Experimental Analysis of the Performance and Scalability of Network Time Security for the Network Time Protocol

Griffin Leclerc[1,2] and Radim Bartos[1]
[1]*Department of Computer Science*
[2]*InterOperability Laboratory*
University of New Hampshire
Durham, NH 03824, USA
{gleclerc@iol, rbartos@cs}.unh.edu

***Abstract*—Network Time Security (NTS) standardizes mechanisms that allow clients to authenticate timing information received via Network Time Protocol (NTP). NTS includes a new key establishment protocol, NTS-KE, and extension fields for NTPv4 which, when utilized together, allow clients to authenticate messages from time servers. Utilizing an open source implementation of each, we determine the existence and severity of any performance or scalability impact introduced by NTS when compared to NTP. We found that conducting individual authenticated time transfer takes approximately 116% longer when utilizing NTS over NTP. Additionally, we found that NTS-KE can only support approximately 2000 requests per second before a substantial and consistent increase in turnaround time is observed.**

I. Introduction

The Network Time Protocol (NTP) [1] is currently used to provide accurate timing information to hundreds of millions of networked devices around the world. Despite its staggering usage, NTP is one of the last remaining, widely-used protocols with no security components. Currently, there is no native method for NTP clients to authenticate time information obtained from NTP servers. This presents an opportunity for the malicious operation of fraudulent NTP sources to distribute incorrect timing information to client machines.

Network Time Security (NTS, RFC8915) [2] was established to augment NTP with a variety of useful security features, including authentication, confidentiality, replay prevention, and non-amplification. While these features address some of the vulnerabilities of NTP, time server operators are concerned that the introduction of the added security features in NTS may negatively impact the performance of time distribution systems and accuracy of the timing information, especially in cases when a large number of clients are connected.

This study determines the existence and severity of any performance or scalability impact introduced by NTS. An NTS implementation was augmented to isolate NTS specific functionality from the NTP processes and benchmarked, so that NTS could be observed independently from standard NTP protocol execution. The isolated operations include the TLS key exchange, server side cookie creation, and message signing. This isolation allowed NTP and NTS to be observed and quantified independently, showing a clear comparison between the time required to perform unauthenticated time transfer and authenticated time transfer. Measurements were gathered under loaded and unloaded circumstances to determine the scalability and performance of NTS respectively. While other works have previously gathered measurements of NTS under no load [3], no investigation was conducted to determine the scalability of NTS.

This research has been supported in part by a gift from the Internet Society to assist in the establishment of a Community Network Time Security Lab.

II. Preliminaries

Network Time Security, published in September 2020 by the IETF, introduced two specific sub-protocols that are critical to the functionality of securing NTP. The first is the NTS Key Establishment Protocol (NTS-KE), where NTS clients contact an NTS key establishment server to obtain a secure cookie containing encryption parameters along with the address of an NTS aware NTP server. The client then transmits this cookie in conjunction with a request for NTP time transfer to that NTP server via NTS Extension Fields for NTPv4. Both of these mechanisms are showcased in Figure 1.

NTS-KE begins when a client initiates a TLS handshake with a key establishment server where a shared encryption method is established. Using this encrypted channel, the client issues a Key Establishment (KE) request message to the server, containing Authenticated Encryption Associated with Data (AEAD) and NTS Next Protocol negotiation. The KE server responds with the negotiation status and a new cookie for NTPv4 if negotiations were successful. Both endpoints issue a TLS *close_notify* immediately after sending their respective messages whether or not negotiations were successful.

In addition to a secure cookie, the client also receives the address of an NTS aware NTP server from the KE server, which it contacts immediately. The client transmits an NTP client packet with the NTP extension fields containing the secure cookie and an authentication tag. Using the implementation

