On Implementing Time

draft-aanchal-time-implementation-guidance-01

Abstract

This document describes the properties of different types of clocks available on digital systems. It provides implementors of applications with guidance on choices they have to make when working with time to provide 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 April 25, 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

1. Introduction

It is hard to understate the importance of time in modern digital systems. The functionality and security of applications (distributed or local to one system) and that of network protocols generally hinge on some notion of time. For implementation, these applications and protocols have to choose one of the types of clocks available on their system, each of which has its own specific properties. However, currently many of these applications seem to be oblivious to the implications of choosing one or the other clock for implementation. This behavior can be attributed to: a) the lack of clear understanding of the distinct properties of these clocks, b) trade-offs of using one or the other for an application, and c) availability and compatibility of these clocks on different systems. This document discusses a) and b).

More specifically, in this document we first define different methods used by protocols and applications to express time. We then define properties of clocks maintained by modern digital systems. Next we describe how systems obtain these values from these clocks and the security considerations of using these values to implement protocols and applications that use time. Finally we discuss trade-offs between security and precision of choosing a clock. The document aims to provide guidance to the implementors make an informed choice with an example of POSIX system.

2. Scope of the document

This document aims to provide software developers implementing protocols and applications that have to deal with time with the knowledge and understanding to make informed decisions regarding the available clocks and their respective trade-offs.

It does not describe functionality that is specific to the architecture of a PC, or other devices such as phones, IoT devices, switches, routers, base stations, or synchrophasors. Nor is the document applicable to a specific operating system. Throughout the document we assume that one or the other clock is available on most devices. How these clocks are available on different PCs or other devices is out of scope of this document.

We do not exactly recommend which clock should be used. We discuss the available options and trade-offs. The final decision would vary
depending on the availability of clocks and the security requirements of the specific application under implementation.

Note: Since there is a lack of standards on terminology related to time, we define some terms in the following section. Also, throughout the document, we define the terms as they become relevant. Different systems, depending on their OS, may use different terms for the same types of clocks. A survey on this is not in the scope of this document. We provide a discussion on how to access these values on POSIX and Windows systems. On other systems, implementors will have to determine themselves which of these values are available.

3. Expressing Time

Protocols and applications can express time in several forms, depending on whether they need to express a point in time or a time interval.

3.1. Absolute Time

Absolute time expresses a universally agreed upon reference to a specific point in time. Such a reference can be expressed in different ways. For instance, Unix Time refers to the number of seconds since midnight UTC, January 1st, 1970, while in everyday life, we reference such a point through year, month, day, and so on.

Because absolute time expresses a shared view of time, a system needs to synchronize its clock with a common reference clock, for instance one based on UTC.

Absolute time is often used to express the start or end of the validity of objects with a limited lifetime that are shared over the network.

3.2. Relative Time

Relative time measures the time interval that has elapsed from some well-defined reference point (e.g., 20 minutes from the time of your query).

Since relative time does not express a point in time, it does not rely on synchronized clocks between systems but only on a shared rate of passage of this time.

Relative time is commonly used in network protocols, for instance to determine when a packet should be considered dropped or to express Time To Live (TTL) values that govern the length of time for which an object is valid or usable.
4. 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 constantly updating our own clocks against a common reference clock. Remaining close to this reference clock is a complex process that comes with its own set of difficulties.

In this section, we will have a look at the different clocks a system uses and how it maintains these clocks.

4.1. Native Clock

Each system has its own perception of time. It gains access to it via its native clock. Typically, this clock counts cycles of an oscillator but some systems use process CPU times or thread CPU timers (via timers provided by the CPU). The quality of the native clock therefore depends on either the stability of the oscillator or the CPU timer.

The timescale of the native clock is purely subjective -- no general meaning can be attached to any specific clock value. One can only obtain relative time by comparing two values. Because the value of the native clock always grows at a steady pace, never decreases, never make unexpected jumps, and never skips, the difference between two clock values provides the time interval between the two measurements.

The independence of the native clock from any external time sources renders it resistant to any manipulation but in return there is no guarantee that its clock rate is similar to that of any other system. This difference in rate, especially when compared to a reference clock, is called clock drift.

Clock drift depends on the quality of the clock itself but also on factors such as system load or ambient temperatur which makes it hard to predict.

4.2. World Clock

The native clock only provides means to measure relative time. In order to be able to also process absolute time, the system needs to be synchronized with a global reference clock. Since this clock strives to be the same on all systems, we call it the world clock.

There are a number of ways to maintain the world clock based on the system's native clock.
The first is to manually maintain an offset between values of the native clock and the reference world clock. Because of the clock drift of the native clock, this offset needs to be updated from time to time if a minimal divergence from the reference clock is to be maintained.

Secondly, a hardware clock provided by the system and set to be equivalent to the reference time can be used, allowing the system to retain the offset across reboots.

Finally, the reference clock can be obtained from an external time source. Typically, the Internet is used through a variety of timing protocols including the Network Time Protocol (NTP) [RFC5905], Chrony, SNTP, OpenNTP and others.

Each of these approaches has own problems attached to it.

Manual configurations can be subject to errors and misconfiguration. Also, for mobile devices, when moving between time zones, the offset must be corrected manually.

Accessing the hardware clock requires an I/O operation which is resource intensive, therefore many systems use the hardware clock only upon reboot, to initialize the clock offset; subsequent updates are made either manually or through timing protocols.

Further, on many systems the quality of the hardware clock isn’t very high, leading to a large clock drift if solely relying on it. Worse, systems like microcontrollers that operate within embedded systems (e.g., Raspberry Pi, Arduino, etc.) often lack hardware clocks altogether. These systems rely on external time sources upon reboot and have no means to process absolute time until synchronization with these sources has completed.

Relying on Internet timing protocols opens up the system time to attack. Recent papers show vulnerabilities in NTP [SECNTP], [MCGP] and SNTP that allow attackers to maliciously alter system’s world clock -- pushing it into the past or even into the future. Moreover, many of these time-shifting attacks can be performed by off-path attackers, who do not occupy a privileged position on the network between the victim system and its time sources on the Internet. Researchers have also demonstrated off-path denial of service attacks on timing protocols that prevent systems from synchronizing their clocks.

In other words, the process of obtaining the offset necessary to provide a world clock creates dependencies that can be exploited.
5. Implementation Approaches

Because absolute time relies on a shared interpretation of a value expressing time, the world clock is necessary when processing such values.

For relative time, however, where only the rate of passage of time needs to be close enough to that of the other systems involved, there is no need to rely on the world clock when determining whether an interval has passed.

Instead, by obtaining a value from the native clock when the interval has started only the native clock is necessary to determine when this interval ends. As the native clock does not rely on any external time sources, the implementation becomes resistant to the difficulties of coordinating with these sources.

However, using the native clock in this way comes with a caveat. Since the native clock is not subject to any adjustments by timing protocols, it is not adjusted for the error introduced by clock drift. While this is likely of little consequence for short intervals, it may become significant for intervals that span long periods of time.

Consequently, the choice of clock to be used is application-specific. If applications can tolerate a certain amount of clock drift or if the time intervals are short, implementers may prefer using the native clock. If the application relies on precise timing over long periods one has no choice but to fall back to the world clock.

6. Accessing the Native Clock on Selected Operating Systems

In most operating systems, the standard functions to access time use the world clock since that is normally what users would expect. This section provides an overview how the native clock can be accessed on some common operating systems.

6.1. POSIX

POSIX defines a system C API function which may provide native 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.
6.2. Microsoft Windows

In the Microsoft Windows operating system, native time is called 'Windows Time' and can be accessed through the GetTickCount and GetTickCount64 API functions. The returned value is normally the number of milliseconds since system start. GetTickCount will return a 32 bit value while GetTickCount64 returns a value 64 bits wide that will wrap around less often.

7. Acknowledgements

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

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

Time is a fundamental component for the security guarantees claimed by various applications. A system that uses a time distribution protocol may be affected by the security aspects of the time protocol. The security considerations of time protocols in general are discussed in [RFC7384]. This document discusses the security considerations with respect to implementing time values in applications in various sections.

10. Informative References

[CLOCKDRIFT]


Authors’ Addresses

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

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany
Email: kristof.teichel@ptb.de

Martin Hoffmann
NLnet Labs
Science Park 400
Amsterdam 1098 XH
Netherlands
Email: martin@nlnetlabs.nl
Network Time Protocol Best Current Practices
draft-ietf-ntp-bcp-13

Abstract

The Network Time Protocol (NTP) is one of the oldest protocols on the Internet and has been widely used since its initial publication. This document is a collection of Best Practices for general operation of NTP servers and clients on the Internet. It includes recommendations for stable, accurate and secure operation of NTP infrastructure. This document is targeted at NTP version 4 as described in RFC 5905.

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, 2019.

Copyright Notice

Copyright (c) 2019 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
   2.1. BCP 38 ........................................... 3
3. NTP Configuration Best Practices ............................... 4
   3.1. Keeping NTP up to date ............................. 4
   3.2. Use enough time sources ............................. 4
   3.3. Use a diversity of Reference Clocks .................. 5
   3.4. Control Messages .................................. 6
   3.5. Monitoring ......................................... 7
   3.6. Using Pool Servers ................................ 7
   3.7. Leap Second Handling ................................ 8
   3.7.1. Leap Smearing ................................... 9
4. NTP Security Mechanisms .................................... 10
   4.1. Pre-Shared Key Approach ............................. 10
   4.2. Autokey ............................................. 11
   4.3. Network Time Security ............................... 11
   4.4. External Security Protocols ........................ 11
5. NTP Security Best Practices ................................ 11
   5.1. Minimizing Information Leakage ...................... 11
   5.2. Avoiding Daemon Restart Attacks .................... 12
   5.3. Detection of Attacks Through Monitoring .............. 14
   5.4. Kiss-o’-Death Packets ................................ 14
   5.5. Broadcast Mode Should Only Be Used On Trusted Networks 15
   5.6. Symmetric Mode Should Only Be Used With Trusted Peers 15
6. NTP in Embedded Devices ................................... 15
   6.1. Updating Embedded Devices ........................... 16
   6.2. Server configuration ................................ 16
7. NTP over Anycast ......................................... 16
8. Acknowledgments .......................................... 18
9. IANA Considerations ....................................... 18
10. Security Considerations ................................... 18
11. References .............................................. 18
   11.1. Normative References ............................... 18
   11.2. Informative References ............................. 19
   11.3. URIs ............................................ 21
Appendix A. Best Practices specific to the Network Time
Foundation implementation ................................... 21
   A.1. Use enough time sources ............................. 22
   A.2. NTP Control and Facility Messages .................. 22
1. Introduction

NTP version 4 (NTPv4) has been widely used since its publication as [RFC5905]. This document is a collection of best practices for the operation of NTP clients and servers.

The recommendations in this document are intended to help operators distribute time on their networks more accurately and more securely. It is intended to apply generally to a broad range of networks. Some specific networks may have higher accuracy requirements that require additional techniques beyond what is documented here.

Among the best practices covered are recommendations for general network security, time protocol specific security, and NTP server and client configuration. NTP operation in embedded devices is also covered.

This document also contains information for protocol implementors who want to develop their own implementations that are compliant to RFC 5905.

1.1. Requirements Language

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


2.1. 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 connection-oriented protocols. NTP control messages can generate a lot of data in response to a small query, which makes it attractive as a vector for distributed denial-of-service attacks. (NTP Control messages are discussed further in Section 3.4). One documented instance of such an attack
can be found here [1], and further discussion in [IMC14] and [NDSS14].

BCP 38 [RFC2827] was published in 2000 to provide some level of remediation against address-spoofing attacks. BCP 38 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 ISP’s and large corporate networks implement ingress and egress filtering. More information is available at the BCP38 Info Web page [2].

3. NTP Configuration Best Practices

This section provides Best Practices for NTP configuration and operation. Application of these best practices that are specific to the Network Time Foundation implementation, including example configuration directives valid at the time of this writing, are compiled in Appendix A.

3.1. Keeping NTP up to date

There are multiple versions of the NTP protocol in use, and multiple implementations, on many different platforms. The practices in this document are meant to apply generally to any implementation of [RFC5905]. NTP users should 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.

3.2. Use enough time sources

An NTP implementation that is compliant with [RFC5905] takes the available sources of time and submits this timing data to sophisticated intersection, clustering, and combining 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 [RFC5905] or the detailed description of NTP in [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, it will be difficult
to know which one is correct without making use of information from outside of the protocol.

- If there are 3 sources of time, there is more data available to converge on the best calculated 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.

- 4 or more sources of time is better, as long as the sources are diverse (Section 3.3). If one of these sources develops a problem there are still at least 3 other time sources.

This analysis assumes that a majority of the servers used in the solution are honest, even if some may be inaccurate. Operators should be aware of the possibility that if an attacker is in control of the network, the time coming from all servers could be compromised.

Operators who are concerned with maintaining accurate time SHOULD use at least 4 independent, diverse sources of time. Four sources will provide sufficient backup in case one source goes down. If four sources are not available, operators MAY use fewer sources, subject to the risks outlined above.

But even with 4 or more sources of time, systemic problems can happen. One example involves the leap smearing concept detailed in Section 3.7.1. For several hours before and after the June 2015 leap second, several operators configured their NTP servers with leap smearing while others did not. 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-smereared time. This is just one of many potential causes of disagreement among time sources.

Operators are advised to monitor all time sources that are in use. If time sources do not generally agree, operators are encouraged to investigate the cause of this and either correct the problems or stop using defective servers. See Section 3.5 for more information.

3.3. Use a diversity of Reference Clocks

When using servers with attached hardware reference clocks, it is suggested that different types of reference clocks be used. Having a diversity of sources with independent implementations 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. Even devices from different vendors may not be truly independent if they share common elements. Are they using the same base chipset? 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. When having the correct time is of critical importance, it’s ultimately up to operators to ensure that their sources are sufficiently independent, even if they are not under the operator’s control.

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. Depending on the application requirements, operators may need to consider backup scenarios in the rare circumstance when the satellite system is faulty or unavailable.

3.4. Control Messages

Some implementations of NTPv4 provide the NTP Control Messages (also known as Mode 6 messages) that were originally specified in Appendix B of [RFC1305] which defined NTPv3. These messages were never included the NTPv4 specification, but they are still used. At the time of this writing, work is being done to formally document the structure of these control messages in [I-D.ietf-ntp-mode-6-cmds].

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. But these facilities can be a vector for amplification attacks when abused. For this reason, it is RECOMMENDED that publicly-facing NTP servers should block NTP Control Message queries from outside their organization.

The ability to use NTP Control Messages beyond their basic monitoring capabilities SHOULD be limited to authenticated sessions that provide a ‘controlkey’. It can also be limited through mechanisms outside of the NTP specification, such as Access Control Lists, that only allow access from approved IP addresses.

The NTP Control Messages responses are much larger than the corresponding queries. Thus, they can be abused in high-bandwidth DDoS attacks. Section 2.1 gives more information on how to provide protection for this abuse by implementing BCP 38.
3.5. Monitoring

Operators SHOULD use their NTP implementation’s remote monitoring capabilities to quickly identify servers which are out of sync, and ensure correctness of the service. Operators SHOULD also monitor system logs for messages so problems and abuse attempts can be quickly identified.

If a system starts to receive NTP Reply packets from a remote time server that do not correspond to any requests sent by the system, that can be an indication that an attacker is forging that system’s IP address in requests to the remote time server. The goal of this attack is to adversely impact the availability of time to the targeted system whose address is being forged. Based on these forged packets, the remote time server might decide to throttle or rate limit packets, or even stop sending packets to the targeted system.

If a system is a broadcast client and its system log 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.

(Broadcast client and server modes are defined in Section 3 of [RFC5905])

If a server’s system log shows messages that indicates it is receiving NTP timestamps that are much 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.

3.6. 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. Network operators and advanced users who want to synchronize their computers MUST 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 this pool, please review their instructions at http://www.pool.ntp.org/en/use.html [3].

Vendors can obtain their own subdomain that is part of the NTP Pool Project. This offers vendors the ability to safely make use of the time distributed by the pool for their devices. Details are available at http://www.pool.ntp.org/en/vendors.html [4].
If there is a need to synchronize many computers, an operator may want to run local NTP servers that are synchronized to the NTP Pool Project. NTP users on that operator’s networks can then be synchronized to local NTP servers.

3.7. Leap Second Handling

UTC is kept in agreement with the astronomical time UT1 [5] 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 International Earth Rotation and Reference Systems Service (IERS) in its Bulletin C [6] 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 Global Navigation Satellite Systems (like GPS) or radio transmitters (like DCF77) broadcast leap second information. If an NTP client is synced to an NTP server that provides leap second notification, the client will get advance notification of impending leap seconds automatically.

Since the length of the UT1 day is generally slowly increasing [7], 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, if that ever becomes 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. When implementing this recommendation, operators should ensure that clients are not configured to use polling intervals greater than 24 hours, so the leap second notification is not missed.

In circumstances where an NTP server is not receiving leap second information from an automated source, certain organizations maintain files which are updated every time a new leap second is announced:

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

Some NTP installations make use of a technique called Leap Smearing. With this method, instead of introducing an extra second (or eliminating a second) on a leap second event, 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.

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.

Operators who have legal obligations or other strong requirements to be synchronized with UTC or civil time SHOULD NOT use leap smearing, because the distributed time cannot be guaranteed to be traceable to UTC during the smear interval.

Clients that are connected to leap smearing servers MUST NOT apply the standard NTP leap second handling. 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.

Any use of leap smearing servers should be limited to within a single, well-controlled environment. 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, and there is no standardized way for a client to detect that a server is using leap smearing. 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 NOT 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.

4. 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 threat analysis for time synchronization protocols is given in [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.

4.1. 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 MUST be exchanged securely by external means, and they MUST be protected from disclosure. It is RECOMMENDED that each association be protected by its own unique key. It is RECOMMENDED that participants agree to refresh keys periodically. However, NTP does not provide a mechanism to assist in doing so. Each communication partner will need to keep track of its keys in its own local key storage.

[RFC5905] specifies using the MD5 hash algorithm for calculation of the MAC, but other algorithms may be supported as well. The MD5 hash is now considered to be too weak and unsuitable for cryptographic usage. [RFC6151] has more information on the algorithm’s weaknesses. Implementations will soon be available based on AES-128-CMAC [I-D.ietf-ntp-mac], and users SHOULD use that when it is available.

Some implementations store the key in clear text. Therefore it MUST only be readable by the NTP process.

An NTP client has to be able to link a key to a particular server in order to establish a protected association. This linkage is implementation specific. Once applied, a key will be trusted until the link is removed.
4.2. Autokey

[RFC5906] specifies the Autokey protocol. It was published in 2010 to provide automated key management and authentication of NTP servers. However, security researchers have identified vulnerabilities [8] in the Autokey protocol.

Autokey SHOULD NOT be used.

4.3. Network Time Security

Work is in progress on an enhanced replacement for Autokey. Refer to [I-D.ietf-ntp-using-nts-for-ntp] for more information.

4.4. External Security Protocols

If applicable, external security protocols such as IPsec and MACsec can be applied to enhance integrity and authenticity protection of NTP time synchronization packets. Usage of such external security protocols can decrease time synchronization performance [RFC7384]. Therefore, operators are advised to carefully evaluate if the decreased time synchronization performance meets their specific timing requirements.

