Time synchronization performance using the network time security protocol

Martin Langer, Kristof Teichel, Dieter Sibold and Rainer Bermbach

Full Citation of the original article published by IEEE:

M. Langer, K. Teichel, D. Sibold and R. Bermbach, " Time synchronization performance using the network time security protocol," 2018 European Frequency and Time Forum (EFTF), Turin, Italy, 2018, pp. 138-144, doi: 10.1109/EFTF.2018.8409017.

Available at:
https://doi.org/10.1109/ EFTF.2018.8409017

Time Synchronization Performance Using the Network Time Security Protocol

Martin Langer[*], Kristof Teichel[†], Dieter Sibold[†] and Rainer Bermbach[*]
Email: mart.langer@ostfalia.de, kristof.teichel@ptb.de, dieter.sibold@ptb.de, r.bermbach@ostfalia.de
[*]Ostfalia University of Applied Sciences, Wolfenbüttel, Germany
[†]Physikalisch-Technische Bundesanstalt, Braunschweig, Germany

***Abstract*—This paper compares the time synchronization performance of standard NTP versus NTP secured using the Network Time Security (NTS) protocol. The measurements were performed using the NTS software of Ostfalia University of Applied Science – the first implementation of NTS based on the IETF internet draft “draft-ietf-ntp-using-nts-for-ntp-06”. The measurements quantify the impact of the security measures on the time synchronization performance and allow conclusions to be drawn regarding efficiency and potential improvements to the protocol.**

***Keywords*—Network Time Security (NTS); Network Time Protocol (NTP); security; authentication; integrity**

I. Introduction

Time synchronization of networked computer systems is of paramount importance for the correct functioning and interoperability of many computer applications. The protection of time synchronization protocols is vital in order to both counter existing threats and comply with legal requirements. To date, time information has usually been disseminated in an unsecured form that creates opportunities for adversaries to maliciously alter time information [1]. An important and widely used example of a tool for disseminating time information is the Network Time Protocol (NTP) [2-4]. NTP meets the accuracy requirements of many application fields but offers only limited security measures. An analysis of security mechanisms specific to NTP demonstrated that they were either not secure or unsuitable for practical use [5]. This prompted the development of Network Time Security (NTS), which is currently in the final specification procedure. NTS was designed to solve the known security issues of NTP without sacrificing the accuracy and stability provided by the time synchronization process using standard NTP.

In this paper, we measure the impact of NTS concerning the efficiency and time synchronization performance of NTS-secured NTP. The measurements presented here are based on the NTS version defined by a set of three internet drafts: draft-ietf-ntp-using-nts-for-ntp-06 [6], draft-ietf-ntp-network-time-security-15 [7], and draft-ietf-ntp-cms-for-nts-message-06 [8]. These drafts were implemented by the Ostfalia University of Applied Sciences in cooperation with PTB [9] and refined in follow-up work, which has now been used to analyze the essential properties of this NTS version. Chapter II introduces the reader to the technical aspects of NTP and NTS. Chapter III describes the measurement setup, while the results of the measurements are presented and discussed in Chapter IV. This analysis also allows the inefficient steps in the NTS protocol to be identified; this information can be used to improve such protocol steps in the ongoing specification process.

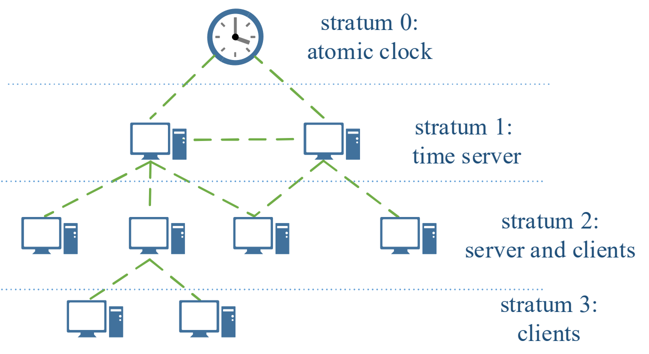


Fig. 1. Hierarchical structure of NTP time servers

II. Preliminaries

This section gives an overview of the main aspects of NTP and NTS, to the extent to which they are relevant to the work presented.

A. The Network Time Protocol (NTP)

One of the most widely used protocols for clock synchronization worldwide is NTP, which is deployed on nearly every computer system in existence. Developed by David L. Mills in 1985 and presented in the IETF’s RFC 958 [3], its Version 4 (NTPv4) has been available as RFC 5905 [4] since 2010. NTP communicates via packet-switched networks using the connectionless communication model of the UDP internet protocol to send and receive timestamps. What is most commonly used is a client/server model combined with a hierarchical structure as seen in Fig. 1. Available time servers are categorized according to their distances from the reference clock. Each level is called a *stratum* and given a number according to the distance. A reference clock would be stratum 0, the time server on the level below would be stratum 1 and so on. As the stratum number increases, the accuracy typically decreases.