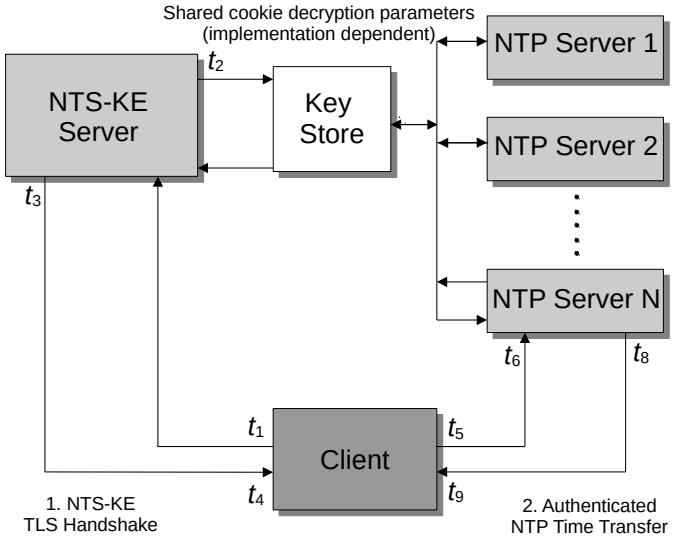

Fig. 1. Overview of NTS-KE and NTS Extension Fields for NTPv4 mechanisms

specific shared medium, illustrated as a key store in Figure 1, the NTP server uses the cookie to both reconstruct the shared keys contained within and subsequently transmit authenticated time information. In addition, the NTP server creates a new encrypted cookie transmitted to the client simultaneously which, according to RFC 8915, should be utilized for future time transfer requests.

Both NTS-KE and NTP servers utilize the same shared medium when they create secure cookies for clients. This interaction allows all members of this secure time domain to interact with any clients regardless of which machine created the cookie. Additionally, by storing relevant security information in the cookie, NTS-KE and NTP servers remain stateless, a highly desirable property due to NTP's widespread usage and subsequent large number of clients.

It is important to note that NTPv4 can still be conducted in parallel with NTS mechanisms by simply not contacting the NTS-KE server and excluding or ignoring the NTP extensions. Clients can still request unauthenticated NTP by ignoring NTS-KE, contacting only the NTS aware NTP server without the required extension fields. Non-NTS Aware NTP servers that cannot or choose not to parse the NTS extensions simply ignore them and respond using a standard, unauthenticated NTPv4 header. While this is undesirable from a security standpoint, proprietary local error correction methods can correct for the lack of security, such as maintaining a sliding window of previously received authenticated time to resist jumps. More importantly, backwards compatibility allows the NTP server to utilize unaltered NTPv4 headers on both the client and server side to perform authenticated NTP.

Thanks to a generous effort by Cloudflare [4], an open source implementation of NTS-KE, the NTPv4 Extension Fields and key storage mechanisms are readily available. The implementations are available as a collection titled CFNTS [5], written in Rust and completed in mid 2020. CFNTS is a prime candidate for experimentation due to its status as a reference implementation. Unlike other open source NTS implementations [6], [7], CFNTS does not conduct continuous time transfer using a daemon process, allowing a single key exchange and time transfer to occur in a user process. This property is useful as it allows a single machine to host multiple NTS clients in parallel which greatly simplifies experimentation. In addition, this simplified architecture results in direct access to the core protocols, meaning almost all of CFNTS' run time is devoted to explicit time transfer rather than other internal mechanisms such as modifying the local system time.

CFNTS utilizes Memcached [8], an open source memory object caching system, to store cookie decryption information generated during NTS-KE. Memcached provides a simple in memory key-value store accessible across processes. CFNTS initializes KE server operation by writing randomly generated 16-byte values into the Memcached store to be used as encryption keys during the cookie creation phase of NTS-KE. These keys are later recalled by the NTP server to decrypt cookies received by clients during an authenticated NTP exchange. It is important to note that the Memcached allows other networked machines access to the key-value store in addition to local processes. However, our experiments host the NTS-KE and NTPv4 servers on the same machine, so there is no network transmission required to read or write keys to Memcached.

III. Methodology

After careful study of CFNTS and its time transfer procedures, key areas of operation in both the client and the server were identified and an experimental benchmarking suite was developed to quantify operational time of the NTS-KE and authenticated NTPv4. Timestamps throughout protocol operation were gathered using Rust's standard library time::Instant::now() function and are represented in Figure 2.

Beginning with client side measurements, the total time required for a client to perform a TLS handshake with the KE server, negotiate NTS-KE encryption parameters and receive an NTS-KE secure cookie is represented by d_{CKE} (1). Similarly, the time required for a client to conduct authenticated time transfer using the cookie is d_{CNTS} (2).

$$d_{\text{CKE}} = t_4 - t_1 \tag{1}$$

$$d_{\text{CNTS}} = t_{10} - t_5 \tag{2}$$

All server side measurements begin when a request is received from a client, and continue measuring until the response is fully formulated and ready for transmission back to a client. For example, the server receives a request for an NTS cookie from the client at t_2. The server proceeds to conduct a TLS handshake, contact the shared key store, and create a cookie for the client. t_3 is taken at this time; when the cookie is generated and ready to be transmitted to the client. This measurement constitutes d_{SKE} (3). d_{SNTS} (4) functions similarly, in that it represents all operations the server needs to take in order to decrypt the incoming NTS cookie, authenticate the NTPv4 header, and create the frame for transmission. Note

