Is the Client Always Right? A Security Evaluation of (S)NTP Clients

Technical Report SIDN Labs 2025-10-16
Update: 2026-11-10

Shreyas Konjerla[1], Giovane C. M. Moura[1,2], Tamme Dittrich[3], and Georgios Smaragdakis[1]

[1] TU Delft, Delft, The Netherlands
s.konjerla@student.tudelft.nl, G.Smaragdakis@tudelft.nl
[2] SIDN Labs, Arnhem, The Netherlands
giovane.moura@sidn.nl
[3] Tweede Golf, Nijmegen, The Netherlands
tamme@tweedegolf.com

Abstract. The Network Time Protocol (NTP) is the Internet's default time synchronization protocol and it is used daily to synchronizes the clocks of billions of devices. Prior work has studied the protocol and its servers; how clients respond to incorrect time has received little attention. Yet the client decides whether to adopt a server's time. NTPv4 specifies two safeguards for this decision, a panic threshold and cross-validation across servers; SNTP omits both, and SNTP clients are the defaults on macOS, Windows, and many Linux distributions. We evaluate eight NTP and SNTP clients, including these three defaults, in normal operation and against a malicious or faulty time source. All three defaults adopt incorrect time: `timesyncd` and `macOS` accept a two-year offset within 13 min and 2 h, respectively, and `W32Time` accepts offsets up to two weeks. An end-to-end attack shifts an unmodified `macOS` client by 30 years. All five NTP clients reject every offset, including when only one of three servers is honest. Their Kiss-o'-Death handling, however, deviates from the standards: none stops querying a server after a DENY, and one sends 1,800 queries/s. We disclosed nine issues to vendors and, with Ubuntu's operators, quantified the 25× traffic increase of their switch to `Chrony` with Network Time Security.

Keywords: NTP · Clock synchronization · Time-shift attacks · NTP clients

1 Introduction

Clock synchronization underpins the security of the Internet. TLS certificate validation [17], DNSSEC signatures [3], RPKI [9], Kerberos [48], and DNS caches [45] all rely on synchronized clocks to establish freshness [40,31,59,16].

Hardware clocks drift over time, as their crystal oscillators run slightly fast or slow relative to true time swing to temperature fluctuations, manufacturing tolerances, and power variations [36]. To keep this drift from accumulating, clients run time synchronization protocols that *periodically* query time servers and realign the local clock with a reference.

The Network Time Protocol (NTP) [40] is the Internet's default protocol for clock synchronization. It is designed to absorb the variable network latency between client and server, achieving accuracy of tens of milliseconds across the public Internet. Clients synchronize by exchanging timestamps with NTP servers, estimating the clock offset, and precisely adjusting the local clock accordingly.

Protocols: NTPv4 [40], standardized in 2010, specifies the wire format together with client-side algorithms for server selection, timestamp filtering, and clock discipline. SNTP [39] retains the packet format but omits these algorithms. NTPv4 traffic is unauthenticated in practice; Network Time Security (NTS) [19] authenticates servers and packets with keys derived from a TLS handshake [58]. NTPv5 [29] is under standardization.

NTP Clients: Client implementations differ in how closely they follow these specifications; `Chrony` [55], for example, omits broadcast and multicast client modes [13]. Because the client decides whether to adopt a server's time and adjust its clock, its behavior determines whether incorrect time propagates.

Incorrect time may propagate to every service that depends on the clock. In November 2012, the US Naval Observatory (USNO) NTP servers returned timestamps 12 years in the past; routers and Active Directory servers that accepted them failed [35,5]. In July 2026, the GPS receiver of a single NTP timing device at Telstra, Australia's largest telecommunications operator, computed a date 1,024 weeks in the past (November 2006). The date propagated through Telstra's mobile core, whose nodes and applications accepted it; core elements reset, and access from about 30,000 IP prefixes was blocked. Restoring most services took more than nine hours, and some users could not place emergency (000) calls [68,67,65]. In both incidents, a single faulty source caused an outage because clients applied a multi-year offset. An adversary that controls a client's time source can induce the same failure.

The client is thus the last line of defense. NTPv4 specifies two safeguards: a *panic threshold*, which rejects offsets above 1,000 s, and cross-validation of multiple servers before the clock is adjusted [40]. SNTP specifies neither [39]. Nonetheless, SNTP clients are the defaults on macOS and Windows; `timesyncd`, an SNTP client, was the default on Ubuntu until October 2025 and remains the default on many other `systemd`-based distributions. Whether these clients, or the NTP clients that could replace them, reject incorrect time has not been measured. Prior work has studied NTP servers [46,59,61,4,23] and server-side attacks [16,14,28,53]. On the client side, Mahlous [30] attacked four NTP clients through a man-in-the-middle, but evaluated neither the OS defaults nor SNTP.

We evaluate eight NTP and SNTP clients (Table 1): the defaults of Windows Server (2025.02.13), `macOS`, and Ubuntu, and five NTP implementations—`ntpd` (closest to the reference implementation), `NTPSec` (a hardened `ntpd` fork with

Client (version)	User base	OS	Release
`macOS` (15.4.1)	2.2B* [1]	macOS	Sept 2024
`W32Time` [38]	1.4B [37]	Windows	April 2025
`timesyncd` (255.4) [2]	-	Ubuntu	Sept 2024
`NTPSec` (1.2.2) [51]	-	Ubuntu	Nov 2024
`NTPD-RS` (1.1.2) [50]	-	Ubuntu	Jan 2025
`ntpd` (4.2.8p18) [47]	-	Ubuntu	June 2023
`OpenNTPd` (1:6.2p3-4.2) [21]	-	Ubuntu	Oct 2022
`Chrony` (4.5) [55]	-	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).

NTS), `Chrony` (Ubuntu's default since October 2025, with NTS [34]), `OpenNTPd` (OpenBSD), and `NTPD-RS` (Rust, with NTS). The adversary, or a fault, controls the time source a client queries. We study the two decisions this source influences: whether the client *adopts* the served time (time-shift attacks, §4) and whether it *continues to query* the source (Kiss-o'-Death (KoD) attacks, §5). Off-path packet injection [31] is out of scope. We address three research questions:

RQ1 How many servers do clients query in normal operation, and how often (§3)? The answer determines the load on time services and the number of servers an adversary must control.

RQ2 Can a malicious or faulty time source shift each client's clock, by how much, and how fast (§4.1, §4.2)?

RQ3 Do implementations enforce the safeguards the standards specify: the panic threshold, multi-server cross-validation, KoD handling, and era handling (§4, §5, §6)?

Our contributions are:

- *All three OS default clients adopt incorrect time* (RQ2, §4.1). `timesyncd` and `macOS` accept every offset tested, up to two years, within 13 min and 2 h, respectively. `W32Time` bounds adjustments to two weeks but accepts a one-week offset within 36 h. An end-to-end attack with in-path spoofing shifts an unmodified `macOS` client by 30 years.
- *All five NTP clients reject incorrect time* (RQ2, RQ3, §4.1, §4.2), whether the adversary controls all servers or all but one. Client-side defenses are thus effective; the OS defaults do not implement them.
- *Query volume differs by up to two orders of magnitude across clients*, and clients range from querying a single server to querying all servers concurrently (RQ1, §3). With Ubuntu's operators, we quantified the cost of their switch from `timesyncd` to `Chrony` with NTS before the 25.10 release: 11× more queries and 25× more bytes per client (§3.4).
- *Implementations enforce safeguards inconsistently* (RQ3, §5, §6): no NTP client stops querying a server after a KoD DENY, one sends 1,800 queries/s,

and the `OpenNTPd` package in Ubuntu and Debian mishandles the 2036 era rollover.
– We disclosed nine client issues and two operational findings through coordinated vulnerability disclosure (CVD) [44], and classify each as specification-permitted, a specification deviation, or an implementation bug (§7).

2 Background

Clock synchronization: NTP organizes time sources into *strata*. Stratum 0 comprises reference clocks—atomic clocks, radio signals (e.g., DCF77 [7]), and Global Navigation Satellite Systems (GNSS) such as GPS and Galileo—which are not reachable over the network. Stratum 1 servers obtain time from these clocks out of band. Stratum n servers ($n \geq 2$) synchronize over NTP with stratum $n-1$ servers. An NTP *client* queries one or more servers, estimates its clock offset θ to each from the exchanged timestamps, and adjusts its clock by the combined estimate.

