Did the Shark Eat the Watchdog in the NTP Pool? Deceiving the NTP Pool's Monitoring System

Jonghoon Kwon, *ETH Zürich;* Jeonggyu Song and
Junbeom Hur, *Korea University;* Adrian Perrig, *ETH Zürich*

https://www.usenix.org/conference/usenixsecurity23/presentation/kwon

This paper is included in the Proceedings of the
32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the
32nd USENIX Security Symposium
is sponsored by USENIX.

Did the Shark Eat the Watchdog in the NTP Pool? Deceiving the NTP Pool's Monitoring System

Jonghoon Kwon
ETH Zürich

Jeonggyu Song
Korea University

Junbeom Hur
Korea University

Adrian Perrig
ETH Zürich

Abstract

The NTP pool has become a critical infrastructure for modern Internet services and applications. With voluntarily joined thousands of timeservers, it supplies millions of distributed (heterogeneous) systems with time. While numerous efforts have been made to enhance NTP's accuracy, reliability, and security, unfortunately, the NTP pool attracts relatively little attention. In this paper, we provide a comprehensive analysis of NTP pool security, in particular the NTP pool monitoring system, which oversees the correctness and responsiveness of the participating servers. We first investigate strategic attacks that deceive the pool's health-check system to remove legitimate timeservers from the pool. Then, through empirical analysis using monitoring servers and timeservers injected into the pool, we demonstrate the feasibility of our approaches, show their effectiveness, and debate the implications. Finally, we discuss designing a new pool monitoring system to mitigate these attacks.

1 Introduction

Time synchronization across distributed systems is essential in modern Internet services and applications, for instance in the validation of certificates [14, 16]. Accurate time is vital also for network infrastructure and its control and data plane operations, e.g., updating routing tables with a precise clock would enable dynamic congestion control and avoid routing loops [2, 36]. The Network Time Protocol (NTP) is the de facto standard in practice for such time synchronization among interconnected entities.

Yet, NTP is known to be vulnerable to Man-In-The-Middle (MITM) attacks that drop, replay, delay, or alter synchronization packets, resulting in time shifting of the victim entities [1, 18, 33, 39, 40]. Given the vital role of NTP, various approaches for securing NTP have been proposed. Early research focuses on introducing cryptographic primitives to authenticate NTP communication [9, 11, 20, 21, 23]. The common intuition behind the ideas is that authentication prevents packet manipulation and hijacking. Nonetheless, delay attacks and compromised timeservers remain effective. Thus, redesigning NTP received attention to achieve Byzantine robustness even in the presence of adversarial timeservers [8, 32]. While numerous efforts to secure NTP communication have been made, only a limited number of studies have focused on the NTP ecosystem [24, 35].

The NTP Pool Project [30], the biggest NTP ecosystem, bundles thousands of public timeservers into regional or vendor-specific domains, and provides NTP clients across the globe centralized access via the domain name service (DNS). With the NTP pool, NTP clients enjoy reliable and available time sources. Indeed, millions of networked devices, including routers, IoT devices, and Android mobile devices, rely on the NTP pool. Given this critical infrastructure, interesting research questions arise: "*What if a determined attacker takes control over the pool?*", "*What if an attacker manages to remove the majority of legitimate timeservers from the pool while keeping malicious timeservers as only available time sources?*". To answer these questions, we conduct an in-depth analysis of the current NTP pool architecture and fundamental vulnerabilities in its centralized management system.

We explore strategic attack approaches targeting the NTP pool's health-check system [29]. The NTP pool has a monitoring system that inspects the status of timeservers in the pool. It frequently sends NTP challenges to the timeservers and checks their clock accuracy and responsiveness. Timeservers with an incorrect time value are discharged and eventually removed from the pool. Our attacks investigate abuse of the monitoring system: (i) introducing an arbitrary asymmetric delay to the NTP communications between the monitoring server and timeservers, (ii) manipulating the local clock of the monitoring server, and (iii) injecting malicious monitoring servers into the pool. The strategic attacks deceive the pool into believing that the legitimate timeservers in a specific region are out-of-sync, inducing the pool to expel the healthy timeservers. This allows an attacker controlling a small number of timeservers to inflate its influence in the target region and potentially affect a large number of NTP clients.

Through experiments and analysis, we show the feasibility of our attacks, demonstrating how the current pool's single monitoring system (and also the new monitoring system currently being tested with 13 monitoring servers) is vulnerable to attacks. We first propose an adaptive delay attack in which an MITM attacker continuously micro-adjusts asymmetric network delay, and analyze the impact of the adaptive delay attack on the target timeserver's offset and monitoring score. Second, we shift a monitoring server's local clock and discuss its implications. We then analyze the multi-monitoring system's vulnerabilities by injecting monitoring servers into the pool. Finally, we propose mitigation strategies in three aspects: a robust reference clock for the monitoring system, delay attack detection, and a new scoring system.

