Are NTP clients always right?

Evaluating NTP clients under normal and attack scenarios

Technical Report SIDN Labs 2025-10-16

Update: 2025-10-30

Shreyas Konjerla
TU Delft
Delft, The Netherlands
s.konjerla@student.tudelft.nl

Giovane C. M. Moura
SIDN Labs and TU Delft
Arnhem and Delft, The Netherlands
giovane.moura@tudelft.nl

Georgios Smaragdakis
TU Delft
Delft, The Netherlands
G.Smaragdakis@tudelft.nl

Tamme Dittrich
Tweede Golf
Nijmegen, The Netherlands
tamme@tweedegolf.com

Abstract

The Network Time Protocol (NTP) is the Internet's default time synchronization protocol. While the protocol itself and attacks against NTP servers have been well-studied, there is a lack of understanding on how NTP clients behave under normal operation and under attacks. Considering their importance – they are the ones who decide to update client's clock or not – this paper focus on evaluating the behavior of eight 8 NTP clients, including the default client's used by the most popular OSes (Windows, Ubuntu and MacOS). We find that the the default client's of all OSes are vulnerable to time shift attacks, which is particularly concerning given they are used by billions of devices daily. In total, we find nine issues and bugs with clients and notify vendors. We find a large variation in client behavior, and worked with Ubuntu operators to measure traffic growth on their switch to `Chrony` from `timesyncd`.

1 Introduction

Clock synchronization is critical for the correct operations of billions of digital devices and services. On the Internet, TLS [10], DNS caches [29], RPKI [4], Kerberos [31], and DNSSEC signatures [3] are examples of applications that depend on clock synchronization to prove cryptographic freshness [8, 16, 25, 40].

CPU clocks drift over time, as their internal oscillators gradually diverge from a reference clock due to temperature fluctuations, manufacturing tolerances, and power variations [20]. To prevent the accumulation of time difference caused by this drift, clients use time synchronization protocols to query *periodically* time servers for updated time information, ensuring alignment with a reference time.

The Network Time Protocol (NTP) [25] is the Internet's default protocol for clock synchronization. It is designed to mitigate the effects of network jitter between client and server while providing accuracy of tenths of milliseconds[1].

NTP servers synchronize their clocks with out-of-band high precision references, such as atomic clocks, radio signals, and global navigation satellite system (GNSS, as GPS and Galileo), or other NTP servers. A client, in turn, synchronize its local clock by (i) querying NTP servers – which provide timestamps that allow them compute the offset between its own clock and the NTP server's clock – and (ii) then adjusting its system clock by the calculated offset value, provided that the NTP servers are deemed trustworthy.

Protocols: NTPv4 [25] is the latest version of the protocol. It specifies not only the on-the-wire protocol used, but also *client-side* algorithms (server selection, timestamps filtering, clock discipline), multiple operational modes (client, server, symmetric active, symmetric passive, broadcast, control, and private) and symmetric cryptographic authentication.

Many applications and devices (such as IoT) do not need all features supported by NTPv4. The Simple Network Time Protocol (SNTP) [24] was introduced for such cases. It is a subset of NTPv4, which uses the *same* protocol format, hence compatible with a NTP server. The Network Time Security (NTS)[11] was later proposed as an extension to NTPv4, and it uses Transport Layer Security (TLS) [39] to provide authentication and integrity to NTP.

Clients: there is a large variety of NTP and SNTP clients. Currently, all three major OSes (Windows, macOS, and Ubuntu Linux) rely (Sept. 2025) on SNTP-based clients. Given client's role in clock synchronization – they make the ultimate decision whether to trust time provided by NTP servers and to updated the system's clock – it is essential to evaluate

[1] Applications requiring micro-second accuracy can use the Precision Time Protocol (PTP) [13], which is mostly used in layer 2 and not on the Internet.

Client (version)	User base	OS	Release
macOS (15.4.1)	2.2B* [1]	macOS	Sept 2024
W32Time [22]	1.4B [21]	Windows	April 2025
timesyncd (255.4) [2]	-	Ubuntu	Sept 2024
NTPSec (1.2.2) [33]	-	Ubuntu	Nov 2024
NTPD-RS (1.1.2) [32]	-	Ubuntu	Jan 2025
ntpd (4.2.8p18) [30]	-	Ubuntu	June 2023
OpenNTPd (1:6.2p3-4.2) [12]	-	Ubuntu	Oct 2022
Chrony (4.5) [36]	-	Ubuntu	Oct 2024

Table 1: Evaluated NTP/SNTP clients. Clients highlighted are their OSes' defaults and SNTP-based. (*2.2B devices running MacOS and iOS combined).

how modern clients behave and if they are vulnerable to attacks.

Currently, there is no comprehensive evaluation of the behavior of modern and popular S/NTP clients under both normal operations and attacks. A previous work [15] carried out man-in-the-middle (MITM) attacks against 4 NTP clients, but it did not include the clients default to the three major OSes (used by billion devices daily), and and neither used attack models that did not require MITM.

To fill this void, we scrutinize the behavior of eight popular NTP and SNTP clients (Table 1) covering macOS, Linux (Ubuntu, a systemd-based [45] OS) and Windows Server (2025.02.13). We evaluate eight clients in total: these OSes' default clients – given they are used by billions of devices daily – and five others, namely: ntpd (NTPv4 reference implementation), NTPSec (ntpd fork with NTS support), Chrony (default on Ubuntu releases from Oct. 2025 that supports NTS) [18]), OpenNTPd (OpenBSD's project NTP implementation), and NTPD-RS, a rust-based client with NTS support.

We configure these clients and observer their behavior under normal operations (§2) and under well-known attacks that aim to change drastically client's clock (time shift attacks) or prevent clients from synchronizing, leading to clock drifts (§3). We analyze the results of these experiments and make the following contributions:

- We demonstrate that all major OSes' default clients are vulnerable to time shift attacks (§3), affecting billions of clients globally. For macOS and timesyncd (Ubuntu's default), we cause a two-year jump in time with 2h and 13min, respectively. Windows took 36h.
- We identify ten issues and bugs with existing software and perform coordinated vulnerability disclosure (CVD) [28] to vendors (§4), and one of the issues has been patch as of the writing of this paper.
- We identify large variation among clients under normal operations (§2), in terms of query volume and server usage.
- We disclose our findings to Ubuntu operators and work with them to estimate traffic increase due to their scheduled switch from timesyncd to Chrony in their Oct. 25 release. We determine a 10× more queries and 25× more traffic volume (§2.4). These findings helped Ubuntu's network operators make informed decisions about their service dimensioning.

2 Client's Baseline Behavior

Next we evaluate NTP/SNTP clients shown in Table 1 under normal operations.

2.1 Expected behavior

SNTP protocol is designed for devices that need lower accuracy than NTP clients. SNTP RFCs does not specify how often a client should query NTP servers, but they do stipulate limits: a client *must not* be sent new NTP queries at intervals shorter than 15 s (§10 in [41]) and should wait at least 1 min between queries (§5 in [24]). Clients are expected to use only one NTP server if configured with multiple ones (§7 in [41]).