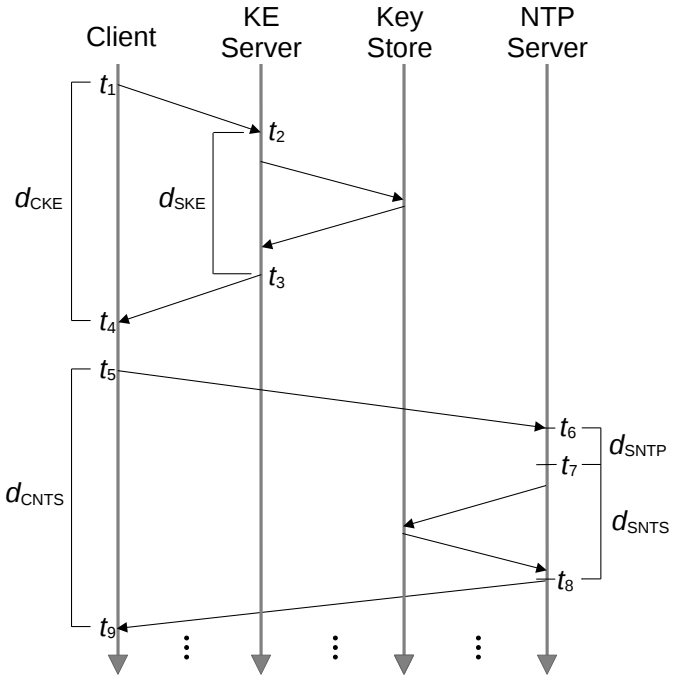

Fig. 2. NTS-KE and authenticated NTP timing diagram and performance measures

d_{SNTS} does not include the creation of a new cookie due to the single time transfer nature of CFNTS. d_{SNTP} (5) represents the creation of the NTPv4 header at the server. It involves no network or security operations. d_{SNTP} and d_{SNTS} represent the total time the server requires to perform authenticated time transfer.

$$d_{\text{SKE}} = t_3 - t_2 \tag{3}$$

$$d_{\text{SNTS}} = t_8 - t_7 \tag{4}$$

$$d_{\text{SNTP}} = t_7 - t_6 \tag{5}$$

Finally, using these measurements, an observer can determine the time consumed during unauthenticated NTP time transfer by:

$$d_{\text{CNTP}} = d_{\text{CNTS}} - d_{\text{SNTS}} = (t_{10} - t_5) - (t_8 - t_7) \tag{6}$$

This study determines both the performance and scalability of NTS, so two separate experiments were developed to complete each of these objectives individually. The first experiment utilized one client connected to one server in an unloaded environment to measure performance of NTS-KE and authenticated NTPv4. The second suite implemented a many-to-one pattern where clients transmitted an increasing number of requests per second to a single server in order to simulate a large network and measure scalability of NTS-KE and NTPv4.

A single dedicated server containing a 4-core 3.3 GHz Intel i5-2500k and 24 GB of memory hosted both the NTS-KE and NTP server for all experiments. This allowed for the shared cookie decryption parameters to be stored in a local Memcached instance and therefore required no network access

TABLE I
OBSERVED AND CALCULATED PERFORMANCE MEASURES

d_{CKE}	Total time required for a client to receive the initial encrypted cookie
d_{CNTP}	Calculated approximation of unauthenticated time transfer
d_{CNTS}	Total time required for a client to conduct authenticated time transfer
d_{SKE}	Time required for a KE server to create an encrypted cookie
d_{SNTP}	Time required for an NTP server to create an NTPv4 header
d_{SNTS}	Time required for an NTP server to process an encrypted cookie and authenticate an NTPv4 message

to decrypt client cookie information. Before any measurements were recorded, ten NTS-KE mechanism and NTP transfers were conducted to allow connections a warm up period and decrease overall variance of data.

IV. Performance Evaluation

To measure performance of NTS mechanisms, an unloaded experimental setup was constructed where a single client issued one request per second to both the KE server and NTP server until 1000 measurements were collected. Figure 3 is a visual representation of the unloaded experiment configuration.

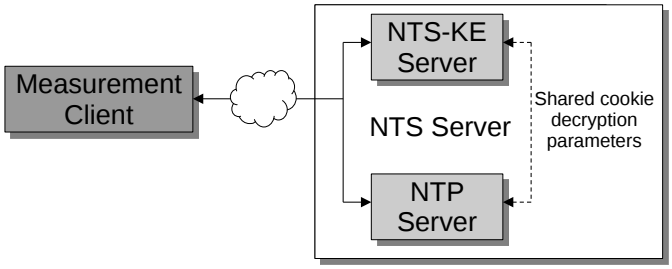

Fig. 3. Experiment setup for performance evaluation

In this scenario, the measurement client is a virtual machine with eight cores and 16 GB of memory. This client transmitted one NTP-KE request and one NTP request per second for 1000 seconds to determine the unloaded operational times of CFNTS. These requests were benchmarked in accordance with Figure 2 which resulted in the following cumulative distribution functions, with minimum and maximum observed values explicitly labeled.