NTP and SNTP: Besides the wire format, NTPv4 [40] specifies client algorithms for server selection, timestamp filtering, and clock discipline. Two of them are relevant here. The *panic threshold* forbids adjusting the clock by an offset larger than 1,000 s. *Server selection* combines the responses of multiple servers and discards outliers before the clock is adjusted.

SNTP [39] retains the packet format but omits these algorithms; RFC 5905 intends it for "clients with a single upstream server and no dependent clients" [40].

Authentication: NTS [19] authenticates the server through a TLS [58] handshake and derives keys that protect each NTP packet. It prevents on-path tampering and redirection to an impostor server, but not incorrect time from a faulty server or from a malicious server with valid credentials.

NTPv5 [29], under standardization, retains only client and server modes, supports NTS, and leaves client algorithms out of scope. PTP [25] and White Rabbit [60] achieve higher accuracy but are confined to local and specialized networks.

3 Baseline Client Behavior

We first measure how clients use their servers when all servers are honest. This behavior matters for security and for operations. A client that queries several servers concurrently can cross-validate them, whereas a client that queries one server must trust it; the number of servers in use is the number an adversary must control. Query volume determines the load that time-service operators must provision for (§3.4). The measurements also serve as the baseline for the KoD experiments (§5). We measure query frequency and the number of servers queried concurrently.

3.1 Expected behavior

SNTP does not prescribe a query interval but bounds it: a client *must not* query more often than every 15 s (§10 in [39]) and should wait at least one minute between queries (§5 in [39]). A client configured with multiple servers should use only one (§7 in [39]).

NTP clients must wait at least 16 s between queries and apply exponential back-off as servers prove stable (§5 in [40]). At start-up, they may send a burst of queries to accelerate the first clock update; `Chrony`, for instance, sends four queries at intervals of at most 2 s [12]. NTP clients should query all configured servers (§5 in [40] and §3.2 in [57]). We test whether clients conform to these requirements.

3.2 Experimental setup

NTP servers: Our setup runs on a single physical machine (`macOS`), except for the Windows client (`W32Time`). We configure three Ubuntu VMs (24.04.3 LTS) with Multipass [66] to serve as our NTP servers, all running `Chrony`. We synchronize these three servers against external references, namely Apple's and Ubuntu's NTP servers (`time.apple.com` and `ntp.ubuntu.com`).

Clients: Each client in Table 1 runs in its own VM: Ubuntu VMs under Multipass and the `macOS` VM under UTM [71]. Each VM maintains its own clock, independent of the host. Each client is configured with our three NTP servers. Because servers and clients share one physical machine, network delay and jitter are minimal; we discuss the effect of this below. We capture all traffic and run each experiment for more than 60 h, which spans several back-off cycles of every client and suffices to reach steady state (Figure 1). The `W32Time` client runs on an AWS EC2 instance (Stockholm) and queries three NTP server VMs configured as above.

3.3 Results

Table 2 lists, per client, the total number of queries, the hourly mean, and the per-server breakdown. Although all clients run under identical, near-ideal network conditions, the hourly query rate ranges from 1.77 to 250.8.

SNTP clients: SNTP clients send the fewest queries, as expected (§3.1). Only `timesyncd` follows the SNTP recommendation to use a single server: it queries one of the three and does not contact the others while that server responds. It therefore cannot cross-validate its source. `W32Time` and `macOS` use all three servers over time, but not concurrently (see *Server usage*, below). All SNTP clients poll at a near-constant rate of 10–12 (17–68 min), visible as flat series in Figure 1.

			Queries Per Server			*Per Server*
Client	**Total**	**Avg/h**	**S1**	**S2**	**S3**	**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 (total and per server) in 60 hours in normal operation. highlighted are SNTP clients.

NTP clients: NTP clients send between 12 (`ntpd`) and 250 (`NTPD-RS`) queries per hour (Table 2). Figure 1 shows each client's poll rate per server over time. A poll rate n corresponds to a *poll interval* $T = 2^n$ s; $n = 10$ means one query every 1,024 s. NTP clients adapt T to network conditions and server stability.

`ntpd`: `ntpd` was long regarded as the NTP reference implementation; it no longer is [27]. It polls frequently at start-up and then backs off exponentially to a poll rate of 16 (18 h), within the maximum of 17 (§3.1; Figure 1).

Troubleshooting ***`NTPD-RS`:*** `NTPD-RS` never exceeds a poll interval of 256 s (rate 8), although its documentation specifies a range of 16–1,024 s (rates 4–10) and no burst mode [50]. Its documentation describes back-off triggered by failures [56]. The traces show no packet loss or retransmissions (90th-percentile RTT: 2.6 ms). The `NTPD-RS` developers confirmed the behavior as intended: for nearby servers, the client favors accuracy over network load.

To verify if these claims were correct, we carried out an additional experiment, configuring `NTPD-RS` to use a NTP server located in Japan, and the client located in Europe. We do not observe a large volume of queries as in Table 2, as can be seen in Figure 2, and did not exhibit the aggressive polling behavior observed in our local experiments.

Effect of latency: Our testbed thus represents ideal network conditions, under which at least one client queries more often than it would across the Internet. Table 2 therefore reflects behavior under ideal conditions. We did not evaluate packet loss or timeouts, which may trigger retransmissions.

Server usage: All clients except `timesyncd` use all three servers (Table 2). They differ in how they distribute queries across them (Figure 3). `ntpd` and `NTPD-RS` query all three concurrently. `macOS` queries two at a time and rotates among them. `W32Time` queries one server and switches after long intervals (Figure 1).

These patterns determine resilience, which we evaluate in §4. A client that queries several servers concurrently can cross-validate them; in our experiments, one honest server out of three sufficed to defeat the attack (§4.2). A client that

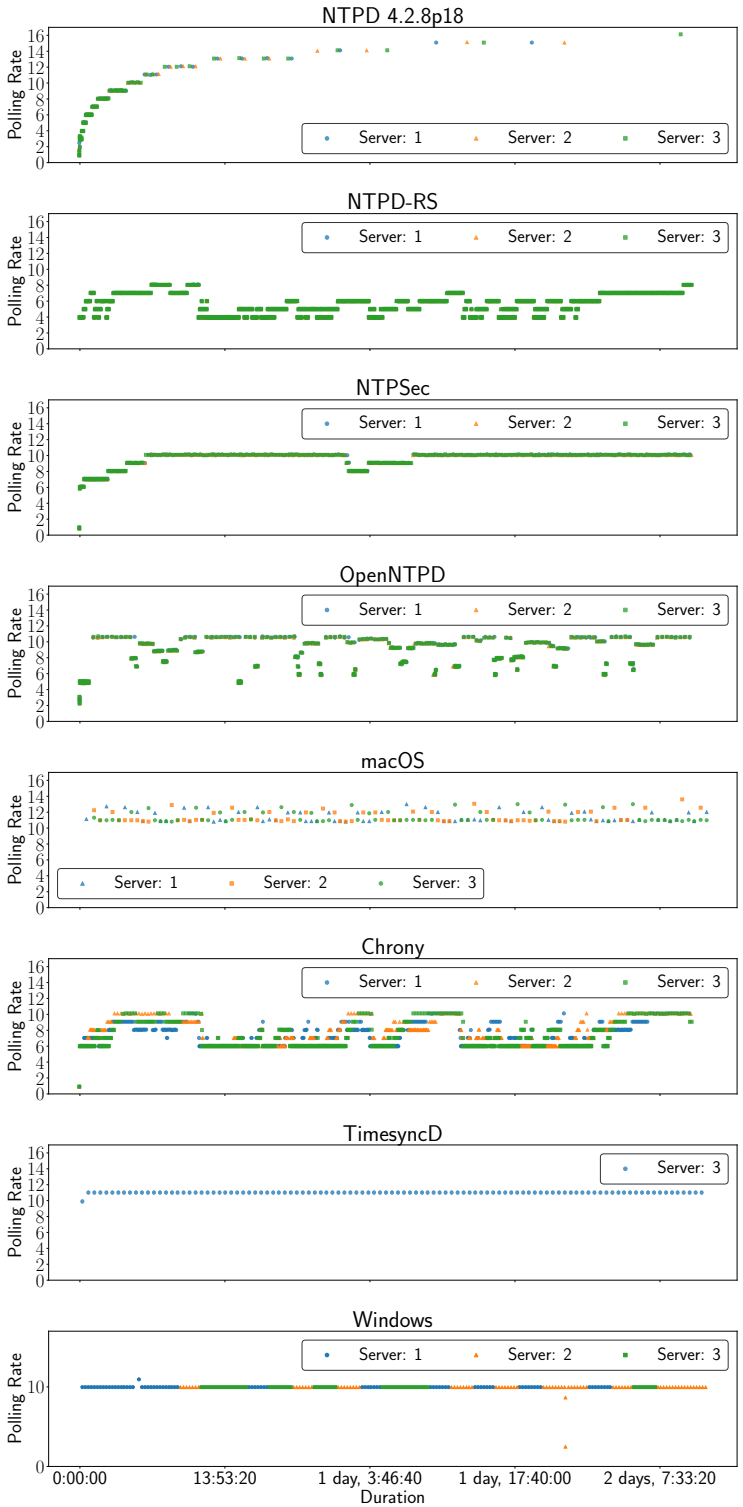


Fig. 1: NTP queries per client. Polling Rate ($n = \log_2 T$) computed from consecutive queries to the same server.