NTP clients, in turn, are tailored for systems that require higher precision and behave differently. NTP clients have to wait 16 s between successive queries and to use exponential back-off to increase the interval between queries if NTP servers are stable (§5 in [25]). At starting conditions, however, NTP clients can be configured in burst mode, allowing them to send multiple queries to speed up its first clock update — chrony, for instance, will send 4 queries spaced 2 seconds or less [5]. Moreover, NTP clients are expected to query all NTP servers they are configured (§5 in [25] and §3.2 in [38]).

2.2 Experimental setup

NTP servers: Our experimental setup consists of a single physical computer (macOS) – except for Windows client case (W32Time). In this setup, we configure three Ubuntu VMs (24.04.3 LTS) using Multipass [46] to serve as our NTP servers, which run Chrony. We synchronize these three NTP server with external sources – Apple and Ubuntu NTP servers as reference NTP servers (time.apple.com and ntp.ubuntu.com).

Clients: we configure a VM for each client from Table 1 – Ubuntu VMs using Multipass and macOS VM using UTM [48]. It is important to note that these VMs maintain their own clocks independently of the host operating system's clock.

We then configure these client VMs to use our previously configured NTP servers . Given that both NTP servers and client VMs run on the same physical machine, network delay and jitter are minimal. We capture traffic in all experiments. We run each experiment for over 60h, which we except to be more than enough to capture the clients behaviors.

Windows : For Windows Server client, we run it on AWS EC2 instance (Stockholm) and set up 3 VMs as NTP servers

Client	Total	Avg/h	Queries Per Server S1	S2	S3	Per Server Poll Rate (avg)
Chrony	4,822	80.37	1,643	1,498	1,681	7.02
macOS	309	5.15	102	101	106	11.6
ntpd	739	12.32	246	246	246	9.41
NTPD-RS	15,050	250.8	5,012	5,012	5,026	5.39
OpenNTPd	1,361	22.68	425	473	463	8.85
NTPSec	1,009	16.82	336	336	337	9.29
W32Time	213	3.55	80	77	56	10.1
timesyncd	108	1.77	108	0	0	10.9

Table 2: Queries sent by each client in 60 hours in normal operation. highlighted are SNTP clients.

(just as in our local setup with Chrony), and configured Windows to query these NTP servers.

2.3 Results

We summarize the traffic captured for each client in Table 2. For each client, we show the total number of NTP queries, average number per hour, and total queries per NTP server . Considering that all the clients have the same ideal network conditions (except for Windows), it is striking to find such large variation in number of queries per hour.

2.3.1 SNTP clients: we see in Table 2 that SNTP send the lowest number of queries, confirming to the expected behavior (§2.1). timesyncd is the only one who adhere to SNTP specifications, querying only one NTP server (§2.1) out of the three. W32Time and macOS query all three servers, which is not expected from SNTP clients (§2.1), and therefore are not dependent on a single time source. All SNTP clients have an average polling interval of roughy 10 (~17min).

2.3.2 NTP clients: NTP clients exhibit a large variation, ranging from 12 to 250 queries per hour (Table 2), being ntpd, the reference implementation, the lowest, whereas NTPD-RS the highest. To investigate these differences, we show in Figure 1 the scatter plots of queries sent by clients. On the x-axis, we show the time, while on the y-axis we show the polling rate, which shows how often a client queries a time server. A polling rate of $n = 10$ means the client sends a query every $2^{10} = 1024$ s – a value known as *polling interval*, denoted by T, and is computed as $T = 2^n$.

Reference implementation: we see in Figure 1 that ntpd polls more aggressively in the beginning and then reducing the rate over time exponentially, until it reaches a polling rate of 16 (18 h polling interval). This value within the standaridzed bound – maximum polling rate being 17 (§2.1).

Troubleshooting NTPD-RS: NTPD-RS, in turn, never has a polling interval larger than 256 s (rate=8). It's documentation stipulates that its poll interval is [16,1024] s (max rate 10) [32] and that it has no burst mode, but in our experiment we do

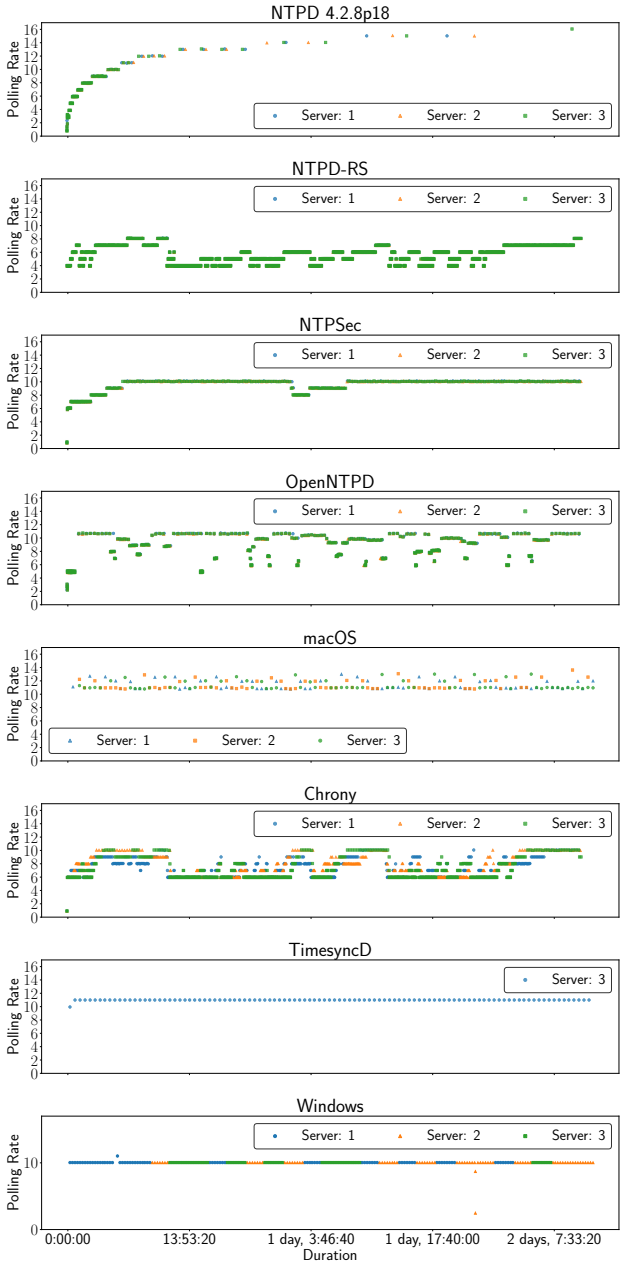


Figure 1: NTP queries per client. Polling interval computed from consecutive queries to the same server.

not see this behavior. The NTPD-RS documentation states that it has "back-offs that kick in case in case of failures" [37] when polling NTP servers. To rule out packet loss and retrial, we analyze the traces and found no packet losses or delayed responses (RTT 90%ile 2.6ms).

We then contacted the NTPD-RS developers, who confirmed our findings as as intended behavior, where the if

	AMS VM		*SYD* VM	
	queries/h	bytes/h	queries/h	bytes/h
`timesyncd`	1.83	201.3	1.83	201.3
`Chrony`	21.24	5213.10	20.38	5043.11
Ratio	11.6	25.1	11.1	25.05

Table 3: Ubuntu outgoing traffic compared (60h)

`NTPD-RS` prioritize accuracy over network load for nearby servers. We verify those claims in Appendix A.