By taking the mean of the explicitly defined values in the cumulative distribution functions, we can conclude the average time consumed by each interval, defined in Table II.

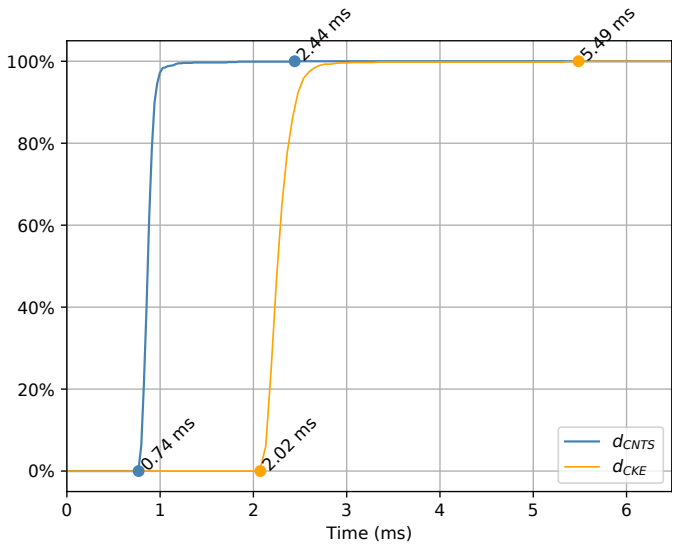

Fig. 4. Cumulative Distribution Functions of client side measurements d_{CKE} and d_{CNTS}

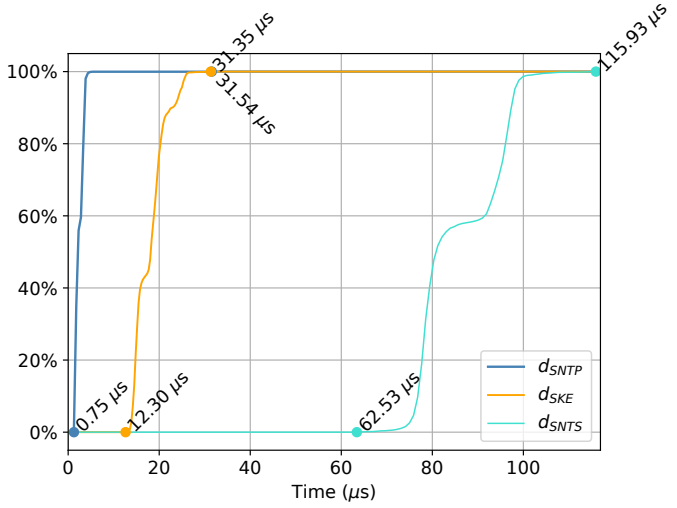

Fig. 5. Cumulative Distribution Functions of server side measurements d_{SKE}, d_{SNTP}, and d_{SNTS}

Additionally, average d_{CNTP}, the time required to perform unauthenticated NTP, is 0.79 ms. When compared to an average d_{CNTS} of 0.87 ms, there is a 9.73% increase in time required to conduct only authenticated time transfer. Accounting for the time needed to conduct NTS-KE, 2.26 ms, there is a 117% increase. These results are consistent with the measurements of the cost of adding security reported in the context of the Precision Time Protocol [9].

d_{CKE} takes approximately three times longer to complete than d_{CNTS}, which is due almost entirely to network latency. To perform d_{CKE}, the server and the client must conduct a TLS handshake, negotiate application specific NTS-KE encryption parameters, and exchange the encrypted cookie. Because CFNTS does not conduct continuous time transfer, no TCP or TLS parameters are stored. Consequently, three network round trips are required for each NTS-KE exchange, regardless of previous requests. Conversely, NTPv4 only requires one network round trip to transfer time, so it is approximately three times faster than NTS-KE from the perspective of the client.

TABLE II
AVERAGE UNLOADED PERFORMANCE MEASURES

d_{CKE}	d_{CNTS}	d_{SNTP}	d_{SKE}	d_{SNTS}
2.26 ms	0.87 ms	2.03 μs	18.13 μs	80.80 μs

Perhaps the most surprising measurement comparison is between d_{SNTS} and d_{SKE}. Intuitively, these server side measurements should take a similar amount of time to complete, as both should need to read from and write to the Memcached store. However, before CFNTS begins serving NTS-KE requests, it loads all cookie encryption keys into a local data structure, removing the need for repeated read operations from Memcached.