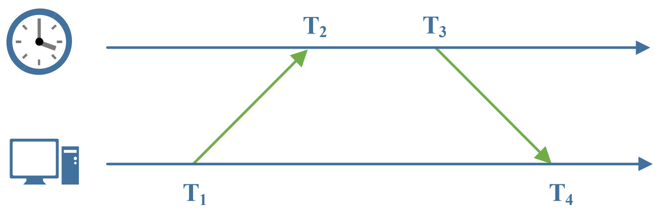


Fig. 2. Timestamps used in the NTPv4 Protocol

NTP uses different modes of operation that determine the kind of communication between NTP protocol objects. In *symmetric mode,* an NTP instance can distribute time information as well as receive it. It synchronizes its clock via calibration with servers of the same stratum. In *broadcast mode,* a server periodically transmits time information that can be used for clock synchronization by any NTP client that receives it. NTP's most prominent mode is the *client/server mode*, which applies a *unicast* communication model. In this mode, a client transmits time requests to an NTP server (once or periodically), whereupon the server responds by sending the time information required [2, 4]. In client/server mode, clock synchronization between an NTP client and an NTP server applies a two-way time transfer approach [10]: At time t_1, the client sends an NTP packet to the server containing the timestamp $T_1 = T(t = t_1)$. Upon arrival at time t_2, the server processes the packet, and at time t_3, it inserts the timestamps T_2 and T_3 and transmits the packet back to the client, where it is received at time t_4. This communication yields the four timestamps displayed in Fig. 2. Note that the clock at the client produces the timestamps T_1, T_4 and the clock at the server creates the timestamps T_2, T_3.

From these timestamps, the NTP client can derive the network delay δ and the time offset θ between the client and the server. Note that δ represents the packet round-trip time excluding the computing time at the server, indicating the time the packets have traveled in the network. Calculation of θ and δ is done according to the following equations [2, 4]:

$$\delta = (T_4 - T_1) - (T_3 - T_2) \tag{1}$$

$$\theta = ((T_2 - T_1) - (T_4 - T_3)) / 2 \tag{2}$$

Assuming that the propagation delay of the packets between the client and the server is the same in both directions, the offset θ quantifies the time difference between the clocks of the NTP client and its server. The NTP process continuously aims to minimize θ by adjusting the frequency of the client's system clock.

1) Issues of Previous Security Measures

In most cases, NTP transmits the time synchronization packets without any cryptographic protection. This allows an adversary to alter, replay or dismiss time synchronization packets, or to inject false packets. A comprehensive list of known threats to time synchronization protocols is compiled in RFC 7384 [1].

To increase security, the *symmetric key* method was introduced in 1992 as part of the NTPv3 protocol. It permits authentication of NTP participants and ensures the integrity of the transmitted time data by means of cryptographic hash functions. Although this method is still considered secure, it does not provide a key exchange mechanism or scale well for large network deployments or for the global internet. In order to address the scalability issue, the Autokey method was introduced with NTPv4 and specified in the informational RFC 5906 [11]. However, an in-depth analysis of Autokey [12, 13] exhibited severe security vulnerabilities. Therefore, Autokey is no longer considered secure or recommended for use [14].

Tunnel solutions such as TLS (Transport Layer Security) and IPSec offer another approach to adding security to time dissemination. Although they provide a secure connection for sending NTP packets, they entail two fundamental disadvantages. On the one hand, they require a permanent, stateful connection between the server and the client. Such a connection involves many resources, especially on the server side. On the other hand, tunnel solutions do not consider the unique requirements associated with the dissemination of time information, thus causing synchronization performance to decline. Consequently, they are of limited use.

B. The Network Time Security Protocol

Originally motivated by demand for cryptographically secured time synchronization mechanisms created within the scope of the Smart Grid Initiative of the German Federal Ministry of Economic Affairs and Energy, the Network Time Security (NTS) protocol offers a solution to the security problem. It extends existing time synchronization protocols and thus allows secure time synchronization in networks. NTS is currently under development, but is approaching finalization and publication as a standards track RFC of the IETF. The present draft version is draft-ietf-ntp-using-nts-for-ntp-11 [15].