2.3.3 Server usage: we have seen in Table 2 that all clients, except for `timesyncd`, use the three available NTP servers they have been set up. While all NTP clients send queries to all three NTP servers simultaneously, we found that `macOS` and `W32Time` do not: `macOS` sends to two at each time, and `W32Time` to one at a time, as can see in Figure 1. We include in Appendix A figures showing shorter time intervals where this behavior is clear.

Windows versions: besides Windows server, we cover in Appendix B how Windows home behave, and how Windows Pro/Server Client behaves if part of a active directory domain-controller (it uses many-cast mode [24]).

2.4 Ubuntu switch to `Chrony`

Ubuntu is scheduled to replace its SNTP client (`timesyncd`) with `Chrony` in Oct. 2025 [18], with NTS enabled by default on all images. As shown in Table 2, `Chrony` issues significantly more queries than `timesyncd`. Since Ubuntu operates its own time service network, we disclosed our findings to Ubuntu developers in Sept. 2025 and raised concerns about a potential traffic surge they will observe due to this change.

To estimate a more precise traffic increase, we reproduced Ubuntu's future setup and captured 60 h of traffic from two VMs in different data centers (AMS and SYD). We then repeated the experiment using `timesyncd` under identical conditions and compare traffic from both experiments.

Table 3 summarizes the results. We see an 11× increase in query counts and a 25× increase in traffic volume per hour, when switching to `Chrony` (Table 3) – NTS queries and responses are larger due the use of extension fields [11].

We shared these results helped Ubuntu's operators, which helped them to make informed choices about the required capacity of their time service network. We cover these experiments in detail in Appendix C.

3 Attacking Clients

Next, we evaluate attacks targeting S/NTP clients. Specifically, we demonstrate two classes of misbehavior: (i) time-shift attacks (§3.1) and Kiss-o'-Death (KoD) attacks (§3.2).

3.1 Time shifting attacks

Time shifting attacks consists of an attacker deliberately manipulating time responses to shift a client's system clock. They can disrupt operations and cause issues on a large scale. For instance, in November 2011, the US Navy Naval Observatory's (USNO) NTP servers [34] had an incident where they reported wrong time information (12 years incorrect), resulting in outages in multiple places, including Active Directory servers and and routers [16, 19]. Even though it was not an intentional time shift attack, it *behaved* like one.

Expected client behavior: To prevent such attacks, NTPv4 specifies a "panic mode", in which clients should not update their clocks if the offset is larger than 1000s [25]. Moreover, NTPv4 also states that clients should compare responses from multiples servers before stepping their clock. SNTP clients are not expected to have such protections, which may explain the reported outages on Windows systems in the USNO event, given it uses a SNTP based client.

3.1.1 Threat model: we assume the goal of the attacker is shift a target systems clock. In our threat model, the attack controls malicious NTP servers. We do not use in or out-of-path attack models; rather, we assume that clients are redirected to the the attackers' services. This can be done in multiple ways. First, using the NTP Pool [35], which is a volunteer-based time service provider consisting of 4k NTP servers, and a very popular service [27]. There is a known vulnerability in which an attacker can take over all NTP queries to the NTP pool for entire countries (or parts of) [27]. It requires adding malicious servers to the NTP Pool and evading their monitoring system [14]. In this way, an attacker can server a large population of clients with wrong time information, evading NTP Pool bad servers detection.

A second way to redirect clients is by using DNS – either by carrying out cache poisoning or compromising a DNS resolver. For instance, it can forward all `macOS` default time servers (`time.apple.com`) to IP addresses of NTP servers under the attackers' control. Once the NTP client is served by the attacker NTP servers, the attack can start.

3.1.2 Experiments: For ethical reasons, we refrain from performing these previously demonstrated attacks (NTP Pool or compromising public DNS servers) and start from the point where the clients are already redirected to the attackers' servers, given our goal is to determine if clients are able to fend-off such attacks.

We configure all clients to use a domain name that pointed to three NTP servers under our control. We allow the clients initially to have a *hot-start*, meaning that they retrieve correct time information after booting. After this initial phase, we start the time shift attacks, by responding queries with wrong time information from all three servers We run 6 experiments

	Offset					
Client	**900s**	**1M**	**2M**	**3M**	**1Y**	**2Y**
macOS	✓	✓	✓	✓	✓	✓
W32Time	✓					
timesyncd	✓	✓	✓	✓	✓	✓
NTPSec						
NTPD-RS						
ntpd						
OpenNTPd						
Chrony						

Table 4: Client behavior to time shift attacks. (✓) shows vulnerable clients. (M = month, Y = Year).

for each client, in which we use different offsets (how long we want clocks to be shifted), from 900 s (under the "panic threshold" [25]) to two years, using the same setup from §2. We run the experiments for at least eight hours and and observe the client's handling of the responses and the system's clocks.

3.1.3 Results: Table 4 summarizes the results. We see that *none of the NTP clients* were vulnerable to time shift attacks, regardless the offset used in the attack.

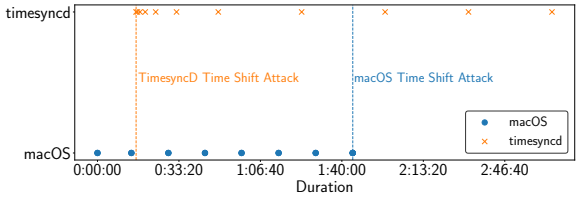
However, we found that *all* SNTP clients are vulnerable. Even though they conform to the SNTP standards, these are the *default* clients in the three major OSes and given their billions of users (Table 1), it makes them very problematic given the impact they may cause.

We managed to timeshift the clocks of macOS and Ubuntu's timesyncd (which is also used in several other Linux distros) by all the offsets we tried. Window's W32Time client, however, could not be time shifted for more than 900s. Windows server documentation states that it has a two-week panic mode value, meaning it accepts ofssets up to 2 weeks but not more than that [22]. Still, it may be vulnerable to time skimming attacks [16], where it can be time shifted by multiple times over longer periods.

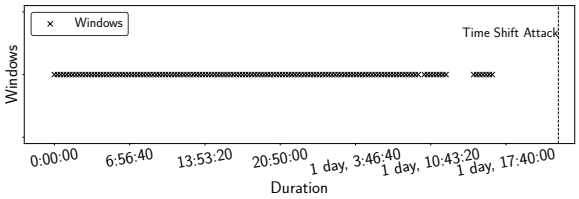
How long does it take? We were surprised to see how fast we could succeed inour attacks: 13 minutes for Ubuntu's timesyncd, and 2 hours for macOS (8 and 13 queries, respectively). We show the time series of queries in Figure 2a.

Windows' W32Time client too longer. We ran an additional experiment using a offset of a week, and it took 136 queries over 36 hours for the attack to succeed (Figure 2b).

CVD: We followed coordinate vulnerable disclosure (CVD) guidelines [28] and disclosed our findings to Apple in March 2025 (issue # OE1100520849262). We even reproduced the attack with DNS hijacks (Appendix D). They concluded that this was not a vulnerability and did not fix it, which therefore keeps their users vulnerable to such attacks. We notified also both timesyncd (systemd) and Debian[7] developers (2025-09-25), considering that Debian is particularly vulnerable given it uses both timesyncd and the NTP Pool. timesyncd developers responded saying that there is "working in progress to add NTS support to timesyncd" [44], which partially fix the issues (a client can be configured to use a bad NTS timekeeper still). We are stil waiting for Debian's response.



