Performance Comparison Between Network Time Security Protocol Drafts

Improvements and Accuracy of the Latest NTS Draft

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, " Performance Comparison Between Network Time Security Protocol Drafts - Improvements and Accuracy of the Latest NTS Draft," 2019 Joint Conference of the IEEE International Frequency Control Symposium and European Frequency and Time Forum (EFTF/IFCS), Orlando, FL, USA, 2019, pp. 1-7.
doi: 10.1109/FCS.2019.8856019

Available at:
https://doi.org/10.1109/FCS.2019.8856019

Performance Comparison Between Network Time Security Protocol Drafts

Improvements and Accuracy of the Latest NTS Draft

Martin Langer, Rainer Bermbach
Ostfalia University of Applied Sciences
Wolfenbüttel, Germany
mart.langer@ostfalia.de, r.bermbach@ostfalia.de

Kristof Teichel, Dieter Sibold
Physikalisch-Technische Bundesanstalt
Braunschweig, Germany
kristof.teichel@ptb.de, dieter.sibold@ptb.de

***Abstract*—This paper compares the time synchronization performance of the standard Network Time Protocol (NTP) versus secured NTP using the Network Time Security (NTS) protocol and describes the improvements of the current draft-ietf-ntp-using-nts-for-ntp-17 (NTS-17) compared to draft-ietf-ntp-using-nts-for-ntp-06 (NTS-06) already investigated in [1]. The measurements are based on the implementations of these NTS drafts by the Ostfalia University of Applied Science, both using an NTS-ready NTP version likewise implemented by the Ostfalia, and show the effects of the NTS security mechanisms on time synchronization accuracy.**

Keywords—Network Time Security (NTS) protocol; Network Time Protocol (NTP); security; authentication; synchronization accuracy

I. Introduction

The dissemination of time information, often done through packet-based time protocols, such as the widely used Network Time Protocol (NTP) [2], is of paramount importance for the correct functioning and interoperability of many computer applications.

The accuracy achievable with this protocol in the lower milliseconds is sufficient for most applications. It stands to reason to protect the time information against any manipulation though to date, NTP time data has usually been disseminated completely unsecured which enables chances for adversaries to maliciously modify those data. NTP itself provides some security mechanisms, but an in-depth analysis revealed several deficiencies and exposed them either insecure or unsuitable for practical use [3]. This led to the development of the Network Time Security (NTS) protocol, which is designed to solve the known security issues of NTP without significantly reducing the accuracy and stability provided by the time synchronization process using standard NTP. Now, the specification of NTS is in a pre-final state and the RFC is expected in 2019.

This paper, which is based on previous work [1], examines the performance of the protocol in its version 06 from 03/2016 [4] and compares the results to its version 17 (02/2019) [5]. Recently, version 18 was published with minor changes that do not affecting the results presented in this elaboration. Furthermore, we compare the results with unsecured standard NTP and determine the real time deviation between the time server and client. Both NTS drafts follow completely different protocol designs and were implemented by the Ostfalia University of Applied Sciences in cooperation with the Physikalisch-Technische Bundesanstalt (PTB). One expects distinct improvements of the new version based on the findings of the examination in [1].

Chapter II introduces the reader to the technical aspects of NTP and both NTS versions. Chapter III describes the measurement setup, while the results of the measurements are presented and discussed in Chapter IV.

II. Preliminaries

This section gives an overview of the protocols considered in the measurements and describes the technical processes, their strengths and weaknesses. The comparison of the NTS protocols shows the improvements in NTS-17 against the obsolete NTS-06 version.

A. *The Network Time Protocol (NTP)*

Time synchronization of computer systems and other appliances typically uses packet-based time dissemination protocols. The most widely used protocol for those applications is the Network Time Protocol (NTP). David L. Mills presented NTP back in 1985 and the RFC 958 of the Internet Engineering Taskforce (IETF) describes it thoroughly. The IETF standardized the revised version 4 (NTPv4) in 2010 in RFC 5905 which is the currently relevant version.

NTP works in packet-switched networks und uses the connectionless UDP protocol to send and receive packets containing the time messages. Typically, NTP deploys a hierarchical architecture, as shown in Fig. 1. On the top level, called stratum 0, the server disseminates the time information to clients of the underlying stratum 1. In turn, clients working also as servers, communicate with clients further down and so on. With increasing stratum numbers, the achievable precision typically decreases. Usually NTP servers on stratum 0 receive time information from an atomic clock or a suitable Global Navigation Satellite System (GNSS) receiver.