The main goals of NTS are to enable NTP clients to cryptographically identify their NTP servers, ensure authenticity and integrity for exchanged time packets and ensure that the time synchronization quality is impacted as little as possible. To this end, an NTS-protected association between an NTP client and a server is established in a first phase, the so-called key exchange phase. In this stage, the client and server will negotiate the cryptographic algorithms, exchange certificates and generate cryptographic keys. In addition, the client will verify the authenticity of the server by means of a Public Key Infrastructure (PKI). In the subsequent phase, the participants exchange secured time synchronization packets protected by a Message Authentication Code (MAC), a digital fingerprint enabling the detection of any data manipulation. See [5] for further information. NTS does not require the NTP server to save the state of the client. Instead, it utilizes stateful clients and stateless servers. The server can process the requests of a client based solely on the information provided in the request packet.

III. Measurement Setup and Configuration

The goal of the measurements was to compare different implementations (unsecured and secured NTP) in terms of their performance, and to compare the resulting accuracy and stability of the synchronized clocks. Hence, the tests recorded the

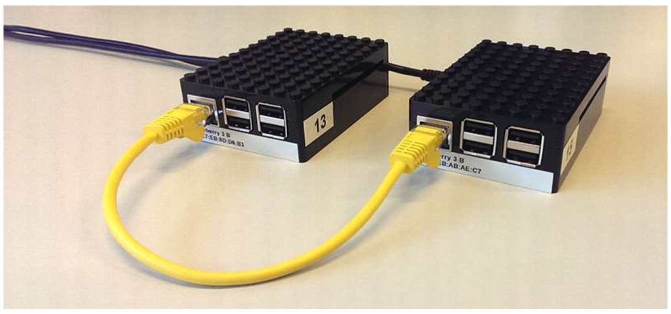


Fig. 3. Measurement setup with a direct client/server connection

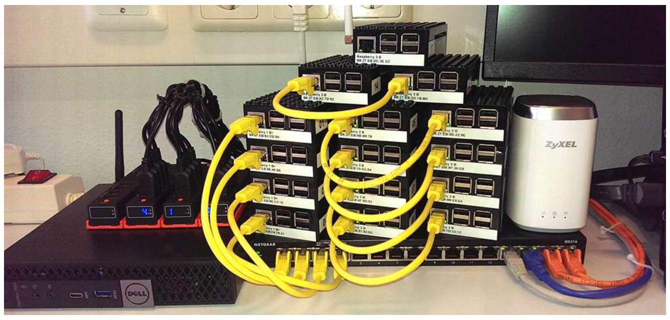


Fig. 4. Composite of several measurement setups

time offset, the delay and the processing time of each implementation. The tests also considered the asymmetrical round-trip time of NTS. To obtain consistent and comparable results, identical conditions and simultaneous measurements were fixed for all measurement setups. The hardware was based on third-generation Raspberry Pi[1] devices, all of which were in closed standard boxes running identical software. They differed only in the individual configuration of the NTP implementation used and in their specific role in the communication process (NTP server or client). Data was exchanged between the client and the server via direct coupling using a Fast-Ethernet cable connection[2] (s. Fig. 3). In this way, one client always interacted with one server. All Raspberry Pis were operated in headless mode[3] and controlled via Wi-Fi, which is already integrated in these models. Thus, network traffic due to control messages had no impact on the measurements. Several of these units were assembled according to Fig. 4.

On the software side, the *Raspbian lite* operating system [16] formed the base of the measurements. All services needed by the system to synchronize the clock were set up manually using three different NTPv4 implementations or configurations. Two different realizations of NTP software were applied: the so-called reference implementation provided by the Network Time Foundation [17] (in the following referred to as **NTPD**[4]), and the implementation given in [18] and referred to throughout this paper as **NTP**. This realization offers an interface to plug in NTS functions, thus enabling secured time synchronization; we refer to this configuration as NTS or **NTP (NTS)**. The NTS service version employed [6] is operational in unicast client/server mode and is designed to ensure security by embedding additional security data into existing time transmission protocols without any further modification to those protocols. In the case of NTP, this is accomplished by encapsulating NTS-related content in NTP extension fields. Such fields are defined in RFC 7822 [19]. Both NTPD and our own NTS-secured NTP solution can each act as either a client or a server.

The timestamps that an NTS-enabled NTP server transmits during one measurement period are based upon its local time. Since this setup does not include any further network connectivity, there was no synchronization of the server's own clock to another party during the measurement. However, synchronization to external servers took place immediately before the measurements and was suppressed deliberately during the measurement periods in order to prevent potential fluctuations of the server's local time. In addition, the configuration regarding NTS took place in an identical way in order to increase comparability of the results.