(a) timesyncd and macOS (2 years offset)



(b) Windows (1 week offset)

Figure 2: Time series of NTP queries for each client. Dashed lines show when the attacked succeed.

Other findings: we found out that NTPD-RS would crash roughly after 30min into the experiment, when its calculated offset is greater than its panic threshold. We disclosed the issue to the developers and yet to hear back from them.

Takeway: the default time clients of the three major OSes are vulnerable to time shift attacks, making billion of devices. Ubuntu's switch to Chrony in Oct. 2025 will eliminate this issue, whereas Microsoft and Apple still need to fix it.

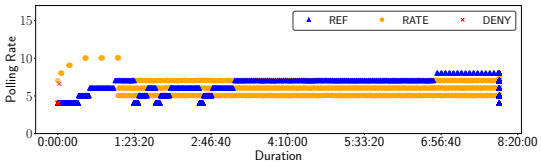
3.2 Kiss-o'-Death (KoD) attacks

The NTP protocol provides KoD packets as a rate-limiting mechanism for servers to tell clients to reduce their query rates. While the conception was well intended, it has been proved to be a dangerous attack vector in multiple attacks: spoofed KoD packets can cause clients to stop synchronize their clocks, which will cause their clocks do drift [16, 38], and can be exploited in DDoS attacks.

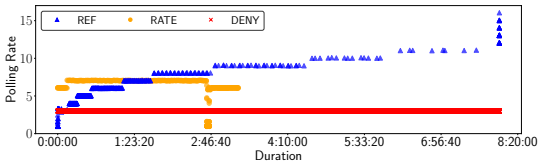
Expected behavior: RFC8633 [38] (§5.4) recommends (but does not mandate) clients to back-off when receiving KoD RATE packets, which signals that clients should reduce their query rate (thus increasing polling interval, but never beyond 13), and states that many clients do not respect them. NTPv4 RFC states that clients should stop using the server after receiving a KoD DENY packet (§7.4).

	RATE	DENY
Chrony	Reduced	Reduced
ntpd	Increased and stopped	Increased
NTPD-RS	Increased (bug fixed)	Increased
NTPSec	Reduced	Reduced
OpenNTPd	Reduced	Reduced

Table 5: Summary of Kiss of Death results.



(a) NTPD-RS



(b) ntpd

Figure 3: Client behavior under KoD attacks.

Threat model : we set out to determine if we could prevent clients from stop synchronization by sending KoD packets, which can lead to clock drifts. We use the same threat model as in §3.1 – except we now instead of sending wrong time information, we send KoD packets after a hot start – where clients receive legitimate responses.

Experiments: We carry out two experiments, using RATE and DENY KoD packets. It is important to note that we did not test SNTP clients since they are not expected to implement KoD handling [24]. .

Results: Table 5 summarizes the results. We see that most clients react to KoD packets. For DENY, only ntpd stopped querying – all the others reduced their query rates. For RATE, all of them reduced, only NTPD-RS had a bug (which caused an increase) – and ntpd kept a constant query rate.

NTPD-RS bug: we found that NTPD-RS instead of either ignoring DENY packets (if choosing to distrust its authenticity) or stopping using the server, it does actually hammers the servers. In a period of 5 minutes, it sent 541613 packets (1800 qps). In Figure 3a, note that we show only the first 5min of KoD DENY experiment, which we stopped after it. (We show the scatter plot of queries in §A.1). We reported the issues to the NTPD-RS developers on GitHub [42] and they acknowledged and fixed the issue.

ntpd: being the reference implementation, we had expected ntpd to conform to the standards. However, that is not what we observed. We see that it does not stop using the NTP servers if it receives a KoD deny packet. We show in Figure 3b the time series of queries for a baseline reference (REF) and the KoD experiments. We see how DENY, ntpd kept on using a polling interval of 6 for more than two hours, constantly, and later had a in increase. For KoD RATE, it simply ignored an kept querying with a constant polling interval of 3 (8s).

We contacted the ntpd developers and they have acknowledged the issue. They said that ntpd only "test for the incoming RATE code". So in this case, it is a deliberate choice that helps ntpd to avoid clock drifts by avoiding stopping contacting the server.

Takeaway: we found that the NTP clients will not accept KoD packets and stop querying NTP servers — therefore they are not vulnerable to KoD attacks in our threat model.

4 Discussion and recommendations

We have identified in total 11 issues with current S/NTP clients and notified vendors and opeators (we summarize it in a table on Appendix E). We discuss the ethics of this paper in Appendix F.

Our results shows how the three major OSes (Windows, MacOS and Ubuntu/Linux) rely on SNTP clients that are vulnerable to time shift attacks. We commend that both Microsoft and Apple deploy robust NTP clients in their systems, with NTS support, to mitigate the risk of time shift attacks, just as Ubuntu is planning for Oct. 2025.

We also recommend SNTP software developers to make their software more robust against time shift attacks. For instance, by validating the responses received in case of odd offsets. For instance, OpenNTPd has feature called "constraints", which allows it to "query 'Date' from trusted HTTPS servers via TLS" [47], and verify if the reported timestamp on NTP responses are within reasonable bounds.

Lastly, we recommend that every systemd-based Linux distribution should reconsider using timesyncd as it currenlty is, and switch to more robust alternatives.

5 Related Work

Client security: The closest work to ours is by Malhous [15], where they peform time-shift attacks on NTP clients (Chrony, ntpd, OpenNTPd, and NTPSec). Differently from us, they perform MITM attacks, whereas our threat model for attacks start from clients being already served by the malicious server (§3.1). They suceeded in changing clocks by multiple of 1000s, whereas we succeded in changing years. We evaluate 8 clients – and 3 being default clients in the most popular

OSes, under normal behavior, time shift attacks using multiple offsets, and under KoD packets.

NTP security: NTPv4 has known security issues; it can be exploited in DDoS amplification attacks [6]. NTS was later proposed as an extension to NTPv4, and it protect against man-in-the-middle attacks, but not against attacks that include DNS hijacking as in our threat model (NTP Pool currently does not support NTS.)

Attacks against NTP. Malhotra et al. [17] exposed how unauthenticated NTP traffic enables a range of attacks, including time-shift and Denial-of-Service (DoS) attacks. They show how clients are vulnerable during initialization and can be manipulated via spoofed or fragmented packets. Whereas their work evaluate only `ntpd`, we evaluate 8 different NTP/S-NTP clients, but against three attacks. While many of the specific vulnerabilities identified have since been addressed [38], time-shift attacks remain a persistent risk, particularly due to clients' acceptance of large time changes at startup.

6 Conclusion

NTP lurks in the background of modern computer systems playing a key role in its well functioning. We have evaluated 8 S/NTP clients and showed that there several good options and yet, the three most popular modern OSes still rely SNTP clients that are vulnerable to time shift attacks, creating unecessary risks for its billion of daily users. We hope our results encourage Microsoft, Apple and `timesyncd` folks to improve their software against time shift attacks.

References

[1] Apple Inc. 2024. Apple Reports First Quarter Results. https://www.apple.com/newsroom/2024/02/apple-reports-first-quarter-results/. Accessed: 2025-02-18.