The main contributions of this paper are the following:

- For the first time, we provide a comprehensive analysis of the NTP pool monitoring system and disclose vulnerabilities caused by a lack of security consideration in its design.
- We introduce strategic attacks exploiting the vulnerabilities of the NTP pool's self-health-checking system and demonstrate their feasibility.
- We present possible mitigation strategies, and discuss the fundamental architecture design for securing the current and future NTP pool monitoring system.

Ethics statement and responsible disclosure. To avoid any potential harm to the existing NTP pool system and its users, the NTP timeservers and monitoring servers that we planted into the pool reacted truthfully except to the queries we generated for testing. That is, the timeservers and monitoring servers recognize the source and destination IP addresses used for the experiment, and distinguish experimental packets. To prevent unexpected collateral damage, we turn the timeserver's status into *monitoring only* (inactive) when conducting an experiment, ensuring none of the NTP clients read a faulty time value. In addition, we carefully chose an attack parameter that is sufficient to confirm its effectiveness while avoiding harm to systems or Internet users. We disclosed our findings to the NTP pool administrator and were requested to post the findings to the NTP pool community channel for further discussion. There is an on-going discussion now at https://community.ntppool.org/c/monitor-operators/13 (monitor operators only).

2 Background: NTP Basics

The concept of NTP was first introduced in 1979 [22], and NTP has been one of the oldest Internet protocols in current use. It is intended to synchronize different computing systems, originally described as a client-server model, over packet-switched, variable-latency networks.

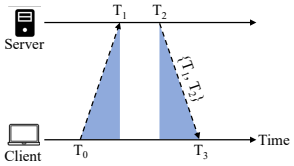

Figure 1: The basic NTP message exchange.

NTP clients. An NTP client synchronizes its time by periodically sending time synchronization requests to NTP timeservers, gathering the servers' local time information to estimate the time offset. Given that the estimated offset could vary depending on the network condition, the client used to query multiple servers—multiple queries to each—prune outliers, and compute the median offset to update its time.

NTP timeservers. An NTP timeserver provides clients with its local time information. For the sake of scalability, NTP timeservers form a hierarchical structure called *strata*. High precision time keeping devices, such as Global Navigation Satellite Systems (GNSS) and atomic clocks, are positioned at the top of the hierarchy. The physical reference clocks are not directly connected to a network, but only provide time information to stratum 1 timeservers that provide the reference clock across the network. The stratum number thus indicates the distance of a timeserver to the reference clocks; a server synchronized to a stratum N server is placed in stratum $N+1$.

Time-offset computation. NTP is a protocol exchanging messages between an NTP client and an NTP timeserver to fetch the server's local time. The client initiates a query to fetch the server's RX and TX timestamps (e.g., T_1 and T_2 in Figure 1). Through the message exchange, the NTP client collects four distinct timestamps, i.e., $T_0 \sim T_3$, and computes the time-offset between the server and client, $\theta = \frac{(T_1-T_0)+(T_2-T_3)}{2}$.

The time-offset computation is based on an intrinsic assumption that the propagation delay is the same in both directions, i.e., $T_0+\theta+\delta/2 = T_1$, where $\delta = (T_3-T_0)-(T_2-T_1)$. In a real system, however, this assumption will (most likely) not hold due to the nature of packet-switched networks, e.g., hot-potato routing [7, 13, 31] and packet buffering [10, 15]. Consequently, the accuracy of the offset computation degrades, e.g., up to the magnitude of 100 milliseconds [25].

NTP Security. There were no security considerations in the early design of NTP. Thus, similar to other early Internet protocols, NTP was vulnerable to MITM attacks where a malicious entity on the communication path intercepts and forges NTP packets. These vulnerabilities bolstered the need for NTP authentication.

Consequently, an authentication method using a pre-shared symmetric key was introduced in NTPv3 [20], adding extension fields for message authentication to NTP packets, including a 32-bit key identifier and 64-bit cryptographic checksum. Nevertheless, the key distribution is considered outside the

scope of NTP. Given that the pre-shared keys require a manual configuration for each client-server pair, it does not scale well. NTPv4 [21] solves this issue using public key cryptography in the Autokey method [12]. Unfortunately, the initial design of Autokey had vulnerabilities to several attacks, such as a brute force attack against the small seed values (32 bits), leading to the evolution to Autokey v2, which became Network Time Security (NTS), published as an IETF RFC [11].

In NTS, an NTS-KE (key exchange) server ensures that the NTP server and client share the same cryptographic algorithms and authenticate them. Through the NTS key establishment protocol, a client retrieves initial cookies, Authenticated Encryption with Associated Data (AEAD) keys, and NTP server addresses. With this information, the client requests a server clock reading. The query is signed with the key and includes a cookie, such that the server can validate the client's query and continue the authenticated time synchronization. However, given that NTS is expensive—adding round-trip latency and server-side public key operations—it has shown only limited adoption in practice. Furthermore, NTP authentication does not protect against network delay attacks.

3 Case Study: NTP Pool Ecosystem

We provide an empirical analysis of the NTP pool ecosystem (§3.1) and present our concerns regarding the security of the NTP pool (§3.2). To better understand the NTP pool ecosystem, we participate in the NTP pool with three monitoring servers and three timeservers located in Europe, North America, and Asia, and observe the operational details of each role for five months (from March to July, 2022).