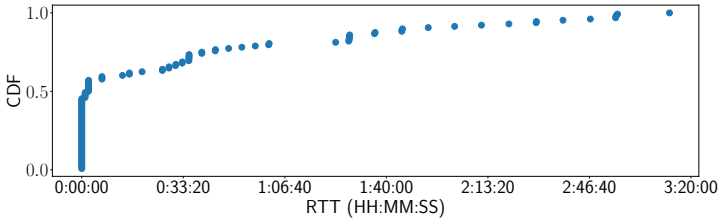


Fig. 2: NTPD-RS time series from Japan.

relies on one server at a time—`timesyncd` always, `W32Time` between switches—adopts that server's time, so a single malicious or faulty server suffices.

Answer to RQ1: Hourly query volume differs by more than two orders of magnitude across clients, and clients range from querying one server to querying all servers concurrently. The OS defaults combine the lowest query volume with the least redundancy.

Windows versions: besides Windows server, we cover in **??** how Windows home behave, and how Windows Pro/Server Client behaves if part of a active directory domain-controller (it uses many-cast mode [39]).

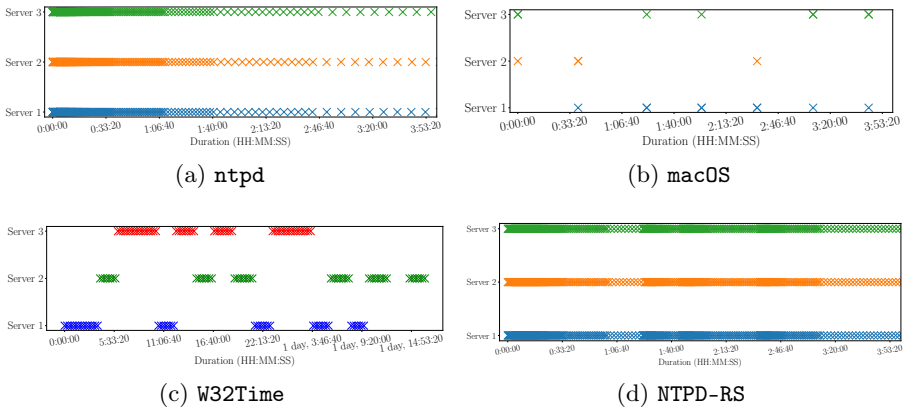
3.4 Ubuntu switch to `Chrony`

Ubuntu replaced `timesyncd` with `Chrony` and NTS as its default in October 2025 [34]. `Chrony` queries more often than `timesyncd`, and NTS extension fields enlarge each packet [19]; we therefore expected increased load on Ubuntu's time servers. In September 2025, before the change, we raised this with Ubuntu's developers. To quantify the increase, we reproduced Ubuntu's planned configuration on two VMs in different data centers (AMS and SYD), captured 60 h of traffic, and repeated the measurement with `timesyncd`.

	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)

Table 3 shows the results: switching to `Chrony` with NTS increases queries per hour 11× and bytes per hour 25×. The byte ratio exceeds the query ratio because NTS adds extension fields to every packet. Robustness thus has a cost: the client that resists the attacks in §4 sends an order of magnitude more traffic. We shared these results with Ubuntu's operators to inform the provisioning of their time service; **??** details the experiments



(a) `ntpd` (b) `macOS`

(c) `W32Time` (d) `NTPD-RS`

Fig. 3: Distinct server utilization patterns.

4 Time-Shift Attacks

A time source influences two client decisions: whether to adopt the time it serves and whether to continue querying it. This section evaluates the first; §5 evaluates the second.

In a time-shift attack, the adversary serves forged timestamps to shift a client's clock. A faulty server has the same effect; the USNO and Telstra incidents (§1) required no adversary [52,31,35]. We first assume that the adversary controls all of a client's servers (§4.1) and then a subset of them (§4.2).

Expected client behavior: NTPv4 clients must reject offsets above the 1,000 s panic threshold and should combine the responses of several servers before adjusting the clock [40] (§2). SNTP requires neither safeguard [39], which may explain the failures of Windows hosts in the USNO incident. SNTP does not, however, preclude bounding accepted offsets; `W32Time` enforces such a bound (§4.1).

4.1 All servers malicious

Threat model: The adversary aims to shift the target's clock. The adversary operates NTP servers that receive the client's queries, through redirection or interception, and answers them with forged timestamps. Because it observes the queries, it can echo their origin timestamps, as a legitimate server does. Off-path injection without observing the queries is out of scope; Malhotra et al. studied it for `ntpd` [31]. The model also covers a faulty server, as in the USNO and Telstra incidents.

Is this threat model realistic? For the OS defaults, controlling all of a client's servers requires controlling a single name or address. `timesyncd` queries one server (§3), and `macOS` and `W32Time` each resolve one vendor hostname; a single

poisoned DNS response, hijacked prefix, or faulty server therefore reaches every server the client uses. For clients of the NTP Pool, including `timesyncd` on Debian, Beverly and Rye show that ten or fewer servers suffice to capture half of the Pool's traffic in about 90% of countries, and that clients continue to query evicted servers for weeks to months [4]. §4.2 relaxes the assumption to a subset of malicious servers, and §4.1 validates the attack end to end against `macOS`.

Prior work has demonstrated two vectors for redirecting NTP clients to attacker-controlled servers and proposed a third. First, off-path DNS cache poisoning can redirect clients to malicious NTP servers [26] (e.g., `pool.ntp.org` → 192.168.1.1). Second, an attacker can join the NTP Pool [54], one of the largest public time services [46], and serve manipulated timestamps while evading the Pool's monitoring [28]. Third, an attacker can hijack the BGP prefixes of legitimate NTP servers and intercept their clients [24,26].

Countermeasures: DNSSEC [3] prevents cache poisoning only if the domain is signed and the resolver validates; fewer than 20% of resolvers validate [49]. RPKI [8] lets routers reject hijacked BGP announcements. Few large time-service providers deploy DNSSEC, RPKI, or NTS [23]. For the NTP Pool, detecting malicious servers requires hidden monitors [28]. NTS [19] prevents redirection to an impostor, but none of the three OS default clients enables it, and it does not protect against a faulty server or a malicious server with valid credentials. The client's safeguards are thus the remaining defense; we evaluate them.

Experiments: Because these redirection techniques are documented and executing them against production infrastructure would affect third parties, each experiment starts with the client already querying the adversary's servers. This isolates the question we study: how the client handles incorrect time. §4.1 validates one complete attack path in a network under our control.

All clients use a domain name that resolves to three NTP servers under our control. Each client first synchronizes correctly (*hot start*). All three servers then answer with incorrect time. We run six experiments per client, with offsets from 900 s (below the panic threshold [40]) to two years, in the setup of §3. Each experiment lasts at least 8 h; we record the client's system clock against the host's reference clock. A client is *vulnerable* to an offset if its clock adopts the adversary's time, by stepping or slewing, within the experiment.

Results: Table 4 summarizes the results. No NTP client adopted the adversary's time, for any offset.

All three SNTP clients adopted it. They comply with the SNTP specification, but they are the *default* clients of the three major OSes (Table 1).

`macOS` and `timesyncd`, which several other Linux distributions also use, adopted every offset. `W32Time` adopted no offset larger than two weeks, consistent with its documented two-week limit [38].

A panic threshold bounds each adjustment, not their sum: any client, including the NTP clients, may remain vulnerable to time-skimming, which shifts the

clock in steps below the threshold [31]. We did not evaluate time-skimming; the two-week bound of `W32Time` makes it a practical target.