Note that none of the security measures described in Section 4 can prevent packet delay manipulation attacks on NTP. Such delay attacks can target time synchronization packets sent as clear-text or even within an encrypted tunnel. These attacks are described further in Section 3.2.6 of [RFC7384].

5. NTP Security Best Practices

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

5.1. 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
As such, mechanisms outside of the NTP protocol, such as Access Control Lists, SHOULD be used to limit the exposure of this information to allowed IP addresses, and keep it from remote attackers not on the list. Hosts SHOULD only respond to NTP control queries from authorized parties.

An NTP client that does not provide time on the network can additionally log and drop incoming mode 3 timing queries from unexpected sources. Note well that the easiest way to monitor the status of an NTP instance is to send it a mode 3 query, so it may not be desirable to drop all mode 3 queries. As an alternative, operators SHOULD either filter mode 3 queries from outside their networks, 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. An exception to this can be made if a leaf-node host is being actively monitored, in which case incoming packets from the monitoring server can be allowed.

Please refer to [I-D.ietf-ntp-data-minimization] for more information.

5.2. Avoiding Daemon Restart Attacks

[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 as described in [NDSS16], when the following two conditions hold:

1. The operating system automatically restarts the NTP client 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. As of the time of this writing, it appears likely to become the default behavior in other systems as they migrate legacy init scripts to process supervisors such as systemd.)

2. The NTP client is configured to ignore the panic threshold on all restarts.

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

Operators need to be aware that when operating with the above two conditions, the panic threshold offers no protection from attacks. The natural solution is not to run hosts with these conditions. Specifically, operators SHOULD NOT ignore the panic threshold in all cold-start situations unless sufficient oversight and checking is in place to make sure that this type of attack cannot happen.

As an alternative, the following steps MAY be taken by operators to mitigate the risk of attack:

- Monitor the 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.
- Configure the ntp client to only ignore the panic threshold in a cold start situation.
- Increase the minimum number of servers required before the NTP client adjusts the system clock. This will make the NTP client wait until enough trusted sources of time agree before declaring the time to be correct.

In addition, the following steps SHOULD be taken by those who implement the NTP protocol:

- Prevent the NTP daemon from taking time steps that set the clock to a time earlier than the compile date of the NTP daemon.
- Prevent the NTP daemon from putting ‘INIT’ in the reference ID of its NTP packets upon initializing. This will make it more difficult for attackers to know when the daemon reboots.
5.3. Detection of Attacks Through Monitoring

Operators 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.

5.4. Kiss-o’-Death Packets

The "Kiss-o’-Death" (KoD) packet includes a rate management mechanism where a server can tell a misbehaving client to reduce its query rate. KoD packets in general (and the RATE packet in particular) are defined in Section 7.4 of [RFC5905]. It is RECOMMENDED that all NTP devices 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 an exposed 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 [RFC5905]. If the client uses the poll interval value sent by the server in the RATE 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 rate management mechanism relies on clients behaving properly in order to be effective. Some clients ignore the RATE packet entirely, and other poorly-implemented clients might unintentionally increase their poll rate and simulate a denial of service attack. Server administrators are advised to be prepared for this and take measures outside of the NTP protocol to drop packets from misbehaving clients when these clients are detected.
Kiss-o’-Death (KoD) packets can be used in denial of service attacks. Thus, the observation of even just one RATE packet with a high poll value could be sign that the client is under attack. And KoD packets are commonly accepted even when not cryptographically authenticated, which increases the risk of denial of service attacks.

5.5. Broadcast Mode Should Only Be Used On Trusted Networks

Per [RFC5905], NTP’s broadcast mode is authenticated using symmetric key cryptography. The broadcast server and all 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 have to 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 clients have to trust each other. Broadcast mode SHOULD only be run from within a trusted network.

5.6. 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 SHOULD only allow symmetric passive associations to be established with trusted peers. Specifically, a host SHOULD require each of its symmetric passive association to be cryptographically authenticated. Each symmetric passive association SHOULD be authenticated under a different cryptographic key.

6. NTP in Embedded Devices

As computing becomes more ubiquitous, there will be many small embedded devices that require accurate time. These devices may not have a persistent battery-backed clock, so using NTP to set the
correct time on power-up may be critical for proper operation. These 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.

6.1. Updating Embedded Devices

Vendors of embedded devices are advised to pay attention to the current state of protocol security issues and bugs in their chosen implementation, 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 an up-to-date and secure NTP implementation.

Vendors of embedded devices SHOULD include the ability to update the list of NTP servers used by the device.

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].

6.2. Server configuration

Vendors of embedded devices with preconfigured NTP servers need to carefully consider which servers to use. 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 MUST only preconfigure NTP servers that they have permission to use.

Vendors are encouraged to invest resources into providing their own time servers for their devices to connect to. This may be done through the NTP Pool Project, as documented in Section 3.6.

Vendors should read [RFC4085], which advises against embedding globally-routable IP addresses in products, and offers several better alternatives.

7. NTP over Anycast

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

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 many 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 that using a single anycast address for NTP presents its own potential issues. 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. This can be mitigated by creating multiple, separate anycast pools so clients can have multiple sources of time while still gaining the configuration benefits of the anycast pools.

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 enter and leave the network, or the network topology changes, 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. In extreme cases where the network topology is changing rapidly, this could cause the server seen by a client to rapidly change as well, which can lead to larger time inaccuracies. It is RECOMMENDED that network operators only deploy anycast NTP in environments where operators know these small shifts can be tolerated by the applications running on the clients being synchronized in this manner.

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 an alternative method of access, such as an alternate Unicast IP address, 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, or a single interface which can support multiple addresses. 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.

8. Acknowledgments

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, Robert Nagy, and Brian Haberman.

9. IANA Considerations

This memo includes no request to IANA.

10. Security Considerations

Time is a fundamental component of security on the internet. The absence of a reliable source of current time subverts many common web authentication schemes, e.g., by allowing the use of expired credentials or by allowing for replay of messages only intended to be processed once.

Much of this document directly addresses how to secure NTP servers. In particular, see Section 2, Section 4, and Section 5.

There are several general threats to time synchronization protocols which are discussed in [RFC7384].

[I-D.ietf-ntp-using-nts-for-ntp] specifies the Network Time Security (NTS) mechanism and applies it to NTP. Readers are encouraged to check the status of the draft, and make use of the methods it describes.

11. References

11.1. Normative References

Internet-Draft  Network Time Protocol BCP  March 2019


11.2. Informative References


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

[I-D.ietf-ntp-mac]

[I-D.ietf-ntp-mode-6-cmds]

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

[IMC14]

[MILLS2006]

[NDSS14]

[NDSS16]

[RFC1305]

[RFC5906]
11.3. URIs

[12] https://support.ntp.org/bin/view/Support/ConfiguringNTP

Appendix A. Best Practices specific to the Network Time Foundation implementation

The Network Time Foundation (NTF) provides a widely used implementation of NTP, known as ntpd [10]. It is an evolution of the first NTP implementations developed by David Mills at the University

of Delaware. This appendix contains additional recommendations specific to this implementation that are valid at the time of this writing.

A.1. Use enough time sources

In addition to the recommendation given in Section 3.2 the ntpd implementation provides the ‘pool’ directive. Starting with ntp-4.2.6, using this directive in the ntp.conf file will spin up enough associations to provide robust time service, and will disconnect poor servers and add in new servers as-needed. The ‘minclock’ and ‘maxclock’ options of the ‘tos’ command may be used to override the default values of how many servers are discovered through the ‘pool’ directive.

A.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:

```
restrict default -4 nomodify notrap nopeer noquery
restrict default -6 nomodify notrap nopeer noquery
restrict source nomodify notrap noquery
```
A.3. Monitoring

The ntpd implementation 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:

```
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.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, substituting the proper path:

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

There may need to restart ntpd to apply this change.

ntpd 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.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 http://bk.ntp.org/ntp-stable/README.leapsmear?PAGE=anno [11].
A.6. Configuring ntpd

See https://support.ntp.org/bin/view/Support/ConfiguringNTP [12] for additional information on configuring ntpd.

A.7. Pre-Shared Keys

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

keyid type key

where "keyid" is a number between 1 and 65534, inclusive, "type" is an ASCII character which defines the key format, and "key" is the key itself.

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

server address key keyid

substituting the server address in the "address" field and the numerical keyid to use with that server in the "keyid" field.

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

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]).

Authors’ Addresses

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

Email: denis.reilly@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
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 September 26, 2019.

Copyright Notice

Copyright (c) 2019 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 synchronizaton. 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 SHOULD be set to either the actual polling interval as specified by RFC 5905 or 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 SHOULD be 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

[1] https://github.com/openbsd/src/commit/1346900e6d0ac3aeb0e3f9eb60b94c66586978c6

Appendix A. Acknowledgements

The possibility of minimizing data in client packets was described in RFC 2030 [RFC2030]. The authors would like to acknowledge Alexander Guy for pioneering the idea of randomization of all bits of the transmit timestamp in the rdate program of the OpenBSD project as early as May 2004 [1].

The authors would also 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
NTP Interleaved Modes
draft-ietf-ntp-interleaved-modes-02

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 November 24, 2019.

Copyright Notice

Copyright (c) 2019 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 a client/server, symmetric, and broadcast mode. The transmit and receive timestamps are two of the four timestamps included in every NTPv4 packet used for time synchronization.

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

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

1. User space (software)
2. Network device driver or kernel (software)
3. Data link layer (hardware - MAC chip)
4. Physical layer (hardware - PHY chip)

Software timestamps captured in the user space in the NTP implementation itself are least accurate. They do not include system calls used for sending and receiving packets, processing and queuing delays in the system, network device drivers, and hardware. Hardware timestamps captured at the physical layer are most accurate.

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

The protocol described in RFC 5905 does not specify any mechanism for a server to provide its clients and peers with a more accurate transmit timestamp that is known only after the transmission. A packet that strictly follows RFC 5905, i.e., it contains a transmit timestamp corresponding to the packet itself, is said to be in basic mode.
Different mechanisms could be used to exchange timestamps known after
the transmission. The server could respond to each request with two
packets. The second packet would contain the transmit timestamp
corresponding to the first packet. However, such a protocol would
enable a traffic amplification, or it would use packets with an
asymmetric length, which would cause an asymmetry in the network
delay and an error in the measured offset.

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

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

This document assumes familiarity with RFC 5905.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "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.

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

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

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

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

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

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

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

If the conditions are not met, the server MUST NOT respond in the interleaved mode. The server MAY always respond in the basic mode. In any case, the server SHOULD save the new receive and transmit timestamps.
The first request from a client is always in the basic mode and so is the server response. It has a zero origin timestamp and zero receive timestamp. Only when the client receives a valid response from the server, it will be able to send a request in the interleaved mode.

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

An example of packets in a client/server exchange using the interleaved mode is shown in Figure 1. The packets in the basic and interleaved mode are indicated with B and I respectively. The timestamps \( t_1' \), \( t_3' \) and \( t_{11}' \) point to the same transmissions as \( t_1 \), \( t_3 \) and \( t_{11} \), 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 \( t_6 \) and \( t_7 \) (e.g. they were dropped to save timestamps for other clients using the interleaved mode).

```
Server   t2   t3               t6   t7              t10  t11
-----+----+----------------+----+----------------+----+-----
      / \                /      \              /      \
Client /        \            /        \            /        \
  +----------+----------+----------+----------+----------+--
  t1         t4         t5         t8         t9        t12
  Mode: B         B           I         I           I         B
  +----+    +----+      +----+    +----+      +----+    +----+
  Org | 0  |    | t1'   |      | t2 |    | t4 |      | t6 |    | t5 |
  Rx  | 0   | t2   | t4   | t6   | t8 | t10 |
  Tx  | t1' | t3'  | t4   | t6   | t8 | t5  |
```

Figure 1: Packet timestamps in interleaved client/server mode

When the client receives a response from the server, it performs the tests described in RFC 5905. Two of the tests are modified for the interleaved mode:
1. The check for duplicate packets SHOULD compare both receive and transmit timestamps in order to not drop a valid response in the interleaved mode if it follows a response in the basic mode and they contain the same transmit timestamp.

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

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

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

\[
\begin{align*}
T_1 & \text{ - local transmit timestamp of the previous request (t1)} \\
T_2 & \text{ - remote receive timestamp from the previous response (t2)} \\
T_3 & \text{ - remote transmit timestamp from the latest response (t3)} \\
T_4 & \text{ - local receive timestamp of the previous response (t4)}
\end{align*}
\]

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.

\[
\begin{align*}
T_1 & \text{ - local transmit timestamp of the latest request (t5)} \\
T_2 & \text{ - remote receive timestamp from the latest response (t6)} \\
T_3 & \text{ - remote transmit timestamp from the latest response (t3)} \\
T_4 & \text{ - local receive timestamp of the previous response (t4)}
\end{align*}
\]

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

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

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

3. Interleaved Symmetric mode

The interleaved symmetric mode uses the same principles as the interleaved client/server mode. A packet in the interleaved symmetric mode has a transmit timestamp which corresponds to the previous packet sent to the peer and an origin timestamp equal to the receive timestamp from the last packet received from the peer.

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

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

A peer A SHOULD send a peer B a packet in the interleaved mode only when the following conditions are met:
1. The peer A has an active association with the peer B which was specified with an option enabling the interleaved mode, OR the peer A received at least one valid packet in the interleaved mode from the peer B.

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

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

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

```
  Peer A: t2 t3 t6 t8 t9 t12 t14 t15
  Peer B: t1 t4 t5 t7 t10 t11 t13 t16

Mode: B B I I B B B B
Org | 0 | t1’ | t2 | t3’ | t4 | t3’ | t3 | t10
Rx  | 0 | t2 | t4 | t4 | t8 | t10 | t10 | t14
Tx  | t1’ | t3’ | t1 | t7’ | t3 | t11’ | t13’ | t9
```

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 T1 and T4 in order to reduce the error in the measured delay.
4. Interleaved Broadcast mode

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

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

Server  
+--------+--------+
<table>
<thead>
<tr>
<th>t1</th>
<th>t3</th>
</tr>
</thead>
</table>
\---------\---------

Client  
+--------+--------+
<table>
<thead>
<tr>
<th>t2</th>
<th>t4</th>
</tr>
</thead>
</table>
\---------\---------

Mode:  
+----+       +----+       +----+       +----+
| B  |       | I   |       | I   |       | I   |
+----+       +----+       +----+       +----+

Org  
| 0  |       | t1  |       | t3  |       | t5  |

Rx   
| 0  |       | 0   |       | 0   |       | 0   |

Tx   
| t1' |       | t3' |       | t5' |       | t7' |
+----+       +----+       +----+       +----+

Figure 3: Packet timestamps in interleaved broadcast mode

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

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

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

5. Acknowledgements

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

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

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

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

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

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

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

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

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

8. References

8.1. Normative References


8.2. Informative References


8.3. URIs


Authors’ Addresses

Miroslav Lichvar
Red Hat
Purkynova 115
Brno 612 00
Czech Republic

Email: mlichvar@redhat.com
Abstract

This document describes the structure of the control messages that were historically used with the Network Time Protocol before the advent of more modern control and management approaches. These control messages have been used to monitor and control the Network Time Protocol application running on any IP network attached computer. The information in this document was originally described in Appendix B of RFC 1305. The goal of this document is to provide a current, but historic, description of the control messages as described in RFC 1305 and any additional commands implemented in NTP.

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

Status of This Memo

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

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

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

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

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction ........................................... 3
   1.1. Control Message Overview ........................... 3
   1.2. Remote Facility Message Overview ................... 4
2. NTP Control Message Format ............................... 5
3. Status Words ............................................. 7
   3.1. System Status Word ................................ 8
   3.2. Peer Status Word .................................. 10
   3.3. Clock Status Word ................................ 12
   3.4. Error Status Word ................................ 12
4. Commands .................................................. 13
5. IANA Considerations ...................................... 16
6. Security Considerations .................................. 16
7. Contributors .............................................. 18
8. Acknowledgements ........................................ 18
9. References ............................................... 18
   9.1. Normative References ................................ 18
   9.2. Informative References ............................. 19
Appendix A. NTP Remote Facility Message Format ............. 19
Author's Address ............................................ 21
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.

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

1.1. Control Message Overview

The NTP 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 [RFC0791]; however, some commands or responses may involve more data than will fit into a single datagram. Accordingly, a simple reassembly feature is included in which each octet of the message data is numbered starting with zero. As each fragment is transmitted the number of its first octet is inserted in the offset field and the number of octets is inserted in the count field. The more-data (M) bit is set in all fragments except the last.

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

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

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

Commands need not necessarily be sent by an NTP peer, so ordinary access-control procedures may not apply; however, the optional mask/match mechanism suggested 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 (ntpdc) 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.