V. SCALABILITY EVALUATION

Our scalability study determines how many requests per second the KE server and the NTP server could process. Due to limitations in hardware available to the research team, a single client could reliably send approximately 1500 requests per second (rps) to the KE server and the NTP server. Because of this, multiple client machines were utilized to issue large numbers of requests. At the beginning of the experiment a single client began by issuing 100 rps for 20 seconds to gather steady state measurements. Once the measurement window had elapsed, all machines in the experiment were power-cycled and the client would increase the request load to the server, issuing 200 rps for 20 seconds. This incremental increase in the number of requests per second would continue until the client was issuing 500 rps. At the beginning of the subsequent measurement window, the client machine would start an auxiliary, or AUX client, whose sole purpose is to continually issue 500 rps. The original client would then issue 100 requests leading to a global load of 600 per second; 500 from the AUX client and 100 from the measurement client. When the measurement client again reached 500 rps, resulting in a global load of 1000 rps, another AUX client would be added, and the next iteration of the experiment would continue with a global load of 1100 rps. This methodology is illustrated in Figure 6.

The measurement client is identical to the client utilized during performance evaluation, a virtual machine with eight cores and 16 GB of memory. The server is also identical, containing a 4-core 3.3 GHz Intel i5-2500k and 24 GB of memory. Each AUX client is an identical virtual machine with four cores and eight GB of memory. Fifteen total AUX clients were available, resulting in an upper limit of 8000 total rps.

The measurements gathered are the same as previously mentioned and illustrated in Figure 2. However, it is important to note that our measurement environment only records

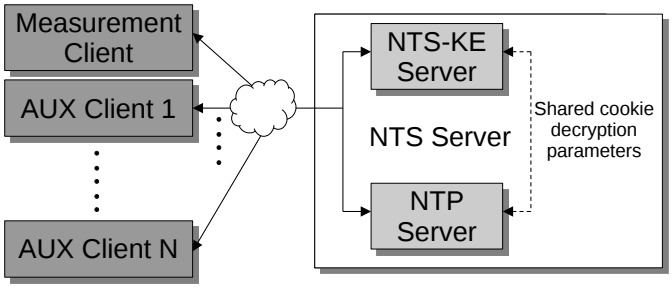

Fig. 6. Experiment setup for scalability evaluation

measurements from the measurement client and not from any AUX clients. This is true for both client and server side measurements. To compensate for this, we ensure that the measurement client issues requests only after all active AUX clients have begun issuing requests. This prevents the measurement client from gathering measurements when no external load is present and provides measurements similar to a worst-case scenario. Additionally, to find the limits of both the NTS-KE and NTP server, three NTPv4 exchanges took place for each NTS-KE exchange. This ratio aligns with the results of the scalability study, where NTS-KE took approximately three times longer to complete than authenticated NTPv4. This lies in contrast with the recommendations of RFC 8915 Section 1.3 [2], which states that only one time transfer should take place per secure cookie to ensure unlinkability. However, that recommendation relies on another attribute of NTS: that an additional, new cookie is provided to the client in conjunction with NTPv4 time transfer, which CFNTS does not do. Using CFNTS, following this recommendation would require an explicit invocation of NTS-KE for each individual time transfer. This is not how the protocol is intended to be executed. To compensate, three time transfers were executed using the same cookie.

As illustrated in Figure 7, the measurement client experiences increased NTS-KE turnaround times beginning at 2000 rps. The median measured response time increases by an average of 50.09 ms, while the 95th percentile measurement increases by 744.64 ms on average. This increase in response time continues throughout the remainder of our observations. In addition, at 2300 rps, the client begins to experience errors after requesting encrypted cookies. The primary error types were connection time out and resource temporarily unavailable. Connection timeout occurs when the KE server was unable to provide a cookie within ten seconds per the protocol timer, and resource temporarily unavailable implies that the KE request could not be processed. The ratios of errors encountered throughout experimentation is present in Figure 8.

Conversely, load was observed to have no effect on d_{CNTS}. Visible in Figure 9, d_{CNTS} remained around the expected value obtained during the performance study of approximately 0.7 ms, with few outliers. Our research team speculates that, because new encrypted cookies can only be obtained through explicit execution of NTS-KE, clients are not issuing a sufficient number of requests to the NTP server in relation to the KE server. Empirical evidence should be gathered and this claim should be investigated in the future.

d_{SKE}, d_{SNTP}, and d_{SNTS}, illustrated in Figures 10, 11, and 12, were relatively uniform during the scalability experiments. This is to be expected, as these local server side operations do not depend on a network and execute in relatively constant time, so naturally they are unaffected by external network load.