Time to compromise: The attacks succeeded after 13 min against `timesyncd` and 2 h against `macOS` (8 and 13 queries, respectively; Figure 4a).

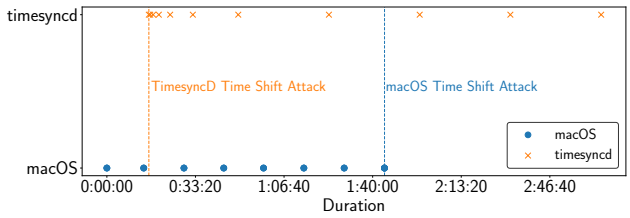
Against `W32Time`, a one-week offset succeeded after 36 h and 136 queries (Figure 4b).

Client	*Offset*					
	900s	**1 month**	**2 months**	**3 months**	**1 year**	**2 years**
`macOS`	✓	✓	✓	✓	✓	✓
`W32Time`	✓					
`timesyncd`	✓	✓	✓	✓	✓	✓
`NTPSec`						
`NTPD-RS`						
`ntpd`						
`OpenNTPd`						
`Chrony`						

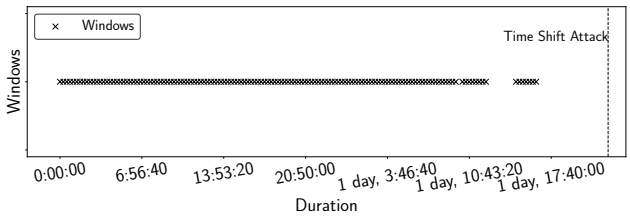
Table 4: Client behavior to time shift attacks. (✓) shows vulnerable clients. (M = month, Y = Year).

End-to-end validation against `macOS`: To show that the attack does not require reconfiguring the client, we attacked an unmodified `macOS` client in a network under our control. DNS cache poisoning failed: `macOS` resolves some names over DNS-over-HTTPS (DoH) and applies further heuristics. We therefore acted as the client's first-hop router, forwarded its NTP queries to our server, and answered with Apple's server address as the source. The client accepted the responses; we shifted its clock by up to 30 years. An adversary in this position, such as a compromised home router, a rogue Wi-Fi access point, or a malicious network operator, requires no access to the device.

Disclosure: We followed coordinated vulnerability disclosure (CVD) guidelines [44]. We disclosed the attack to Apple in March 2025; Apple tracked it as issue OE1100520849262. Apple initially responded that changing the configured time source is not a realistic attack vector; after we reported the end-to-end attack, it deemed an attacker-controlled router outside the scope of `macOS` security and did not fix the issue. We notified the `timesyncd` (`systemd`) and Debian [15] developers on 2025-09-25; Debian is particularly exposed because it combines `timesyncd` with the NTP Pool. The `timesyncd` developers replied that NTS support is in progress [64]; NTS would prevent redirection but not incorrect time from a server with valid credentials. Debian has not responded.



(a) timesyncd and macOS (2 years offset)



(b) Windows (1 week offset)

Fig. 4: Queries sent during the time-shift attack; each marker is one query, and the dashed lines mark when the client's clock adopted the adversary's time. `timesyncd` adopted a two-year offset after 13 min (8 queries) and `macOS` after 2 h (13 queries); `W32Time` adopted a one-week offset after 36 h (136 queries).

Other findings: `NTPD-RS` crashes about 30 min into the experiment, when its computed offset exceeds its panic threshold. We reported this to the developers, who have not responded.

Answer to RQ2 (all servers malicious): The default clients of the three major OSes adopt incorrect time from a malicious or faulty source within minutes to hours; all five NTP clients reject it. Ubuntu's switch to `Chrony` in October 2025 removes this exposure for Ubuntu; other `systemd`-based distributions that retain `timesyncd` remain exposed. Ubuntu has since announced plans to replace `Chrony` with `NTPD-RS` as its default client and server [10].

4.2 A subset of servers malicious

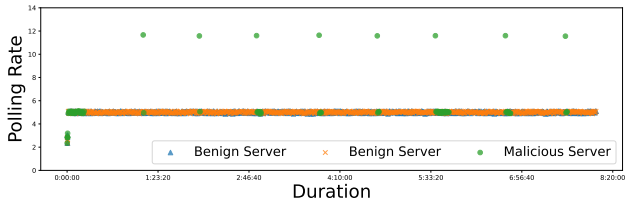
§4.1 assumed that the adversary controls all of a client's servers. We now configure three servers per client, of which one (first scenario) or two (second scenario) are malicious. We test the NTP clients. The question does not apply to `timesyncd`, which queries one server (§3). `macOS` and `W32Time` query several servers; we did not test how they combine honest and malicious responses. Each client first synchronizes; after 10 min, the malicious servers serve a one-week offset. Each run lasts 8 h.

No NTP client adopted the adversary's time in either scenario: with at least one honest server, every client rejects the one-week offset. The clients differ only

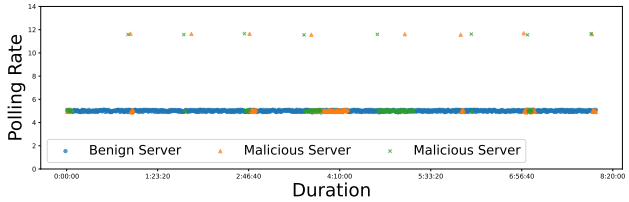
in how they treat the malicious servers. All except `OpenNTPd` continue to poll them as in §4.1.

`OpenNTPd` polls the honest servers at rate 5 and backs the malicious servers off to rate 12 (about once per hour) without dropping them (Figure 5).

Answer to RQ2 (subset of servers malicious): With one honest server out of three, no NTP client adopted the adversary's time.



(a) 1 malicious server (Server 3)



(b) 2 malicious servers (Server 3 benign)

Fig. 5: Timeseries of polling rate for `OpenNTPd` clients under the time-shift attack with one and two malicious servers.

5 Kiss-o'-Death Attacks

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

Table 5: Change in query rate after a server starts answering with KoD packets, relative to normal operation. No client stops querying after DENY, as RFC 5905 requires; `ntpd` and `NTPD-RS` query *more*. Table 6 and Table 7 give hourly rates.

This section evaluates the second client decision: whether to continue querying a time source. Kiss-o'-Death (KoD) packets are NTP's only server-to-client control messages: a RATE packet asks the client to reduce its query rate, and a DENY packet asks it to stop using the server. Spoofed KoD packets can stop clients from synchronizing, causing clock drift [31,57]; a client that mishandles them may instead flood the server.

Expected behavior: RFC 8633 (§5.4) [57] recommends, without mandating, that a client receiving a RATE packet increase its poll interval, up to a poll rate of 13, and notes that many clients ignore such packets. RFC 5905 states that a client should stop using a server after receiving a DENY packet (§7.4 [40]).

Threat model: We ask whether a time source can stop a client from synchronizing by sending KoD packets. We use the threat model of §4.1: after a hot start, all three servers answer every query with a KoD packet instead of a timestamp.

Experiments: We run one experiment with RATE packets and one with DENY packets. We test only the NTP clients; SNTP clients are not required to handle KoD [39].

Table 5 summarizes the results. Under RATE, all clients reduced their query rate (Figure 8). `ntpd` stopped querying after 3.3 h.

Under DENY, no client stopped querying, contrary to RFC 5905. All except `ntpd` and `NTPD-RS` reduced their query rate; `ntpd` polled at a fixed interval of 8 s (Figure 8a).

Figure 6 and Figure 7 show the poll rate of every client under RATE and DENY, respectively; the dashed red line marks the first KoD response after the hot start. Compared with normal operation (Figure 1), `ntpd` polls at a fixed rate under both RATE and DENY, whereas `Chrony` reduces its query rate under both. The `Chrony` developers confirmed that `Chrony` honors KoD and temporarily reduces its query rate [43].

Table 6 and Table 7 list queries per hour in normal operation and under RATE and DENY, respectively, and their ratio (Amp.). Under RATE, only `NTPSec` largely ignores KoD packets; the other clients reduce their query rate. Under DENY, `ntpd` and `NTPD-RS` increase their query rate; the ratio for `NTPD-RS` exceeds 10,000 because of the bug described below. Ref. is the 60-h mean of Table 2, which includes each client's steady-state back-off, whereas each KoD run lasts about 8 h; the ratios therefore overstate increases relative to the same period of normal operation, which Table 5 and Figure 8 use.