[2] Arch Linux. 2024. *systemd-timesyncd.* https://wiki.archlinux.org/title/Systemd-timesyncd Accessed: 2025-05-15.

[3] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. 2005. *DNS Security Introduction and Requirements.* RFC 4033. IETF. http://tools.ietf.org/rfc/rfc4033.txt

[4] Randy Bush and Rob Austein. 2013. *The Resource Public Key Infrastructure (RPKI) to Router Protocol.* RFC 6810. IETF. http://tools.ietf.org/rfc/rfc6810.txt

[5] Chrony Project. 2025. *chrony.conf - chronyd configuration file.* https://chrony-project.org/doc/3.4/chrony.conf.html Accessed: 2025-05-13.

[6] Jakub Czyz, Michael Kallitsis, Manaf Gharaibeh, Christos Papadopoulos, Michael Bailey, and Manish Karir. 2014. Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks. In *Proceedings of the 2014 ACM Conference on Internet Measurement Conference* (Vancouver, BC, Canada) *(IMC)*. ACM, 435–448. https://doi.org/10.1145/2663716.2663717

[7] Debian Project. 2025. *Debian – The Universal Operating System.* https://www.debian.org/ Accessed: 2025-09-25.

[8] Omer Deutsch, Neta Rozen Schiff, Danny Dolev, and Michael Schapira. 2018. Preventing (Network) Time Travel with Chronos. In *Network and Distributed Systems Security (NDSS) Symposium 2018.* San Diego, CA, USA. https://doi.org/10.14722/ndss.2018.23231

[9] Ubuntu Developers. 2025. *chrony (version 4.7-1ubuntu1) — Ubuntu package in questing.* https://packages.ubuntu.com/questing/chrony Accessed: 2025-09-17.

[10] Tim Dierks and Eric Rescorla. 2008. *The Transport Layer Security (TLS) Protocol Version 1.2.* RFC 5246. IETF. http://tools.ietf.org/rfc/rfc5246.txt

[11] Daniel Franke, Dieter Sibold, Kristof Teichel, Marcus Dansarie, and Ragnar Sundblad. 2020. *Network Time Security for the Network Time Protocol.* RFC 8915. IETF. http://tools.ietf.org/rfc/rfc8915.txt

[12] Henning Brauer. 2022. *OpenNTPd.* https://www.openntpd.org Accessed: 2025-05-13.

[13] IEEE. 2002. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std. 1588-2002* (2002). https://standards.ieee.org/ieee/1588/3140/

[14] Jonghoon Kwon, Jeonggyu Song, Junbeom Hur, and Adrian Perrig. 2023. Did the Shark Eat the Watchdog in the NTP Pool? Deceiving the NTP Pool's Monitoring System. In *30th USENIX Security Symposium.* https://www.usenix.org/conference/usenixsecurity23/presentation/kwon

[15] Ahmed R. Mahlous. 2024. Quantitative Risk Analysis of Network Time Protocol (NTP) Spoofing Attacks. *IEEE Access* 12 (2024), 164891–164910. https://doi.org/10.1109/ACCESS.2024.3493759

[16] Aanchal Malhotra, Isaac E Cohen, Erik Brakke, and Sharon Goldberg. 2016. Attacking the Network Time Protocol. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS 2016)* (San Diego, California).

[17] Aanchal Malhotra and Sharon Goldberg. 2016. Attacking NTP's Authenticated Broadcast Mode. *SIGCOMM Comput. Commun. Rev.* 46, 2 (may 2016), 12–17.

[18] Lukas Märdian. 2025. PSA: Installing chrony by default, to enable Network Time Security (NTS). Ubuntu-devel mailing list. https://lists.ubuntu.com/archives/ubuntu-devel/2025-May/043355.html Message posted on May 22, 2025.

[19] Mark Morowczynski. 2012. Did Your Active Directory Domain Time Just Jump To The Year 2000? https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/did-your-active-directory-domain-time-just-jump-to-the-year-2000/ba-p/255873.

[20] Hicham Marouani and Michel R. Dagenais. 2008. Internal Clock Drift Estimation in Computer Clusters. *Journal of Computer Networks and Communications* 2008, 1 (2008), 583162. https://doi.org/10.1155/2008/583162

[21] Microsoft. 2022. Windows Devices by the Numbers. https://web.archive.org/web/20220419050729/https://news.microsoft.com/bythenumbers/en/windowsdevices. Accessed: 2025-02-18.

[22] Microsoft Corporation. 2024-04-23. [MS-SNTP]: Network Time Protocol (NTP) Authentication Extensions. (2024-04-23).

[23] Microsoft Corporation. 2024-04-23. [MS-W32T]: W32Time Remote Protocol. (2024-04-23).

[24] David Mills. 2006. *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI.* RFC 4330. IETF. http://tools.ietf.org/rfc/rfc4330.txt

[25] David Mills, Jim Martin, Jack Burbank, and William Kasch. 2010. *Network Time Protocol Version 4: Protocol and Algorithms Specification.* RFC 5905. IETF. http://tools.ietf.org/rfc/rfc5905.txt

[26] Giovane C. M. Moura. 2025. *Tech report on S/NTP clients behavior.* https://community.ntppool.org/t/tech-report-on-s-ntp-clients-behavior/4107/3 Post on the NTP Pool Project community forum.

[27] Giovane C. M. Moura, Marco Davids, Caspar Schutijser, Cristian Hesselman, John Heidemann, and Georgios Smaragdakis. 2024. Deep Dive into NTP Pool's Popularity and Mapping. 8, 1, Article 15 (feb 2024), 30 pages. https://doi.org/10.1145/3639041

[28] Giovane C. M. Moura and John Heidemann. 2023. Vulnerability Disclosure Considered Stressful. *SIGCOMM Comput. Commun. Rev.* 53, 2 (April 2023), 2–10. https://doi.org/10.1145/3610381.3610383

[29] Giovane C. M. Moura, John Heidemann, Ricardo de O. Schmidt, and Wes Hardaker. 2019. Cache Me If You Can: Effects of DNS Time-to-Live. In *Proceedings of the ACM Internet Measurement Conference*. ACM, Amsterdam, the Netherlands, 101–115. https://doi.org/10.1145/3355369.3355568

[30] Network Time Foundation. 2025. *NTPD 4.2.8p-series*. https://www.ntp.org/documentation/4.2.8-series/#building-and-installing-ntp Accessed: 2025-05-13.

[31] Clifford Neuman, Tom Yu, Sam Hartman, and Kenneth Raeburn. 2005. *The Kerberos Network Authentication Service (V5)*. RFC 4120. IETF. http://tools.ietf.org/rfc/rfc4120.txt

[32] ntpd-rs Project. 2025. *ntp.toml - Configuration File for the ntpd-rs NTP Daemon*. https://docs.ntpd-rs.pendulum-project.org/man/ntp.toml.5/ Accessed: 2025-05-13.

[33] NTPsec project. 2022. *NTPSec*. https://www.ntpsec.org Accessed: 2025-05-13.

[34] United States Naval Observatory. 2022. Information about NTP, the time backbone of the Internet. (Nov. 5 2022). https://www.cnmoc.usff.navy.mil/Our-Commands/United-States-Naval-Observatory/Precise-Time-Department/Network-Time-Protocol-NTP/

[35] NTP Pool. 2025. *pool.ntp.org: the internet cluster of ntp servers*. https://www.ntppool.org/en/ Accessed: 2025-04-23.