NTS servers and clients each use local certification chains[5] with 2048-bit RSA keys and with sha256WithRSAEncryption as their signature algorithm. For authentication of the NTP time packets, the HMAC_SHA512 algorithm was used. Furthermore, the NTP poll interval was set to 16 seconds on all NTP clients. The typical duration of a measurement series in the data presented was 48 to 72 hours. Each dataset started at midnight and ended 24 hours later. The data was collected via the implementation itself, which saves the (raw) data as text files. Before a measurement series started, the devices went through several hours of warm-up time; here, the NTP clients became attuned to the same conditions as their servers. Additionally, this enabled temperature control and thereby minimized the quartz fluctuations that occur due to temperature differences. A fully temperature-stable environment was not available at the time of the measurement.

IV. Measurement Results

The measurements listed here observe the essential aspects of the NTS implementation and range from resource requirements and accuracy/stability achieved to non-correctible deviations in the time synchronization.

A. *Computational Cost and Resource Requirements*

The more complex processing chain and the employment of cryptographic functions[6] in NTS increase processing times and therefore the CPU power required. The values listed in Table 1 show the performance of the corresponding implementations for comparison, each differentiated by the different roles as client or server. Since the measurements were taken on relatively weak single-board computers, computation times are significantly higher than they would be in a desktop or server system. However, the relationship between the values can be expected to be preserved on a desktop computer. The processor time required daily was determined based upon the execution

[1] Raspberry Pi 3 Model B Rev 1.2

[2] 100 Mbit bandwidth using LAN9514-Chip

[3] No connection of other peripherals such as mouse, keyboard or monitor

[4] Network Time Protocol (NTP) Daemon v4.2.8p10

[5] In each case, the primary certificate, two intermediate and one root certificate

[6] The basis for this is the OpenSSL library in version 1.1.0

TABLE I. COMPARING THE REQUIRED PROCESSING TIME

Implementation	Role	CPU Time per Day [s]	Average CPU Usage [%]
NTPD	Client	7.18	0.008
NTPD	Server	6.57	0.008
NTP	Client	25.46	0.029
NTP	Server	11.28	0.013
NTP (NTS)	Client	41.33	0.048
NTP (NTS)	Server	22.85	0.026

time of the NTP daemon and the actual processing time of the CPU. To capture the values in Linux, the *top* and *htop* processor observation tools were used[7].

The data demonstrates that an NTS client generates approximately six times the computing load of an NTPD client for processing and transmitting the same number of NTP packets. However, the cryptography employed affects the values only marginally. A detailed analysis performed with the *KCachegrind* profiler tool [20] revealed that an NTP client with activated NTS functionality uses only 3 % of its required CPU time for the actual computation of the MAC, and only 8 % for the NTS service. The majority of the NTS-specific CPU time is consumed by the ASN.1[8] library (asn1c [21]), certificate processing, and debugging and logging functions employed. Furthermore, the direct comparison between NTPD and Ostfalia's NTP implementation indicates that there is potential for optimization. However, due to the *proof of concept* character of the implementation and the unfinished NTS specification, no effort to optimize this implementation has been made yet.

B. *Obtainable Synchronization Accuracy*

This section discusses the synchronization quality that an NTP/NTS client can achieve relative to its time server. Since the NTP protocol must work from the assumption of symmetric network delays, any asymmetry in packet transport times is hard to correct. However, symmetric packet delays can be assumed for the given measurement configuration because conditions are deliberately uniform on all devices, including identical hardware and software, and because unnecessary external devices such as routers in the network communication chain are eliminated wherever possible.

1) Determination of Minimum Packet Delays

To improve the measurability of the efficiency of the NTP/NTS service, assessment of the minimum possible packet round-trip delays is important. These are delays that not even a well-optimized implementation can go below. The necessary measurements were performed by *ping* commands on the corresponding Ethernet interfaces of the devices employed. For the fixed packet size chosen for this context, the payload (embedded in IPv4) matched the size of an NTS-secured NTP packet transported via UDP[9]. The diagram in Fig. 5 shows the

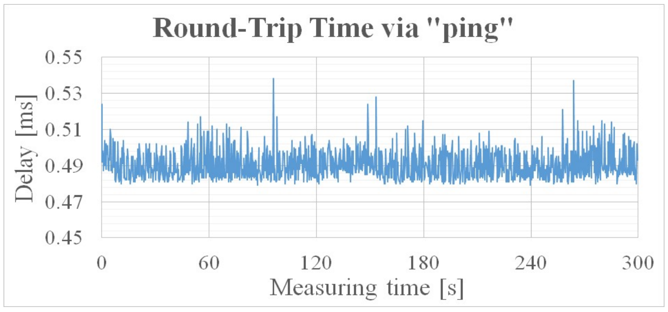


Fig. 5. Minimally achievable round-trip times

development of the measurement that involved firing such a *ping* four times per second. To minimize distortions during these experiments, the data was saved directly into RAM and console outputs were deactivated. The mean value detected of 0.49 ms and the standard deviation of 0.008 ms were consistently found for all measurement objects, even under variation of *ping* intervals during further tests.