`NTPD-RS` bug: Instead of ignoring DENY packets or ceasing to query the server, `NTPD-RS` sent 541,613 queries in five minutes (1,800 queries/s). We stopped the experiment after five minutes; Figure 8b shows this interval, and Figure 9 the intervals between consecutive queries. We reported the bug to the `NTPD-RS` developers [62], who have fixed it.

The `ntpd` developers acknowledged its DENY behavior: `ntpd` "tests for the incoming RATE code" only, a deliberate choice that avoids clock drift by never ceasing to query a server.

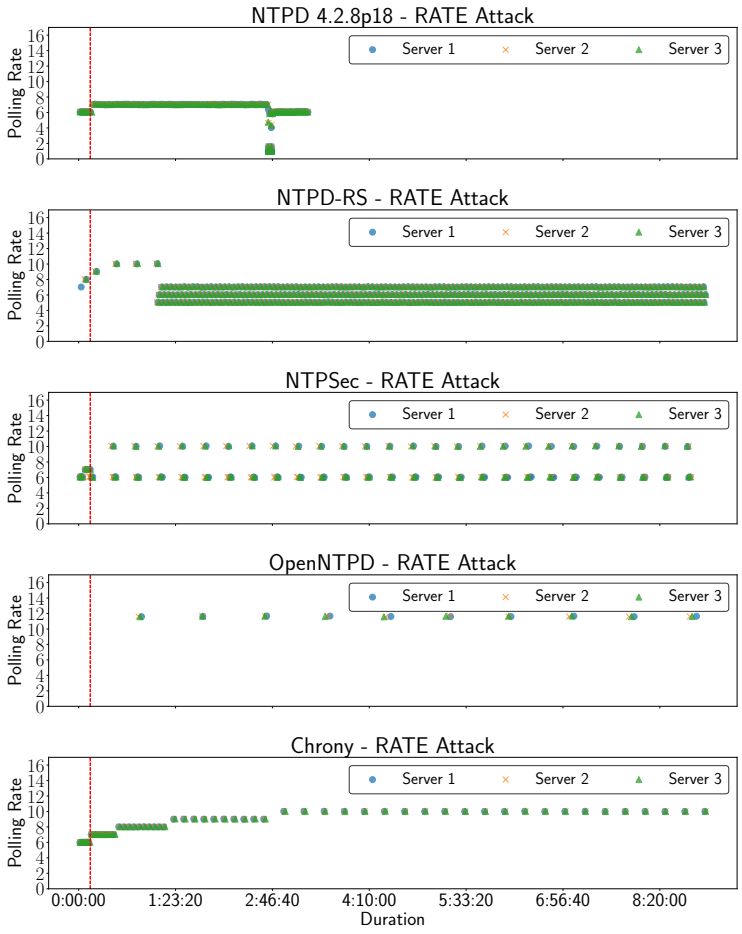


Fig. 6: Poll rate per server under KoD RATE; the dashed red line marks the first KoD response. `ntpd` stops querying after 3.3 h.

	Ref.	RATE	Amp.
NTPSec	16.82	27.83	1.65
NTPD-RS	250.8	136.7	0.54
NTPD	12.32	112.5	9.13
OpenNTPD	22.68	3.48	0.15
Chrony	80.37	12.72	0.15

Table 6: Queries per hour in normal operation (Ref.) and under KoD RATE, and their ratio.

	Ref.	DENY	Amp.
NTPSec	16.82	35.25	2.09
NTPD-RS	250.8	2691993.8	10733
NTPD	12.32	1350.1	109.5
OpenNTPD	22.68	4.33	0.19
Chrony	80.37	33.11	0.41

Table 7: Queries per hour in normal operation (Ref.) and under KoD DENY, and their ratio.

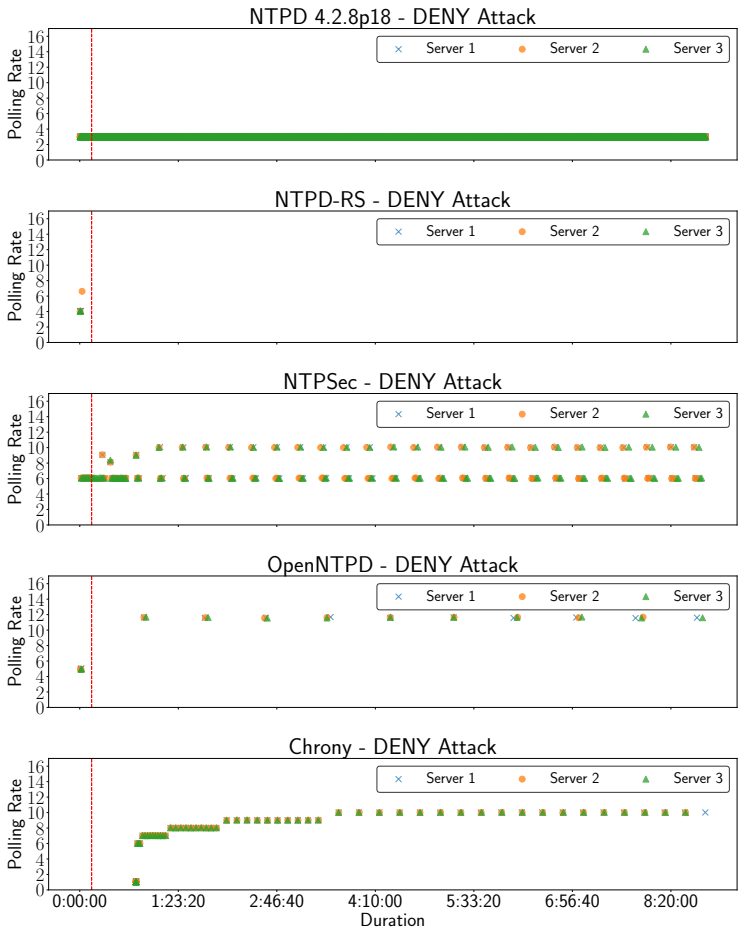
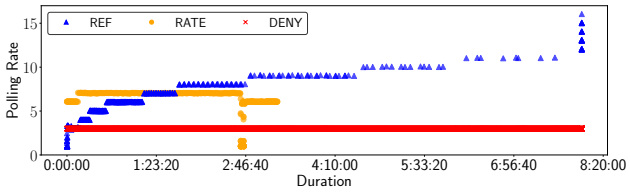
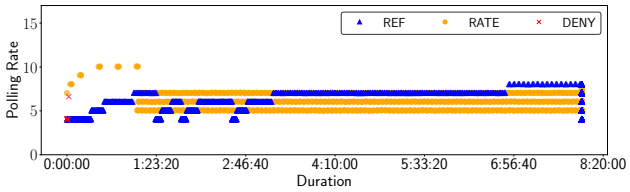


Fig. 7: Poll rate per server under KoD DENY; the dashed red line marks the first KoD response. No client stops querying: `ntpd` polls every 8 s, and `NTPD-RS` floods the servers (run stopped after 5 min).



(a) `ntpd`



(b) `NTPD-RS`

Fig. 8: Poll rate under KoD RATE and DENY, compared with normal operation (REF). Under DENY, `ntpd` (a) polls every 8 s (rate 3) throughout the run instead of stopping; `NTPD-RS` (b) enters a tight query loop, so we stopped the DENY run after five minutes.

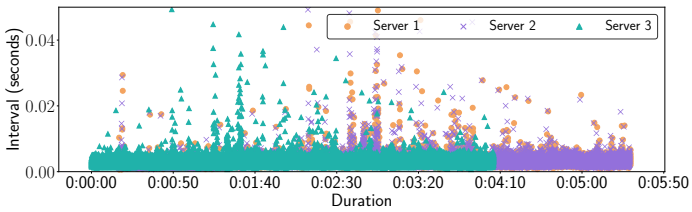


Fig. 9: Intervals between consecutive `NTPD-RS` queries in the first minutes of the KoD DENY experiment; most queries follow the previous one within milliseconds (1,800 queries/s).

Answer to RQ3 (KoD): No NTP client conforms to RFC 5905 for DENY: none stops using the server, and two increase their query rate, one by four orders of magnitude.

6 Era Rollover

NTP timestamps have 64 bits: 32 for seconds and 32 for fractions of a second. The seconds field wraps every 2^{32} s, about 136 years; each such period is an *era*. Era 0 began at 00:00:00 UTC on 1 January 1900, and era 1 begins on 7 February 2036. A client that does not account for the wrap interprets timestamps from era 1 as dates in 1900. RFC 5905 therefore requires the era to be "derived from external means, such as the file system or dedicated hardware" [40].