```
+---------------+---------------+---------------+---------------+
| LI | VN | Mode | OpCode | Sequence Number |
+---------------+---------------+---------------+---------------+
| Status | Association ID |
+---------------+---------------+---------------+---------------+
| Offset | Count |
+---------------+---------------+---------------+---------------+
| Data (up to 468 bytes) |
+---------------+---------------+---------------+---------------+
| Padding (optional) |
+---------------+---------------+---------------+---------------+
| Authenticator (optional, 96 bits) |
+---------------+---------------+---------------+---------------+
```

Figure 1: NTP Control Message Header

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

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

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

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

Error Bit (E): Set to zero for normal response, one for error response.
More Bit (M): Set to zero for last fragment, one for all others.

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

<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.
### 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>

---

Figure 2: Status Word Formats

---

3.1. System Status Word

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

<table>
<thead>
<tr>
<th>LI</th>
<th>Clock Src</th>
<th>Count</th>
<th>Code</th>
<th>System Status Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-------------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>SEL</th>
<th>Count</th>
<th>Code</th>
<th>Peer Status Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clock Status</th>
<th>Code</th>
<th>Radio Status Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Reserved</th>
<th>Error Status Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Count</th>
<th>Code</th>
<th>Clock Status Word</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Haberman Expires March 31, 2019 [Page 8]
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:
### Code | Meaning
---|---
0 | unspecified
1 | frequency correction (drift) file not available
2 | frequency correction started (frequency stepped)
3 | spike detected and ignored, starting stepout timer
4 | frequency training started
5 | clock synchronized
6 | system restart
7 | panic stop (required step greater than panic threshold)
8 | no system peer
9 | leap second insertion/deletion armed for the of the current month
10 | leap second disarmed
11 | leap second inserted or deleted
12 | clock stepped (stepout timer expired)
13 | kernel loop discipline status changed
14 | leapseconds table loaded from file
15 | leapseconds table outdated, updated file needed

#### 3.2. Peer Status Word

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

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

<table>
<thead>
<tr>
<th>Peer Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>configured (peer.config)</td>
</tr>
<tr>
<td>1</td>
<td>authentication enabled (peer.authenable)</td>
</tr>
<tr>
<td>2</td>
<td>authentication okay (peer.authentic)</td>
</tr>
<tr>
<td>3</td>
<td>reachability okay (peer.reach != 0)</td>
</tr>
<tr>
<td>4</td>
<td>broadcast association</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:
<table>
<thead>
<tr>
<th>Sel</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rejected</td>
</tr>
<tr>
<td>1</td>
<td>discarded by intersection algorithm</td>
</tr>
<tr>
<td>2</td>
<td>discarded by table overflow (not currently used)</td>
</tr>
<tr>
<td>3</td>
<td>discarded by the cluster algorithm</td>
</tr>
<tr>
<td>4</td>
<td>included by the combine algorithm</td>
</tr>
<tr>
<td>5</td>
<td>backup source (with more than sys.maxclock survivors)</td>
</tr>
<tr>
<td>6</td>
<td>system peer (synchronization source)</td>
</tr>
<tr>
<td>7</td>
<td>PPS (pulse per second) peer</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Peer Event Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>unspecified</td>
</tr>
<tr>
<td>1</td>
<td>association mobilized</td>
</tr>
<tr>
<td>2</td>
<td>association demobilized</td>
</tr>
<tr>
<td>3</td>
<td>peer unreachable (peer.reach was nonzero now zero)</td>
</tr>
<tr>
<td>4</td>
<td>peer reachable (peer.reach was zero now nonzero)</td>
</tr>
<tr>
<td>5</td>
<td>association restarted or timed out</td>
</tr>
<tr>
<td>6</td>
<td>no reply (only used with one-shot ntpd -q)</td>
</tr>
<tr>
<td>7</td>
<td>peer rate limit exceeded (kiss code RATE received)</td>
</tr>
<tr>
<td>8</td>
<td>access denied (kiss code DENY received)</td>
</tr>
<tr>
<td>9</td>
<td>leap second insertion/deletion at month’s end armed by peer vote</td>
</tr>
<tr>
<td>10</td>
<td>became system peer (sys.peer)</td>
</tr>
<tr>
<td>11</td>
<td>reference clock event (see clock status word)</td>
</tr>
<tr>
<td>12</td>
<td>authentication failed</td>
</tr>
<tr>
<td>13</td>
<td>popcorn spike suppressed by peer clock filter register</td>
</tr>
<tr>
<td>14</td>
<td>entering interleaved mode</td>
</tr>
<tr>
<td>15</td>
<td>recovered from interleave error</td>
</tr>
</tbody>
</table>
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:
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 <<identifier>>[=<<value>>],<<identifier>>[=<<value>>],... where <<identifier>> is the ASCII name of a system or peer variable specified in RFC 5905 and <<value>> is expressed as a decimal, hexadecimal or string constant in the syntax of the C programming language. Where no ambiguity exists, the <169>sys.<170> or <169>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 follows: IPv4 addresses are written in the form [n.n.n.n], where n is in decimal notation and the brackets are optional; IPv6 addresses are formulated based on the guidelines defined in [RFC5952]. Timestamps, including reference, originate, receive and transmit values, as well as the logical clock, are represented in units of seconds and fractions, preferably in hexadecimal notation. Delay, offset, dispersion and distance values are represented in units of milliseconds and fractions, preferably in decimal notation. All other values are represented as-is, preferably in decimal notation.

Implementations may define variables other than those described in RFC 5905. Called extramural variables, these are distinguished by the inclusion of some character type other than alphanumeric or <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. Inclusion of the nonce by a management agent demonstrates to the server that the agent can receive datagrams sent to the source address of the request, making source address "spoofing" more difficult in a similar way as TCP’s three-way handshake.

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

5. IANA Considerations

This document makes no request of IANA.

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

6. Security Considerations

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

NTP’s control query interface allows reading and writing of system, peer, and clock variables remotely from arbitrary IP addresses using commands mentioned in Section 4. Traditionally, overwriting these variables, but not reading them, requires authentication by default. However, this document argues that an NTP host must authenticate all control queries and not just ones that overwrite these variables. Alternatively, the host can use 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-DOS]). These attacks only use
NTP as a vector for DoS attacks on other protocols, but do not affect the time service on the NTP host itself. To limit the sources of these malicious commands, NTP server operators are recommended to deploy ingress filtering [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. The specific attack is described in [CVE-SPOOF].

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

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

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

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.

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

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

7. Contributors

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

8. Acknowledgements

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

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

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

9. References

9.1. Normative References

Specification, Implementation and Analysis", RFC 1305,
DOI 10.17487/RFC1305, March 1992,
9.2. Informative 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.

Author’s Address

Brian Haberman (editor)
JHU

Email: brian@innovationslab.net
Guidelines for Defining Packet Timestamps
draft-ietf-ntp-packet-timestamps-06

Abstract

Various network protocols make use of binary-encoded timestamps that are incorporated in the protocol packet format, referred to as packet timestamps for short. This document specifies guidelines for defining 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.

The rationale behind defining a relatively small set of recommended formats is that it enables significant reuse; network protocols can typically reuse the timestamp format of the Network Time Protocol (NTP) or the Precision Time Protocol (PTP), 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.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 15, 2019.

Copyright Notice

Copyright (c) 2019 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
2. Terminology ................................................... 3
   2.1. Requirements Language .................................. 3
   2.2. Abbreviations .......................................... 3
   2.3. Terms used in this Document ............................ 4
3. Packet Timestamp Specification Template ........................ 4
4. Recommended Timestamp Formats ................................. 5
   4.1. Using a Recommended Timestamp Format .................... 6
   4.2. NTP Timestamp Formats ................................... 6
       4.2.1. NTP 64-bit Timestamp Format ....................... 6
       4.2.2. NTP 32-bit Timestamp Format ....................... 8
   4.3. The PTP Truncated Timestamp Format ...................... 9
5. Synchronization Aspects ........................................ 11
6. Timestamp Use Cases ........................................... 12
   6.1. Example 1 .............................................. 13
   6.2. Example 2 .............................................. 13
7. Packet Timestamp Control Field ............................... 13
   7.1. High-level Control Field Requirements .................. 14
8. IANA Considerations ........................................... 15
9. Security Considerations ....................................... 15
10. Acknowledgments ............................................... 16
11. References ................................................... 16
    11.1. Normative References .................................. 16
    11.2. Informative References ............................... 16
Internet-Draft              Packet Timestamps              February 2019

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 field that has a fixed size, and are not intended to have a human-friendly format. Packet timestamps are also very common in the RFC series, and are used for example for measuring delay and for synchronizing clocks, e.g., [RFC5905], [RFC4656], and [RFC1323].

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 these recommended timestamp formats. In some cases a network protocol may use more than one of the recommended timestamp formats. However, 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>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
<td>[RFC5905]</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol</td>
<td>[IEEE1588]</td>
</tr>
<tr>
<td>TAI</td>
<td>International Atomic Time</td>
<td></td>
</tr>
<tr>
<td>UTC</td>
<td>Coordinated Universal Time</td>
<td></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.

Synchronization aspects:

The specification of a network protocol that makes use of a packet timestamp is expected to include the synchronization aspects of using the timestamp. While the synchronization aspects are not strictly part of the timestamp format specification, these aspects provide the necessary context for using the timestamp within the scope of the protocol. Further details about synchronization aspects are discussed in Section 5.

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 variety of 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.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+---------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Seconds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+---------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fraction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+---------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

Note: As pointed out in [RFC5905], strictly speaking, UTC did not exist prior to 1 January 1972, but it is convenient to assume it has existed for all eternity. The current epoch implies that the timestamp specifies the number of seconds since 1 January 1972 at 00:00 UTC plus 2272060800 (which is the number of seconds between 1 January 1900 and 1 January 1972).

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. Thus, during and possibly after the occurrence of a leap second, the value of the timestamp may temporarily be ambiguous, as further discussed in Section 5.

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] and [I-D.ietf-sfc-nsh-dc-allocation]. 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.

Note: As pointed out in [RFC5905], strictly speaking, UTC did not exist prior to 1 January 1972, but it is convenient to assume it has existed for all eternity. The current epoch implies that the timestamp specifies the number of seconds since 1 January 1972 at 00:00 UTC plus 2272060800 (which is the number of seconds between 1 January 1900 and 1 January 1972).

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. Thus, during and possibly after the occurrence of a leap second, the value of the timestamp may temporarily be ambiguous, as further discussed in Section 5.

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.

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. Note that the recommended timestamp formats defined in this document (Section 4) do not include the synchronization aspects of these timestamp formats, but it is expected that specifications of network protocols that make use of these formats should include the synchronization aspects. Examples of a Synchronization Aspects 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 and precision.

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, a leap second is a one-second adjustment that is occasionally applied to UTC in order to keep it aligned to the solar time. A leap second may be either positive or negative, i.e., the clock may either be shifted one second forwards or backwards. All leap seconds that have occurred up to the publication of this document have been in the backwards direction, and although forward leap seconds are theoretically possible, the text throughout this document focuses on the common case, which is the backward leap second. 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 the leap second and adjacent time intervals until the new time value catches up. The interval for this process, commonly referred to as leap smear, can range from
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.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Recommended formats</th>
<th>Other format</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTP [RFC5905]</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>OWAMP [RFC4656]</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>TWAMP [RFC5357]</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>TWAMP [RFC8186]</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>TRILL [RFC7456]</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>MPLS [RFC6374]</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>TCP [RFC1323]</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>RTP [RFC3550]</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>IPFIX [RFC7011]</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>[I-D.ietf-ippm-initial-registry]</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>[I-D.ietf-sfc-nsh-dc-allocation]</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

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-ntp-packet-timestamps].

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 4.3 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, Miroslav Lichvar, Denis Reilly, Daniel Franke, Watson Ladd, 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


11.2. Informative References


[IEEE1588v1]

[ITU-T-Y.1731]


Authors' Addresses

Tal Mizrahi
Huawei Network.IO Innovation Lab
Israel

Email: tal.mizrahi.phd@gmail.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/
Network Time Protocol REFID Updates
draft-ietf-ntp-refid-updates-05

Abstract

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 two recognized shortcomings to this approach, and this proposal addresses them. 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.

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH BEFORE PUBLISHING:

The source code and issues list for this draft can be found in https://github.com/hstenn/ietf-ntp-refid-updates

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 26, 2019.
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, as this would cause a timing loop.

1.2. NOT-YOU REFID

The traditional 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 bare minimum 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 its 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 degrading 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, unless or 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 an 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 an 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 [DRAFT-I-DO] protocol this situation cannot happen.

1.4. 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 one of two special IP addresses that is interpreted to mean "not you". The "not you" IP addresses are 127.127.127.127 and 127.127.127.128. If an IPv6 query is received from an address whose four-octet hash equals one of these two addresses and we believe the querying host is not our system peer, the other NOT-YOU address is returned as the REFID.

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. This 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 [DRAFT-REFID-SUGGESTION] or I-DO [DRAFT-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.

4. 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.

5. IANA Considerations

This memo requests IANA to allocate a pseudo Extension Field Type of 0xFFFF so the proposed "I-Do" exchange can report whether or not the "IPv6 REFID Hash" is supported.

6. 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.

7. References

7.1. Normative References


7.2. Informative References


Authors’ Addresses
Abstract

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

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

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 November 1, 2019.
Copyright Notice

Copyright (c) 2019 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 .................................................. 4
   1.1. Objectives ............................................... 4
   1.2. Protocol Overview ....................................... 5
2. Requirements Language .......................................... 7
3. TLS profile for Network Time Security ............................ 7
4. The NTS Key Establishment Protocol ................................ 8
   4.1. NTS-KE Record Types ..................................... 10
      4.1.1. End of Message ..................................... 10
      4.1.2. NTS Next Protocol Negotiation ....................... 11
      4.1.3. Error ............................................... 11
      4.1.4. Warning ............................................ 11
      4.1.5. AEAD Algorithm Negotiation .......................... 12
      4.1.6. New Cookie for NTPv4 ............................... 12
      4.1.7. NTPv4 Server Negotiation ............................ 13
      4.1.8. NTPv4 Port Negotiation .............................. 13
   4.2. Key Extraction (generally) ................................ 14
5. NTS Extension Fields for NTPv4 ................................ 14
   5.1. Key Extraction (for NTPv4) ............................... 14
   5.2. Packet Structure Overview ............................... 15
   5.3. The Unique Identifier Extension Field .................... 15
   5.4. The NTS Cookie Extension Field .......................... 16
   5.5. The NTS Cookie Placeholder Extension Field .............. 16
   5.6. The NTS Authenticator and Encrypted Extension Fields Extension Field ..................... 16
   5.7. Protocol Details ........................................ 19
6. Suggested Format for NTS Cookies ................................ 24
7. IANA Considerations ........................................... 25
   7.1. Service Name and Transport Protocol Port Number Registry 25
   7.2. TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs Registry ................... 25
   7.3. TLS Exporter Labels Registry ............................ 26
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 public key infrastructure, 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.

- Non-amplification: Implementations (especially server implementations) may avoid acting as distributed denial-of-service (DDoS) amplifiers by never responding to a request with a packet larger than the request packet.
Scalability: Server implementations 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. Additionally, for certain message types control mode may require confidentiality as well as authentication. Client-server mode places more stringent requirements on resource utilization than other modes, because servers may have vast number of clients and be unable to afford to maintain per-client state. However, client-server mode also has more relaxed security needs, because only the client requires replay protection: it is harmless for stateless servers to process replayed packets. The security demands of symmetric and control modes, on the other hand, are in conflict with the resource-utilization demands of client-server mode: any scheme which provides replay protection inherently involves maintaining some state to keep track of what messages have already been seen.

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

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

The "NTS Key Establishment" protocol (NTS-KE) is a mechanism for establishing key material for use with the NTS Extension Fields for NTPv4. It uses TLS to exchange keys, provide the client with an initial supply of cookies, and negotiate some additional protocol options. After this exchange, the TLS channel is closed with no per-client state remaining on the server side.
The typical protocol flow is as follows: The client connects to an NTS-KE server on the NTS TCP port and the two parties perform a TLS handshake. Via the TLS channel, the parties negotiate some additional protocol parameters and the server sends the client a supply of cookies along with a list of one or more IP addresses to NTP servers for which the cookies are valid. The parties use TLS key export [RFC5705] to extract key material which will be used in the next phase of the protocol. This negotiation takes only a single round trip, after which the server closes the connection and discards all associated state. At this point the NTS-KE phase of the protocol is complete. Ideally, the client never needs to connect to the NTS-KE server again.

Time synchronization proceeds with one of the indicated NTP servers 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 key exchange server) and an authentication tag, computed using key material extracted from the NTS-KE handshake. The NTP server uses the cookie to recover this key material and send back an authenticated response. The response includes a fresh, encrypted cookie which the client then sends back in the clear in a subsequent request. (This constant refreshing of cookies is necessary in order to achieve NTS’s unlinkability goal.)

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

2. Requirements Language

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

3. TLS profile for Network Time Security

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

Since the NTS protocol is new as of this publication, no backward-compatibility concerns exist to justify using obsolete, insecure, or otherwise broken TLS features or versions. Implementations MUST conform with [RFC7525] or with a later revision of BCP 195. In
particular, failure to use cipher suites that provide forward secrecy will make all negotiated NTS keys recoverable by anyone that gains access to the NTS-KE server’s private certificate. Furthermore:

Implementations MUST NOT negotiate TLS versions earlier than 1.2, SHOULD negotiate TLS 1.3 [RFC8446] or later when possible, and MAY refuse to negotiate any TLS version which has been superseded by a later supported version.

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

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 2. Requests and non-error responses each SHALL include exactly one NTS Next Protocol Negotiation record. The sequence SHALL be terminated by a "End of Message" record. The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting.

Clients and servers MAY enforce length limits on requests and responses, however, servers MUST accept requests of at least 1024 octets and clients SHOULD accept responses of at least 65536 octets.
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 Number: A 15-bit integer in network byte order. The semantics of record types 0-7 are specified in this memo. Additional type numbers SHALL be tracked through the IANA Network Time Security Key Establishment Record Types registry.

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

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

For clarity regarding bit-endianness: the Critical Bit is the most-significant bit of the first octet. In 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]’.

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

Figure 3 provides a schematic overview of the key 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 a zero-length body. It MUST occur exactly once as the final record of every NTS-KE request and response. The Critical Bit MUST be set.
4.1.2. NTS Next Protocol Negotiation

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

The Protocol IDs listed in the client’s NTS Next Protocol Negotiation record denote those protocols which the client wishes to speak using the key material established through this NTS-KE session. 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.