3.1 NTP Pool Architecture

The NTP pool is a group of public NTP timeservers, enabling NTP clients to synchronize. For usability, reliability, and scalability, the NTP pool system leverages DNS, unifying access to the participating timeservers distributed across the globe; the participating timeservers become part of the *pool.ntp.org* domain—each IP address is assigned to a subdomain—and DNS polling is used to provide clients with the best server IP address based on their geographic proximity. Currently, as of Oct. 2022, more than 4500 timeservers are part of the pool serving approximately a million hosts per day, including routers, IoT devices, and home appliances.

Considering that the NTP Pool Project is voluntary-oriented, managing such heterogeneous systems is challenging. The pool employs two key management artifacts to simplify the complexity: the watchdog system and management central. A watchdog, i.e., a monitoring server, periodically checks the status of the participating timeservers. A centralized management server oversees the monitoring server's operation to keep a consistent view over time across the pool. Figure 2 illustrates the overall NTP pool structure.

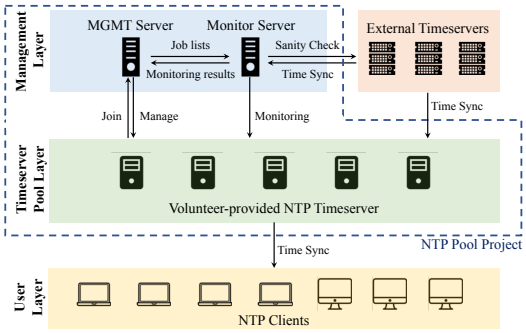

Figure 2: The basic structure of the NTP pool.

Management system. The management system's main tasks are twofold: administration and supervision. The management system provides an interface through which participants can register their timeservers with a publicly accessible and static IP address. The management system registers the new timeservers into different DNS zones according to their location, leading NTP clients to the most suitable, the least erroneous timeservers nearby. Another main task is to ensure the reliability and integrity of the pool. To this end, the management system utilizes the monitoring server to track the status of the participating timeservers; it delivers scheduled job lists to the monitoring server, collects monitoring results, and removes unhealthy timeservers from the pool.

A decision to expel a timeserver from the pool is made through a scoring algorithm [29]. At every inspection round, the management server estimates a new score for each timeserver. The score ranges from 20 to -100, depending on the inspection result. According to their scores, timeservers are classified as either active or inactive: between 20 and 10, a timeserver is recognized as an active server appearing in DNS responses, and a timeserver with a score below 10 is an inactive server and thus invisible in DNS responses. If a timeserver's score drops below -15, i.e., `BAD_SERVER_THRESHOLD`, an out-of-band notification (e.g., an email alert) is sent to the server's administrator.

The scoring algorithm applies the following hard-coded linear step function to preclude sudden shifts in the score:

$$score_{new} = min(max\_score, (score_{old} * 0.95) + step), \quad (1)$$

where 0.95 is the aging rate, max_score is the upper limit of $score$ ($= 20$ by default), and $step$ is the step variable. The step variable ranges from $+1$ to -5 according to the target server's response (see, Algorithm 1). More precisely, $step = -5$ if the target server is offline (i.e., unreachable or not responding). For the target servers with time offset > 75 ms, it would be $-4 \leq step \leq +0.7$; otherwise $+1$. We confirmed that the scoring algorithm is being used in the live monitoring server through an experiment using a test timeserver to which an arbitrary delay is applied; after the test server's score reached 17.2 due to the -1.8 step value by the artificial offset of 700 ms,

Algorithm 1: Step Formula (https://github.com/ntppool/monitor/client/localok/local-check.go, commit 6005ff4)

```
if no_response or stratum == 0 then
    step = −5
else
    if |offset| > 3 or stratum >= 8 then // 3 s
        step = −4
        if |offset| > 3 then
            max_score = −20
        end
    else if |offset| > 0.75 then          // 750 ms
        step = −2
    else if |offset| > 0.075 then         // 75 ms
        step = −4 ∗ |offset| + 1
    else
        step = +1
    end
end
```

it took about 80 rounds of inspection (i.e., approximately 18 hours) to reach the maximum score of 20 again.

Monitoring server. The monitoring server is responsible for checking the health of timeservers in the pool, more precisely the timeservers' responsiveness and time accuracy. By simply exchanging packets through NTP, it inspects if the target timeserver replies to the requests (i.e., responsiveness) and calculates the timeserver's time offset to the monitoring server's local time (i.e., accuracy).