We test whether each client handles the rollover. In the setup of §3, we set the client's clock to one hour before the end of era 0, let it synchronize across the rollover, and monitor its clock.

Table 8 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 §4.1). We notified the `OpenNTPd` developers in May 2025 and the issue was fixed in May 2026 [63].

Client	Clock set before rollover	Era-aware
`ntpd`	Yes	Yes
`NTPD-RS`	Yes	Yes
`NTPSec`	Yes	Yes
`OpenNTPd`	Yes	No
`timesyncd`	Yes	Yes
`Chrony`	Yes	Yes
`macOS`	No	n/t
`W32Time`	No	n/t

Table 8: Era rollover. *Clock set before rollover*: whether we could bring the client's clock to the end of era 0. Of the six clients we could test, only `OpenNTPd` (as packaged in Ubuntu) is not era-aware. n/t: not testable.

7 Discussion and recommendations

Is a malicious server a client vulnerability? One may object that a client cannot be faulted for trusting its server, since NTP cannot distinguish a malicious server from an honest one. This objection does not hold, for three reasons. First, the standards anticipate incorrect time: RFC 5905 specifies the panic threshold and multi-server cross-validation because servers can be faulty or malicious [40]. Second, client design determines the outcome. All five NTP clients rejected the

forged responses, and `W32Time`, an SNTP client, rejected offsets beyond two weeks; only `macOS` and `timesyncd` adopted every offset (Table 4). Third, bounding offsets is inexpensive: `W32Time` does so within SNTP, and `OpenNTPd` validates NTP time against the dates reported by HTTPS servers [69]. No client can prevent every shift, since an adversary can skim the clock below any threshold, but a client can bound the damage; two of the three OS defaults do not.

Classification of findings. Table 9 lists eleven findings: nine client issues and two operational findings. We classify each as *spec-permitted* (the client complies with a specification that permits unsafe behavior), a *deviation* from the specification, or an *implementation bug*. The time-shift findings are spec-permitted, because SNTP does not require a panic threshold. Fixing them requires replacing or hardening the client, not patching a bug. The KoD findings are deviations; the crash and the query flood are bugs. `OpenNTPd`, however, has fixed their era bug [63] one year after our initial contact.

#	Software	Finding	Type	Vendor response
1	`NTPD-RS`	Queries surge to nearby servers (§3.3)	Design choice	Intended behavior
2	Ubuntu	25× traffic growth with `Chrony` and NTS (§3.4)	Operational	Acknowledged
3	`macOS`	Adopts any time shift (§4.1)	Spec-permitted	"Not a bug"
4	`timesyncd`	Adopts any time shift (§4.1)	Spec-permitted	NTS support in progress
5	`W32Time`	Adopts time shifts up to two weeks (§4.1)	Spec-permitted	Acknowledged, not a bug
6	Debian Linux	`timesyncd` with the NTP Pool by default (§4.1)	Spec-permitted	No response
7	`NTPD-RS`	Crashes during time-shift attacks (§4.1)	Bug	No response
8	`NTPD-RS`	Floods servers after KoD DENY (§5)	Bug	Fixed [62]
9	`ntpd`	Stops querying after repeated KoD RATE (§5)	Deviation	Acknowledged
10	`ntpd`	Keeps querying every 8 s after KoD DENY (§5)	Deviation	Acknowledged
11	`OpenNTPd`	Not era-aware in Ubuntu/Debian package (§6)	Bug (outdated)	Fixed [63]

Table 9: Findings, their type, and vendor responses as of November 2025. Rows 1–2: operational findings; rows 3–11: client issues. One issue has been fixed in a deployed release.

Cost of robustness. The NTP clients that resisted every attack send 7 to 142 times more queries than `timesyncd` (Table 2), and `Chrony` with NTS sends 25 times more bytes (Table 3). Per host, the overhead is small (`Chrony` with NTS sends about 5 KB per hour), but providers serving billions of devices must provision for it, as Ubuntu did. We did not measure CPU or memory overhead.

7.1 Recommendations

We make the following recommendations:

1. Apple and Microsoft: deploy NTP clients. `macOS` and Windows ship SNTP clients, although RFC 5905 intends SNTP for "clients with a single upstream server and no dependent clients" [40]. Both vendors should replace them with NTP clients that enforce the panic threshold and cross-validate multiple servers. Until then, `macOS` should bound accepted offsets, as `W32Time` already does.

2. Linux distributions: replace `timesyncd`. Distributions that ship `timesyncd` should replace it with an NTP client such as `Chrony` or `NTPD-RS`, as Ubuntu did in October 2025. This applies to server, desktop, and container distributions, including Flatcar Linux [18]. Debian is particularly exposed, because it combines `timesyncd` with the NTP Pool (§4.1).

3. OS vendors and time-service providers: enable NTS. NTS prevents redirection to impostor servers (§2). OS vendors should enable it by default, as Ubuntu has, and time-service providers should support it. NTS does not protect against a faulty server or a malicious server with valid credentials; it complements client-side safeguards rather than replacing them.

4. SNTP implementers: reject implausible offsets. SNTP does not preclude bounding accepted offsets (§4.1). Implementations should enforce such a bound, for instance by validating server time against the dates reported by trusted HTTPS servers, as `OpenNTPd` does [69].

Limitations. Our testbed has near-zero latency and no loss (§3.2); as the `NTPD-RS` experiment with a server in Japan shows, query volumes on the Internet may differ. We evaluated one version of each client, in virtual machines. We did not evaluate time-skimming, off-path injection, or how `macOS` and `W32Time` combine honest and malicious responses.

8 Ethics

All experiments ran on machines under our control: a local machine and cloud VMs rented in Amsterdam, Sydney, and Stockholm. All NTP servers queried were our own, except a public server in Japan that `NTPD-RS` queried at its normal rate (§3.3). No third-party users or production time infrastructure were affected. We did not execute DNS cache poisoning, NTP Pool injection, or BGP hijacking against production infrastructure; the end-to-end attack against `macOS` ran in a network under our control (§4.1).

The principal ethical consideration is the disclosure of the vulnerabilities. We followed coordinated vulnerability disclosure (CVD) [44] for every issue in Table 9 and gave all affected vendors and operators time to respond before publication. At the time of writing, one issue has been fixed; the others are pending or were not classified as vulnerabilities by the vendor. Apple, for example, declined to classify the time-shift attack as a vulnerability, stating 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".

We consider publication warranted: the affected clients run on billions of devices, and users and operators need this information to assess their exposure.

9 Related Work

NTP client security. The work closest to ours is by Mahlous [30], who ran time-shift attacks against four NTP clients (`Chrony`, `ntpd`, `OpenNTPd`, and `NTPSec`). Their attacker is a man-in-the-middle, while our threat model assumes the client is already being served by a malicious server (§4.1). They shifted clocks by multiples of 1000 s; we shifted them by years. We also go further in scope: we evaluate eight clients, including the defaults of the three most popular OSes, both under normal operation and under attack. Malhotra et al. [31] showed that unauthenticated NTP traffic enables time-shift and DoS attacks, and that clients are especially vulnerable during initialization, when spoofed or fragmented packets can manipulate them. They evaluated only `ntpd`; we cover eight NTP/SNTP clients.

Prior studies show that clients are susceptible to malicious time servers [16,20] and to third-party attacks [31,33,6], including attacks on broadcast mode [32,70]. Off-path attackers can also redirect clients through DNS cache poisoning or BGP hijacking [26,31]. NTS [19] authenticates the server, which defeats on-path tampering and redirection to an impostor, but not a faulty server or a malicious one with valid credentials; moreover, the NTP Pool and the three OS default clients do not use it.

Time server infrastructure. Several studies look at the server side instead of the clients; they complement our work. Beyond its use in DDoS amplification [14], the NTP Pool has been studied extensively [59,53,46,28], including its architecture and client-to-server mapping [46]. Beverly and Rye [4] measured the Pool over nine months and found that only about a fifth of its active servers are fully independent, and that a small number of cloud providers host much of it. They show that an attacker with ten or fewer servers could capture half of the Pool's traffic in most countries, and that clients keep querying evicted servers for weeks to months. That persistence is a client-side behavior, and it echoes the server-selection differences we observe in §3. Huppert et al. [23] characterized seven large time service providers that are the defaults for most OSes, including Microsoft, Apple, Google, and Ubuntu. They found wide variation in how these providers replicate their service and map clients to servers, poor accuracy and single-address assignments at Microsoft, and limited deployment of NTS, DNSSEC, and RPKI. Rytilahti et al. [59] had covered several of the same providers earlier. Other studies map NTP servers across the whole IPv4 space [22,59,42,41] or measure the accuracy of stratum-1 servers [11]. These studies share a premise: the servers decide what time is offered. We study the other half, namely how the clients those providers serve decide whether to accept it. Our findings also matter to providers themselves: Ubuntu's switch from `timesyncd` to `Chrony` [34] changes the load on its servers, which we quantify in §3.4.