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. It is empty if the server supports none of the algorithms offered. In requests, the list MUST include at least one algorithm. In responses, it MUST include at most one. Honoring the client’s preference order is OPTIONAL: servers may select among any of the client’s offered choices, even if they are able to support some other algorithm which the client prefers more.

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

4.1.6. New Cookie for NTPv4

The New Cookie for NTPv4 record has a Record Type number of 5. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 6 for a suggested construction.
Clients MUST NOT send records of this type. Servers MUST send at least one record of this type, and SHOULD send eight of them, if the Next Protocol Negotiation response record contains Protocol ID 0 (NTPv4) and the AEAD Algorithm Negotiation response record is not empty. The Critical Bit SHOULD NOT be set.

4.1.7. NTPv4 Server Negotiation

The NTPv4 Server Negotiation record has a Record Type number of 6. Its body consists of an ASCII-encoded [ANSI.X3-4.1986] string. The contents of the string SHALL be either an IPv4 address in dotted decimal notation, an IPv6 address, or a fully qualified domain name (FQDN). IPv6 addresses MUST conform to the "Text Representation of Addresses" as specified in [RFC4291] and MUST NOT include zone identifiers [RFC6874]. If internationalized labels are needed in the domain name, the A-LABEL syntax specified in [RFC5891] MUST be used.

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

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

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

4.1.8. NTPv4 Port Negotiation

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

When NTPv4 is negotiated as a Next Protocol and this record is sent by the server, the body specifies the port number of the NTPv4 server with which the client should associate and which will accept the supplied cookies. If no record of this type is sent, the client SHALL assume a default of 123 (the registered port number for NTP).
When this record is sent by the client in conjunction with a NTPv4 Server Negotiation record, it indicates that the client wishes to associate with the NTP server at the specified port. The NTS-KE server MAY incorporate this request when deciding what NTPv4 Server Negotiation and NTPv4 Port Negotiation records to respond with, but honoring the client’s preference is OPTIONAL.

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

4.2. 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 [RFC5705] provides that disambiguating label strings beginning with "EXPERIMENTAL" MAY be used without IANA registration.

5.2. Packet Structure Overview

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

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

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

5.3. The Unique Identifier Extension Field

The Unique Identifier extension field provides the client with a cryptographically strong means of detecting replayed packets. It has a Field Type of [[TBD2]]. When the extension field is included in a client packet (mode 3), its body SHALL consist of a string of octets generated 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. All server packets generated by NTS-implementing servers in response to client packets containing this extension field MUST also contain this field with the same content as in the client’s request. The field’s use in modes other than client–server is not defined.
This extension field MAY also be used standalone, without NTS, in which case it provides the client with a means of detecting spoofed packets from off-path attackers. Historically, NTP’s origin timestamp field has played both these roles, but for cryptographic purposes this is suboptimal because it is only 64 bits long and, depending on implementation details, most of those bits may be predictable. In contrast, the Unique Identifier extension field enables a degree of unpredictability and collision resistance more consistent with cryptographic best practice.

5.4. The NTS Cookie Extension Field

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

5.5. The NTS Cookie Placeholder Extension Field

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

Whenever an NTS Cookie Placeholder extension field is present, it MUST be accompanied by an NTS Cookie extension field. The body length of the NTS Cookie Placeholder extension field MUST be the same as the body length of the NTS Cookie extension field. This length requirement serves to ensure that the response will not be larger than the request, in order to improve timekeeping precision and prevent DDoS amplification. The contents of the NTS Cookie Placeholder extension field’s body 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]]. It SHALL be formatted according to Figure 4 and include the following fields:
Nonce Length: Two octets in network byte order, giving the length of the Nonce field, excluding any padding, interpreted as an unsigned integer.

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

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

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

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

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Nonce Length         |      Ciphertext Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.          Nonce, including up to 3 octets padding              .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.        Ciphertext, including up to 3 octets padding           .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                      Additional Padding                       .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: NTS Authenticator and Encrypted Extension Fields Extension Field Format
The Ciphertext field SHALL be formed by providing the following inputs to the negotiated AEAD Algorithm:

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

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

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

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

The purpose of the Additional Padding field is to ensure that servers can always choose a nonce whose length is adequate to ensure its uniqueness, even if the client chooses a shorter one, and still ensure that the overall length of the server’s response packet does not exceed the length of the request. For mode 4 (server) packets, no Additional Padding field is ever required. For mode 3 (client) packets, the length of the Additional Padding field SHALL be computed as follows. Let ‘N_LEN’ be the padded length of the the Nonce field. Let ‘N_MAX’ be, as specified by RFC 5116 [RFC5116], the maximum permitted nonce length for the negotiated AEAD algorithm. Let ‘N_REQ’ be the lesser of 16 and N_MAX, rounded up to the nearest multiple of 4. If N_LEN is greater than or equal to N_REQ, then no Additional Padding field is required. Otherwise, the Additional Padding field SHALL be at least N_REQ - N_LEN octets in length. Servers MUST enforce this requirement by discarding any packet which does not conform to it. Senders are always free to include more Additional Padding than mandated by the above paragraph. Theoretically, it could be necessary to do so in order to bring the extension field to the minimum length required by [RFC7822]. This should never happen in practice because any reasonable AEAD algorithm will have a nonce and an authenticator long enough to bring the extension field to its required length already. Nonetheless, implementers are advised to
explicitly handle this case and ensure that the extension field they emit is of legal length.

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

5.7. Protocol Details

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

Exactly one Unique Identifier extension field which MUST be authenticated, MUST NOT be encrypted, and whose contents MUST 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 has been previously provided to the client; either from the key exchange server during the NTS-KE handshake or from the NTP server in response to a previous NTS-protected NTP request.

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

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

The client MAY include one or more NTS Cookie Placeholder extension fields which MUST be authenticated and MAY be encrypted. The number of NTS Cookie Placeholder extension fields that the client includes SHOULD be such that if the client includes N placeholders and the server sends back N+1 cookies, the number of unused cookies stored by the client will come to eight. The client SHOULD NOT include more than seven NTS Cookie Placeholder extension fields in a request. When both the client and server adhere to all cookie-management guidance provided in this memo, the number of placeholder extension fields will equal the number of dropped packets since the last successful volley.
In rare circumstances, it may be necessary to include fewer NTS Cookie Placeholder extensions than recommended above in order to prevent datagram fragmentation. When cookies adhere the format recommended in Section 6 and the AEAD in use is the mandatory-to-implement AEAD_AES_SIV_CMAC_256, senders can include a cookie and seven placeholders and still have packet size fall comfortably below 1280 octets if no non-NTS-related extensions are used; 1280 octets is the minimum prescribed MTU for IPv6 and is in practice also safe for avoiding IPv4 fragmentation. Nonetheless, senders SHOULD include fewer cookies and placeholders than otherwise indicated if doing so is necessary to prevent fragmentation.
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, the server SHALL include the following extension fields in its response:

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

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

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

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

The server MAY include additional (non-NTS-related) extension fields which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension field (therefore authenticated but not encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). 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.

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.

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

If the NTP server has previously responded with authentic NTS-protected NTP packets (i.e., packets containing the NTS Authenticator and Encrypted Extension Fields extension field), the client MUST verify that any KoD packets received from the server contain the Unique Identifier extension field and that the Unique Identifier matches that of an outstanding request. If this check fails, the packet MUST be discarded without further processing. If this check passes, the client MUST comply with RFC 5905, Section 7.4 [RFC5905] where required. A client MAY automatically re-run the NTS-KE protocol upon forced disassociation from an NTP server. In that case, it MUST be able to detect and stop looping between the NTS-KE and NTP servers by rate limiting the retries using e.g. exponential retry intervals.

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

To allow for NTP session restart when the NTS-KE server is unavailable and to reduce NTS-KE server load, the client SHOULD keep at least one unused but recent cookie, AEAD keys, negotiated AEAD algorithm, and other necessary parameters on persistent storage. This way, the client is able to resume the NTP session without performing renewed NTS-KE negotiation.
6. Suggested Format for NTS Cookies

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

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

Servers should select an AEAD algorithm which they will use to encrypt and authenticate cookies. The chosen algorithm should be one such as AEAD_AES_SIV_CMAC_256 [RFC5297] which resists accidental nonce reuse. It need not be the same as the one that was negotiated with the client. Servers should randomly generate and store a 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 between NTS-KE and NTP servers as well as across load-balanced clusters can make automatic key rotation challenging. However, the task can be accomplished without the need for central key-management infrastructure by using a ratchet, i.e., making each new key a deterministic, cryptographically pseudo-random function of its predecessor. A recommended concrete implementation of this approach is to use HKDF [RFC5869] to derive new keys, using the key’s predecessor as Input Keying Material and its key identifier as a salt.

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

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

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

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

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

7. IANA Considerations

7.1. Service Name and Transport Protocol Port Number Registry

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

Service Name: ntske
Transport Protocol: tcp
Assignee: IESG <iesg@ietf.org>
Contact: IETF Chair <chair@ietf.org>
Description: Network Time Security Key Exchange
Reference: [[this memo]]
Port Number: [[TBD1]], selected by IANA from the User Port range

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

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

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

Protocol: Network Time Security Key Establishment, version 1

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

Reference: [[this memo]], Section 4

7.3. TLS Exporter Labels Registry

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

<table>
<thead>
<tr>
<th>Value</th>
<th>DTLS-OK</th>
<th>Recommended</th>
<th>Reference</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORTER-network-time-security/1</td>
<td>Y</td>
<td>Y</td>
<td>[[this memo]], Section 4.2</td>
<td></td>
</tr>
</tbody>
</table>

7.4. NTP Kiss-o’-Death Codes Registry

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTSN</td>
<td>Network Time Security (NTS) negative-</td>
<td>[[[this memo]],</td>
</tr>
<tr>
<td></td>
<td>acknowledgment (NAK)</td>
<td>Section 5.7</td>
</tr>
</tbody>
</table>

7.5. NTP Extension Field Types Registry

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

<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]], Section 5.3</td>
</tr>
<tr>
<td>[[TBD3]]</td>
<td>NTS Cookie</td>
<td>[[this memo]], Section 5.4</td>
</tr>
<tr>
<td>[[TBD4]]</td>
<td>NTS Cookie Placeholder</td>
<td>[[this memo]], Section 5.5</td>
</tr>
<tr>
<td>[[TBD5]]</td>
<td>NTS Authenticator and Encrypted Extension Fields</td>
<td>[[this memo]], Section 5.6</td>
</tr>
</tbody>
</table>

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

7.6. Network Time Security Key Establishment Record Types Registry

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

- **Record Type Number (REQUIRED):** An integer in the range 0-32767 inclusive.
- **Description (REQUIRED):** A short text description of the purpose of the field.
- **Reference (REQUIRED):** A reference to a document specifying the semantics of the record.

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

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

Applications for new entries SHALL specify the contents of the Description, Set Critical Bit, and Reference fields as well as which of the above ranges the Record Type Number should be allocated from. Applicants MAY request a specific Record 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>Record Type Number</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End of Message</td>
<td>[[this memo]], Section 4.1.1</td>
</tr>
<tr>
<td>1</td>
<td>NTS Next Protocol Negotiation</td>
<td>[[this memo]], Section 4.1.2</td>
</tr>
<tr>
<td>2</td>
<td>Error</td>
<td>[[this memo]], Section 4.1.3</td>
</tr>
<tr>
<td>3</td>
<td>Warning</td>
<td>[[this memo]], Section 4.1.4</td>
</tr>
<tr>
<td>4</td>
<td>AEAD Algorithm Negotiation</td>
<td>[[this memo]], Section 4.1.5</td>
</tr>
<tr>
<td>5</td>
<td>New Cookie for NTPv4</td>
<td>[[this memo]], Section 4.1.6</td>
</tr>
<tr>
<td>6</td>
<td>NTPv4 Server Negotiation</td>
<td>[[this memo]], Section 4.1.7</td>
</tr>
<tr>
<td>7</td>
<td>NTPv4 Port Negotiation</td>
<td>[[this memo]], Section 4.1.8</td>
</tr>
<tr>
<td>16384-32767</td>
<td>Reserved for Private &amp; Experimental Use</td>
<td>[[this memo]]</td>
</tr>
</tbody>
</table>

### 7.7. Network Time Security Next Protocols Registry

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

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

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

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

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

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

The initial contents of this registry SHALL be as follows:
7.8. Network Time Security Error and Warning Codes Registries

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

- Number (REQUIRED): An integer in the range 0-65535 inclusive
- Description (REQUIRED): A short text description of the condition.
- Reference (REQUIRED): A reference to a relevant specification
document.

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

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

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

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unrecognized Critical</td>
<td>[[this memo]],</td>
</tr>
<tr>
<td></td>
<td>Extension</td>
<td>Section 4.1.3</td>
</tr>
<tr>
<td>1</td>
<td>Bad Request</td>
<td>[[this memo]],</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 4.1.3</td>
</tr>
<tr>
<td>32768-65535</td>
<td>Reserved for Private or</td>
<td>Reserved by [[this</td>
</tr>
<tr>
<td></td>
<td>Experimental Use</td>
<td>memo]]</td>
</tr>
</tbody>
</table>

The Network Time Security Warning Codes Registry SHALL initially be
empty except for the reserved range, i.e.:
8. Implementation Status - RFC EDITOR: REMOVE BEFORE PUBLICATION

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

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

8.1. Implementation 1

Organization: Ostfalia University of Applied Science

Implementor: Martin Langer

Maturity: Proof-of-Concept Prototype

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

8.1.1. Coverage

This implementation covers the complete specification.
8.1.2. Licensing

The code is released under a Apache License 2.0 license.

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

8.1.3. Contact Information

Contact Martin Langer: mart.langer@ostfalia.de

8.1.4. Last Update

The implementation was updated 25. February 2019.

8.2. Implementation 2

Organization: Netnod

Implementor: Christer Weinigel

Maturity: Proof-of-Concept Prototype

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

8.2.1. Coverage

This implementation covers the complete specification.

8.2.2. Licensing

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

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

8.2.3. Contact Information

Contact Christer Weinigel: christer@weinigel.se

8.2.4. Last Update

The implementation was updated 31. January 2019.
8.3. Implementation 3

Organization: Red Hat
Implementor: Miroslav Lichvar
Maturity: Prototype

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

8.3.1. Coverage

This implementation covers the complete specification.

8.3.2. Licensing

Licensing is GPLv2.

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

8.3.3. Contact Information

Contact Miroslav Lichvar: mlichvar@redhat.com

8.3.4. Last Update

The implementation was updated 28. March 2019.

8.4. Implementation 4

Organization: NTPsec
Implementor: Hal Murray and NTPsec team
Maturity: Looking for testers. Servers running at ntp1.glypnod.com:123 and ntp2.glypnod.com:123

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

8.4.1. Coverage

This implementation covers the complete specification.
8.4.2. Licensing

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

8.4.3. Contact Information

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

8.4.4. Last Update

The implementation was updated 2019-Apr-10.

8.5. Implementation 5

Organization: Cloudflare

Implementor: Watson Ladd

Maturity:

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

8.5.1. Coverage

This implementation covers the server side of the NTS specification.

8.5.2. Licensing

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

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

8.5.3. Contact Information

Contact Watson Ladd: watson@cloudflare.com

8.5.4. Last Update

The implementation was updated 21. March 2019.

8.6. Implementation 6

Organization: Netnod

Implementor: Michael Cardell Widerkrantz et. al.
Maturity: Early proof of concept

8.6.1. Coverage

NTS-KE client and server.

8.6.2. Licensing

The source code is available at: https://github.com/mchackorg/gonts

8.6.3. Contact Information

Contact Michael Cardell Widerkrantz: mc@netnod.se

8.6.4. Last Update

The implementation was updated 24. March 2019.

8.7. Interoperability

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

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

9. Security Considerations

9.1. Sensitivity to DDoS attacks

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

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

NTS users should also consider that they are not fully protected against DDoS attacks by on-path adversaries. In addition to dropping packets and attacks such as those described in Section 9.4, an on-path attacker can send spoofed kiss-o’-death replies, which are not authenticated, in response to NTP requests. This could result in significantly increased load on the NTS-KE server. Implementers have to weigh the user’s need for unlinkability against the added resilience that comes with cookie reuse in cases of NTS-KE server unavailability.

9.2. Avoiding DDoS Amplification

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

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

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

9.3. Initial Verification of Server Certificates

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

Check whether the system time is in fact unreliable. 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.

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

Use multiple time sources. The ability to pass off an expired
certificate is only useful to an adversary who has compromised the
corresponding private key. If the adversary has compromised only
a minority of servers, NTP’s selection algorithm (RFC 5905 section
11.2.1 [RFC5905]) will protect the client from accepting bad time
from the adversary-controlled servers.
9.4. 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.

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

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

9.5. 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, cryptographically secure random number generation MUST be used. RFC 4086 [RFC4086] contains guidelines concerning this topic.

9.6. NTS Stripping

Implementers must be aware of the possibility of "NTS stripping" attacks, where an attacker tricks clients into reverting to plain NTP. Naive client implementations might, for example, revert automatically to plain NTP if the NTS-KE handshake fails. A man-in-the-middle attacker can easily cause this to happen. Even clients that already hold valid cookies can be vulnerable, since an attacker can force a client to repeat the NTS-KE handshake by sending faked NTP mode 4 replies with the NTS NAK kiss code. Forcing a client to repeat the NTS-KE handshake can also be the first step in more advanced attacks.
For the reasons described here, implementations SHOULD NOT revert from NTS-protected to unprotected NTP with any server without explicit user action.

10. Privacy Considerations

10.1. Unlinkability

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

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

The unlinkability objective only holds for time synchronization traffic, as opposed to key 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.

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

10.2. Confidentiality

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

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

11. Acknowledgements

The authors would like to thank Richard Barnes, Steven Bellovin, Patrik Fältstrom (Faltstrom), 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, Joachim Stroombergsson (Strombergsson), Martin Thomson, Richard Welty, and Christer Weinigel for contributions to this document and comments on the design of NTS.

12. References

12.1. Normative References


12.2. Informative References

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

[Mizrahi]
Mizrahi, T., "A game theoretic analysis of delay attacks against time synchronization protocols", in Proceedings of Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2012, pp. 1-6, September 2012.

[RFC0768]

[RFC0793]

[RFC4086]

[RFC5077]

[RFC5280]

Franke, et al. Expires November 1, 2019 [Page 41]
Appendix A. Terms and Abbreviations

**AEAD** Authenticated Encryption with Associated Data [RFC5116]

**ALPN** Application-Layer Protocol Negotiation [RFC7301]

**C2S** Client-to-server

**DDoS** Distributed Denial-of-Service

**EF** Extension Field [RFC5905]

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

**IANA** Internet Assigned Numbers Authority

**IP** Internet Protocol

**KoD** Kiss-o’-Death [RFC5905]

**NTP** Network Time Protocol [RFC5905]

**NTS** Network Time Security

**NTS-KE** Network Time Security Key Exchange

**S2C** Server-to-client

**SCSV** Signaling Cipher Suite Value [RFC7507]

**TCP** Transmission Control Protocol [RFC0793]

**TLS** Transport Layer Security [RFC8446]
UDP     User Datagram Protocol [RFC0768]

Authors’ Addresses

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

Email: dafranke@akamai.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

Marcus Dansarie

Email: marcus@dansarie.se

Ragnar Sundblad
Netnod

Email: ragge@netnod.se
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", "MUST 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 June 20, 2019.

1. Introduction


The data model conveys 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.
Table 1: Prefixes and corresponding YANG modules

<table>
<thead>
<tr>
<th>Prefix</th>
<th>YANG module</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>yang</td>
<td>ietf-yang-types</td>
<td>[RFC6991]</td>
</tr>
<tr>
<td>inet</td>
<td>ietf-inet-types</td>
<td>[RFC6991]</td>
</tr>
<tr>
<td>if</td>
<td>ietf-interfaces</td>
<td>[RFC8343]</td>
</tr>
<tr>
<td>ianach</td>
<td>iana-crypt-hash</td>
<td>[RFC7317]</td>
</tr>
<tr>
<td>key-chain</td>
<td>ietf-key-chain</td>
<td>[RFC8177]</td>
</tr>
<tr>
<td>acl</td>
<td>ietf-access-control-list</td>
<td>[I-D.ietf-netmod-acl-model]</td>
</tr>
<tr>
<td>rt-types</td>
<td>ietf-routing-types</td>
<td>[RFC8294]</td>
</tr>
</tbody>
</table>

2. NTP data model

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

module: ietf-ntp
  +++-rw ntp
    +++-rw port? inet:port-number {ntp-port}?
    +++-rw refclock-master!
      | +++-rw master-stratum? ntp-stratum
    +++-rw authentication