VI. Conclusions

Given that NTS includes additional security features, including a TLS handshake and implementation specific distribution of encryption parameters, it was expected that it would be more resource intensive than NTP. The objective of this study was to quantify both the performance and scalability of NTS-KE and authenticated NTPv4 when compared to base NTP. Utilizing CFNTS, we found that authenticated NTP takes approximately 117% longer to conduct time transfer than base NTP from the client's perspective. Regarding the server, creating a secure cookie during NTS-KE adds an overhead of 18.13 μs, and repeated server operations saw a significant increase on average from 2.03 μs to 80.8 μs. Scalability of NTS was also quantified, where we found that the NTS-KE server could only process 2000 requests per second before a substantial and consistent increase in response time and unprocessed requests were observed. All other measurements were unaffected by scaling.

Future opportunities for research include further increasing the number of requests per second issued from clients, or increasing the number of time transfers per cookie to identify scalability limits of authenticated NTPv4. In addition, other open source NTS implementations can be augmented to observe the performance of sustained time transfer using NTS.

Acknowledgments

The authors would like to thank Sawyer Bergeron for outstanding technical support related to virtualization, dynamic CPU utilization and Rust build procedures.

References

[1] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, Jun. 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5905
[2] D. F. Franke, D. Sibold, K. Teichel, M. Dansarie, and R. Sundblad, "Network Time Security for the Network Time Protocol," RFC 8915, Sep. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8915
[3] 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)*, 2018, pp. 138–144.
[4] Cloudflare. [Online]. Available: https://www.cloudflare.com
[5] CFNTS. GitHub Repository. [Online]. Available: https://github.com/cloudflare/cfnts
[6] Chrony. [Online]. Available: https://chrony.tuxfamily.org
[7] NTPSec. [Online]. Available: https:/ntpsec.org/
[8] Memcached. [Online]. Available: https:/memcached.org/
[9] D. Maftei, R. Bartos, B. Noseworthy, and T. Carlin, "Implementing proposed IEEE 1588 integrated security mechanism," in *2018 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2018, pp. 1–6.