2) Comparison of Delay and Time Offset between the Implementations

After ascertaining the minimum delay in IV.B.1, the comparison of the recorded delays can indicate the efficiency and the synchronization accuracy of the NTP services. The comparison of these measurements can be seen in Fig. 6. This diagram shows a 6-hour section of the various implementations. Each program calculates and stores its own delay values. Note that, in this case, the delay not only includes the round-trip time of the packets, but also contains implementation-dependent delays. As can be seen, the delay of NTPD is around 0.5 ms and very close to the minimum delay determined. Therefore, this service appears to be well optimized. By comparison, our own NTP service shows higher delays in the range of 0.9 ms, as well as a higher jitter. Hence, the missing optimizations and the debug functions presumably have a direct effect on the synchronization accuracies. Enabling NTS in this NTP service yields an increased mean and jitter of the network delay (approximately 1.5 µs larger than the NTPD implementation), due to the time needed for the ASN.1 encoding and the cryptographic operations. The latter accumulate completely to the delay, because the calculation of the MAC can only be performed after adding the timestamps T_1 or T_3, respectively, to the NTP packet.

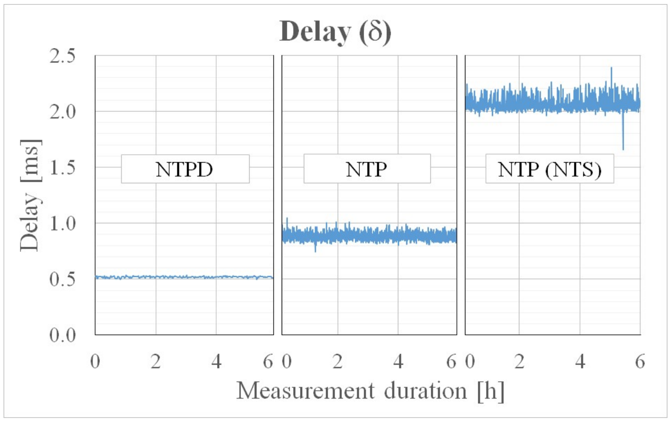


Fig. 6. Delay comparison between the different implementations

[7] Recording and evaluation of values: ELAPSED, TIME and TIME+

[8] Abstract Syntax Notation One

[9] Refers to the NTS server's TimeResponse message. The complete UDP frame thus has a total size of 152 Bytes.