Keeping the monitoring server's time accurate is critical, because timeservers that are either offline or appear out-of-sync with the monitoring server's clock will be marked as inactive and are eventually removed from the pool. To this end, the monitoring server performs continuous self-checkups before pulling a job list of timeservers to be inspected from the management server. The monitoring server performs a sanity check on the local clock using NTP with external timeservers in the hard-coded external timeserver list. If the sanity check is passed, i.e., the number of *hosts* (external timeservers) with local offset < 3.5 ms is bigger than *failureThreshold*, where

$$failureThreshold = len(hosts) - ((len(hosts)+2)/2), \quad (2)$$

the monitoring server retrieves a job list from the management server and executes monitoring. Otherwise, it retries again after a second. Note that the management server makes job lists by grouping the timeservers registered in the pool into subgroups, allowing the monitoring server to pull a scheduled job list. According to the current monitoring schedule, timeservers are inspected approximately every 13 minutes.

The monitoring server reports the inspection results to the pool's management server, including the target timeserver's IP, offset, and RTT. Based on the report, the management server calculates the step value and updates the corresponding score for each timeserver. The front end of the management server has an interface that allows anyone to search for timeserver scores and download CSV logs from the last 24 hours that contain inspection time, offset, step value, score, and the responsible monitoring server.

Table 1: Hard-coded reference time sources for the sanity check (commit 6005ff4).

Source	Active (as of May 2022)
time.apple.com	✓
ntp.ubuntu.com	
time.google.com	
ntp1.net.berkeley.edu	✓
tock.ucla.edu	✓
ntp.inet.tele.dk	✓
uslax1-ntp-001.aaplimg.com	✓
defra1-ntp-002.aaplimg.com	✓
uklon5-ntp-001.aaplimg.com	✓
ntp.stupi.se	✓
ntp.se	
ntp.nict.jp	✓
ntp.ripe.net	✓
time.fu-berlin.de	✓

3.2 Vulnerabilities at the Top of the Hierarchy

The pool has been operating a single monitoring server, i.e., Monitor SJ (*monsjc1.ntppool.net*), San Jose, CA, US. The simple monitoring setup delivers a good enough quality of self-healing mechanism with reasonable management overhead, yet it has fundamental challenges.

Circular dependency in the time measurement. The single monitoring server inspects all timeservers in the pool whether their time is coherent with the UTC standard. Considering the vital role of overseeing the thousands of timeservers' time, the accuracy of the monitoring server's local clock is essential. Nevertheless, the monitoring server's local clock also has a *weak foundation of precision*, a software clock that needs to be frequently synchronized with high-precision external clocks. Unlike other critical infrastructures, such as tier-1 ISPs, financial networks, or transportation systems, where their time is tightly synchronized with the physical reference clocks, however, the pool's monitoring system does not seem to have such luxury. The monitoring server keeps sanity checking its time with external stratum 1 or 2 timeservers (listed in Table 1), indicating that the server's clock is likely being synchronized with external timeservers with a similar stratum rather than highly precise reference clocks. A concern is that the monitoring server and timeservers in the pool get fed their time by the same external time sources? What if these external sources propagate inaccurate time? This potential circular dependency may banish timeservers with the correct time and keeps ones with the same incorrect time sources, jeopardizing the time of the entire pool and further all NTP clients synchronized with the surviving timeservers.

Asymmetric network delay. Given that NTP is the basic protocol for the pool's time-related operations, i.e., synchronization with external time sources and monitoring of the pool's timeservers, the architecture inherits NTP's weakness to the asymmetric network delays. As briefly described in Section 2, NTP works very well under ideal network condition

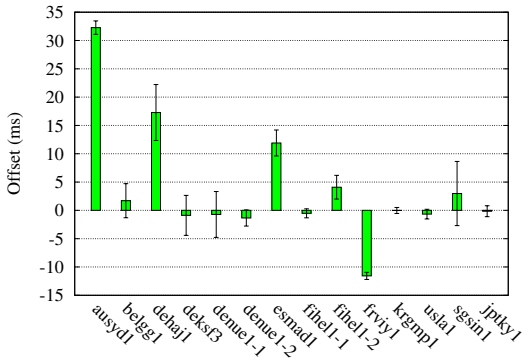

Figure 3: Offsets logged by various monitoring servers for a test timeserver.

where network latencies for both directions are symmetric. The NTP standards clearly state the possible inaccuracy; depending on the versions of `ntpd` running on the servers, monitoring servers' local clocks might vary up to a few tens of milliseconds (NTPv4 [21]) or a few hundreds of milliseconds (NTPv3 [20]). A problem is that asymmetric propagation delays can be deliberately induced by an adversary.

Single-point-of-failure. A single monitoring server represents a *single point of failure*. Due to a variety of reasons, such as server maintenance or network link failure, the monitoring server may go offline, leaving timeservers with no supervision. It is also problematic if the monitoring server's time deviates from the real time or the accuracy of NTP is not guaranteed due to a network problem.

3.3 New Monitoring System under Pilot Testing and Limitations

Multiple monitoring stations. Since May 2022, the pool has employed multiple monitoring servers in its beta testbed to test a new monitoring system*. The participating monitoring servers collect scores for a subgroup in the pool (under *ntp.beta.grundclock.com*) and report the monitoring results to the log server. From our observations so far, however, no distinct changes have been addressed other than configuring multiple monitoring servers; the scoring logic has not changed, and the log server simply averages the aggregated scores. Nevertheless, here we share our insight learned from the observations that might be useful to improve the new monitoring system with multiple servers.