10 Conclusion

Research on the security of network time has focused on the protocol and its servers: authenticating servers, hardening the NTP Pool, and detecting malicious time sources. Our results show that the decisive link is the client's acceptance policy. The default clients of macOS, Windows, and many Linux distributions adopt multi-year offsets from the servers they query, within minutes to hours. NTP clients that already exist reject the same responses, even when only one of three servers is honest, at a traffic cost that is negligible per host. The exposure of billions of devices is therefore primarily a deployment problem: the defenses exist but are not shipped.

Two further lessons follow. First, server authentication and client-side validation are complementary. NTS authenticates the channel, not the time: it prevents redirection to an impostor, but a client without a panic threshold still adopts incorrect time from a faulty or compromised server with valid credentials; NTS would not have prevented the USNO or Telstra incidents. Second, the specifications permit the unsafe behavior. The time-shift findings are spec-permitted, because SNTP requires no offset bound, and NTPv5 leaves client algorithms out of scope. Minimum client safeguards—an offset bound and validation against more than one source—should be specified normatively, so that new implementations do not repeat SNTP's omissions.

Future work. Four directions remain open. (i) Time-skimming: a panic threshold bounds each adjustment, not their sum; how far an adversary can move a bounded client such as `W32Time` over time is unmeasured. (ii) Clients beyond desktop and server OSes: mobile, embedded, and IoT devices often rely on SNTP clients, which we did not evaluate. (iii) Exposure in the wild: server-side traces could estimate which clients, and how many, query public time services, and thus how many hosts each finding affects. (iv) Continuous testing: client behavior changes across releases; an open, automated test suite that runs each release against faulty and malicious servers would detect regressions and make our evaluation reproducible.

References

1. Apple Inc.: Apple reports first quarter results. https://www.apple.com/newsroom/2024/02/apple-reports-first-quarter-results/ (2024), accessed: 2025-02-18
2. Arch Linux: systemd-timesyncd (2024), https://wiki.archlinux.org/title/Systemd-timesyncd, accessed: 2025-05-15
3. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS Security Introduction and Requirements. RFC 4033, IETF (Mar 2005), http://tools.ietf.org/rfc/rfc4033.txt
4. Beverly, R., Rye, E.: On borrowed time: Measurement-informed understanding of the NTP pool's robustness to monopoly attacks. In: Proceedings of the Network and Distributed System Security (NDSS) Symposium 2026. Internet Society, San Diego, CA, USA (Feb 2026). https://doi.org/10.14722/ndss.2026.240541, https://dx.doi.org/10.14722/ndss.2026.240541
5. Bicknell, L.: NTP Issues Today. https://mailman.nanog.org/pipermail/nanog/2012-November/053449.html (2012)
6. Bishop, M.: A security analysis of the ntp protocol version 2. In: [1990] Proceedings of the Sixth Annual Computer Security Applications Conference. pp. 20–29 (1990). https://doi.org/10.1109/CSAC.1990.143746
7. Bundesanstalt, P.T.: FDCF77 - PTB.de (Nov 5 2022), https://www.ptb.de/cms/en/ptb/fachabteilungen/abt4/fb-44/ag-442/dissemination-of-legal-time/dcf77.html
8. Bush, R., Austein, R.: The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1. RFC 8210, IETF (Sep 2017), http://tools.ietf.org/rfc/rfc8210.txt
9. Bush, R., Austein, R.: The Resource Public Key Infrastructure (RPKI) to Router Protocol. RFC 6810, IETF (Jan 2013), http://tools.ietf.org/rfc/rfc6810.txt
10. Canonical: ntpd-rs: it's about time! Ubuntu Community Hub (Mar 2026), https://discourse.ubuntu.com/t/ntpd-rs-its-about-time/79154, accessed 2026-06-02
11. Cao, Y., Veitch, D.: Toward trusted time: Remote server vetting and the misfiring heart of internet timing. IEEE/ACM Transactions on Networking **28**(2), 944–956 (2020). https://doi.org/10.1109/TNET.2020.2977024
12. Chrony Project: chrony.conf - chronyd configuration file (2025), https://chrony-project.org/doc/3.4/chrony.conf.html, accessed: 2025-05-13
13. Chrony Project: Faq: Can `chronyd` be driven from broadcast/multicast ntp servers? https://chrony-project.org/faq#_can_chronyd_be_driven_from_broadcastmulticast_ntp_servers (2025), accessed: 2025-11-21
14. Czyz, J., Kallitsis, M., Gharaibeh, M., Papadopoulos, C., Bailey, M., Karir, M.: Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks. In: Proceedings of the 2014 ACM Conference on Internet Measurement Conference. pp. 435–448. IMC, ACM (Nov 2014). https://doi.org/10.1145/2663716.2663717, http://doi.acm.org/10.1145/2663716.2663717
15. Debian Project: Debian – The Universal Operating System (2025), https://www.debian.org/, accessed: 2025-09-25
16. Deutsch, O., Schiff, N.R., Dolev, D., Schapira, M.: Preventing (network) time travel with chronos. In: Network and Distributed Systems Security (NDSS) Symposium 2018. San Diego, CA, USA (February 18-21 2018). https://doi.org/10.14722/ndss.2018.23231, https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02A-2_Deutsch_paper.pdf

17. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF (Aug 2008), http://tools.ietf.org/rfc/rfc5246.txt
18. Flatcar: Flatcar container linux, https://www.flatcar.org/, accessed: 2025-02-14
19. Franke, D., Sibold, D., Teichel, K., Dansarie, M., Sundblad, R.: Network Time Security for the Network Time Protocol. RFC 8915, IETF (Sep 2020), http://tools.ietf.org/rfc/rfc8915.txt
20. Golf, T.: Manipulating time through (s)ntp (2024), https://tweedegolf.nl/en/blog/142/manipulating-time-through-sntp, accessed: 2025-11-18
21. Henning Brauer: Openntpd (2022), https://www.openntpd.org, accessed: 2025-05-13
22. Huang, Z., Wang, S., Chen, L., Li, D., Liu, Y.: Measuring the time source vulnerabilities in the ntp ecosystem. In: Proceedings of the 2025 ACM Internet Measurement Conference. p. 493–510. IMC '25, Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3730567.3764478, https://doi.org/10.1145/3730567.3764478
23. Huppert, P., Moura, G.C.M., Davids, M., de Boer, P.T., Holz, R., Fernhout, R., Smaragdakis, G., Hesselman, C.: Bigtime – characterizing large time service providers. Proc. ACM Netw. **4**(CoNEXT4) (Dec 2026). https://doi.org/10.1145/3856960
24. Huston, G., Rossi, M., Armitage, G.: Securing bgp — a literature survey. IEEE Communications Surveys & Tutorials **13**(2), 199–222 (2011). https://doi.org/10.1109/SURV.2011.041010.00041
25. IEEE: Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008) pp. 1–499 (2020). https://doi.org/10.1109/IEEESTD.2020.9120376
26. Jeitner, P., Shulman, H., Waidner, M.: The Impact of DNS Insecurity on Time. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (2020)
27. Konjerla, S.: Are ntp clients always right? (2025), https://ripe91.ripe.net/programme/meeting-plan/sessions/17/RNBMHH/, presented at RIPE 91: Tuesday, 21 October 2025, 11:30 (UTC+03:00), Main Room. Accessed: 2025-11-12
28. Kwon, J., Song, J., Hur, J., Perrig, A.: Did the Shark Eat the Watchdog in the NTP Pool? Deceiving the NTP Pool's Monitoring System. In: 30th USENIX Security Symposium (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/kwon
29. Lichvar, M., Mizrahi, T.: Network Time Protocol Version 5. Internet-Draft draft-ietf-ntp-ntpv5-10, Internet Engineering Task Force (Oct 2026), https://datatracker.ietf.org/doc/draft-ietf-ntp-ntpv5/10/, work in Progress
30. Mahlous, A.R.: Quantitative risk analysis of network time protocol (ntp) spoofing attacks. IEEE Access **12**, 164891–164910 (2024). https://doi.org/10.1109/ACCESS.2024.3493759
31. Malhotra, A., Cohen, I.E., Brakke, E., Goldberg, S.: Attacking the Network Time Protocol. In: Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS 2016) (2016)
32. Malhotra, A., Goldberg, S.: Attacking NTP's Authenticated Broadcast Mode. SIGCOMM Comput. Commun. Rev. **46**(2), 12–17 (may 2016)
33. Malhotra, A., Van Gundy, M., Varia, M., Kennedy, H., Gardner, J., Goldberg, S.: The Security of NTP's Datagram Protocol. In: Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21. pp. 405–423. Springer (2017)