[36] Chrony Project. 2025. *Chrony: A Versatile Implementation of NTP*. https://chrony-project.org/ Accessed: 2025-04-23.

[37] Pendulum Project. 2025. *Algorithm Documentation*. https://github.com/pendulum-project/ntpd-rs/blob/main/docs/algorithm/algorithm.pdf

[38] D. Reilly, H. Stenn, and D. Sibold. 2019. *Network Time Protocol Best Current Practices*. RFC 8633. IETF. http://tools.ietf.org/rfc/rfc8633.txt

[39] E. Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. IETF. http://tools.ietf.org/rfc/rfc8446.txt

[40] Teemu Rytilahti, Dennis Tatang, Janosch Köpper, and Thorsten Holz. 2018. Masters of Time: An Overview of the NTP Ecosystem. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*. 122–136. https://doi.org/10.1109/EuroSP.2018.00017

[41] S. Sakane, K. Kamada, M. Thomas, and J. Vilhuber. 2006. *Kerberized Internet Negotiation of Keys (KINK)*. RFC 4430. IETF. http://tools.ietf.org/rfc/rfc4430.txt

[42] Shreyas Konjerla. 2025. *KISS codes not ignored #1867*. https://github.com/pendulum-project/ntpd-rs/issues/1867 Accessed: 2025-05-14.

[43] Shreyas Konjerla. 2025. *OpenNTPd not keeps track of the Era #77*. https://github.com/openntpd-portable/openntpd-portable/issues/77 Accessed: 2025-05-16.

[44] squell. 2025. NTS support for systemd-timesyncd (Pull Request #39010). https://github.com/systemd/systemd/pull/39010. Accessed: 2025-10-16.

[45] systemd project. 2025. systemd - System and Service Manager. https://systemd.io/ Accessed: 2025-09-24.

[46] Canonical Multipass Team. 2025. canonical/multipass: Multipass orchestrates virtual Ubuntu instances. https://github.com/canonical/multipass. Accessed: 2025-09-02.

[47] The OpenBSD Project. 2025. *ntpd.conf(5) - OpenBSD Manual Pages*. OpenBSD. https://man.openbsd.org/ntpd.conf Accessed: 2025-09-19.

[48] LLC Turing Software. 2025. UTM: Virtual Machines for Mac. https://mac.getutm.app/ Accessed: 2025-08-27.

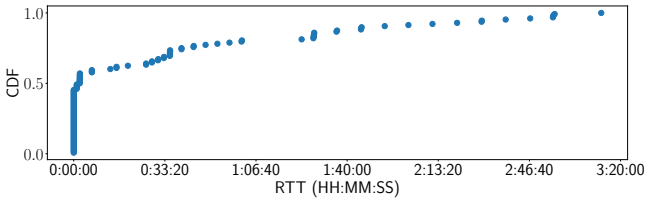


Figure 4: NTPD-RS time series from Japan.

Appendix

A Extra figures

Figure 4 provides a timeseries of polling interval when we use NTPD-RS against a server listed with the NTP Pool from Japan. We do not observe a large volume of queries as in Table 2.

This server had a polling interval of 3, therefore 8s. We run it on 2025-05-24 from a VP located in Europe. We saw that the NTPD-RS was not as aggressive as we saw in our local experiments. This may suggest NTPD-RS is more conservative when it is not in a local environment. This was confirmed when asked about to the developers. This is also mentioned in §2.2.

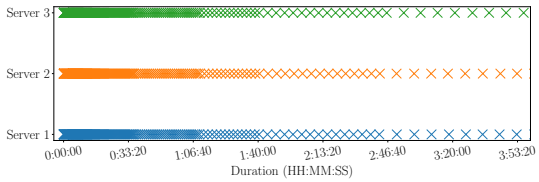
In Figure 5 shows the server utilization patterns of different clients. In these figures, a mark is a NTP request sent to that specific server. Each of these clients was provided with 3 servers. Looking at this we can see how different clients used their servers. ntpd is the reference point which uses all the servers at all time sending request all of them. macOS uses only 2 servers at a time and frequently switches between them to balance the load. W32Time uses only one server at a time and switches between them after long periods of time. NTPD-RS is another client which uses all the servers at the same time.

This again highlights the different design decisions made by the developers of these clients and how they impact the security of the clients. Clients which use multiple servers at the same time are more resilient to attacks since they can cross-verify the responses. Clients which use only one server at a time are more vulnerable and can be affected a single malicious/malfinctioning server.

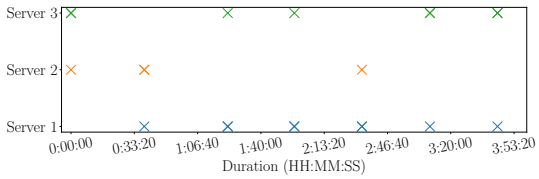
A.1 KoD figures

Figure 6 shows the scatter plot of queries during the KoD Deny bug of NTPD-RS.

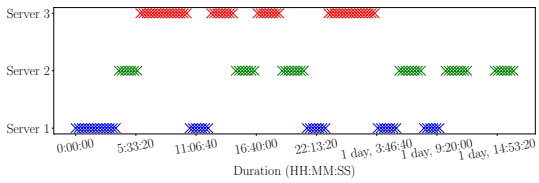
Figure 7 show the timeseries of queries and polling rate for each client under KoD Rate and Deny packets. In the figure, the dashed red vertical line shows when we the experiment start sending KoD messages – after a hot start. Comparing it with Figure 1, we see how ntpd use a fixed rate for DENY and RATE, and Chrony seems to reduce its rates in both



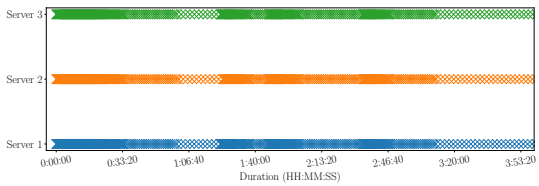
(a) `ntpd`



(b) `macOS`



(c) `W32Time`



(d) `NTPD-RS`

Figure 5: Distinct server utilization patterns.