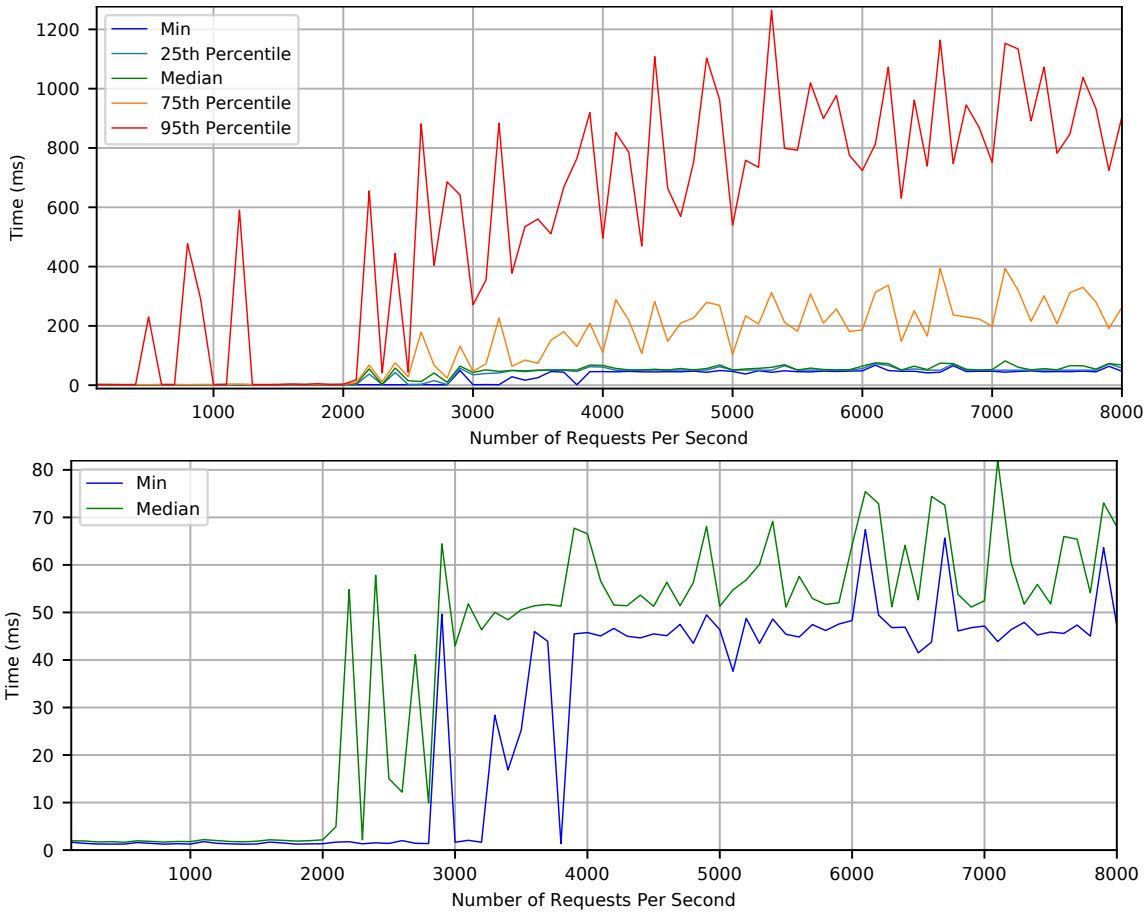

Fig. 7. d_{CKE} under load

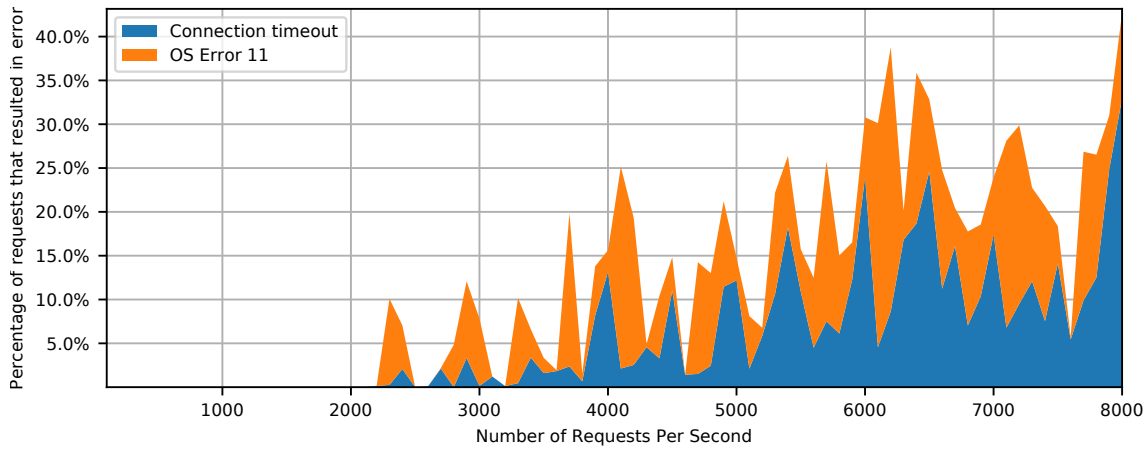

Fig. 8. Percentage of requests that resulted in an error encountered during collection of d_{CKE}.