NTP provides several modi operandi. Thus, in symmetric mode, servers on the same stratum communicate with each other whereas in broadcast mode a server sends out time

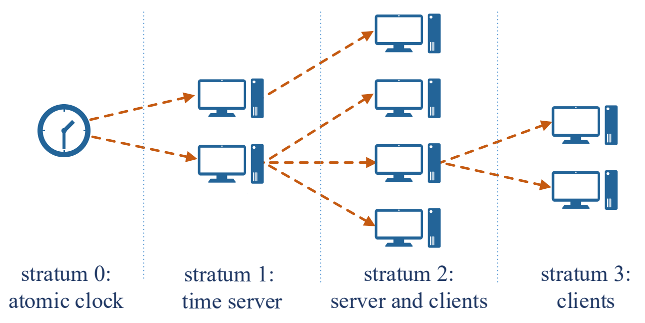

Fig. 1. NTP time distribution architecture

messages continuously with the clients only listening (one-way synchronization). The most widely used mode is the so-called unicast mode that comprises the modes 3 (client) and 4 (server). In this two-way synchronization, the client sends time request messages to a server on a higher stratum level, which in turn answers with the respective response packet, as is shown in Fig. 2.

When sending a request, the client saves and stores a timestamp T_1 in the time request message, T_1 representing the point in time when it sends out the packet. On receiving the request, the server registers the timestamp T_2. It analyses the request and builds the response packet wherein it stores T_2 and the timestamp T_3 that is the sending time of the response message. The packet then arrives at the client again. The client finally registers the timestamp T4, characterizing the receive time at the client. Thus, four timestamps are available, T_1 and T_4 taken from the client's local clock as well as T_2 and T_3 gained from the server's clock.

The client now calculates the delay δ and the time offset θ with those timestamps using the equations (1) and (2).

$$\delta = (T_4 - T_1) - (T_3 - T_2) \quad (1)$$

$$\theta = ((T_2 - T_1) - (T_4 - T_3)) / 2 \quad (2)$$

The delay δ characterizes the time the packet travelled in the network, i.e. the so-called round-trip time (RTT). On the other hand, the offset θ describes the difference in time between the server time and the client's local clock. With the time offset θ NTP adapts the local clock using special algorithms. NTP assumes a symmetrical round-trip time, i.e. the travelling time from client to server being identical to the way back. Any asymmetries in communication lead to uncorrectable time offset errors.

To this day, plain NTP transfers the time data completely unsecured. As mentioned above, it stands to reason to protect the time information against any manipulation, at least in security critical applications. RFC 7384 [6] presents a comprehensive list of known threats to time synchronization protocols.

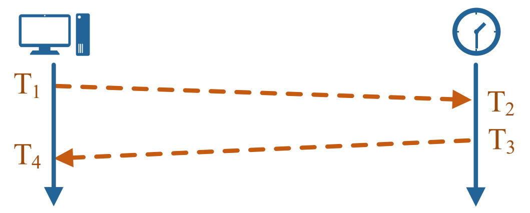

Fig. 2. Acquisition of timestamps via NTP unicast communication

In principle, NTP knows two different security measures to protect the time messages. The pre-shared key scheme [2] and the Autokey protocol [7]; neither of them meets state-of-the-art security demands. The pre-shared key approach does not provide any means to exchange the key for an association between a client and a server. It therefore requires exchanging the key for each association by external means, which reduces its scalability and thus its applicability especially in larger networks, which is the typical use case for NTP unicast mode. The Autokey protocol provides the desired scalability. However, due to systemic vulnerabilities it does not provide appropriate protection for the NTP packets [3]. External security means such as cryptographically based tunnel protocols like MACsec or IPsec provide the desired protection of the NTP traffic. Nevertheless, these means can affect timestamping accuracy that results in a decreased time synchronization performance [6].

B. The Network Time Security Protocol (NTS)

NTS is a new approach to provide cryptography-based protection to time synchronization protocols, especially to NTP. The main goals of NTS are to enable NTP clients to cryptographically identify their NTP servers, to ensure authenticity and integrity for exchanged time packets, to provide good scalability and to ensure that the time synchronization performance is impacted as little as possible.