      +++-rw auth-enabled? boolean
      +++-rw authentication-keys* [key-id]
        | +++-rw key-id uint32
        | +++-rw algorithm? identityref
        | +++-rw key? ianach:crypt-hash
        | +++-rw istrusted? boolean
    +++-rw access-rules
      +++-rw access-rule* [access-mode]
        | +++-rw access-mode access-mode
        | +++-rw acl? -> /acl:acls/acl/name
  +++-ro clock-state
    | +++-ro clock-state ntp-clock-status
    | +++-ro clock-stratum ntp-stratum
    | +++-ro clock-refid union
    | +++-ro associations-address?
      |   | -> /ntp/associations/address
    | +++-ro associations-local-mode?
      |   | -> /ntp/associations/local-mode
    | +++-ro associations-isconfigured?
      |   | -> /ntp/associations/isconfigured
    | +++-ro nominal-freq decimal64
+--ro actual-freq          decimal64
+--ro clock-precision     uint8
+--ro clock-offset?       decimal64
+--ro root-delay?         decimal64
+--ro root-dispersion?    decimal64
+--ro reference-time?     yang:dateTime
+--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)?
        +--rw key-id?        leafref
    +--rw prefer?          boolean
    +--rw burst?           boolean
    +--rw iburst?          boolean
    +--rw source?          if:interface-ref
    +--rw minpoll?         ntp-minpoll
    +--rw maxpoll?         ntp-maxpoll
    +--rw port?            inet:port-number {ntp-port}?
    +--rw version?         ntp-version
++ro associations* [address local-mode isconfigured]
    +--ro address          inet:host
    +--ro local-mode       association-mode
    +--ro isconfigured     boolean
    +--ro stratum?         ntp-stratum
    +--ro refid?           union
        -> /ntp/authentication/authentication-keys/key-id
    +--ro prefer?          boolean
    +--ro peer-interface?  if:interface-ref
    +--ro minpoll?         ntp-minpoll
    +--ro maxpoll?         ntp-maxpoll
    +--ro port?            inet:port-number {ntp-port}?
    +--ro version?         ntp-version
    +--ro reach?           uint8
    +--ro unreach?         uint8
    +--ro poll?            uint8
    +--ro now?             uint32
    +--ro offset?          decimal64
    +--ro delay?           decimal64
    +--ro dispersion?      decimal64
    +--ro originate-time?  yang:dateTime
    +--ro receive-time?    yang:dateTime
    +--ro transmit-time?   yang:dateTime
    +--ro input-time?      yang:dateTime
    +--ro ntp-statistics
---ro packet-sent? yang:counter32
---ro packet-sent-fail? yang:counter32
---ro packet-received? yang:counter32
---ro packet-dropped? yang:counter32

---rw interfaces
---rw interface* [name]
  ---rw name if:interface-ref
  ---rw broadcast-server!
    ---rw ttl? uint8
    ---rw authentication
    | ---rw (authentication-type)?
    | | ---:(symmetric-key)
    | | ---rw key-id? leafref
    ---rw minpoll? ntp-minpoll
    ---rw maxpoll? ntp-maxpoll
    ---rw port? inet:port-number {ntp-port}?
    ---rw version? ntp-version

---rw broadcast-client!
---rw multicast-server* [address]
  ---rw address rt-types:ip-multicast-group-address
  ---rw ttl? uint8
  ---rw authentication
  | ---rw (authentication-type)?
  | | ---:(symmetric-key)
  | | ---rw key-id? leafref
  ---rw minpoll? ntp-minpoll
  ---rw maxpoll? ntp-maxpoll
  ---rw port? inet:port-number {ntp-port}?
  ---rw version? ntp-version

---rw multicast-client* [address]
  ---rw address rt-types:ip-multicast-group-address

---rw manycast-server* [address]
  ---rw address rt-types:ip-multicast-group-address

---rw manycast-client* [address]
  ---rw address rt-types:ip-multicast-group-address
  ---rw authentication
  | ---rw (authentication-type)?
  | | ---:(symmetric-key)
  | | ---rw key-id? leafref
  ---rw ttl? uint8
  ---rw minclock? uint8
  ---rw maxclock? uint8
  ---rw beacon? uint8
  ---rw minpoll? ntp-minpoll
  ---rw maxpoll? ntp-maxpoll
  ---rw port? inet:port-number {ntp-port}?
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

<table>
<thead>
<tr>
<th>YANG data nodes in /ntp/clock-state/system-status</th>
<th>NTPv4-MIB objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock-state</td>
<td>ntpEntStatusCurrentMode</td>
</tr>
<tr>
<td>clock-stratum</td>
<td>ntpEntStatusStratum</td>
</tr>
<tr>
<td>clock-refid</td>
<td>ntpEntStatusActiveRefSourceId</td>
</tr>
<tr>
<td>clock-precision</td>
<td>ntpEntTimePrecision</td>
</tr>
<tr>
<td>clock-offset</td>
<td>ntpEntStatusActiveOffset</td>
</tr>
<tr>
<td>root-dispersion</td>
<td>ntpEntStatusDispersion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>YANG data nodes in /ntp/associations/</th>
<th>NTPv4-MIB objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>ntpAssocAddressType</td>
</tr>
<tr>
<td>stratum</td>
<td>ntpAssocStratum</td>
</tr>
<tr>
<td>refid</td>
<td>ntpAssocRefId</td>
</tr>
<tr>
<td>offset</td>
<td>ntpAssocOffset</td>
</tr>
<tr>
<td>delay</td>
<td>ntpAssocStatusDelay</td>
</tr>
<tr>
<td>dispersion</td>
<td>ntpAssocStatusDispersion</td>
</tr>
<tr>
<td>ntp-statistics/packet-sent</td>
<td>ntpAssocStatOutPkts</td>
</tr>
<tr>
<td>ntp-statistics/packet-received</td>
<td>ntpAssocStatInPkts</td>
</tr>
<tr>
<td>ntp-statistics/packet-dropped</td>
<td>ntpAssocStatProtocolError</td>
</tr>
</tbody>
</table>

### YANG NTP State Data Nodes and Related NTPv4-MIB Objects

4. Relationship with RFC 7317

This section describes the relationship with NTP definition in Section 3.2 System Time Management of [RFC7317]. YANG data nodes in /ntp/ also supports per-interface configurations which is not supported in /system/ntp. If the yang model defined in this document is implemented, then /system/ntp SHOULD NOT be used and MUST be ignored.
### YANG NTP Configuration Data Nodes and counterparts in RFC 7317 Objects

<table>
<thead>
<tr>
<th>YANG data nodes in /ntp/</th>
<th>YANG data nodes in /system/ntp</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntp!</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/iburst</td>
<td>server/association-type</td>
</tr>
<tr>
<td>unicast-configuration/prefer</td>
<td>server/iburst</td>
</tr>
<tr>
<td></td>
<td>server/prefer</td>
</tr>
</tbody>
</table>

5. **Access Rules**

As per [RFC1305], NTP could include an access-control feature that prevents unauthorized access and controls which peers are allowed to update the local clock. Further it is useful to differentiate between the various kinds of access (such as peer or server; refer access-mode) and attach different acl-rule to each. For this, the YANG module allow such configuration via /ntp/access-rules. The access-rule itself is configured via [I-D.ietf-netmod-acl-model].

6. **Key Management**

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

7. **NTP YANG Module**

<CODE BEGINS> file "ietf-ntp@2018-12-17.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";
}

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";
}

/* Note: The RFC Editor will replace XXXX with the number assigned
to the RFC once draft-ietf-netmod-acl-model becomes an RFC.*/

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

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 (http://trustee.ietf.org/license-info).

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

revision 2018-12-17 {
  description
    "Updated revision.";
  reference
    "RFC YYYY: A YANG Data Model for NTP.";
}

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

/* Typedef Definitions */
typedef ntp-stratum {
  type uint8 {
    range "1..16";
  }
  description
    "The level of each server in the hierarchy is defined by a stratum. Primary servers are assigned with stratum one; secondary servers at each lower level are assigned with one stratum greater than the preceding level";
}
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 maximum poll exponent for this NTP association."
}

typedef access-mode {
type enumeration {
enum peer {
value "0";
description "Enables the full access authority. Both time request and control query can be performed on the local NTP service, and the local clock can be synchronized with the remote server.";
}
enum server {
value "1";
description "Enables the server access and query. Both time requests and control query can be performed on the local NTP service, but the local clock cannot be synchronized with the remote server."
}
enum synchronization {

typedef unicast-configuration-type {
  type enumeration {
    enum server {
      value "0";
      description
      "Use client association mode. This device
      will not provide synchronization to the
      configured NTP server.";
    }
    enum peer {
      value "1";
      description
      "Use symmetric active association mode.
      This device may provide synchronization
      to the configured NTP server.";
    }
  }
  description
  "This defines NTP unicast mode of operation.";
}
typedef association-mode {
  type enumeration {
    enum client {
      value "0";
      description
      "Use client association mode (mode 3).
      This device will not provide synchronization
      to the configured NTP server.";
    }
    enum active {

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

description

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

/* features */
feature ntp-port {
    description
    "This defines NTP clock status.";
}
feature authentication {
    description "Support for NTP symmetric key authentication";
    reference "RFC1305 - 3.2.6 Authentication Variables";
}

date-time
    description "Support for NTP port configuration";
    reference "RFC1305 - 3.2.7 Parameters";
}

feature access-rules {
    description "Support for NTP access control";
    reference "RFC1305 - 3.6 Access Control Issues";
}

feature unicast-configuration {
    description "Support for NTP client/server or active/passive in unicast";
    reference "RFC1305 - 3.3 Modes of Operation";
}

feature broadcast-server {
    description "Support for broadcast server";
    reference "RFC1305 - 3.3 Modes of Operation";
}

feature broadcast-client {
    description "Support for broadcast client";
    reference "RFC1305 - 3.3 Modes of Operation";
}

feature multicast-server {
    description "Support for multicast server";
    reference "RFC1305 - 3.3 Modes of Operation";
}

feature multicast-client {
    description "Support for multicast client";
    reference "RFC1305 - 3.3 Modes of Operation";
}

feature manycast-server {
description
    "Support for manycast server";
reference "RFC5905 - 3.1 Dynamic Server Discovery";
}

feature manycast-client {
    description
    "Support for manycast client";
    reference "RFC5905 - 3.1 Dynamic Server Discovery";
}

/* Groupings */
grouping authentication-key {
    description
    "To define an authentication key for a Network Time
    Protocol (NTP) time source.";
    leaf key-id {
        type uint32 {
            range "1..max";
        }
        description
        "Authentication key identifier.";
    }
    leaf algorithm {
        type identityref {
            base key-chain:crypto-algorithm;
        }
        description
        "Authentication algorithm.";
    }
    leaf key {
        type ianach:crypt-hash;
        description
        "The key";
    }
    leaf istrusted {
        type boolean;
        description
        "Key-id is trusted or not";
    }
}

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

grouping statistics {
  description "NTP packet statistic.";
  leaf packet-sent {
    type yang:counter32;
    description "The total number of NTP packets delivered to the transport service by this NTP entity for this association. Discontinuities in the value of this counter can occur upon cold start or reinitialization of the NTP entity, the management system and at other times as indicated by discontinuities in the value of sysUpTime.";
  }
  leaf packet-sent-fail {
    type yang:counter32;
    description "The number of times NTP packets sending failed.";
  }
  leaf packet-received {
    type yang:counter32;
    description "The total number of NTP packets delivered to the NTP entity from this association. Discontinuities in the value of this counter can occur upon cold start or reinitialization of the NTP entity, the management system and at other times as indicated by discontinuities in the value of sysUpTime.";
  }
  leaf packet-dropped {
    type yang:counter32;
    description "The total number of NTP packets that were delivered to this NTP entity from this association and this entity";
  }
}
was not able to process due to an NTP protocol error. Discontinuities in the value of this counter can occur upon cold start or reinitialization of the NTP entity, the management system and at other times as indicated by discontinuities in the value of sysUpTime.;


grouping common-attributes {
  description "NTP common attributes for configuration.";
  leaf minpoll {
    type ntp-minpoll;
    description "The minimum poll interval used in this association.";
  }
  leaf maxpoll {
    type ntp-maxpoll;
    description "The maximum poll interval used in this association.";
  }
  leaf port {
    if-feature ntp-port;
    type inet:port-number {
      range "123 | 1025..max";
    }
    default "123";
    description "Specify the port used to send NTP packets.";
  }
  leaf version {
    type ntp-version;
    description "NTP version.";
  }
}

grouping association-ref {
  description "Reference to NTP association mode";
  leaf associations-address {
    type leafref {
      path "/ntp:ntp/ntp:associations/ntp:address";
    }
    description "Indicates the association’s address which result in clock synchronization.";
  }
}
leaf associations-local-mode {
  type leafref {
    path "/ntp:ntp/ntp:associations/ntp:local-mode";
  }
  description
  "Indicates the association’s local-mode which result in clock synchronization.";
}

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

/* Configuration data nodes */
container ntp {
  presence
  "NTP is enabled and system should attempt to synchronize the system clock with an NTP server from the 'ntp/associations' list.";
  description
  "Configuration parameters for NTP.";
  leaf port {
    if-feature ntp-port;
    type inet:port-number {
      range "123 | 1025..max";
    }
    default "123";
    description
    "Specify the port used to send and receive NTP packets.";
  }
  container refclock-master {
    presence
    "NTP master clock is enabled.";
    description
    "Configures the local clock of this device as NTP server.";
    leaf master-stratum {
      type ntp-stratum;
      default "16";
      description
      "Stratum level from which NTP clients get their time synchronized.";
    }
  }
}
container authentication {
    description
        "Configuration of authentication.";
    leaf auth-enabled {
        type boolean;
        default false;
        description
            "Controls whether NTP authentication is enabled or disabled on this device.";
    }
    list authentication-keys {
        key "key-id";
        uses authentication-key;
        description
            "List of authentication keys.";
    }
}

container access-rules {
    description
        "Configuration to control access to NTP service by using NTP access-group feature.
The access-mode identifies how the acl is applied with NTP.";
    list access-rule {
        key "access-mode";
        description
            "List of access rules.";
        leaf access-mode {
            type access-mode;
            description
                "NTP access mode. The definition of each possible values:
                peer(0): Both time request and control query can be performed.
                server(1): Enables the server access and query.
                synchronization(2): Enables the server access only.
                query(3): Enables control query only.";
        }
        leaf acl {
            type leafref {
                path "/acl:acls/acl:acl/acl:name";
            }
            description
                "Control access configuration to be used.";
        }
        reference
            "RFC 1305";
    }
container clock-state {
    config "false";
    description
        "Clock operational state of the NTP.";
}

container system-status {
    description
        "System status of NTP.";
    leaf clock-state {
        type ntp-clock-status;
        mandatory true;
        description
            "The state of system clock. The definition of each possible value is:
             synchronized(0): Indicates local clock is synchronized.
             unsynchronized(1): Indicates local clock is not synchronized.";
    }
    leaf clock-stratum {
        type ntp-stratum;
        mandatory true;
        description
            "The NTP entity’s own stratum value. Should be a stratum of syspeer + 1 (or 16 if no syspeer).";
    }
    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.";
}
leaf nominal-freq {
  type decimal64 {
    fraction-digits 4;
  }
  units Hz;
  mandatory true;
  description
    "The nominal frequency of the
    local clock.";
}
leaf actual-freq {
  type decimal64 {
    fraction-digits 4;
  }
  units Hz;
  mandatory true;
  description
    "The actual frequency of the
    local clock.";
}
leaf clock-precision {
  type uint8;
  units Hz;
  mandatory true;
  description
    "Clock precision of this system in integer format
     (prec=2^(-n)). A value of 5 would mean 2^-5 = 31.25 ms.";
}
leaf clock-offset {
  type decimal64 {
    fraction-digits 3;
  }
  units milliseconds;
  description
    "The time offset to the current selected reference time
     source e.g., '0.032' or '1.232'.";
}
leaf root-delay {
  type decimal64 {
    fraction-digits 3;
  }
  units milliseconds;
  description
    "Total delay along the path to root clock.";
}
leaf root-dispersion {
  type decimal64 {
fraction-digits 3;
}
units milliseconds;
description
 "The dispersion between the local clock 
and the root clock, e.g., '6.927'."
}
leaf reference-time {
 type yang:date-and-time;
description
 "The reference timestamp."
}
leaf sync-state {
 type ntp-sync-state;
 mandatory true;
description
 "The synchronization status of 
 the local clock."
}
}
}
list unicast-configuration {
 key "address type";
description
 "List of NTP unicast-configurations.";
leaf address {
 type inet:host;
description
 "Address of this association.";
}
leaf type {
 type unicast-configuration-type;
description
 "Use client association mode. This device 
 will not provide synchronization to the 
 configured NTP server."
}
container authentication{
description
 "Authentication used for this association.";
 uses authentication;
}
leaf prefer {
 type boolean;
default "false";
description
 "Whether this association is preferred or not.";
}
leaf burst {
  type boolean;
  default "false";
  description
    "If set, a series of packets are sent instead of a single
     packet within each synchronization interval to achieve faster
     synchronization."
}

leaf iburst {
  type boolean;
  default "false";
  description
    "If set, a series of packets are sent instead of a single
     packet within the initial synchronization interval to achieve
     faster initial synchronization."
}

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

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

list associations {
  key "address local-mode isconfigured";
  config "false";
  description
    "List of NTP associations. Here address, local-mode
     and isconfigured is required to uniquely identify
     a particular association. Lets take following examples -

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

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

    Thus all 3 leaves are needed as key to unique identify
    the association."
leaf address {
    type inet:host;
    description
        "The address of this association. Represents the IP
         address of a unicast/multicast/broadcast address.";
}
leaf local-mode {
    type association-mode;
    description
        "Local mode of this NTP association.";
}
leaf isconfigured {
    type boolean;
    description
        "Indicates if this association is configured or
dynamically learned.";
}
leaf stratum {
    type ntp-stratum;
    description
        "The association stratum value.";
}
leaf refid {
    type union {
        type inet:ipv4-address;
        type binary {
            length "4";
        }
        type string {
            length "4";
        }
    }
    description
        "The refclock driver ID, if available.
        -- a refclock driver ID like '127.127.1.0' for local clock sync
        -- uni/multi/broadcast associations will look like '20.1.1.1'
        -- sync with primary source will look like 'DCN', 'NIST',
        'ATOM';
        reference
        "RFC 1305";
    }
leaf authentication{
    type leafref {
        path "/ntp:ntp/ntp:authentication/
           + "ntp:authentication-keys/ntp:key-id";
    }
    description
        "The refclock driver ID, if available.
        -- a refclock driver ID like '127.127.1.0' for local clock sync
        -- uni/multi/broadcast associations will look like '20.1.1.1'
        -- sync with primary source will look like 'DCN', 'NIST',
        'ATOM';
        reference
        "RFC 1305";"}
leaf prefer {
    type boolean;
    default "false";
    description "Indicates if this association is preferred.";
}
leaf peer-interface {
    type if:interface-ref;
    description "The interface which is used for communication.";
}
uses common-attributes {
    description "Common attributes like port, version, min and max poll.";
}
leaf reach {
    type uint8;
    description "The reachability of the configured server or peer.";
}
leaf unreach {
    type uint8;
    description "The unreachability of the configured server or peer.";
}
leaf poll {
    type uint8;
    units seconds;
    description "The polling interval for current association";
}
leaf now {
    type uint32;
    units seconds;
    description "The time since the NTP packet was not received or last synchronized.";
}
leaf offset {
    type decimal64 {
        fraction-digits 3;
    }
    units milliseconds;
description
"The offset between the local clock
and the peer clock, e.g., ‘0.032’ or ‘1.232’";
}
leaf delay {
    type decimal64 {
        fraction-digits 3;
    }
    units milliseconds;
    description
    "The network delay between the local clock
    and the peer clock.";
}
leaf dispersion {
    type decimal64 {
        fraction-digits 3;
    }
    units milliseconds;
    description
    "The root dispersion between the local clock
    and the peer clock.";
}
leaf originate-time {
    type yang:date-and-time;
    description
    "This is the local time, in timestamp format,
    when latest NTP packet was sent to peer(T1)."
    reference
    "RFC 1305";
}
leaf receive-time {
    type yang:date-and-time;
    description
    "This is the local time, in timestamp format,
    when latest NTP packet arrived at peer(T2).
    If the peer becomes unreachable the value is set to zero.";
    reference
    "RFC 1305";
}
leaf transmit-time {
    type yang:date-and-time;
    description
    "This is the local time, in timestamp format,
    at which the NTP packet departed the peer(T3).
    If the peer becomes unreachable the value is set to zero.";
    reference
    "RFC 1305";
}
leaf input-time {
  type yang:date-and-time;
  description
    "This is the local time, in timestamp format,
     when the latest NTP message from the peer arrived(T4).
     If the peer becomes unreachable the value is set to zero.";
  reference
    "RFC 1305";
}

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

container interfaces {
  description
    "Configuration parameters for NTP interfaces.";
  list interface {
    key "name";
    description
      "List of interfaces.";
    leaf name {
      type if:interface-ref;
      description
        "The interface name.";
    }
  }

  container broadcast-server {
    presence
      "NTP broadcast-server is configured";
    description
      "Configuration of broadcast server.";
    leaf ttl {
      type uint8;
      description
        "Specifies the time to live (TTL) for a
         broadcast packet.";
    }

    container authentication{
      description
        "Authentication used for this association.";
      uses authentication;
    }
}
uses common-attributes {
  description
    "Common attribute like port, version, min and max poll.";
}

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) for a multicast packet.";
  }
  container authentication{
    description
      "Authentication used for this association.";
    uses authentication;
  }
  uses common-attributes {
    description
      "Common attributes like port, version, min and max poll.";
  }
}

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
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 used for this association.";
        uses authentication;
    }
    leaf ttl {
        type uint8;
        description "Specifies the maximum time to live (TTL) for the expanding ring search.";
    }
    leaf minclock {
        type uint8;
        description "The minimum manycast survivors in this association.";
    }
    leaf maxclock {
        type uint8;
        description "The maximum manycast candidates in this association.";
    }
}
leaf beacon {
  type uint8;
  description
    "The maximum interval between beacons in this
    association.";
}
uses common-attributes {
  description
    "Common attributes like port, version, min and
    max poll.";
  reference
    "RFC 5905";
}
}
}
container ntp-statistics {
  config "false";
  description
    "Total NTP packet statistics.";
  uses statistics {
    description
      "NTP send and receive packet statistics.";
  }
}
"CODE ENDS"

8. Usage Example

8.1. 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 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: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>
8.2.  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>

8.3. Authentication configuration

Below is the example on how to enable authentication and configure trusted authentication key 10 with mode as md5 and key as abcd
<edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <target>
    <running/>
  </target>
  <config>
    <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
      <authentication>
        <auth-enabled>true</auth-enabled>
        <authentication-keys>
          <key-id>10</key-id>
          <algorithm>md5</algorithm>
          <key>abcd</key>
          <istrusted>true</istrusted>
        </authentication-keys>
      </authentication>
    </ntp>
  </config>
</edit-config>

Below is the example on how to get authentication related configuration

<get>
  <filter type="subtree">
      <sys:authentication>
        </sys:authentication>
    </sys:ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <authentication>
      <auth-enabled>false</auth-enabled>
      <trusted-keys/>
      <authentication-keys>
        <key-id>10</key-id>
        <algorithm>md5</algorithm>
        <key>abcd</key>
        <istrusted>true</istrusted>
      </authentication-keys>
    </authentication>
  </ntp>
</data>
8.4. Access configuration

Below is the example on how to configure access mode "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/>
    </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">
    <access-rules>
      <access-rule>
        <access-mode>peer</access-mode>
        <acl>2000</acl>
      </access-rule>
    </access-rules>
  </ntp>
</data>
```