Split views of the distributed monitoring servers. The new scoring algorithm must achieve resilience against potential bias in the aggregated monitoring results from the distributed monitoring servers. Despite the sanity check with the same external timeservers, each monitoring server might have different local clock values, because of: (i) different external timeservers they are synchronized with, (ii) different geographical distances between the external timeservers and monitoring servers, and (iii) different network conditions, such as instant network congestion. Figure 3 supports the possibility of the monitoring results' inconsistency. We investigated how the distributed monitoring servers file the monitoring results for our test timeserver. As shown in the figure, each monitoring server logs different offsets ranging from 32.28 ms to -11.57 ms. Recall that all the monitoring servers have passed the sanity check, which requires an accuracy of ± 3.5 ms.

Malicious monitoring servers. The NTP pool's management system allows the monitoring servers to perform a self sanity check before monitoring. This implementation choice assumes that the monitoring server is fully committed to the integrity of the pool. The assumption, however, only stands if all the monitoring servers are under the pool's authority. Employing voluntary monitoring servers as currently tested cannot guarantee such trust; monitoring servers planted with malicious intent may blindfold the self sanity check to feed the pool's management server with forged monitoring results. Other than authenticating the participating monitoring servers through the pool-issued certificates, there are currently no security mechanisms in place to prevent misbehavior.

4 Attack Modeling

Goal. The attacker's goal is to shift the local time of NTP clients in a wide area, e.g., a country or even a continent, by leveraging the NTP pool system. To illustrate the severity of such a large-scale time shift attack, an attacker could gradually disrupt critical infrastructures over an entire region, e.g., disrupt finance or health care operations. To this end, we consider an attacker attempting to dominate the NTP pool ecosystem by removing legitimate timeservers from the pool.

Adversary model. We assume a benign NTP pool management and monitoring server infrastructure, as otherwise the security of the system is trivially compromised.

The adversary controls a subset of the timeservers in the target region, which were obtained either by compromising existing timeservers, or by injecting its own servers into the pool. Timeservers under the adversary's control can distinguish between regular NTP clients and the NTP pool's monitoring server. Adversarial time servers report different local clock readings; accurate clock readings to the monitoring server and incorrect time values to NTP clients. Their selective response enables to pass the pool's inspection, ensuring their survivability.

The adversary may also perform on-path attacks: modifying or injecting packets into unauthenticated connections, or delaying or dropping packets in case of authenticated connections. In addition, the network attacks can be performed off-path in conjunction with BGP prefix hijacking, enabling the attack from diverse vantage points.

Attack strategies. Chronos [8, 37] and Ananke [32] enable NTP clients to collect time information from randomly sam-

*There is an ongoing discussion in the NTP pool community for a new monitoring system and scoring logic, issued on April 29, 2022 [26].

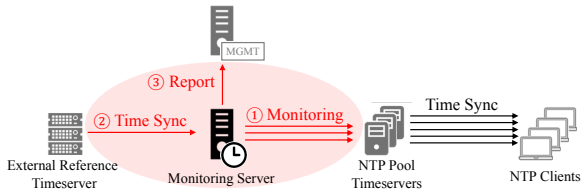

Figure 4: Three targets to exploit the monitoring system.

pled multiple NTP timeservers, remove outliers, and compute seemingly accurate time (i.e., the average value of the surviving time information), improving resilience against the time-shifting attack. Hence, to be successful in the attack, the attacker must have a legion of malicious timeservers to occupy the majority of the NTP client's query list, tamper with the majority of response packets directed to the client, or attract the clients to malicious time servers, e.g., through DNS cache poisoning. However, there are practical difficulties: (i) to affect a target area or the entire pool, hundreds or even thousands of timeservers must be infected or injected, (ii) packet tempering can be blocked if the response packets are authenticated, and (iii) it can only increase the probability of the attacker's timeserver getting queried by multiple clients, but cannot guarantee that the client's query list will be populated with the attacker's timeservers.

In our attack, the attacker aims to make legitimate timeservers in the pool invisible, maximizing the influence of the attacker's timeservers. To this end, the attacker leverages the pool's monitoring system. As described in Section 3.1, the NTP pool monitoring system removes timeservers exhibiting abnormal offsets or non-responsiveness from the pool through periodic health checks. By exploiting this monitoring process, the attacker excludes competing (legitimate) timeservers from the pool and attracts more queries from NTP clients to the attacker's timeservers.

We understand that this radical attack model may cause drastic changes in the NTP ecosystem and thus can be easily detected by the pool administrator. Therefore, for the attacker, it is vital to maintain a low profile to avoid being caught on the pool administrator's radar. The attacker, therefore, leverages `BAD_SERVER_THRESHOLD`. Recall that the monitoring server classifies each time server into active or inactive, and remove stages according to their score: if the score is 10 or less, the corresponding timeserver is tagged as inactive and excluded from the DNS table. When the score drops below -15 (i.e., `BAD_SERVER_THRESHOLD`), the timeserver's administrator will be notified. Therefore, the attacker's strategic goal is to keep the scores of benign timeservers between 10 and -15, the so-called *grey zone*.