1) Obsolete Protocol Design (up to NTS-06)

Up to draft version 06, NTS used its own custom designed handshake protocol to authenticate the server and to exchange keys and certificates. To this end it defined different message types that were embedded in the NTP extension fields which were piggy-backed onto NTP packets (see Fig. 3). The access messages protect the client against amplification attacks. The association messages authenticate the server and negotiate the necessary crypto parameters. Finally, the cookie messages supply the client with a single cookie. Among others things, it contains the key material, which the server provides to the client allowing the server to operate in a stateless fashion. However, thorough analysis, e.g. in [1], identified uncorrectable time offsets because of asymmetric packet sizes and performance disadvantages caused by the usage of ASN.1 and CMS in particular. In addition, problems like IP fragmentation and privacy issues were suspected, some of which were confirmed by our implementation of this draft version.

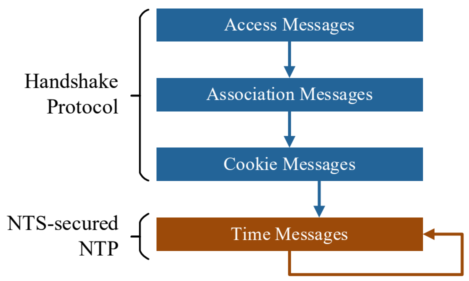

Fig. 3. Obsolete NTS protocol sequence up to NTS-06

The main point of critique, however, was that a custom handshake protocol was being used at all when established solutions such as TLS [8] were available. Formal analysis and verification of the custom protocol [9] was not deemed sufficient to refute this criticism. It also turned out that a key aspect of this issue was that the custom protocol necessarily had to reverse-engineer and copy security features of established protocols. Examples for TLS features that NTS attempted to emulate included mitigation of amplification attacks on handshake exchanges, as well as secure version negotiations and derivation of security guarantees from exchanged certificate chains. Overall, the switch towards a TLS-based handshake was decided and a complete overhaul of the NTS protocol design was performed.

2) *Current Protocol Design (NTS-17)*

The overall structure of the protocol has not changed in the overhaul: As seen in Fig. 4, NTS still has two separate phases, one for handshake operations (performed once) and another for the actual secured time synchronization exchanges (performed repeatedly). NTS now also aims to protect the privacy of the client. It shall not leak any information which enables an adversary to track a client if it crosses networks. Another aspect that has changed under the new design is that the key material is kept fresh by exchanges in the second phase, eliminating any urgent need for re-initialization via repetition of the first phase after a set amount of time.

a) Handshake Protocol

The latest NTS version employs TLS1.2/1.3 [8] for key establishment and exchanges required parameters and algorithms via TLS Records (Application Data Protocol). This solves the IP fragmentation problem that the old design had and allows for optional separation of key server and time server. At the end of the handshake protocol, the server transmits several cookies to the client. These cookies contain the cryptographic state information and enable the server to re-establish the state of an association upon receipt of a time request message. The client is advised to use a cookie only once. It will receive a new cookie with each time reply message from the server.

b) Time Messages

Exchanged time synchronization messages are secured by NTP extension fields as in the old design, but the new design eliminates the need for CMS or ASN.1, which results in improving processing time. For this purpose, four extension fields (EF) have been specified. First: NTS Unique-Identifier extension, which contains a 32-octet random value that serves as nonce and protects against replay attacks. Second: NTS Cookie extension. This EF contains the cryptographic state information. Note that the stateless server dynamically generates cookies upon each time request of the client. This behavior fulfills the privacy requirement added to the NTS specification. Note that the cookies are opaque to the client. Third: NTS Cookie Placeholder extension. The client will send this EF if it wishes to receive a new cookie. Its usage guarantees that the size of the time request equals the size of the time response message. Fourth: NTS Authenticator and Encrypted Extensions extension. This EF contains the actual security tag. It is computed over the NTP header itself and any preceding EF. The new design utilizes AEAD schemes for the calculation of the security tag. Usage of the AEAD scheme also allows for parts of the information in NTP extension fields to only ever be transmitted encrypted. This capability is exploited in NTS itself, where the dynamically generated cookies are encrypted before transmitted to the client. However, the capability to transfer information in secret could also be adapted to interact with any other designs using NTP extension fields.