8.5. 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.
<get>
  <filter type="subtree">
      <sys:interfaces>
        <sys:interface>
          <sys:multicast-server/>
        </sys:interface>
      </sys:interfaces>
    </sys:ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <interfaces>
      <interface>
        <name>Ethernet3/0/0</name>
        <multicast-server>
          <address>224.1.1.1</address>
          <ttl>224.1.1.1</ttl>
          <authentication>
            <symmetric-key>
              <key-id>10</key-id>
            </symmetric-key>
          </authentication>
          <minpoll>6</minpoll>
          <maxpoll>10</maxpoll>
          <port>1025</port>
          <version>3</version>
        </multicast-server>
      </interface>
    </interfaces>
  </ntp>
</data>

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>
8.6. 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

<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:manycast-server>
        </sys:interface>
      </sys:interfaces>
    </sys:ntp>
  </filter>
</get>

<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <interfaces>
      <interface>
        <name>Ethernet3/0/0</name>
        <manycast-server>
          <address>224.1.1.1</address>
        </manycast-server>
      </interface>
    </interfaces>
  </ntp>
</data>
8.7. Clock state

Below is the example on how to get clock current state

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

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

8.8. Get all association

Below is the example on how to get all association present

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

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ntp xmlns="urn:ietf:params:xml:ns:yang:ietf-ntp">
    <clock-state>
      <system-status>
        <clock-state>synchronized</clock-state>
        <clock-stratum>7</clock-stratum>
        <clock-refid>192.0.2.1</clock-refid>
        <associations-address>192.0.2.1</associations-address>
        <associations-local-mode>client</associations-local-mode>
        <associations-isconfigured>yes</associations-isconfigured>
        <nominal-freq>100.0</nominal-freq>
        <actual-freq>100.0</actual-freq>
        <clock-precision>18</clock-precision>
        <clock-offset>0.025</clock-offset>
        <root-delay>0.5</root-delay>
        <root-dispersion>0.8</root-dispersion>
        <reference-time>10-10-2017 07:33:55.258 Z+05:30</reference-time>
      </system-status>
    </clock-state>
  </ntp>
</data>
```
<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>10-10-2017 07:33:55.253 Z+05:30</originate-time>
      <receive-time>10-10-2017 07:33:55.258 Z+05:30</receive-time>
      <transmit-time>10-10-2017 07:33:55.300 Z+05:30</transmit-time>
      <input-time>10-10-2017 07:33:55.305 Z+05:30</input-time>
      <ntp-statistics>
        <packet-sent>20</packet-sent>
        <packet-sent-fail>0</packet-sent-fail>
        <packet-received>20</packet-received>
        <packet-dropped>0</packet-dropped>
      </ntp-statistics>
    </associations>
  </ntp>
</data>
8.9. Global statistic

Below is the example on how to get clock current state

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

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

9. IANA Considerations

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


Registrant Contact: The NTP 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 YYYY
10. Security Considerations

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

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

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

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

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

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

/ntp/interfaces/interface - The entries in the list include all per-interface configurations related to broadcast, multicast and anycast 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.

11. 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.

12. References

12.1. Normative References

[I-D.ietf-netmod-acl-model]
Jethanandani, M., Agarwal, S., Huang, L., and D. Blair,
"Network Access Control List (ACL) YANG Data Model",
draft-ietf-netmod-acl-model-21 (work in progress),
November 2018.

Specification, Implementation and Analysis", RFC 1305,
DOI 10.17487/RFC1305, March 1992,

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
DOI 10.17487/RFC3688, January 2004,

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

"Definitions of Managed Objects for Network Time Protocol
Internet-Draft                YANG for NTP                 December 2018

           the Network Configuration Protocol (NETCONF)", RFC 6020,
           DOI 10.17487/RFC6020, October 2010,

           and A. Bierman, Ed., "Network Configuration Protocol
           (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,

[RFC6242]  Wasserman, M., "Using the NETCONF Protocol over Secure
           Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011,

           RFC 6991, DOI 10.17487/RFC6991, July 2013,

           RFC 7950, DOI 10.17487/RFC7950, August 2016,

           Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017,

[RFC8174]  Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
           2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,

           Zhang, "YANG Data Model for Key Chains", RFC 8177,
           DOI 10.17487/RFC8177, June 2017,

[RFC8294]  Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger,
           "Common YANG Data Types for the Routing Area", RFC 8294,
           DOI 10.17487/RFC8294, December 2017,

           BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018,

           Access Control Model", STD 91, RFC 8341,
           DOI 10.17487/RFC8341, March 2018,

12.2. Informative References


Authors’ Addresses

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

Email: eric.wu@huawei.com

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

Email: dhruv.ietf@gmail.com

Ankit kumar Sinha
RtBrick Inc.
Bangalore, Kanataka
India

Email: ankit.ietf@gmail.com
Enterprise Profile for the Precision Time Protocol With Mixed Multicast and Unicast Messages
draft-ietf-tictoc-ptp-enterprise-profile-15

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.

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 October 6, 2019.

Copyright Notice

Copyright (c) 2019 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 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 IEEE 1588-2008 [IEEE1588] needs to know the identity of the time sources in the network (the Master Clocks).

The "Best Master Clock Algorithm" (IEEE 1588-2008 [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 (IEEE 1588-2008 [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 IEEE 1588-2008 [IEEE1588] subclause 13.4), defining an optional Best Master Clock Algorithm and a few other ways. PTP has its own management protocol (defined in IEEE 1588-2008 [IEEE1588] subclause 15.2) but allows a PTP profile specify an alternative management mechanism, for example SNMP.

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. 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 RFC 5905 [RFC5905]

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.
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 [ISPCS].

5. Network Technology

This PTP profile SHALL operate only in networks characterized by UDP RFC 768 [RFC0768] over either IPv4 RFC 791 [RFC0791] or IPv6 RFC 8200 [RFC8200], as described by Annexes D and E in IEEE 1588 [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, ITU-T G.8271 [G8271].

6. Time Transfer and Delay Measurement

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

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

Sync messages MUST be sent as PTP event multicast messages (UDP port 319) to the PTP primary IP address. Two step clocks SHALL send Follow-up messages as PTP general messages (UDP port 320). Announce messages MUST be sent as multicast messages (UDP port 320) to the PTP primary address. The PTP primary IP address is 224.0.1.129 for IPv4 and FF0X:0:0:0:0:0:0:181 for IPv6, where X can be a value between 0x0 and 0xFF, see IEEE 1588 [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 forbidden PTP options (Section 13).

8. Requirements for Master Clocks

Master Clocks SHALL obey the standard Best Master Clock Algorithm from IEEE 1588 [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 IEEE 1588 [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 IEEE 1588 [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 IEEE 1588 [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. Acknowledgements

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

There are no IANA requirements in this specification.

18. 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 [RFC6241].

General security considerations of time protocols are discussed in RFC 7384 [RFC7384].

19. References

19.1. Normative References


19.2. Informative References


Authors’ Addresses

Doug Arnold
Meinberg Funkuhren GmbH & Co. KG
Lange Wand 9
Bad Pyrmont 31812
Germany

Email: doug.arnold@meinberg.de
Heiko Gerstung  
Meinberg Funkuhren GmbH & Co. KG  
Lange Wand 9  
Bad Pyrmont  31812  
Germany  

Email: heiko.gerstung@meinberg.de
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 October 27, 2019.

Copyright Notice

Copyright (c) 2019 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 receive and transmit timestamps from the NTP header.

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.

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.

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.

Delay Correction
A signed fixed-point number of nanoseconds with 48 integer bits and 16 fractional bits, which represents the current correction of the network delay that has accumulated for this packet on the path from the source to the destination. The format of this field is identical to the PTP correctionField.

Path ID
A 16-bit identification number of the path where the delay correction was updated.

Checksum Complement
A field which can be modified in order to keep the UDP checksum of the packet valid. This allows the UDP checksum to be transmitted before the Correction Field is received and modified. The same field is described in RFC 7821 [RFC7821].

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.

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.

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 calculates 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. A conversion is necessary as the corrections are in different units than the timestamps (nanoseconds vs seconds).

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
NTPv4 Short Extension Fields
draft-mlichvar-ntp-short-extension-fields-00

Abstract

This document specifies a new packet format for the Network Time Protocol version 4 (NTPv4) which is compatible with RFC 7822, but allows NTPv4 packets to contain shorter extension fields.

Status of This Memo

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

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

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

This Internet-Draft will expire on March 30, 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

RFC 7822 [RFC7822] specifies a minimum length of extension fields in NTPv4 packets in order to prevent ambiguities in their parsing. Without these rules, an extension field in a valid NTPv4 packet could be parsed as a Message Authentication Code (MAC), or a MAC could be parsed as an extension field.

The minimum length of 28 octets forces extension fields that do not contain enough data to reach the minimum length to waste space. With multiple extension fields in a packet the wasted space accumulates.

A different issue with extension fields in NTPv4 packets is that servers/clients cannot pad a response/request to a specific length, e.g. to make their length symmetric when they contain different extension fields, or the sums of their lengths are different, unless one of the extension fields included in the request/response supports padding.

This document specifies a new NTPv4 format using three new extension fields:

1. An extension field which contains other extension fields with no requirements on minimum length
2. An extension field which does not contain any information and can always be used for padding
3. An extension field which contains MAC

Together, these extension fields allow NTPv4 packets to contain short extension fields, minimize the wasted space, and allow the packets to be padded to any length that meets the requirements of RFC 7822.

Older NTP implementations which follow RFC 7822 will parse a packet in the new format as a valid packet which contains a single unknown extension field, skipping all extension fields and/or MAC, and can respond as appropriate.

An implementation which supports the new format will parse all extension fields and/or MAC contained in the packet.

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. New extension fields

2.1. Packing Field

The Packing Field is an NTP extension field following RFC 7822 [RFC7822], which contains one or more other extension fields. The format of the extension field is shown below.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Field Type           |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Subfield 1 Type         |        Subfield 1 Length      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                           .                                                               .
                           .               Subfield 1 Data                                   .
                           .                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                           .                                                               .
                           .                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Subfield N Type         |        Subfield N Length      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                           .                                                               .
                           .               Subfield N Data                                   .
                           .                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Format of Packing Field

The extension field has the following fields:

Field Type
The type which identifies the Packing Field. TBD

Length
The length of the extension field, which is at least 28 octets.

Subfield 1..N Type
The types of the contained extension fields.

Subfield 1..N Length
The lengths of the contained extension field, which are divisible by 4 and can be smaller than 28.

Subfield 1..N Data
Data specific to the included extension fields.

2.2. Padding Field

The Padding Field is an NTP extension field which does not contain any useful data. It does not follow the requirements from RFC 7822 [RFC7822] and it MUST be contained in the Packing Field.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Field Type           |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                 Padding
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Format of Padding Field

The extension field has the following fields:

Field Type
The type which identifies the Padding Field. TBD

Length
The length of the extension field.

Padding
Octets filling the space of the extension field with any value.

2.3. MAC Field

The MAC Field is an NTP extension field which contains a MAC as specified in RFC 5905 [RFC5905]. It does not follow the requirements from RFC 7822 [RFC7822] and it MUST be contained in the Packing Field.
The extension field has the following fields:

Field Type
  The type which identifies the MAC Field. TBD

Length
  The length of the extension field.

Key Identifier
  The ID of the key which is used for calculating the digest.

Message Digest
  Digest calculated over all UDP data before the Key Identifier, including the length of the MAC Field and Packing field.

3. New NTPv4 format

An NTPv4 packet in the new format consists of:

1. NTPv4 header per RFC 5905 [RFC5905] (48 octets)
2. Field Type of the Packing Field (2 octets)
3. Length of all data following the NTP header (2 octets)
4. Extension fields with no restrictions on their minimum length, optionally including the Padding and/or MAC Fields (at least 24 octets)

The packet MUST have exactly one Packing Field and it MUST contain all other extension fields. The packet MUST NOT have a MAC outside the Packing Field. If there is not enough data to reach the minimum
length of 28 octets, the Packing Field MUST include at least one Padding Field to increase the length of the Packing Field.

4. Parsing of NTPv4 packets

An implementation SHOULD check if the following applies to the UDP data before parsing it as an NTPv4 packet in the new format:

1. NTP version (in the first octet) is 4.
2. NTP mode (in the first octet) is 1, 2, 3, 4, or 5.
3. Length is at least 76 octets.
4. 49th and 50th octets contain the type of the Packing Field.
5. 51st and 52nd octets contain a value that is equal to the length of the UDP data minus 48.

5. IANA Considerations

IANA is requested to allocate Extension Field Types for the Packing, Padding, and MAC Extension Fields.

6. Normative References


Author’s Address
Miroslav Lichvar
Red Hat
Purkynova 115
Brno 612 00
Czech Republic

Email: mlichvar@redhat.com
Abstract

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

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 November 28, 2019.

Copyright Notice

Copyright (c) 2019 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

1. Motivation

Time synchronization is essential to Internet security as many security protocols and other applications require synchronization. Unfortunately, widely deployed protocols such as the Network Time Protocol (NTP) [RFC5905] lack essential security features, and even newer protocols like Network Time Security (NTS) [I-D.ietf-ntp-using-nts-for-ntp] fail to ensure that the servers behave correctly. Authenticating time servers prevents network adversaries from modifying time packets, but an authenticated time server still has full control over the contents of time packet and may go rogue. The Roughtime protocol provides cryptographic proof of malfeasance, enabling clients to detect and prove to a third party server's attempts to influence the time a client computes.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Authenticated Server</th>
<th>Server Malfeasance Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTP, Chronos</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>NTP-MD5</td>
<td>Y*</td>
<td>N</td>
</tr>
<tr>
<td>NTP-Autokey</td>
<td>Y**</td>
<td>N</td>
</tr>
<tr>
<td>NTS</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Roughtime</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Security Properties of current protocols

Table 1

Y* For security issues with symmetric-key based NTP-MD5 authentication, please refer to Message Authentication Code for the Network Time Protocol draft [I-D.ietf-ntp-mac]

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

More specifically,

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

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

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

2. Requirements Language

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

3. Protocol Overview

Roughtime is a protocol for rough time synchronization that enables clients to provide cryptographic proof of server malfeasance. It does so by having responses from servers include a signature with a certificate rooted in long term public/private key pair over a portion of the initial request, thus providing cryptographic proof that the timestamp was issued after previous responses and before future ones.

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

The server proves freshness of its response as follows: The request contains a random challenge. The server incorporates the challenge into its signed response so that its needed to verify the signature. This proves that the signed response could only have been generated after the challenge was issued if the challenge has sufficient entropy.

Chaining multiple servers: For subsequent requests, the client generates its nonce by hashing the reply from the first server with a random value. This proves that the nonce was created after the reply from the first server. It sends that to the second server and
receives a signature from it covering that nonce and the time from the second server.

Cryptographic proof of misbehavior: If the time from the second server is before the first, then the client has proof of misbehavior; the reply from the second server implicitly shows that it was created later because of the way that the client constructed the nonce. If the time from the second server is after, then the client can contact the first server again and get a signature that was provably created afterwards, but with an earlier timestamp.

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

4. The guarantee

A Roughtime response to a query sent at t_1, received at t_2, and with timestamp t_3 is guaranteed to have been created between the transmission of the query and its reception. If t_3 is not within that interval, a server inconsistency may be detected and used to impeach the server. The use of such a guarantee in synchronization is currently beyond the grasp of this document.

5. Message Format

A uint32 is a 32 bit unsigned integer. It is serialized in bytes with the least significant byte first. A uint64 is a 64 bit unsigned integer. It is also serialized with the least significant byte first. 8 byte timestamps are described in Section 7.

A Roughtime packet is a UDP packet whose contents are interpreted as a map from uint32s to strings of bytes. The byte strings must all have lengths a multiple of four. All uint32 are encoded with the least significant byte first. The keys of this map are called tags, and we speak of the value of a tag as the string of bytes it is mapped to.

A Roughtime packet is serialized as follows: First there is a header, The first four bytes in the header are the uint32 number of tags N, and hence of (tag, value) pairs. 4*(N-1) bytes are offsets, each offset a uint32. The last 4*N bytes are the tags.
Tags are in ascending order, and no tag can be repeated. Offsets are all a multiple of four and MUST be strictly increasing. The offset array is considered to have a not explicitly encoded value of 0 as its zeroeth entry.

Immediately following the header is a concatenation of all the strings. The first post-header byte is at offset 0, and the end of the final byte string is indicated by the end of the packet. The ith byte string ends at offset[i+1]-1, counting of course from 0, and begins at offset[i]. It is the value associated to the ith tag.

This encoding may be recursive: a value may be said to be in Roughtime format and thus have a header, etc. Tags may be listed as four ASCII characters [RFC0020]. In this case the tag when serialized will be those four ASCII characters. For example NONC would be the numeric value 0x434e4f4e. They may also be listed as fewer than four ASCII characters with hex escape codes at the end.

6. Protocol

6.1. Queries

A query is a Roughtime packet with the tag NONC. The contents of NONC are 64 bytes. The request packet MUST be a minimum of 1024 bytes. To attain this size the tag PAD\xff MAY be added at the end of the packet with a content of all zeros. Other tags MUST be ignored by the server. Future versions may specify additional tags and their semantics, so clients MUST NOT add other tags.

6.2. Responses

A response contains the following tags: SREP, SIG\x00, CERT, INDX, PATH, SREP value is itself in Roughtime format that contains the following tags: ROOT, MIDP, RADI. SIG\x00 is an Ed25519 signature [RFC8032] over the SREP value using the public key contained in CERT as explained later.

CERT in Roughtime format and contains the following tags: DELE, SIG\x00. This SIG\x00 is an Ed25519 signature over DELE that can be verified using the long term public key of the server. DELE is itself in Roughtime format containing tags MINT, MAXT, PUBK.

6.2.1. SREP

- ROOT contains the root hash value of a Merkle tree using SHA512 as described when we reach the PATH and INDX blocks
- MIDP contains an 8 byte timestamp of the moment of processing
RADI is a u32 contains the server’s estimate of the accuracy of MIDP in microseconds. Servers MUST ensure the true time is within (MIDP-RADI, MIDP+RADI) at the time they compose the response packet.

6.2.2. DELE

MINT is the minimum 8 byte timestamp at which the key in PUBK is trusted to begin signing time. MIDP > MINT for validity.

MAXT is the maximum 8 byte timestamp at which PUBK may sign. MIDP < MAXT for validity.

PUBK is a temporary Ed25519 public key. The use of this field is to enable seperation of a root public key from keys on devices exposed to the public Internet.

6.2.3. INDX and PATH

INDX is a uint32 determining the position of NONC in a Merkle tree. PATH contains the values to be hashed with the running hash as one ascends the tree. PATH is a multiple of 64 bytes long. The following algorithm verifies inclusion in the Merkle tree:

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

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

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

If the current bit is 1 one hashes \x01, the value from PATH, and the current hash.

6.3. Validity of response

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

The signature in CERT was made with the long-term key of the server.
The DELE timestamps and the MIDP value are consistent.

The INDX and PATH values prove NONC was included in the Merkle tree with value ROOT.

The signature of SREP in SIG\x00 validates with the public key in DELE.

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

7. Time

An 8 byte timestamp contains a 4 byte Modified Julian Date (as in [MJD]) followed by a 4 byte count of the number of microseconds since midnight on that day. This is not a unique representation: leap seconds are handled by changing the day number early or late, and hence having the number of microseconds increase even more. Unlike NTP this is not a representation that uses the full number of bits in the fraction part.

8. Cheater detection

A chain of responses is a series of responses where the SHA-512 hash of the preceding response, H, is concatenated with a 64 byte blind X, and then SHA-512(H, X) is the NONC used in the subsequent response. These may be represented as an array of objects in JSON where each object may have keys "blind" and "packet". Packet has the base64 encoded bytes of the packet and blind is the blind used for the next nonce. The last packet needs no blind.

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

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

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

Servers MAY send back a fraction of responses that are syntactically invalid or contain invalid signatures as well as incorrect times. Clients MUST properly reject such responses. Servers MUST NOT send back responses with incorrect times and valid signatures. Either signature MAY be invalid for this application.

10. Roughtime Servers

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

roughtime.int08h.com:2002;
AW5uAoT3STdFg3NfY1bTh08 gumq1Rb+HVhbj3ODJvsE=

roughtime.cloudflare.com:2002; gD63hSj3ScS+wu0eGrubX1q35N1c5Lby/S+T7MNTjxo=

roughtime.sandbox.google.com:2002;
etPaaIxcBMY1oUeGpwvPMcJw1RVNvx51KK/tktoJTQ=

11. Trust anchors and policies

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

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

12. Acknowledgements

Thomas Peterson corrected multiple nits. Marcus Dansarie, Kristof Teichel, Tal Mizrahi, and the other members of the NTP working group contributed comments and suggestions.
13. IANA Considerations

We request IANA assign a UDP port and create a new registry for Roughtime tags.

14. Security Considerations

This protocol will not survive the advent of quantum computers. Currently only one signature scheme is supported. Maintaining a list of trusted servers and adjudicating violations of the rules by servers are not discussed in this document and are essential for security. Arithmetic on the adjusted timescale is interesting with intervals, and this may impact the interpretation of the MAXT and MINT fields. Servers carry out a significant amount of computation in response to clients, and thus may experience vulnerability to denial of service attacks.

This protocol does not provide any confidentiality, and given the nature of timestamps such impact is minor. The compromise of a PUBK’s private key, even past MAXT, is a problem as the private key can be used to sign invalid times that are in the range MINT to MAXT, and thus violate the good behavior guarantee of the server.

Roughtime clients MUST update their view of which servers are trustworthy in order to benefit from the detection of misbehavior.

Packets sent by the client MUST be at least 1024 bytes in length in order to mitigate amplification attacks, and servers MUST ignore request packets that are smaller than this length.

15. Privacy Considerations

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

16. References

16.1. Normative References


16.2. Informative References


Authors’ Addresses

Aanchal Malhotra
Boston University
111 Cummington Mall
Boston 02215
USA

Email: aanchal4@bu.edu

Adam Langley
Google

Watson Ladd
Cloudflare
101 Townsend St
San Francisco
USA

Email: watson@cloudflare.com
A Secure Selection and Filtering Mechanism for the Network Time Protocol
Version 4
draft-schiff-ntp-chronos-02

Abstract

The Network Time Protocol version 4 (NTPv4) defines the peer process, the clock filter algorithm, the system process and the clock description algorithm. The clock filter algorithm and the system process, as defined in RFC 5905, are the mechanism according to which an NTP client chooses the NTP servers it synchronized with. This document specifies an alternative set of client mechanisms, named Chronos, that is backward compatible with NTPv4, and offers an improved level of security against time shifting attacks.

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 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

According to RFC 5905 [RFC5905], the NTP servers used for updating the client’s time are chosen by the clock filter algorithm and the system process. However, this method may be vulnerable to time shifting attacks, in which the attacker’s goal is to shift the local time of an NTP client. Time shifting attacks on NTP are possible even if all NTP communications are encrypted and authenticated. This document introduces an improved system process with a secure algorithm called Chronos. Chronos is backwards compatible with NTPv4, as an NTP client that runs Chronos is interoperable with [RFC5905]-compatible NTPv4 servers.

Chronos achieves accurate synchronization even in the presence of powerful attackers who are in direct control of a large number of NTP servers. Chronos leverages ideas from distributed computing literature on clock synchronization in the presence of adversarial (Byzantine) behaviour.
A Chronos client iteratively "crowdsource"s time queries across multiple NTP servers and applies a provably secure algorithm for eliminating "suspicious" responses and averaging over the remaining responses. Chronos is carefully engineered to minimize communication overhead so as to avoid overloading NTP servers. Chronos’ security was evaluated both theoretically and experimentally with a prototype implementation. The experimental results indicate that in order to implement a successful time-shifting attack on a Chronos client by over 100ms from the UTC, even a powerful man-in-the-middle attacker requires over 20 years of effort in expectation. The full paper is in [Chronos_paper].

Chronos differs from the current NTPv4 in two aspects. First, the Chronos client relies on a large number of NTP servers, from which only few are chosen at random in order to avoid overloading the servers. Second, the selection algorithm uses an approximate agreement technique to remove outliers, thus limiting the attacker’s ability to contaminate the chosen time samples. These Chronos client mechanisms have provable security guarantees against man-in-the-middle attackers and attackers who are capable of compromising a large number of NTP servers.

2. Conventions Used in This Document

2.1. Terminology

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

2.2. Terms and Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTPv4</td>
<td>Network Time Protocol version 4 [RFC5905].</td>
</tr>
<tr>
<td>Selection</td>
<td>Clock filter algorithm and system process [RFC5905].</td>
</tr>
</tbody>
</table>

2.3. Notations
Describing Chronos algorithm, the following notation are used.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>An upper bound on the distance from the local time at any NTP server with an accurate clock (&quot;truechimer&quot; as in [RFC5905])</td>
</tr>
<tr>
<td>Cest</td>
<td>the client’s estimate for the time that passed since its last synchronization to the server pool (sec)</td>
</tr>
<tr>
<td>ERR</td>
<td>(2W*Cest)/1000</td>
</tr>
<tr>
<td>K</td>
<td>panic trigger</td>
</tr>
<tr>
<td>tc</td>
<td>the current time, as indicated by the client’s local clock [sec]</td>
</tr>
</tbody>
</table>

Table 1: Chronos Notations

3. Extension for NTP Selection Process

A client that runs Chronos does not implement the functionality described in Sections 10 and 11 in [RFC5905]. Instead, the client implements the behavior described in this section and the next one.

3.1. Peer Calibration Process

The peer calibration process gathers a server pool of hundreds of servers. Each NTP client conducts the peer process as in Section 9 in [RFC5905], on an hourly basis for 24 consecutive hours and generates the union of all received IP addresses. Importantly, this is executed in the background once in a long time (e.g., every few weeks/months).

3.2. Chronos Selection Process

The Chronos selection process samples the server pool and removes outliers (replaces the clock filter algorithm and the system process as in [RFC5905]). First, a subset on the order of tens of the servers in the server pool is selected at random. Then, out of the tens of collected samples, the third lowest-value samples and third highest value samples are discarded.

Given the remaining samples, Chronos checks two conditions:

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

R. Schiff, et al. Expires September 12, 2019
The average value of the remaining samples is at a distance of at most \( \text{ERR} + 2w \) from the client’s local clock.

(where \( w, \text{ERR} \) are described in Table 1).

In the event that both of these conditions are satisfied, the average of the remaining samples is the "final offset". Otherwise, a few tens of the servers from the pool are sampled again, in the exact same manner. This re-sampling process continues until the two conditions are finally satisfied or the number of times the servers are re-sampled exceeds a "Panic Trigger" (\( K \) in Table 1), in which case, Chronos enters a "Panic Mode".

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

As in [RFC5905], the final offset is passed to the clock discipline algorithm to steer the system clock to the correct time.

4. Chronos Pseudocode

The Chronos pseudocode Time Sampling Scheme is the following:

```plaintext
counter := 0
While counter < K do
    S := sample(m) //gather sample from tens randomly chosen servers
    T := bi-side-trim(S,1/3) //trim third lowest and highest values
    if (max(T) - min(T) <= 2w) and (|avg(T) - tc| < \text{ERR} + 2w) Then
        return avg(t)
    end
    counter ++;
end
// panic mode;
S := sample(n);
T := bi-sided-trim(S,n/3) //trim bottom and top thirds;
return avg(T)
```

5. Precision Vs. Security

Chronos client changes the list of the sampled servers more frequently than NTPv4 [Chronos_paper], without using NTPv4 filters. This enables Chronos to be provably more secure than NTPv4 [RFC5905] but might adversely affect its precision and accuracy. Therefore we
add the following smoothing mechanism: Chronos returns the offset with minimal absolute value unless its distance from the average offset is larger than a predefined value. Another approach we considered was to use the same set of servers as in the previous sample, unless the difference between the current offset and the new offset is larger than a predefined value.

In our experiments we observed that with the smoothing mechanism, Chronos and NTP are similar in terms of precision and accuracy when there is no attack.

6. Acknowledgements

The authors would like to thank Miroslav Lichvar, Yaakov.J.Stein and Karen O’Donoghue for contributions to this document and helpful discussions and comments.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

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

Analytical results (in [Chronos_paper]) indicate that in order to succeed in shifting time at a Chronos client by even a small time
shift (e.g., 100ms), even a powerful man-in-the-middle attacker requires many years of effort (e.g., over 20 years in expectation).

It should be noted that Chronos provides resilience to MitM attacks that cannot be achieved by cryptographic authentication protocols. However, adding an authentication and crypto-based security layer to the Chronos layer is important for achieving high security guarantees and detection of various spoofing and modification attacks.

Further details about the Chronos security considerations and guarantees are discussed in [Chronos_paper].

9. References

9.1. Normative References


9.2. Informative References


[roughtime]


Authors’ Addresses

Neta Rozen Schiff
Hebrew University of Jerusalem
Jerusalem
Israel
Phone: +972 2 549 4599
Email: neta.r.schiff@gmail.com

Danny Dolev
Hebrew University of Jerusalem
Jerusalem
Israel
Phone: +972 2 549 4588
Email: danny.dolev@mail.huji.ac.il

Tal Mizrahi
Huawei Network.IO Innovation Lab
Israel
Email: tal.mizrahi.phd@gmail.com

Michael Schapira
Hebrew University of Jerusalem
Jerusalem
Israel
Phone: +972 2 549 4570
Email: schapiram@huji.ac.il
Network Time Protocol Extended Information Extension Field
draft-stenn-ntp-extended-information-04

Abstract

The core network packet used by NTP has no spare bits available for reporting additional state information and no larger data areas available for larger amounts of information. This proposal offers a new extension field that would contain this additional information.

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 26, 2019.

Copyright Notice

Copyright (c) 2019 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

The core NTP packet format has changed little since RFC 958 [RFC0958] was published in 1985. Since then, there has been demonstrated need to convey additional information about NTP’s state in an NTP packet but no backward-compatible way to usurp the few otherwise potentially available bits has been found, and no larger data areas are available in the core packet structure. This proposal offers a new extension field that would contain this additional information.

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. The Extended Information Extension Field

The Field Type of the Extended Information EF includes a version number field in the low-order bits of the first octet, to make it easier to evolve this specification. The initial specification for this proposal uses Version 0, which equates to 0x0009 [ADJUST AS NEEDED BASED ON IANA, IF AN IANA REGISTRY IS USED]. A future revision for Version 1 would use 0x0109 [IBID].

The payload for Version 0 is comprised of a two octet Content Descriptor followed by a two octet Content Data field, as described below.
NTP Extension Field: Extended Information

Field Type: TBD (Recommendation for IANA: 0x0009 (Extended-Information, Version 0))

Field Length: as needed

2.1. Version 0 Content Descriptor and Content Data fields

There are 16 bits available for state information in the Version 0 Extended Information Content Descriptor. These bits are allocated as follows:

0x0001: TAI Offset is stored in the low-order 8 bits (the second octet) of the Content Data.

0x0002: Interleave Mode indicator in the low order bit of the first octet of the Content Data. [NOTE: this may not be useful, and it can be removed if desired. It can serve as a belt-and-suspenders way to identify when a packet contains interleaved timestamps.]

0xFFFD: Reserved for future versions. SHOULD be zeroes for Version 0, and the meaning of any nonzero values is unspecified.

The Content Data field of the Version 0 Extended Information extension field is comprised of two octets, with the contents allocated as follows:

0xXXNN: The low-order 8 bits (NNNN) are the TAI Offset. Any data in the high-order 8 bits (XXXX) are not part of the TAI Offset.

0xX0XX: A value of 0 in the low-order bit of the first octet indicates that the timestamps in the base packet are not interleave-mode timestamps.

0xX1XX: A value of 1 in the low-order bit of the first octet indicates that the timestamps in the base packet are interleave-mode timestamps.

0xN2XX: thru
0xNDXX: Any of the seven high-order bits in the first octet are reserved for future versions and SHOULD be zero for Version 0. The meaning of any nonzero values is unspecified.

Content Descriptor 1       Content Data 1
0x0001            TAI offset in the low-order 8 bits, 24-31
0x0002            Interleave Mode indicator in Bit 23
0xFFFD            Reserved (Zeroes)

Interleave Mode: 1 if the sender is in interleave mode, 0 otherwise

NTP Extension Field: Extended Information, Version 0 Content Fields

Example: A system that wants to convey an offset to TAI of 36 seconds, and show it is in interleave mode.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|    Field Type (0x0009)        |   Field Length (0x0008)       |
+-------------------------------+-------------------------------+
|            0x0003             |           0x0124              |
+-------------------------------+-------------------------------+

NTP Extension Field: Extended Information V0, Example
```

3. Acknowledgements

The author wishes to acknowledge the contributions of Martin Burnicki and Sam Weiler.

4. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Type

```
0x0009  (Extended-Information, Version 0)
```

for this proposal.

5. Security Considerations

No unusual or special security considerations are known to be associated with this proposal.
6. Normative References


Author’s Address

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

Email: stenn@nwtime.org
Network Time Protocol I-Do Extension Field
draft-stenn-ntp-i-do-06

Abstract

This proposal defines and describes a mechanism by which cooperating
NTP instances may communicate any optional features they are willing
to admit they support.

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH BEFORE PUBLISHING:

The source code and issues list for this draft can be found in
https://github.com/hstenn/ietf-ntp-i-do

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 26, 2019.

Copyright Notice

Copyright (c) 2019 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 first implementation of NTPv4 was released in 2003, and was defined by RFC 5905 [RFC5905]. It contains an optional and now obsolete public-key security protocol, Autokey, which is defined by RFC 5906 [RFC5906]. Until very recently, Autokey has been the only implemented use of NTP packet Extension Fields. New proposals for extension fields are being written and there is currently no convenient way to learn if a remote instance of NTP supports any extension fields or not. This proposal contains a method to tell a remote instance of NTP what we (are willing to admit we) support, and ask what they (are willing to admit they) support.

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. The I-Do Extension Field

2.1. Overview

The purpose of the I-DO EF is to provide information to the remote side about our capabilities.

If an incoming packet contains an unrecognized extension field, one of several things will happen. While that unrecognized extension
field SHOULD be ignored, an implementation MAY choose to drop the entire packet.

If any extension field is present there ordinarily SHOULD be a MAC following the extension field. However, an older conforming NTP implementation will require that any EF MUST be followed by a MAC.

Some extension fields are unable to be "signed" by a MAC, regardless of whether or not that MAC is a traditional MAC or an extension field MAC.

In the previous two cases, a conforming legacy system that receives these types of packets will interpret the unrecognized EF as a missing or legacy MAC, and return a crypto-NAK.

If the remote system replies with a crypto-NAK, that is a good indication that it is running older software that does not recognize EFs and thinks we have sent an invalid MAC. In this case, we SHOULD NOT send that system newer EFs.

If the remote system replies without including an I-DO-RESPONSE EF, we at least know they can handle EFs, but they either don’t understand I-DO or are not willing to tell us anything. In this case, we SHOULD NOT send any newer EFs.

If the remote system replies with a packet that includes an I-DO-RESPONSE EF, then we SHOULD remember what they told us, and use that information appropriately. In other words, we can exchange packets containing any new EFs that we agree on, and we should not exchange packets containing any new EFs that we have not agreed on.

In client/server mode, it makes sense for the client to send an I-DO to the server, and notice how the server responds. While the server SHOULD respond with an I-DO-RESPONSE EF, it likely does not make sense for the server to send an I-DO EF in response to a client request.

In symmetric mode, either side may initiate sending an I-DO EF, and the receiving side SHOULD reply with an I-DO-RESPONSE EF.

In broadcast mode, the broadcast server MAY send broadcast packets that include an I-DO EF, but note that if, counter to recommended practice, these packets are unauthenticated they MAY cause client machines to misinterpret the packet as having invalid authentication. In this situation, the broadcast server SHOULD alternate sending broadcast server packets with and without an I-DO EF, to insure that all clients receive time packets they will accept. Note that if, as recommended, broadcast packets are authenticated, a conforming client
SHOULD have no difficulty in receiving a broadcast (mode 5) packet from a server that includes an I-DO EF.