We now present two viable attack approaches: interference and invasive attacks. With the interference attack approach, the adversary launches network attacks to exploit the monitoring operations and, eventually, influence the inspection results (① and ② in Figure 4). This approach is suitable for the current pool system with a single monitoring server under the pool's administration. Another attack approach targets the multiple monitoring server system currently being tested. The adversary performs a more invasive attack, directly manipulating the monitoring results (③ in Figure 4).

5 Interference Attacks

5.1 Adding Asymmetric Delays to Monitoring Packets

The attacker interferes with the monitoring server's inspection process to mislead the health checks on timeservers. The most intuitive attack method is to directly modify the time value in the payload of the NTP response packet. For this, however, the attacker must be on the vantage point to which most NTP response packets towards the monitoring server aggregates (locale limitation). In addition, it is inefficient to parse the payload, compute the target offset, and modify the payload for all NTP response packets from thousands of timeservers (performance degradation). Finally, payload modification will not be feasible if NTP packets are protected using Transport Layer Security (TLS) with cryptographic keys shared via AutoKey/NTS. Note that, in this attack, we do not assume direct control over the monitoring server; the current monitoring system is under direct management by the pool's administrator, so any intention of system penetration might be detected via, e.g., software auditing.

Delay attack. Considering the NTP basics, a delay attack is the most effective and difficult to mitigate. The attacker can achieve the same results by simply adding an asymmetric network delay to the NTP communication. The delay attack does not require any changes in the exchanged synchronization packets and thus sidesteps any potential NTP authentication schemes. Furthermore, above all, the asymmetric network delay is not a conspicuous phenomenon in the current Internet due to many reasons, such as network congestion or routing path asymmetry, so detecting the delay attack is challenging. Driven by this, the attacker performs asymmetric delay attacks such that $target\_delay = 2 * target\_offset$. Since the attacker aims to keep the target timeservers' scores between 10 and -15, *target_offset* should be adjusted to ideally meet $step = 0$. According to line 12 in Algorithm 1, $target\_offset = 1/4 =$ 250 ms and consequently, $target\_delay = 500$ ms. Note that the default value for `MAXDIST`, which denotes the maximum RTT that NTPv4 tolerates for accuracy against delay attacks, is one second [21], and thus the additional 500 ms is expected not to cause an NTP failure in most practical cases.

BGP hijacking. To effectively perform the asymmetric delay attack on all timeservers, the attacker must be at the point where all traffic is aggregated, or all traffic must be diverted to the attacker. Obviously, the most certain aggregation point would be right before the monitoring server. The attacker, however, cannot always achieve this, and thus the off-path

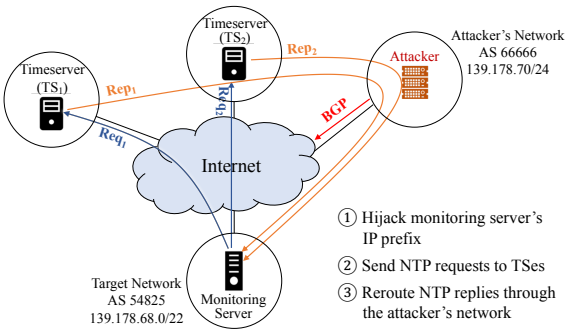

Figure 5: BGP hijacking against the IP prefix of the pool's monitoring server reroutes NTP response packets, introducing asymmetric latencies only into the monitoring process.

attack model using BGP hijacking would be a better approach.

The attacker modifies the routing path towards the target IP prefixes through forged BGP announcements. The off-path attacker can either become the destination of the hijacked packets or forward them to the real destination. Considering the attacker hijacking packets between a monitoring server and thousands of timeservers in the pool, targeting the IP prefix containing the monitoring server is obviously the simplest. Figure 5 illustrates the off-path attacker. We confirmed from RIPEstat [34] that the current monitoring server (*mon-sjc1.ntppool.net*) resides within an AS (54825) that owns a /22 IP prefix. By broadcasting a forged BGP announcement that the attacker claims to possess a more specific prefix (e.g., /24), the attacker is able to update the routers' routing information bases. Given that the attacker hijacks the monitoring server's IP prefix, the NTP request packets will be routed through the original paths. On the other hand, the response packets will take a detour through the attacker, causing an additional arbitrary delay $\alpha = OWD(Rep) - OWD(Req)$ and resulting in asymmetric latencies.

The additional delay caused by BGP hijacking varies depending on the attacker's location. For example, as shown in the figure, Rep_1 will demonstrate a higher additional delay than Rep_2 due to the longer detour. A practical challenge for the attack is that the attacker cannot directly infer how much additional delay α_i has been introduced for each timeserver. Considering that typical Internet latency is likely below *target_delay*, 500 ms—for intercontinental communications such as Europe-Asia, it is around 300 $\sim$ 350 ms—the attacker needs to generate another artificial delay $\beta = target\_delay - \alpha$.