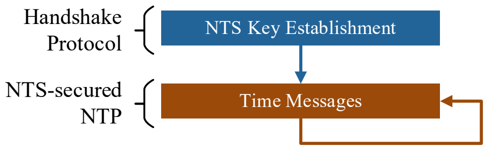

Fig. 4. Current NTS protocol sequence (NTS-17)

As mentioned above, the time request and the time response messages now always have the same size, which avoids asymmetries thus enhancing time synchronization performance and possibilities for amplification attacks employing messages from the time synchronization phase. Amplification attacks employing messages from the handshake phase are defended against via the relevant features of TLS.

With all the above being said, the main benefit of the overhaul is arguably the use of established security protocols for handshake operations, the security of which has already been accepted as proven.

It should be added that where there were only a couple of implementations of the old design even attempted, the new design has seen significant improvement in this area. The most recent interoperability tests at the IFTF meeting 104 in Prague on April 2019 involved five different implementations, four of which successfully interoperated in both possible pairings of client and server (for details, see Section 8 of [5]). This, combined with the general maturity of the document, suggests that the current draft version of NTS will likely achieve the status of a standards track document with only minor changes.

III. Measurement Setup And Configuration

In order to investigate the effects of the design differences between the NTS versions described in chapter II.B, a specific measurement setup for these protocols was necessary. The measurements focused on an unsecured NTP with native NTS support, an NTS-06-secured NTP (NTS-06) and an NTS-17-secured NTP (NTS-17). All these implementations are available as open source software on Gitlab [10] [11]. This chapter describes the measurement concept and the necessary configurations to compare these implementations. The procedure is based on the measurement setup described in [1].

A. *General Measurement Setup*

In general, the comparison of the implementations requires fixed conditions for the measurement series to make the results comparable with each other. One of these is the simultaneous execution of several measurements per measurement series, whereby each measurement considers one of the three implementations (NTP/NTS-06/NTS-17). The duration of a measurement was at least 72 hours in which multiple data sets were generated. Each data set started at 00:00 o'clock and ended 24 hours later. For each measurement, the client-server message exchange took place every 16 seconds, generating 5,400 measurement points per day. The server-side instances had been synchronized with an external time server before the start of a measurement, but no synchronization took place during an active recording. Thus, possible fluctuations could not affect the time of the client. However, the time synchronicity of a server is not critical for the measurement and primarily serves a better assignment of different raw data and data sets. Before the measurement started, the devices used were set into operation for several hours in order to control the temperature and thus minimize possible distortions of the measured values caused by the wander of the crystal and subsequent large adjustments of the frequency offset.

1) Hardware Configuration

The hardware was based on single board computers (Raspberry Pi 3B), each in a closed casing. Since all devices used the same hardware models, hardware-related measurement deviations were minimal and negligible. A measuring unit consisted of a pair of these devices, one acting as a client and the other one as a server. Both also worked with a defined software configuration, which is described below. This device pair always communicated via a direct Fast Ethernet connection (see Fig. 5) to prevent network fluctuations. Thus, no connection to an external network was available, so disturbances or load fluctuations were excluded. Since all Raspberry Pi devices operated in the headless mode, the measurement was controlled via an SSH access using a wireless LAN connection. During active measurements, the devices were also stored in a temperature-protected environment to prevent a time drift due to short-term temperature changes.

2) Software Configuration

The Raspberry Pi devices operated with a Linux system that had been adapted to the platform (*Raspbian Stretch Lite*). This also included the OpenSSL v1.1.1 crypto library, which allowed the use of the new TLS 1.3 protocol for the new handshake mechanism in NTS-17. Because of the mirroring of the operating system including the software on it, all devices used the same software base. Thus, deviations in the operating system configurations or system services were excluded. The only exception to that affected the NTP/NTS configuration that had been used on the respective devices. Depending on the measurement, either unsecured NTP, NTS-06 or NTS-17 ran on the devices in the role of a client or a server.

The implementations used had also been compiled as a release build under the same conditions to prevent measurement deviations due to runtime differences between implementations. Furthermore, these did not generate any console or log output other than the required measurement values, as these could have a negative effect on the results. To minimize side effects caused by the operating system, the measurement data was also stored in the RAM drive, since write access to the read-only memory (SD card) is significantly slower.