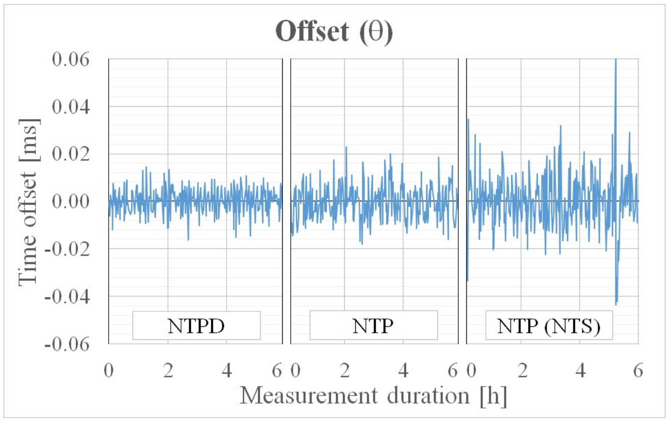


Fig. 7. Time offset comparison between the different implementations

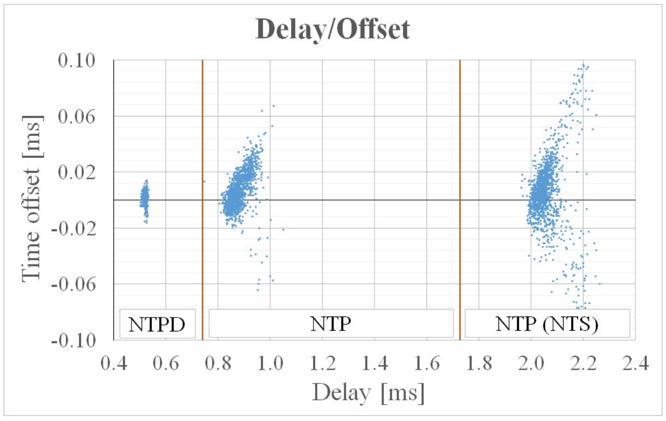


Fig. 8. Delay/offset scattering in comparison

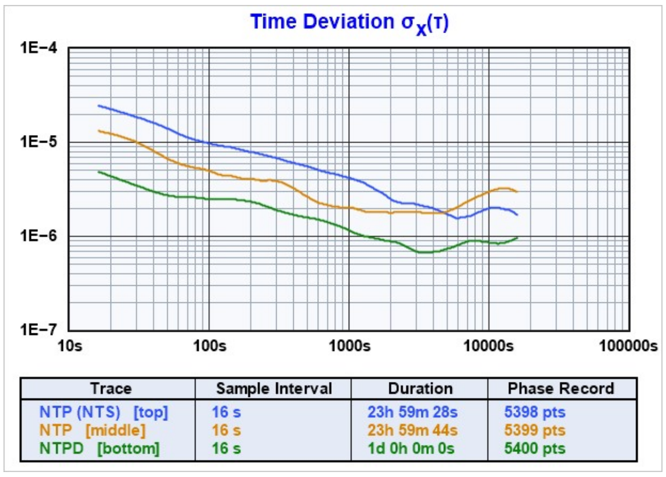
Due to the higher delay and jitter caused by NTS, the jitter of the time offset also increases. The comparison in Fig. 7 shows the difference between the implementations in the same 6-hour measurement section. The standard deviation of the time offset of 4.7 µs in NTPD is smaller than that of 7.9 µs in NTP and much smaller than that of NTP (NTS), 11.6 µs. The combination of delay and offset in a scattering diagram (s. Fig. 8) additionally shows a positive bias of NTP and NTP (NTS) due to the implementation.

3) *Time Stability*

Fig. 9 compares the time deviation (TDEV) of the different measurements. The slope of the blue line (NTP/NTS) is approximately -0.5, which indicates a phase jitter with the character of white phase noise. The slope for NTP (orange) and NTPD (green) indicate that the noise processes are dominated by white phase noise. With increasing averaging time, the jitter (and thus, the instability) decreases. For averaging times larger than 2000 s, other noise processes such as flicker phase noise become dominant. As observed above, the NTPD implementation displays the best stability, followed by the NTP and NTS implementations.

C. *Uncorrectable Time Offset When Using NTS*

Due to NTS's applied cryptography, as well as the differently sized NTS-secured NTP packets, there is an inevitable asymmetry of the packet transmit delays. This is not correctable by NTP itself, and consequently causes a systematic error



Trace	Sample Interval	Duration	Phase Record
NTP (NTS) [top]	16 s	23h 59m 28s	5398 pts
NTP [middle]	16 s	23h 59m 44s	5399 pts
NTPD [bottom]	16 s	1d 0h 0m 0s	5400 pts

Fig. 9. Time deviation (TDEV) of the implementations

in the steering of the client's local clock. This section examines the causes and extent of the major errors in order to identify further areas of research and propose countermeasures.

1) *Asymmetric Packet Sizes*

The complete frame of an NTP network packet that has been sent can be divided into four parts: The Ethernet header (14 bytes), the IPv4 header (20 bytes), the UDP header (8 bytes) and the NTPv4 header (48 bytes). Using unsecured NTP, both the client request and the server response always have the same length (90 bytes). The transfer of this data via Fast-Ethernet (100 Mbit/s) takes 7.2 µs. If the NTP packet contains a *TimeRequest* message from the NTS client, then the size of the complete Ethernet frame with 214 bytes will be 124 bytes larger than that of an unprotected NTPv4 packet. This increases the transmit duration from client to server by an additional 9.92 µs. However, an NTP header (*TimeResponse*) from the server to the client that is 96 bytes larger needs 7.68 µs. Since NTP assumes symmetric packet transmission times, the difference of 2.24 µs leads to a permanent time offset at the client that is half of the packet round-trip time (1.12 µs). The data transfer over a gigabit Ethernet connection can be a temporary countermeasure to reduce the asymmetry of the packet delays. However, the asymmetry remains in this case, and the entire transmission path must have this bandwidth to achieve the desired effect. The alignment of the packet sizes is a better alternative in order to completely compensate for this systematic error. The resulting additional network load increases slightly to 28 bytes[10] per message exchange.