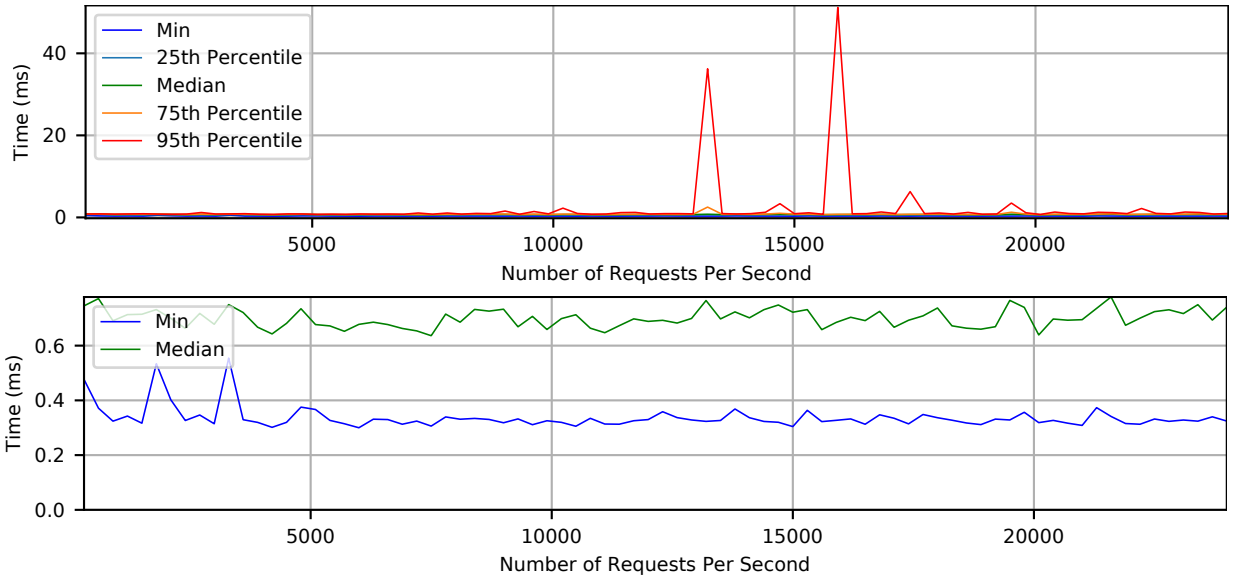

Fig. 9. d_{CNTS} under load

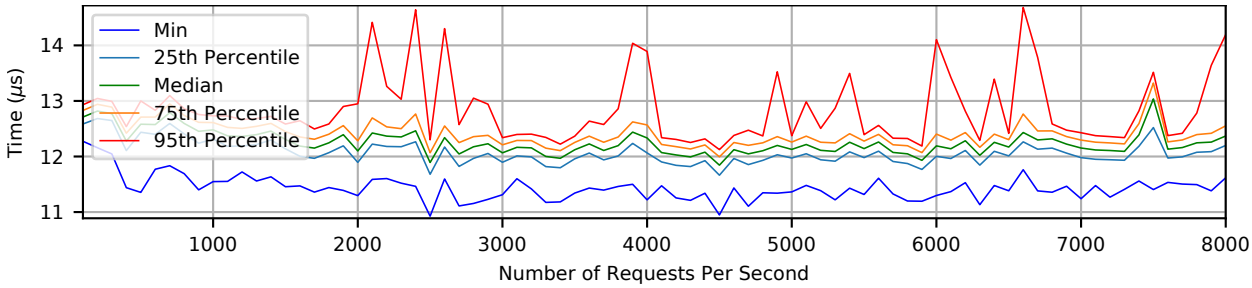

Fig. 10. d_{SKE} under load

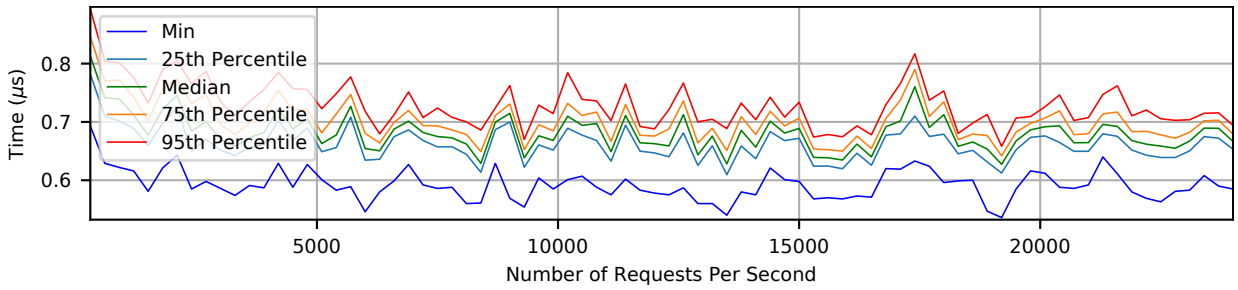

Fig. 11. d_{SNTP} under load

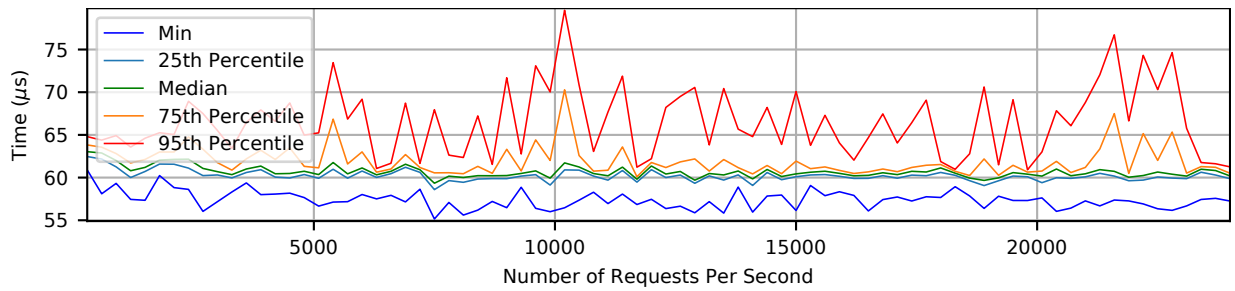

Fig. 12. d_{SNTS} under load