The measurements with the unsecured NTP version provide reference values to determine the additional performance requirements by NTS-06 or NTS-17. The NTS-06 and NTS-17 measurements used the NTP implementation, which embedded the respective NTS version and used it to secure the time messages. NTS-06 servers and clients each used local certificate chains with 2048-bit RSA keys and *sha256WithRSAEncryption* as the signature algorithm. An *HMAC_SHA512* algorithm was applied to protect the NTP packets by generating a MAC. In NTS-17, however, elliptical curves are used in the certificates, which are now transferred using TLS. For integrity protection, client and server used the AES-based AEAD algorithm *AEAD_AES_SIV_CMAC_256*. However, due to the hardware used, no AES acceleration is available on the devices.

B. Common View Measurement Setup

In order to determine the absolute time deviation between client and server, a measurement setup extension was necessary to perform a Common View (CV) measurement. Usually the GPS time is used as reference to determine the time difference between client and server. However, this could not be applied to the measurement setup because the conditions for a sufficiently accurate GPS time could not be met. Instead, a third measurement system (so-called *Reference System*) was used, which observed the local clock of client and server and achieved a sufficiently high measurement accuracy even without GPS time.

The measuring setup consisted of two Raspberry Pi devices (client/server), the Reference System (RS) and a PC for recording the measured values (see Fig. 6). The RS was composed of a microcontroller board (stm32nucleo) equipped with an external crystal oscillator to improve the accuracy of the measurements. The Reference System was directly connected to the Raspberry Pi devices via the GPIO pins and to

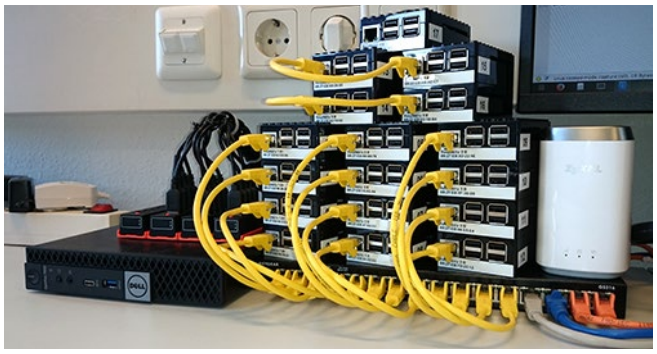
Fig. 5. Hardware setup of the general measurement system

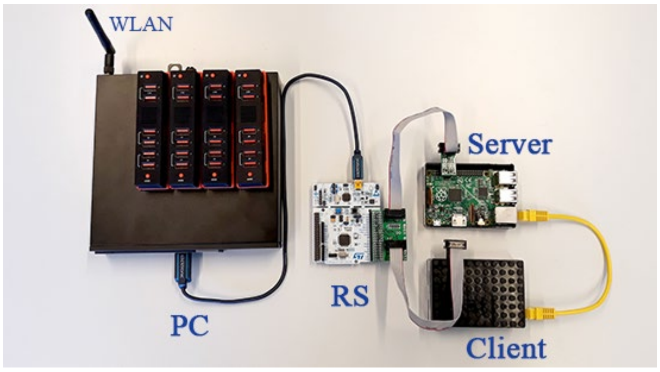

Fig. 6. Hardware setup of the Common View (CV) measurement system

the PC via a serial connection. Another link existed between the Raspberry Pi devices and the PC over wireless LAN in the form of TCP communication.

The PC started the measurement by execution of a Python script that in turn started a measurement every second after a short connection check. During a measurement, the RS initiated a simultaneous time measurement on client and server, which took place independently of the NTP/NTS measurement running on it. The Reference System controlled the process by setting and resetting the GPIO pins. The Raspberry Pi devices reacted to the signals using a specially developed kernel module. Furthermore, the kernel module enabled intelligent interrupt control and fast and efficient timekeeping. Collected time information was then transferred from both Raspberry Pi devices to the PC via the TCP connection. The RS also transferred data to the PC via the serial interface after the measurement had been completed. The data set contained information about the individual phases of the time measurement of both devices, allowing the exact duration of the time measurement to be determined. Using the three data sets per measurement, the time difference between client and server can be determined with an accuracy of about 0.7 µs. For time synchronization via NTP, these values are sufficiently accurate.