2) *Cryptography and Performance Differences*

Two critical issues related to the absolute synchronization quality are the use of cryptography and the performance differences between the client and the server. To measure this offset, the NTS software was provided with timestamps at the appropriate places in the implementation and decoupled from the NTP implementation. In this mode, the NTS service uses dummy NTP packets to prevent possible variations from being caused by the NTP implementation. Furthermore, in this test, the service works both as a server and as a client in the same program instance to avoid influences caused by communication between

[10] The size difference between TimeRequest and TimeResponse

two devices. This allows the time measurement of internal NTS processes that are included as delays in the packet round-trip time. The diagram in Fig. 10 shows measurement data obtained on a Raspberry Pi. In this measurement, 600 packages were produced and processed in direct succession. It can be shown that the weak hardware of the Raspberry Pis has a considerable influence on the delay. Due to the higher computational effort[11], the server's delay, at a mean value of 468 µs, is even higher than the client's, at 363 µs. The difference in the processing times represents the asymmetry, half of which results in an uncorrectable time offset during the synchronization of the client. Therefore, the measured difference of 105 µs leads to a permanent deviation of 52.5 µs. The frequent peaks are probably due to operating system-specific services that may affect the measurement but have not yet been determined. When comparing these results with the values on a desktop PC (s. Fig. 11), the delay on the client and server is reduced, but moves in the same proportion.

The results show two general problems. The interaction between the client and the server, where great differences exist in their performance, leads to a large asymmetry. Furthermore, correcting dynamic errors that occur as variations in the processing time for securing the time messages is virtually impossible. In addition, the measurements on the Raspberry Pi devices showed other cyclic increases in processing time in addition to the jitter. The origin of this behavior has not yet been clarified and presumably depends on the write access of the recorded measurement data to the storage medium or the internal Linux services.

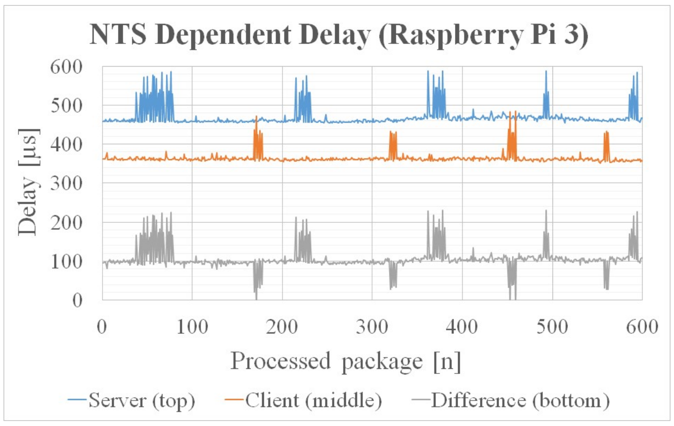


Fig. 10. Additional NTS-related delay on the Raspberry Pi 3

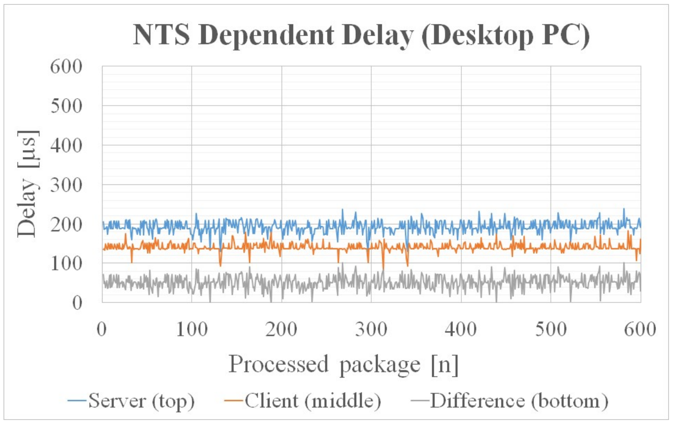


Fig. 11. Additional NTS-related delay on a desktop PC

Note that the measurement of these delays is distorted to some extent. The time measurements (as part of the implementation), the logging of information and the storage of measurement data during this critical processing time have a strong influence on the measurement results. With time recording activated, comparative measurements showed an increase in the delay of about 30-50 % (depending on the hardware). As mentioned in IV.A, the cryptography has shown little impact on processing time, which is currently 8 % of NTS. The optimization of the NTS service offers the possibility of reducing the delay, but cannot prevent it completely. A solution to compensate for these inaccuracies is currently unavailable and requires further research. For this reason, the optimization of the implementation is the best way to minimize the delays.

V. Conclusion and Further Work

The evaluation of the measurement results confirms that it is possible to secure time synchronization traffic with NTS, albeit at the cost of some precision. The relative accuracy that undergoes higher fluctuations due to NTS (and that is hard to correct for via NTP) can be significantly compensated for by optimizing the implementation code. A more problematic issue when securing time synchronization messages is that of performance differences caused by different hardware. Significant differences in processing time during time-critical phases of NTS can lead to a permanent systematic increase in the offset between the clocks of a client and its server. Additional fluctuations during these phases further impede potential solutions such as using correction data.