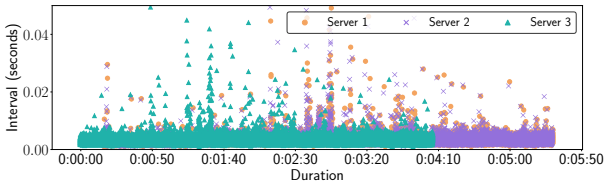
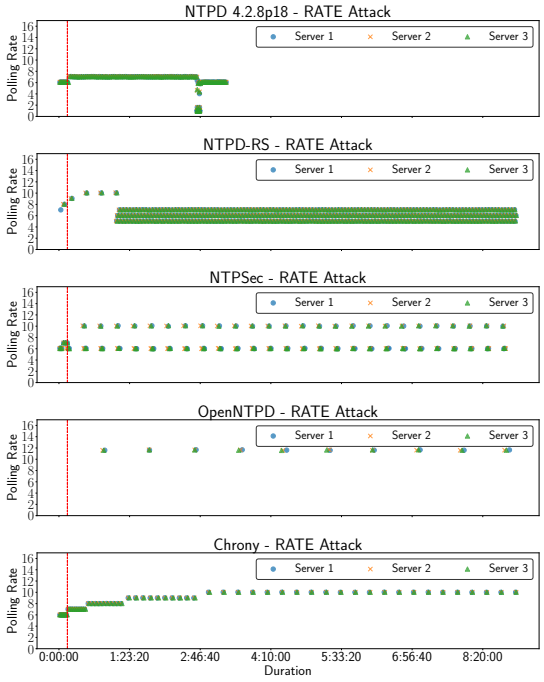
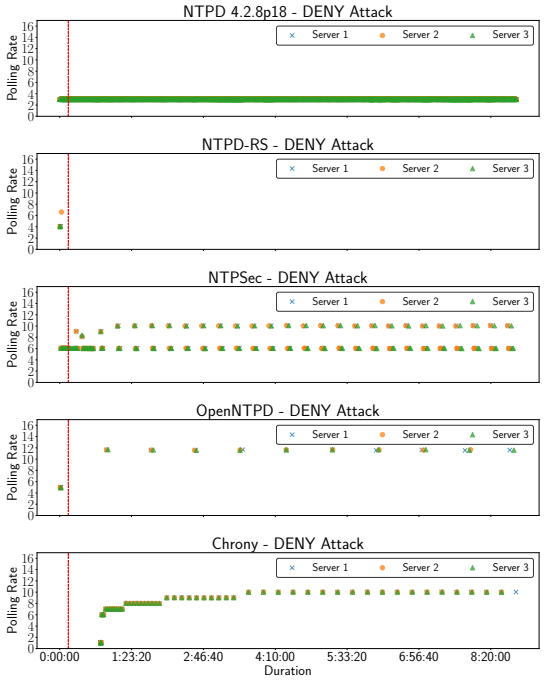


Figure 6: NTPD-RS `DENY` scatter plot during KoD Deny bug.



(a) KoD Rate



(b) KoD Deny

Figure 7: Client behavior under KoD attacks – all clients

cases (we discussed the results with `Chrony` developers and they pointed that `Chrony` supports KoD and "temporarily" reduces querying rates [26].)

B Windows versions and different behaviors

Windows' `W32Time` is also a SNTP-based client, similar to macOS. It operates in a similar fashion by sending requests at long intervals, using the `time.windows.com` server pool. However, the behavior differs between Windows Server (2025) and Windows Home (2025). While both function similarly, the key difference is the polling rate. Windows Server has a polling rate of 10, meaning it requests time updates every 17 minutes, while Windows Home has a polling rate of 15, meaning it requests updates every 9 hours.

B.1 Windows as a Standalone System.

Windows devices that are not part of a domain rely on public NTP servers provided by `time.windows.com`. These machines trust the servers assigned by DNS, similar to macOS. If an attacker successfully hijacks DNS resolution, they could redirect the system's NTP requests to a malicious time server, allowing them to manipulate the clock without the user's knowledge.

Windows Home updates its clock every 9 hours. However, if the system is hibernating or shut down at the scheduled update time, Windows does not immediately synchronize upon wake-up. Instead, it skips the update and attempts it 9 hours later. This means that if a user only uses their device intermittently, their system can remain unsynchronized for days or even weeks and drift by itself significantly.

For Windows Home, the minimum and maximum polling rates are 10 and 15, whereas for Windows Server, they are 6 and 10 [23]. Windows Server can be configured to accept time only from internal sources, but by default, it still allows synchronization with `time.windows. com` when no domain controller is configured. Additionally, Windows automatically rejects time shifts greater than two weeks, but this limit can be lowered via Group Policy settings in Pro and Enterprise editions [23].

B.2 Windows as Part of a Domain.

(Active Directory & Group Policy Enforcement.) In domain-joined environments, Windows devices do not typically use `time. windows.com` for synchronization. Instead, they rely on time updates from Active Directory domain controllers (AD DCs), which adds additional security and accuracy. The primary domain controller (PDC) serves as the main time authority for domain members.

Microsoft has also developed a variant of their SNTP client, as defined in [22]. This client is built on top of NTPv3, defined in RFC1305, and SNTPv4, defined in RFC2030. It introduces additional fields for packet authentication, which is only performed when messages remain within the domain and are not sent to public servers. Two new fields are added: an `Authenticator` field (160 bits) or an `ExtendedAuthenticator` field (576 bits). This significantly increases the packet size from 48 bytes to either 68 or 120 bytes.

The `Authenticator` field includes a Key Identifier and a 128-bit cryptographic checksum for the packet. The `Extended Authenticator` field contains the key identifier, flags, hashes, and a 512-bit cryptographic checksum. The key identifier is shared between the server and clients when a client joins the domain. Typically, a client only sends requests to its designated domain controller, as enforced by the domain's Group Policy Objects (GPOs) [22].

Domain controllers themselves synchronize their clocks by sending requests to their parent DCs in the forest, with only the PDC making requests to external sources. This hierarchical structure creates a potential attack vector: if an attacker can hijack and force the PDC to synchronize with a rogue time source, they could, in theory, poison the entire domain.

C Ubuntu Chrony with NTS Experiment

We have shown in §2 that `Chrony`, as a NTP cliends, sends more queries to NTP servers than `timesyncd`, a SNTP-based client.

Next we configure two Ubuntu VMS (Ubuntu 24.04.3 LTS, one in AMS and the other in SYD) and use them to extimate the increase in NTP traffic for a Ubuntu system when it switches to `Chrony`– as it will be the default set in Oct. 2025 [18].

C.1 `timesyncd` setup

Listing 1 shows the default `timesyncd` configuration file shipped with Ubuntu. As can be seen, it will use only one server: one address from `ntp.ubuntu.com`

We run this configuration for 60 h and capture NTP packets on both servers using `tcpdump`.

```
# /etc/systemd/timesyncd.conf

[Time]
#NTP=
#FallbackNTP=ntp.ubuntu.com
#RootDistanceMaxSec=5
#PollIntervalMinSec=32
#PollIntervalMaxSec=2048
#ConnectionRetrySec=30
#SaveIntervalSec=60
```

Listing 1: `timesyncd` Config File

C.2 `Chrony` setup

After running the experiment with `timesyncd`, we installed `Chrony` using the default Ubuntu's package.

However, we had to change its original configuration, given the `Chrony` package it ships with this released (24.04.03) is not yet configured to use NTS by default. To reproduce the

setup that October 2025 (25.10) release will have, we downloaded the file `chrony_4.7-1ubuntu1.debian.tar.xz` from Ubuntu's packages site [9], and copied Chrony's configuration files from it, replacing the original ones from our VMs with those we manually downloaded.