IV. Measurement Results

This chapter discusses the results and compares the respective performance differences of the NTS versions. The measurements examine the resource requirements as well as the synchronization quality (offset/jitter), and the systematic deviation of the client clock from its server's.

A. Computational Cost

The first measurement considers the performance requirements of the implementations, since cryptographic operations can be problematic especially on weak hardware. Therefore, the processor load of NTP, NTS-06 and NTS-17 respectively was measured, each in the role of client and server. The determination of the required processor time is based on the execution time of the tested implementation and the actual processing time of the processor. The processor monitoring tools *ps* and *htop*, available on the Linux system, provided the necessary information for this.

The results in Table 1 show that even with NTS protection, the performance requirement is low and requires less than 0.1% of the CPU power. The difference in performance between client and server is due to several processing routines and has no direct effect on synchronization accuracy. Compared to the unsecured NTP version, the obsolete NTS-06 increases the processor load by about 35 to 40%. The bracketed values in the table represent the absolute differences compared to the values of the unsecured NTP. In comparison, the current NTS-17 version requires only 25% more computing power to secure the time data than the unsecured connection and is therefore approximately 35% more efficient than NTS-06. The difference is primarily due to the elimination of CMS and the ASN.1 coding of the data in NTS-17.

TABLE I. Comparision of the Computational Cost

Implementation	Role	CPU Time / Day [s]	Avg. CPU Usage [%]
NTP	Server	29.2	0.03
NTP	Client	45.1	0.05
NTS-06	Server	40.8 (+11.6)*	0.05
NTS-06	Client	61.0 (+15.9)*	0.07
NTS-17	Server	36.4 (+7.2)*	0.04
NTS-17	Client	55.6 (+10.5)*	0.06

*Performance difference to unsecured NTP

B. Determined Synchronism by NTP

Standard NTP calculates delay and offset between client and server using the four timestamps T_1 to T_4, as described in II.A. From this, the NTP client derives its synchronicity to the connected time server.

1) Delay (Round-Trip Time)

The delay determined by NTP normally represents the packet round-trip time in the network. However, time elapses between setting the timestamps in the NTP packet and actually sending the message, which distorts this value and makes it slightly higher. The securing of the packets by NTS leads to an additional distortion, because the required processing time for securing the time message is completely included in the delay. This results in higher delay values, which in turn can affect the algorithms used by NTP (e.g. clock selection).

As the results in Fig 7 show, the unsecured NTP reaches a delay value of about 1.0 ms. Further optimization of our NTP implementation could probably reduce that to a magnitude of approximately 0.5 ms, which is the typical delay of NTPD [12] already presented in [1]. In comparison, the delay of NTS-06 is more than one millisecond higher. The time responsible for securing the message consists primarily of cryptographic operations and the ASN.1 coding of the data. In comparison, the delay in NTS-17 with an additional 0.5 ms is only one half of that of NTS-06. This is primarily due to the elimination of ASN.1 encoding and the resulting faster embedding of integrity information in the NTP packet. Hardware that offers AES acceleration could further reduce this delay, but this would not be possible for the old NTS-06 version, because of its hash-based protection.

2) Time Offset

The measurement results in Fig. 8 show the determined offset of the received and unfiltered NTP packets. As expected, the client synchronizes correctly with all measurements. With a standard deviation of 10.7 µs, NTS-17 is comparable to the