Adaptive asymmetric latency. To estimate β, the attacker reverse-engineers the monitoring server's log records retrieved from the management server. More precisely, the attacker first collects the addresses of the hijacked timeservers by sniffing the NTP response packets destined for the monitoring server. After sufficient time has elapsed for the pool to update a new score for the timeserver, e.g., one minute, the attacker retrieves the timeserver's CSV log from the management server and estimates α by comparing the recently logged offset with the one collected a few hours prior, and accordingly β. Finally, on the subsequent monitoring round on the same timeserver, that attacker introduces an additional delay β to the NTP response packets. This can be repeated until the desired offset is met, implying the logs are leveraged as the correction references for the attacker to calibrate β. We analyze the feasibility of this approach in Section 7.1.

Attacker's timeservers. BGP hijacking will also reroute the NTP response packets originating from timeservers under the attacker's control. Nevertheless, the attacker's timeservers can compensate α by responding with modified time values, remaining as active timeservers in the pool.

5.2 Shifting the Monitoring Server's Local Time

Shifting the monitoring server's local time is another effective attack model, since it causes the monitoring server to believe that all the functioning timeservers in the pool are out-of-sync. In this attack, we consider an in-network attacker whose goal is to shift the local time of the monitoring server by tricking its time synchronization process from outside.

Hijacking synchronization packets. To shift the monitoring server's time while still enabling it to pass the sanity checks, the attacker must intervene in two communication channels: the reference clock of the monitoring server and external timeservers with which the monitoring server performs its sanity checks. Similar to the previous attack, the attacker can attempt to hijack the packets with BGP. However, BGP hijacking effects to a wider area, so it does not fit well with the targeted attack. If possible, performing on-path delay attacks targeting only the two types of communication will be the most efficient and effective approach. However, as we discuss earlier, the on-path attack places a strong assumption on the location. Another practical approach is DNS cache poisoning, which specifically targets the monitoring server's external time sources, where the monitoring server's queries can be redirected to the attacker's fake timeservers.

The attacker targets the monitoring server's DNS resolver as shown in Figure 6. By following the hijackers guide [3, 6, 14], the attacker disguises as the legitimate name server with the spoofed IP and injects fake responses (or IP fragments) to the resolver when it issues a query to the upstream authoritative name servers. The attacker knows in advance the victim domains, such as the reference timeserver's domain (i.e., *ussjc2-ntp-001.aaplimg.com*), and the corresponding name server's IP address (i.e., `17.253.207.1`). Suppose the elaborately forged fake response arrives before the legitimate response. In that case, the resolver accepts the fake response, updates its cache, and replies to the monitoring server with the fake IP address. Indeed, DNS cache poisoning is practical considering that: (i) only 12% of resolvers enabled DNSSEC [5], (ii) over 30% of ASes do not block

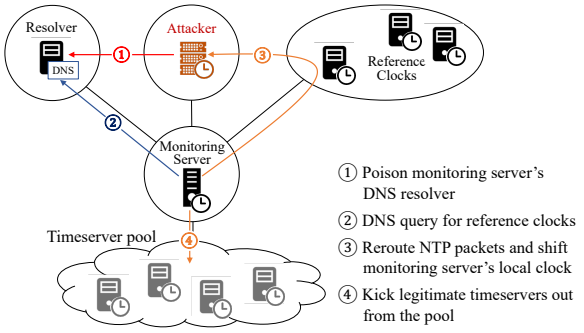

Figure 6: The attacker hijacks monitoring server's synchronization process with external reference clocks, shifting the local clock at the monitoring server.

source address spoofing [17], and (iii) new DNS cache poisoning attacks targeting the lower layers of the DNS hierarchy, e.g., OS-side stub resolvers, seem viable [19].

6 Invasive Attack

Injecting malicious monitoring servers. A new monitoring server can be registered and participate in the beta test through the dedicated management URL [28]. The server owner needs to download and install the monitoring agent binary from the NTP pool developer repository [27]. Once an IP address and e-mail address are given, the NTP pool administrator issues API keys (including a certificate) for communication with the management server. We were able to join the pool's monitoring system with three servers in Europe, the Americas, and Asia, respectively. The attacker can also register multiple monitoring servers into the pool through the same process. Section 7.3 will show how easy it is to enter the pool's monitoring system and deactivate target timeservers using a few monitoring servers.

Bypassing sanity checks. The legitimacy of the monitoring server, for now, solely depends on its self-sanity check. The monitoring server self-estimates the accuracy of the local clock through NTP packet exchanges with external public timeservers. However, bypassing the sanity check is simple: omitting the sanity check line of the source code or bypassing the binary execution downloaded from the repository. The current monitoring system does not consider any integrity check of the (participating) monitoring server, e.g., authentication of the sanity check or code auditing. If the attacker's monitoring server bypasses the self-sanity check, then its entire NTP pool is now the attacker's playground.