In this way, we have exactly the same setup that Ubuntu will ship in 25.10 with Chrony and NTS by default. Listing 2 shows the main configuration (we excluded comments for brevity), and it shows in line 2 how the NTP servers are set in another file, which we show in Listing 3.

```
1  # /etc/chrony/chrony.conf
2  sourcedir /etc/chrony/sources.d
3  keyfile /etc/chrony/chrony.keys
4  driftfile /var/lib/chrony/chrony.drift
5  ntsdumpdir /var/lib/chrony
6  logdir /var/log/chrony
7  maxupdateskew 100.0
8  rtcsync
9  makestep 1 3
10 confdir /etc/chrony/conf.d
```

Listing 2: Chrony main

```
1  # sources.d/ubuntu-ntp-pools.sources
2  # Use NTS by default
3  # NTS uses an additional port to negotiate security:
       4460/tcp
4  # The normal NTP port remains in use: 123/udp
5  pool 1.ntp.ubuntu.com iburst maxsources 1 nts prefer
6  pool 2.ntp.ubuntu.com iburst maxsources 1 nts prefer
7  pool 3.ntp.ubuntu.com iburst maxsources 1 nts prefer
8  pool 4.ntp.ubuntu.com iburst maxsources 1 nts prefer
9  # The bootstrap server is needed by systems without a
       hardware clock, or a very
10 # large initial clock offset. The specified certificate
       set is defined in
11 # /etc/chrony/conf.d/ubuntu-nts.conf.
12 pool ntp-bootstrap.ubuntu.com iburst maxsources 1 nts
       certset 1
```

Listing 3: Chrony sources

Just like `timesyncd`, we run it for roughly 60 h, and capture traffic with tcpdump.

Next we compare the results.

C.3 Results

Table 6 and Table 7 summarize the experiments for both AMS and SYD VMs. As expected, `timesyncd` uses a single NTP server, whereas Chrony use multiple servers (5 in each case), and send more queries per hour to each server than `timesyncd`.

Notice that only Chrony uses NTS, and NTS queries and responses are larger, given they use NTP extension fields (NTS Cookie and NTS Authenticator and Encrypted Extensions), more than doubling its size (110 to 250).

D Apple vulnerability Discussion

As mentioned in §3.2, macOS has a vulnerability that can be exploited by an attacker to shift the time of the system clock by any amount. It does not have any restrictions or panic thresholds such as windows or other NTP clients. We reported this issue to Apple at first they mentioned since we changed the time source itself, it was a severe issue and an attack vector cannot be executed. We later tried the same without changing the time source by first DNS poisoning. This was not successful since Apple uses other heuristics and DNS-over-HTTPS (DoH) for some queries. We then tried IP spoofing where the router will route NTP queries to the attacker and the attacker will respond with a malicious response with a spoofed IP source address. This attack was successful and we were able to shift the clock by up to 30 years. We also reported this back to attack but after their reviews, they concluded was beyond the scope of macOS security as it involved an attacker-controlled router.

E Extra tables

Table 8 sumamrizes the issues and bugs we found with S/NTP clients we evaluated.

We also found that `OpenNTPd` cannot handle NTP ERA rollover, which we include in Appendix G.

F Ethics and CVD

Our work poses one ethical challenge: the notification of vendors of the bugs/vulnerabilities in their software. In total, we have identified eleven issues/bugs and notified all vendors (Table 8). As of the moment of this writing, only one was fixed, and the others are either waiting or considered not to be a bug.

The response we obtained from Apple in case of time shift attacks issue, was that "Users are allowed to point to the time server of their choosing in System Settings after authenticating and should always consider care when overriding system defaults".

G Era Rollover bug

NTP timestamps are 64-bit timestamp format, consisting of a 32-bit part for seconds and a 32-bit part for fractional seconds. Specifically, the 32-bit seconds field overflows approximately every 2^{32} seconds, or about 136 years, each period defined as eras. The first NTP era (era 0) began at 00:00:00 UTC on 1 January 1900, and the era 1 will commence on 7 February 2036.

Without proper handling of these era transitions, otherwise when era 1 arrives, they may assume that time is 1900. Consequently, modern NTP implementations need to have mechanisms to infer the correct era "derived from external means, such as the file system or dedicated hardware" [25].

In this section, we evaluate whether the NTP clients are era-aware. We use the same setup used in the previous experiments and set the clients clock to one hour before the end of era 0. Then, we monitor the systems clock.

timesyncd

NTP Server	# Que	# Resp.	Que. (Bytes)	Resp (Bytes)	Avg Que.	Avg Resp.	Span (s)	Queries/hr
2620:2d:4000:1::41	128	128	14080	14080	110	110	251871.51	1.83

Chrony

NTP Server	# Que	# Resp.	Que. (Bytes)	Resp (Bytes)	Avg Que.	Avg Resp.	Span (s)	Queries/hr
2620:2d:4000:1::1123	333	333	83250	83250	250.00	250.00	243106.66	4.93
2620:2d:4000:1::3123	301	301	75250	75250	250.00	250.00	242736.12	4.46
91.189.91.112	270	270	62100	62100	230.00	230.00	242586.90	4.01
2620:2d:4002:1::3123	265	223	70398	59354	265.65	266.16	243004.06	3.93
185.125.190.122	263	263	60490	60490	230.00	230.00	242112.52	3.91

Table 6: AMS Server Traffic Summary

timesyncd

NTP Server	# Que	# Resp.	Que. (Bytes)	Resp (Bytes)	Avg Que.	Avg Resp.	Span (s)	Queries/hr
2620:2d:4000:1::40	128	128	14080	14080	110.00	110.00	251903.33	1.83

Chrony

NTP Server	# Que	# Resp.	Que. (Bytes)	Resp (Bytes)	Avg Que.	Avg Resp.	Span (s)	Queries/hr
185.125.190.123	299	293	69586	67798	232.73	231.39	242767.78	4.43
185.125.190.122	277	272	64118	62900	231.47	231.25	242283.66	4.12
2620:2d:4002:1::2123	267	259	67362	65294	252.29	252.10	242266.92	3.97
2620:2d:4002:1::3123	266	209	72688	57010	273.26	272.78	242472.44	3.95
2620:2d:4000:1::1123	263	263	65750	65750	250.00	250.00	242276.70	3.91

Table 7: SYD Server Traffic Summary

#	Software	Issue	Vendor response
1	NTPD-RS	Queries surge to nearby servers (§2.3.2)	Feature, not a bug
2	Ubuntu	Traffic growth by using Chrony and NTS (§2.4)	Acknowledged
3	macOS	Vulnerable to time shift (§3.1.3)	“Not a bug”
4	timesyncd	Vulnerable to time shift (§3.1.3)	Waiting
5	W32Time	Vulnerable to time shift (§3.1.3)	
6	Debian Linux	Vulnerable to time shift (§3.1.3)	Responded, add NTS support
7	NTPD-RS	Crashes during time shift attacks (§3.1.3)	Notified
8	NTPD-RS	KoD Bug (too many queries – §3.2)	Bug patched [42]
9	ntpd	Stops operating with responded with RATE (§3.2)	Acknowledged
10	ntpd	Keeps sending queries after receiving DENY (§3.2)	Acknowledged
11	OpenNTPd	Cannot handle ERA rollover	Notified

Table 8: Clients issues observed in the paper and CVDs status (2025-09-25).

Client	Start up offset	Era aware
NTPD	Yes	Yes
NTPD-RS	Yes	Yes
NTPSec	Yes	Yes
OpenNTPd	Yes	No
TimesyncD	Yes	Yes
Chrony	Yes	Yes
macOS	No	Yes
Windows	No	Yes

Table 9: Eras rollover experiment results

Table 9 shows the experiment results. From the clients we manage to bring the clock to the end of era 0, only OpenNTPd was not era aware: it computed a large offset (136 years) but refused to update its clock, likely assuming it was a time shift attack (as in §3.1). We notified the OpenNTPd developers and waiting to hear back from them [43].