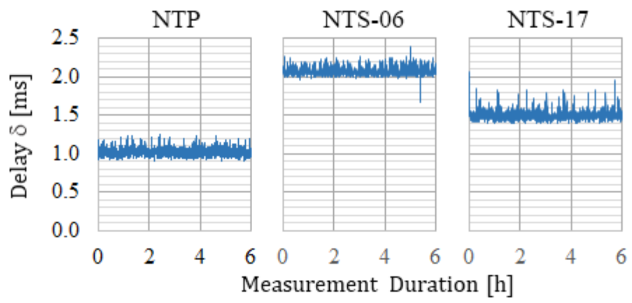

Fig. 7. Delay comparison between unsecured NTP, NTS-06 and NTS-17

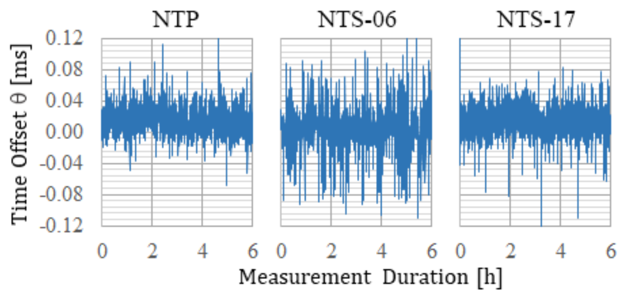

Fig. 8. Comparison of the determined time offset

unsecured NTP connection with 11.1 µs. However, NTS-06 shows a standard deviation of 25.8 µs, which is more than twice the jitter of NTS-17. The actual synchronization accuracy is slightly better, since NTP uses the packet with the lowest delay from the last eight measurements to adjust the time, since it usually has the lowest offset.

3) Delay/Offset Correlation

The combination of the measured values presented in IV.B.1 and IV.B.2 results in a delay/offset correlation diagram. This visualizes the dispersion range of the time information and the synchronization accuracy determined by the NTP client. Ideal measurement values would result in a point or a small circular area. The delay fluctuations and the resulting jitter in the offset, results in a V-shaped spread open to the right, as can be seen in NTS-06 in Fig. 9. The opening angle of the spread reflects the offset jitter. The diagram shows that a smaller delay is always accompanied by a smaller offset. In addition, it can be seen that the unsecured NTP has a strong bias in the positive direction. This is caused by asymmetries in the RTT, which are provoked here by the NTP implementation and can potentially be corrected by optimizations. The bias of NTP is also reflected in the NTS implementations. NTS-17 is almost identical to NTP and only shifted to the right by the delay caused by the crypto time. The shift of NTS-06 is more pronounced and shows a significantly higher offset jitter.

C. Critical Processing Time

In this step, the critical processing time (or critical delay) between timestamping of the NTP message and transmitting it was measured. The focus lies on the part caused by NTS when securing a message. In this measurement, NTP was decoupled from NTS and replaced by an NTP dummy. The dummy resolves the implementation-specific dependencies and prevents measurement value distortions that could be caused by a normal NTP implementation.

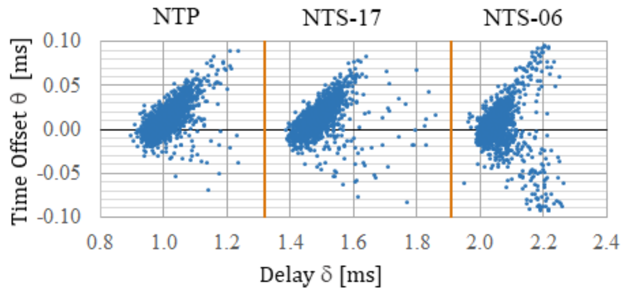

Fig. 9. Delay/Offset correlation diagram of NTP, NTS-06 and NTS-17

Using the NTP dummy one measures the sole NTS processing time, approximately. The results in Fig. 10 show clear differences between NTS-06 and NTS-17, whereby the critical delay caused by NTS-17 is much smaller. In NTP, the delay difference of the processing times between client and server causes an asymmetry in the calculated RTT by NTP. The delay differences between client and server are partly due to the measurement itself, since various debug functions were active during this measurement. The actual values are therefore slightly lower than the presented results.

The critical processing time depends on the crypto algorithms used, the implementation and the available computing power. While with the Raspberry Pi the NTS-17 server causes a delay of approximately 100 µs, with a typical desktop system this drops below 10 µs (depending on computer performance). Thus, a high-performance server and a low-performance client could cause a larger difference and therefore a larger asymmetry. This inevitably leads to a systematic offset error between client and server (see IV.D).

D. Systematic Time Offset

Since the NTP protocol assumes symmetrical packet runtimes due to its design, time shifts caused by asymmetrical runtimes can only be partially corrected. However, due to the homogeneous measurement setup in these test series with identical hardware and software, almost symmetrical packet runtimes can be assumed.

The measurement of the client's deviation from the server was realized using the Common View method described in chapter III.B. In addition to the NTP/NTS implementations mentioned above, measurements with the unsecured reference implementation NTPD were also performed in order to compare the accuracy of the own NTP implementation. The results showed a systematic offset of 20 ± 10 µs for NTPD, which remained constant during the measurement period. The own unsecured NTP implementation caused a higher systematic offset of 40 ± 10 µs due to the lack of optimization of the NTP software.