Since the current draft version of NTS, draft-ietf-ntp-using-nts-for-ntp-11, shows massive design changes from the basis of our implementation [6], it is critically important that new measurements be carried out after the update is fully implemented. Currently, the implementation of said current draft version is still in progress (parallel to the standardization process in the IETF). The changes relate to the communication sequence, the NTP modes supported and increased privacy protection. Furthermore, the current NTS draft does not use ASN.1, which contributes a significant part of the non-correctable asymmetries. Completion of the NTS specification is currently estimated for mid-2018.

Acknowledgment

We are especially grateful to Simon Häußler, Christian Jütte, Thorben Kompa, Silvan König and Thilo Tuschik, who provided Ostfalia's NTPv4 implementation. Special thanks to Dirk Piester who contributed to the analysis of time stability.

References

[1] T. Mizrahi, "Security Requirements of Time Protocols in Packet Switched Networks," RFC 7384, doi 10.17487/rfc7384, October 2014.

[2] D. L. Mills, "Computer network time synchronization: The Network Time Protocol," NY: CRC Press, 2006.

[3] D. L. Mills, "Network Time Protocol (NTP)," RFC 958, doi 10.17487/RFC0958, September 1985.

[4] D. L. Mills, J. Burbank, and W. Kasch, J. Martin, Ed., "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, doi 10.17487/rfc5905, June 2010.

[5] D. Sibold and K. Teichel, "Network Time Security Specification - Protecting Network-based Time Synchronization," in 2016 European Frequency and Time Forum (EFTF), York, UK, 2016.

[6] D. Sibold, S. Roettger, and K. Teichel, "Using the Network Time Security Specification to Secure the Network Time Protocol," Internet Draft, draft-ietf-ntp-using-nts-for-ntp-06, September 2016 (Work in Progress).

[7] K. Teichel, D. Sibold, and S. Roettger, "Network Time Security," Internet Draft, draft-ietf-ntp-network-time-security-15, September 2016 (Work in Progress).

[8] D. Sibold, S. Roettger, K. Teichel, and R. Housley, "Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS)," Internet Draft, draft-ietf-ntp-cms-for-nts-message-06, February 2016 (Work in Progress).

[9] M. Langer, "Implementierung des Network-Time-Security-Protokolls für den Unicast-Betrieb," Master Thesis, Elektrotechnik, Ostfalia University of Applied Science, 2016.

[10] J. Levine, "A review of time and frequency transfer methods," Metrologia, vol. 45, no. 6, pp. S162-S174, Dec 2008.

[11] D. L. Mills, B. Haberman, Ed., "Network Time Protocol Version 4: Autokey Specification," RFC 5906, doi 10.17487/rfc5906, June 2010.

[12] S. Röttger, "Analysis of the NTP Autokey Procedures," Project Thesis, Technische Universität Braunschweig, Institute of Theoretical Information Technology, Braunschweig, 2012.

[13] D. L. Mills, "NTP Security Analysis," [Online] Available (03/06/2018): https://www.eecis.udel.edu/~mills/security.html, May 2012

[14] D. Reilly, H. Stenn, and D. Sibold, "Network Time Protocol Best Current Practices," Internet Draft, draft-ietf-ntp-bcp-06, December 2017 (Work in Progress).

[15] D. F. Franke, D. Sibold, and K. Teichel, "Network Time Security for the Network Time Protocol," Internet Draft, draft-ietf-ntp-using-nts-for-ntp-11, March 2018 (Work in Progress).

[16] Raspberry Pi Foundation, "Raspbian Stretch Lite," [Online] Available (03/06/2018): http://downloads.raspberrypi.org/raspbian_lite/images/raspbian_lite-2017-09-08/, 2017

[17] Network Time Foundation, "Network Time Protocol (NTP) Daemon v4.2.8p10," [Online] Available (03/06/2018): https://www.eecis.udel .edu/~ntp/ntp_spool/ntp4/ntp-4.2, 2017

[18] S. Häußler, C. Jütte, T. Kompa, S. König, and T. Tuschik, "Entwicklung einer NTPv4-Implementation zur Unterstützung von NTS," Team Project, Ostfalia University of Applied Science, 2016.

[19] T. Mizrahi and D. Mayer, "Network Time Protocol Version 4 (NTPv4) Extension Fields," RFC 7822, doi 10.17487/RFC7822, March 2016.

[20] J. Weidendorfer, "KCachegrind v0.7.4," [Online] Available (03/06/2018): https://kcachegrind.github.io/html/Download.html, 2013

[21] L. Walkin, "The ASN.1 Compiler (asn1c v0.9.27)," [Online] Available (03/06/2018): https://github.com/vlm/asn1c/releases/tag/v0.9.27, 2014