34. Märdian, L.: PSA: Installing chrony by default, to enable Network Time Security (NTS). Ubuntu-devel mailing list (May 2025), https://lists.ubuntu.com/archives/ubuntu-devel/2025-May/043355.html, message posted on May 22, 2025
35. Mark Morowczynski: 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 (Nov 2012)
36. Marouani, H., Dagenais, M.R.: Internal Clock Drift Estimation in Computer Clusters. Journal of Computer Networks and Communications **2008**(1), 583162 (2008). https://doi.org/https://doi.org/10.1155/2008/583162
37. Microsoft: Windows devices by the numbers. https://web.archive.org/web/20220419050729/https://news.microsoft.com/bythenumbers/en/windowsdevices (2022), accessed: 2025-02-18
38. Microsoft Corporation: [ms-sntp]: Network time protocol (ntp) authentication extensions (2024)
39. Mills, D.: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. RFC 4330, IETF (Jan 2006), http://tools.ietf.org/rfc/rfc4330.txt
40. Mills, D., Martin, J., Burbank, J., Kasch, W.: Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, IETF (Jun 2010), http://tools.ietf.org/rfc/rfc5905.txt
41. Mills, D.L., Thyagarjan, A., Huffman, B.C.: Internet timekeeping around the globe (1997), https://www.eecis.udel.edu/~mills/database/papers/survey5.pdf
42. Minar, N.: A survey of the ntp network (1999), https://www.eecis.udel.edu/~mills/database/reports/ntp-survey99-minar.pdf, accessed: 2026-10-08
43. Moura, G.C.M.: Tech report on s/ntp clients behavior (Oct 21 2025), https://community.ntppool.org/t/tech-report-on-s-ntp-clients-behavior/4107/3, post on the NTP Pool Project community forum
44. Moura, G.C.M., Heidemann, J.: Vulnerability disclosure considered stressful. SIGCOMM Comput. Commun. Rev. **53**(2), 2–10 (Apr 2023). https://doi.org/10.1145/3610381.3610383, https://doi.org/10.1145/3610381.3610383
45. Moura, G.C.M., Heidemann, J., de O. Schmidt, R., Hardaker, W.: Cache me if you can: Effects of DNS Time-to-Live. In: Proceedings of the ACM Internet Measurement Conference. pp. 101–115. ACM, Amsterdam, the Netherlands (Oct 2019). https://doi.org/https://doi.org/10.1145/3355369.3355568, https://www.isi.edu/%7ejohnh/PAPERS/Moura19b.html
46. Moura, G.C., Davids, M., Schutijser, C., Hesselman, C., Heidemann, J., Smaragdakis, G.: Deep Dive into NTP Pool's Popularity and Mapping. Proceedings of the ACM on Measurement and Analysis of Computing Systems (SIGMETRICS 2024) **8**(1), 1–30 (2024)
47. Network Time Foundation: Ntpd 4.2.8p-series (2025), https://www.ntp.org/documentation/4.2.8-series/#building-and-installing-ntp, accessed: 2025-05-13
48. Neuman, C., Yu, T., Hartman, S., Raeburn, K.: The Kerberos Network Authentication Service (V5). RFC 4120, IETF (Jul 2005), http://tools.ietf.org/rfc/rfc4120.txt
49. Nosyk, Y., Korczynski, M., Duda, A.: Guardians of DNS Integrity: A Remote Method for Identifying DNSSEC Validators Across the Internet . In: 2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1470–1479. IEEE Computer Society, Los Alamitos, CA, USA (Nov 2023). https://doi.org/10.1109/TrustCom60117.2023.00201, https://doi.ieeecomputersociety.org/10.1109/TrustCom60117.2023.00201

50. ntpd-rs Project: ntp.toml - Configuration File for the ntpd-rs NTP Daemon (2025), https://docs.ntpd-rs.pendulum-project.org/man/ntp.toml.5/, accessed: 2025-05-13
51. NTPsec project: Ntpsec (2022), https://www.ntpsec.org, accessed: 2025-05-13
52. Observatory, U.S.N.: 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/
53. Perry, Y., Rozen-Schiff, N., Schapira, M.: A Devil of a Time: How Vulnerable is NTP to Malicious Timeservers? In: Proceedings of the 28th Network and Distributed System Security Symposium (NDSS 2021) (2021)
54. Pool, N.: pool.ntp.org: the internet cluster of ntp servers (2025), https://www.ntppool.org/en/, accessed: 2025-04-23
55. Project, C.: Chrony: A Versatile Implementation of NTP (2025), https://chrony-project.org/, accessed: 2025-04-23
56. Project, P.: Algorithm documentation (2025), https://github.com/pendulum-project/ntpd-rs/blob/main/docs/algorithm/algorithm.pdf
57. Reilly, D., Stenn, H., Sibold, D.: Network Time Protocol Best Current Practices. RFC 8633, IETF (Jul 2019), http://tools.ietf.org/rfc/rfc8633.txt
58. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF (Aug 2018), http://tools.ietf.org/rfc/rfc8446.txt
59. Rytilahti, T., Tatang, D., Köpper, J., Holz, T.: Masters of Time: An Overview of the NTP Ecosystem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS P). pp. 122–136 (2018). https://doi.org/10.1109/EuroSP.2018.00017
60. Serrano, J., Alvarez, P., Cattin, M., Garcia Cota, E., Lewis, J., Moreira, P., Wlostowski, T., et al.: White Rabbit: A PTP application for robust sub-nanosecond synchronization. In: Proceedings of the IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS). pp. 25–30 (2011). https://doi.org/10.1109/ISPCS.2011.6070148
61. Sherman, J.A., Levine, J.: Usage Analysis of the NIST Internet Time Service. Journal of Research of the National Institute of Standards and Technology **121**, 33 (Mar 2016). https://doi.org/10.6028/jres.121.003, https://doi.org/10.6028/jres.121.003
62. Shreyas Konjerla: Kiss codes not ignored #1867 (2025), https://github.com/pendulum-project/ntpd-rs/issues/1867, accessed: 2025-05-14
63. Shreyas Konjerla: Openntpd not keeps track of the era #77 (2025), https://github.com/openntpd-portable/openntpd-portable/issues/77, accessed: 2025-05-16
64. squell: Nts support for systemd-timesyncd (pull request #39010). https://github.com/systemd/systemd/pull/39010 (2025), accessed: 2025-10-16
65. Taylor, J.: Neglected software update caused widespread telstra network outage, ceo tells senate inquiry (Jul 2026), https://www.theguardian.com/business/2026/jul/17/telstra-missing-software-update-undocumented-design-change-outage, accessed: 2026-10-05
66. Team, C.M.: canonical/multipass: Multipass orchestrates virtual ubuntu instances. https://github.com/canonical/multipass (2025), accessed: 2025-09-02
67. Telstra: What we've learned from the external investigation of our july outage (2026), https://www.telstra.com.au/exchange/what-we-ve-learned-from-the-external-investigation-of-our-july-o, accessed: 2026-10-05

68. Telstra Technical Advisory Panel: Findings for Telstra outage of 8 July 2026 (2026), https://www.telstra.com.au/content/dam/tcom/dynamic-media-projects/luke-campbell/TAP-Findings-for-Telstra-Outage.pdf, accessed: 2026-11-02
69. The OpenBSD Project: ntpd.conf(5) - OpenBSD Manual Pages. OpenBSD (2025), https://man.openbsd.org/ntpd.conf, accessed: 2025-09-19
70. Tripathi, N., Hubballi, N.: Preventing time synchronization in ntp broadcast mode. Computers & Security **102**, 102135 (2021). https://doi.org/https://doi.org/10.1016/j.cose.2020.102135, https://www.sciencedirect.com/science/article/pii/S0167404820304089
71. Turing Software, L.: Utm: Virtual machines for mac (2025), https://mac.getutm.app/, accessed: 2025-08-27