As mentioned above, the process of securing the NTP messages by NTS generates an asymmetry due to performance differences of the hardware or different runtimes in the implementation. Fig. 11 shows the sole systematic offset of the two NTS versions, which are generated in addition to the NTP deviations. The value of NTP was averaged out to obtain the pure NTS offset. The current NTS-17 version thus generates an additional offset of 20 µs, while the old NTS-6 version produced a 60 µs offset.

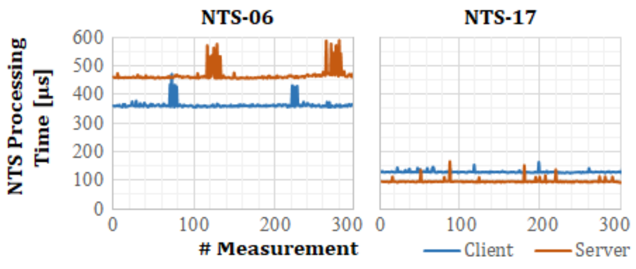

Fig. 10. Introduced delay for securing NTP messages by NTS

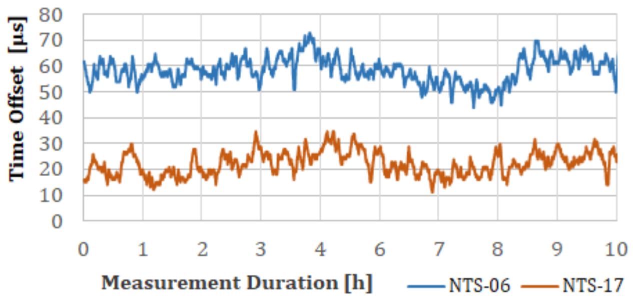

Fig. 11. Systematic time offset caused by NTS

V. Conclusion And Further Work

We have shown that the latest NTS draft provides much better results in performance and accuracy. Moreover, the asymmetry caused by the critical delay is reduced and the systematic error is negligible in typical NTP applications. Additionally, because the current NTS version has fixed all known problems of NTS-06 like IP fragmentation or privacy issues etc., the Network Time Security protocol seems ready for practical application. Therefore, it stands to reason that NTS should be applied in every NTP service.

The focus of our further work lies on the reduction or even the compensation of the critical processing time of NTS. As even perfect cryptography cannot avoid delay attacks, mitigation of their effects as well as the securing of other NTP modes (such as broadcast) are also being pursued.

References

[1] M. Langer, K. Teichel, D. Sibold and R. Bermbach, "Time Synchronization Performance Using the Network Time Security Protocol," in 2018 European Frequency and Time Forum (EFTF), doi 10.1109/EFTF.2018.8409017, Turin, Italy, 2018.

[2] D. L. Mills, et al., "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, doi 10.17487/rfc5905, June 2010.

[3] S. Röttger, "Analysis of the NTP Autokey Procedures," Project Thesis, Technische Universität Braunschweig, Institute of Theoretical Computer Science, Braunschweig, 2012.

[4] 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, Sep 2016.

[5] D. Franke, D. Sibold, K. Teichel, M. Dansarie, R. Sundblad, "Network Time Security for the Network Time Protocol," Internet Draft, draft-ietf-ntp-using-nts-for-ntp-17, Feb. 2019.

[6] T. Mizrahi, "Security Requirements of Time Protocols in Packet Switched Networks," Internet Requests for Comments, IETF Secretariat, RFC 7384, Okt. 2014, https://tools.ietf.org/html/rfc7384.

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

[8] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, 10.17487/rfc8446, Aug. 2018.

[9] K. Teichel, D. Sibold and S. Milius, "First Results of a Formal Analysis of the Network Time Security Specification," 218-245. 10.1007/978-3-319-27152-1_12, Dec 2015.

[10] M. Langer, "Network Time Security v0.9.0," [Online] Available (04/26/2019): https://gitlab.com/MLanger/nts/tags/v0.9.0.

[11] M. Langer, "Network Time Protocol v0.6.0," [Online] Available (04/26/2019): https://gitlab.com/MLanger/ntp/tags/v0.6.0.

[12] Network Time Foundation, "NTPD," [Online] Available (04/26/2019): https://github.com/ntp-project/ntp4