2.2. I-DO Packet Format

The content of the I-DO extension field is an ordinary four octet Extension Field header followed by a payload consisting of an appropriate number of two octet I-DO values that use nonzero values to indicate a supported feature. An I-DO value of zero is ignored. The payload section must end on a four-octet boundary.

There are two types of nonzero I-DO values that may be used. They are both defined in the IANA NTP Extension Field Table (Section 4). These values are either Extension Field Types, where only the low-order values (0x01 thru 0xFE) are used, or I-DO Types, where all 16 bits are used and the bottom octet is currently always 0xFF.

The examples below are built using information from the following Standards and proposals:

- RFC 5906 [RFC5906]
- NTP-EXTENSION-FIELDS [NTP-EXTENSION-FIELD]
- MAC-LAST-EF [DRAFT-MAC-LAST-EF]

| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 |
|---------------|------------------|
| | Field Type    | Field Length     |
| | I-Do 1        | ...              |
| | I-Do N        | Padding          |

NTP Extension Field: I-DO - Overview

Field Type: TBD (Recommendation for IANA: 0x0007 (I-Do), 0x8007 (I-Do Response))

Field Length: as needed

Payload: An enumeration of the supported base Field Types, followed by any zero padding (0x0000) needed to fill the payload to the desired 32-bit boundary.
Example: A system that wants to advertise support for Autokey and I-DO, sending to a system that responds with support for I-DO, NTS, MAC-EF, and LAST-EF.

```
+---------------+---------------+-------------------------------+
|    Field Type (0x0007)        |   Field Length (0x0008)       |
+-------------------------------+-------------------------------+
|            0x0007             |           0x0002              |
+-------------------------------+-------------------------------+
```

NTP Extension Field: I-Do - Example

```
+---------------+---------------+-------------------------------+
|    Field Type (0x8007)        |   Field Length (0x000a)       |
+-------------------------------+-------------------------------+
|            0x0003             |           0x0004              |
|            0x0007             |           0x0008              |
+-------------------------------+-------------------------------+
```

NTP Extension Field: I-Do Response - Example

2.3. Behavior

The sender of any I-Do extension field MUST send an extension field with a Field Type of 0x0007 (I-Do) and SHOULD include a payload with any 0x0000 padding values after enumerating the supported base Extension Field Types. If the responding system recognizes the I-Do extension field, its response MUST include an extension field with a Field Type of 0x8007 (I-Do Response), and SHOULD include a payload with any 0x0000 padding values after enumerating the supported base Extension Field Types.

Any system that receives an I-Do extension field as either an "offer" or a "response" SHOULD scan the entire payload looking for nonzero values that specify the capabilities of the remote association.

Any system that receives an I-Do "offer", 0x0007, SHOULD reply with an I-Do "response", 0x8007.

Any system that sends an I-Do "offer" or "response" may send as few or as many of its supported Field Types as it chooses. At any subsequent time, either side may re-negotiate the list of supported
field types it is prepared to accept from the other system by sending a new I-Do extension field.

The most-recently received I-Do list replaces any previous I-Do list.

3. Acknowledgements

The author wishes to acknowledge the contributions of Sam Weiler.

4. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types:

- 0x0007 (I-DO)
- 0x8007 (I-DO Response)

and NTP Extension Field I-DO types:

- 0x00FF through
- 0xFDFF Reserved for future I-DO types
- 0xFEFF (I-DO Leap Smear REFIDs)
- 0xFFFF (I-DO IPv6 REFID hash)

for this proposal.

5. Security Considerations

No additional or unusual security considerations are expected if this proposal is adopted.

No feedback has been received suggesting this proposal creates any new security considerations.

6. References

6.1. Normative References


6.2. Informative References

[DRAFT-MAC-LAST-EF]

[NTP-EXTENSION-FIELD]


Author’s Address

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

Email: stenn@nwtime.org
Abstract

NTP packets can be authenticated by a Message Authentication Code (MAC) if a MAC is present at the end of an NTP packet. The legacy format for this MAC is not formatted as an NTP Extension Field, and its presence may cause some implementations a parsing ambiguity.

This proposal introduces two ways to resolve this problem. One is to provide a MAC Extension Field. The other is an extension field that unambiguously declares itself to be the last extension field in an NTP packet (so any additional data MUST be a legacy MAC).

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH BEFORE PUBLISHING:

The source code and issues list for this draft can be found in https://github.com/hstenn/ietf-ntp-mac-last-ef

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 26, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

NTPv4 is defined by RFC 5905 [RFC5905], and it and earlier versions of the NTP Protocol have supported symmetric private key Message Authentication Code (MAC) authentication. MACs were first described in Appendix C of RFC 1305 [RFC1305] and are further described in RFC 5905 [RFC5905]. As the number of Extension Fields grows there is an increasing chance some implementations will find a parsing ambiguity when deciding if the "next" set of data is an Extension Field or a legacy MAC. This proposal defines two new Extension Fields to avoid this potential ambiguity. One, LAST-EF, is used to signify that it is the last Extension Field in the packet. If the LAST-EF is present, any subsequent data MUST be considered to be a legacy MAC, or if you prefer, any subsequent data MUST NOT be considered to be an EF. The other, MAC-EF, allows one or more MACs to be encapsulated in an Extension Field. If all parties in an association support MAC-EF, the use of a legacy MAC may be avoided.

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. The Last Extension Field Extension Field - LAST-EF

Now that multiple extension fields are a possibility, additional packet data could be either an Extension Field or a legacy MAC. Having a means to indicate that there are no more Extension Fields in an NTP packet and any subsequent data MUST be something else, almost certainly a legacy MAC, is a valuable facility.

The format of a LAST-EF is an Extension Field comprised of an identified Field Type and an appropriate Field Length.

In the example below the Field Length in the LAST-EF is 4, because there is clearly no need in this case for the 28 octets required by RFC 7822 [RFC7822]. But the LAST-EF could have any supported length, as any payload is ignored.

NTP Extension Field: Last Extension Field - LAST-EF

Field Type: TBD (Recommendation for IANA: 0x0008 (Last Extension Field))

Field Length: 4 (minimum)

Payload: Ignored if present - none needed. SHOULD be zeroes.

Example:
3. MAC Extension Field

Now that multiple extension fields are a possibility, there is a chance that additional packet data could be either an Extension Field or a legacy MAC. There is benefit to encapsulating the MAC in an extension field. By encapsulating the MAC in an EF, we also have the option to include multiple MACs in a packet, which may be of use in broadcast scenarios, for example.

There are two forms of this extension field. The first supports a single MAC, requiring 4 octets’ overhead for the EF header. The second form supports one or more MACs in the EF payload, and requires at least 8 octets.

The format of a MAC-EF is an Extension Field comprised of an identified Field Type and an appropriate Field Length.

A Field Type value of TBD (0x0003 is suggested) identifies this extension field as a MAC Extension field for a single MAC. In this case, the payload consists of the four octet MAC Key ID followed by the MAC digest, and any desired (possibly random data) padding.

A Field Type value of TBD (0x0103 is suggested) identifies this extension field as a MAC extension field for one or more MACs. In this case, the payload consists of an unsigned 16-bit MAC Count (N) followed by N unsigned 16-bit MAC length fields. If there are an even number of MACs specified there is an unused 16-bit field which SHOULD be 0x0000 at the end of the set of MAC length values so that the subsequent MAC data is longword (4-octet) aligned. Each MAC
SHALL be padded so that any subsequent MAC starts on a 4-octet boundary. Optional (possibly random data) padding is allowed.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|     Field Type (0x0003)       |        Field Length           |
|-------------------------------+-------------------------------|
+-------------------------------+-------------------------------+
.    MAC 1 Key ID              .
.    +-------------------------.  
.    MAC 1 Key Data  | Random Data Padding |
.    +-------------------------.  

NTP Extension Field: MAC EF Format (Single MAC)

Field Type: TBD (Recommendation for IANA: 0x0003 (MAC-EF: Single MAC))

Field Length: As needed.

Payload: As described.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|     Field Type (0x0103)       |        Field Length           |
|-------------------------------+-------------------------------|
+-------------------------------+-------------------------------+
|          MAC Count            |        MAC 1 Length           |
|-------------------------------+-------------------------------|
|          MAC 2 Length         |        MAC 3 Length           |
|-------------------------------+-------------------------------|
+-------------------------------+-------------------------------+
.    MAC 1 Key ID              .
.    +-------------------------.  
.    MAC 1 Key Data  | Random Data Padding |
.    +-------------------------.  
.    MAC 2 Key ID              .
.    +-------------------------.  
.    MAC 2 Key Data  | Random Data Padding |
.    +-------------------------.  
.    MAC 3 Key ID              .
.    +-------------------------.  
.    MAC 3 Key Data  | Random Data Padding |
.    +-------------------------.  
.    Padding (as needed)      .
+-------------------------.  

NTP Extension Field: MAC EF Format (1 or more MACs)
Field Type: TBD (Recommendation for IANA: 0x0003 (MAC-EF: Single MAC), 0x0103 (MAC-EF: 1 or more MACs))

Field Length: As needed.

Payload: As described.

A MAC consisting of 4 octets of zeros means the MAC is a crypto-NAK, as defined by RFC5905 [RFC5905].

Additional MACs SHOULD NOT be present if there is a crypto-NAK present in the packet.

Each MAC within the extension field consists of a 32-bit key identifier which SHOULD be unique to the set of key identifiers in this MAC extension field followed by ((MAC Length) - 4) octets of data, optionally followed by random octets to pad the key data to the length specified earlier in the extension field. That key identifier is a shared secret which defines the algorithm to be used and a cookie or secret to be used in generating the digest. The MAC digest is produced by hashing the data from the beginning of the NTP packet up to but not including the start of the MAC extension field. The calculation of the digest SHOULD be a hash of this data concatenated with the 32-bit keyid (in network-order), and the key. When sending or receiving a key identifier each side needs to agree on the key identifier, algorithm and the cookie or secret used to produce the digest along with the digest lengths. Note that the sender may send more bytes than are required by the digest algorithm. This would be done to make it more difficult for a casual observer to identify the algorithm being used based on the length of the data. The digest data begins immediately after the key ID, and any padding octets SHOULD be random.

4. Acknowledgements

MAC-EF: The authors gratefully acknowledge Dave Mills for his insightful comments. Hal Murray asked if there was a way for the MAC-EF to require only 4 octets of overhead if there was only a single MAC in the payload.

5. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types:

0x0003 MAC-EF (Single MAC)

0x0103 MAC-EF (1 or more MACs)
6. Security Considerations

The security considerations of time protocols in general are discussed in RFC7384 [RFC7384], and the security considerations of NTP are discussed in RFC5905 [RFC5905].

Digests MD5, DES and SHA-1 are considered compromised and should not be used [COMP].

[DISCUSS] Each MAC length should be at least 20 octets long to allow for 4 octets of key ID and at least 16 octets of digest and random padding. For a 128-bit digest, there would be 4 octets of key ID, 16 octets of digest, plus any desired octets of random padding. For SHA-256 digests there are 4 octets of key ID, 32 octets digest, plus any desired octets of random padding. Using MAC lengths that include random padding may make it more difficult for an attacker to know which digest algorithms are used.

7. Normative References


Authors' Addresses

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

Email: stenn@nwtime.org

Danny Mayer
Network Time Foundation
P.O. Box 918
Talent, OR  97540
US

Email: mayer@ntp.org
Abstract

NTP’s Reference ID, or REFID, identifies the source of time in a timestamp or time packet. In NTP packets sent over the network the REFID is used to identify the "system peer", and in the long-term general case its fundamental purpose is to prevent a one-degree timing loop. Each instance of NTP decides for itself what REFID it will put in its outgoing packets, and there is currently no way for an external time source to tell or recommend this value in the case where that external time source is selected as the "system peer."

The SUGGESTED-REFID NTP Extension Field proposal is a backward-compatible way for a time source to tell its peers or clients "If you use me as your system peer, use this nonce as your REFID."

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 26, 2019.

Copyright Notice

Copyright (c) 2019 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

NTP has been widely used through several revisions, with the latest being RFC 5905 [RFC5905]. A core component of the protocol and the algorithms is the Reference ID, or REFID, which is used to identify the time source. Traditionally, when the source of time was another system the REFID was the IPv4 address of that other system. If the remote system was using IPv6 for its connection, a 4 octet digest value of the IPv6 address was used. The general case core purpose of the REFID is to prevent a one-degree timing loop (where if A has several timing sources that include B, if B decides to get its time from A we don’t want A then deciding to get its time from B). The REFID is considered to be "public data" and is a vital core-component of the base NTP packet. In an increasingly hostile Internet, knowledge of a system’s time source is abusable information. If a system’s REFID is the IPv4 address of its system peer, an attacker can try to use that information to send spoofed time packets to either or both the target or the target’s server, attempting to cause a disruption in time service. There is also a clear use-case for having a special REFID for use if systems are exchanging leap-smeared time. This proposal is a backward-compatible way for a time source to tell its peers or clients "If you use me as your system peer, use
This nonce, a Suggested REFID, SHOULD be untraceable to the sending system. When used to hide the identity of a server, if the receiving system uses this Suggested REFID nonce instead of the IPv4 address as its REFID, this type of attack and information disclosure is prevented. When used to indicate that a system is either offering leap-smeared time or is synchronized to a leap-smeared time source, this information can be used to prevent unwanted synchronization to a source that is not offering the "flavor" of time we want, and, in the case where a leap smear correction continues into the next day, the second half of a leap smear correction can be applied in the expected manner.

This SUGGESTED-REFID NTP Extension Field proposal is a simple, clean, backward-compatible way for an external time source to request that the receiving system use the provided nonce in the case where the receiving system uses the sending system as its system peer.

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. The REFID

The core reason for the REFID in the NTP Protocol is to prevent a timing loop of degree 1. Put another way, if servers A and B are exchanging time with each other and server B decides to follow A as its system peer, the REFID that B will use must be able to identify server A. The interpretation of a REFID is based on the stratum, as documented in RFC 5905 [RFC5905], section 7.3, "Packet Header Variables". At Stratum 2+, which will be the case if servers A and B are exchanging packets over IPv4, if server B follows A, then B will have A's IPv4 address as its REFID. When A asks B for its time, A will see that B is synchronized to A because B will tell A that its REFID is A's IPv4 address, so when A sees its IP address as B’s REFID, A knows that if it were to follow B for its time then there would be a timing loop. In this case, A will not select B as a potential source of time.

Another related use case for the REFID centers around the increasing use of leap-smeared time servers when the insertion (or any eventual deletion) of a leap second occurs. It is critical that operators and client systems be able to identify when a server is offering leap-smeared time. Furthermore, with the current practice of smearing the insertion of a leap second starting at noon UTC on the day of the leap event and completing the smear at noon UTC on the day after the leap event, a server that is operating during a leap smear event must
be able to immediately identify if it should respond with either
correct or leap-smeared time.

3. The Suggested REFID Extension Field

Since there is no way in the base NTP packet for "this" instance of
an NTP server to tell the "other" instance what REFID it should use
if the "other" instance decides to use "this" instance as its system
peer, the best available way to convey this information is via an
extension field.

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
+---------------+---------------+-------------------------------+
<p>|          Field Type           |        Field Length           |
|---------------------------------------------------------------|</p>
<table>
<thead>
<tr>
<th>Suggested REFID</th>
</tr>
</thead>
</table>

NTP Extension Field: REFID Suggestion

Field Type: TBD (Recommendation for IANA: 0x0006 (Suggested REFID))

Field Length: 0x0008

Suggested REFID: The 4 octets of the suggested REFID. Random nonce
REFID values SHOULD be 0xFDxxxxxx, where the bottom 3 octets SHOULD
be random values.

Examples: When decoded as an IPv4 address, a random nonce suggested
REFID would decode as 253.0.0.0 thru 253.255.255.255.

4. Generating and Sending a Nonce as the Suggested REFID Extension
Field

A system that decides to send a nonce as a Suggested REFID extension
field SHOULD generate a new Suggested REFID nonce for each new
association. It MAY generate a new Suggested REFID nonce for any
association in any response. In addition to remembering the IP-based
REFID, the sender MUST also remember its most-recent Suggested REFID
nonce.

Since the core NTPv4 and earlier protocols do not contain any way to
tell the recipient what to use as a REFID and RFC 5905 [RFC5905] uses
the IPv4 address of the sender as the REFID if the association is
effected over an IPv4 connection, this means that an attacker can
simply send an NTP client request to a server knowing that server’s
system peer will be returned as the REFID in the response packet. At
this point, an attacker can, if that REFID is an IPv4 address, begin
to launch attacks at the target forging the putative IP of the
target’s time source, or the attacker can start forging packets to
the putative time server claiming to be from the target, in an
attempt to cause the time server to limit or deny time service to the
target.

Using a nonce for the REFID that is only recognized by the sending
machine effectively prevents this type of attack.

If servers S1, S2, and S3 are all exchanging time with each other and
are all using the Suggested REFID mechanism, there is a 3 in
16,777,216 (2^24) chance that two different servers in the same group
will happen to choose the same nonce, and that would produce a false-
positive timing loop detection. If a nonce Suggested REFID is never
changed, this false-positive condition will occur for potentially a
long time. This small risk can be reduced by periodically generating
a new Suggested REFID.

5. Remembering a Nonce Suggested REFID Extension Field

An NTP server keeps track of the IP address it uses to talk to its
peers. If an NTP server chooses to send a Suggested REFID to an
associated peer, the server MUST remember this value. When checking
for a timing loop, the Suggested REFID must also be included in the
list of tested REFID values.

A set of NTP servers that are acting as a group of time servers
SHOULD be using peer associations (NTP mode 1 and 2 packets), and
SHOULD NOT be using client/server (NTP mode 3 and 4) exchanges.
Nevertheless, implementors should be aware that the recommendation
against using client/server associations for time groups may be
ignored, and should be conscious of the choices they make and the
configuration options they offer in order to accommodate (or at least
document) this situation.

6. The Suggested REFID Extension Field and Leap Smear REFIDs

The Suggested REFID can play an important part when a server has a
client population that receives leap-smeared time.

The current preferred behavior for servers that offer leap-smeared
time is to offer leap-smeared time in response to appropriate client
(mode 3) requests. There are two competing forces at play during
this time:

- Clients that want correct time should get correct time.
Clients that want leap-smeared time should get leap-smeared time. An additional complication is that a leap-second insertion event begins at noon UTC, when the Leap Indicator is 1, but the smear is only halfway applied at midnight UTC, when the Leap Indicator changes back to 0. There is no simple way for the client to let its server(s) know that it is using leap-smeared time.

One simple way for the client to let its server(s) know that it is using and wants leap-smeared time is for the client to use a Leap Smear REFID [DRAFT-LEAP-SMEAR-REFID] in its client (mode 3) requests during the entire leap smear period.

7. Acknowledgements

The author wishes to acknowledge the contributions of Martin Burnicki and Sam Weiler.

8. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Type 0x0006 (Suggested REFID) for this proposal.

9. Security Considerations

Adopting this proposal will provide a much needed mechanism by which cooperating systems can agree on a less trackable and less identifiable nonce for the REFID. It will also provide a means to properly and better handle leap-smearing events with populations where some clients want correct time and other clients want leap-smeared time, thus enabling better time synchronization.

No reports of adverse consequences of adopting this proposal have been received.

10. References

10.1. Normative References


10.2. Informative References

[DRAFT-I-DO]

[DRAFT-LEAP-SMEAR-REFID]

Author’s Address

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

Email: stenn@nwtime.org