Manipulating monitoring results. The attacker can download the monitoring job list from the management server through a legitimate API, `GetServers()`. For each timeserver's IP address in the job list, time is measured through NTP packet exchanges, offset is calculated, and then the target timeserver's status is reported with the job list ID through another API, `SubmitResults()`. To intentionally lower the score of the target timeserver, a false offset value will be reported. The simplest way is to shift the monitoring server's local clock. The faulty local clock will add an arbitrary offset to all NTP results for normally functioning timeservers.

A bolder attack approach is directly adding the desired offset to the target timeservers' monitoring results through monitor code manipulation. By abusing the monitoring report process, the attacker can overwrite any desired value in the offset field of `ServerStatus` regardless of the NTP measurement result. If the target server's stratum and RTT value are known, the attacker does not even need to perform NTP. In addition, the attack can target specific IPs, surgically excluding timeservers assigned for a target area. Note that the attacker can use a DNS crawler [35] to collect the NTP timeservers' IP addresses assigned to domains for specific zones, such as continents, countries, and vendors. The management server will update these targeted timeservers' scores to a lower value and eventually turn them into an inactive state.

A few monitoring servers is enough. The injected monitoring server's influence is inversely proportional to the number of legitimate monitoring servers registered in the monitoring server pool. The target timeserver's score intentionally lowered by the attacker will be restored by the other monitoring servers. However, due to the biased sensitivity of the step formula in the current scoring algorithm—drops fast but recovers slowly, i.e., -5 to $+1$ in one round—the attacker's monitoring server is comparable to theoretically five legitimate servers. The aging rate in the scoring algorithm even exacerbates this further; the aging rate always affects negatively when the score value ranges $20 \sim 0$. Therefore, injecting a few monitoring servers (e.g., less than 20 % of the monitoring server pool) would give the attacker a significant attack power (see, Section 7.3). It is important to note that this attack is technically not scale-limited; rather, an injected monitoring server can cover all the timeservers in the pool if the job lists are given, as in the case of a single monitor.

7 Attack Analysis

To quantify the feasibility of the attacks aiming at the heart of the NTP pool monitoring system, we mainly analyze the following three factors: if (i) the attacker can micro-control the target server's time offset with delay attacks (§7.1), (ii) the monitoring server whose local clock is arbitrarily shifted can still perform the sanity check, monitoring, and reporting (§7.2), and (iii) a small proportion of monitoring servers can successfully manipulate the target timeserver's score (§7.3). Furthermore, and most importantly, we analyze if the attacker can keep a low profile without being caught by the NTP administrator during and after the attacks (§7.4). For ethical experiments, all tests are conducted on/to/from servers we planted into the pool; none of the existing NTP pool entities were affected by the experiments for correct operation.

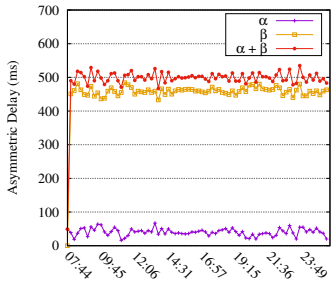

Figure 7: Asymmetric delays.

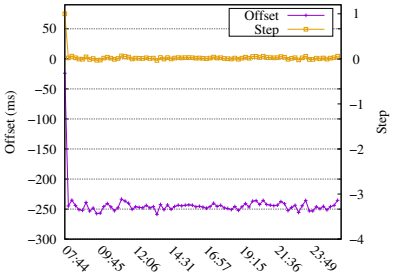

Figure 8: Offset and step values.

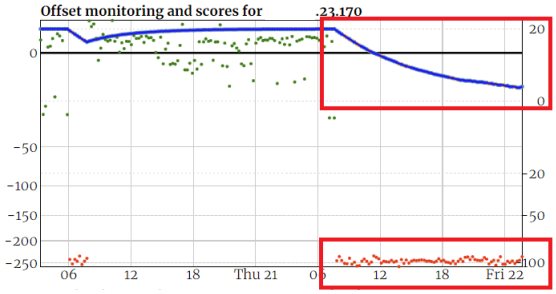

Figure 9: Target timserver's score (blue line).

7.1 Distorting the Vision of the Monitor

We set a goal of 500 ms asymmetric delay to overwhelm the NTP pool silently. To achieve this in a dynamic network environment, an attacker performs the adaptive delay attack described in Section 5.1. That is, through continuous monitoring log analysis, the attacker infers the previous delay and feedback on the current delay, micro-adjusting the attack configuration. We applied the following methodology.

1. On the target timeserver, a random delay 50 ms ($\pm$ 20 ms) to all outgoing traffic is added at its network interface.
2. The attacker sniffs packets between the target timeserver and the monitoring server on the communication path.

3-1) The attacker pulls the monitoring logs from the management server every 30 seconds, and tracks the latest offset value of the target timeserver.

3-2) The attacker infers α from the offset, calculates β, and updates its DB with the {IP, β} pair.

3-3) When a new NTP response packet arrives (e.g., destined for the monitoring server), the attacker performs a DB lookup and applies an additional delay β to the packet.

4. The attacker repeats step 3, adjusting the total delay.

Regarding the experiment, we would like to note that (i) we configured the target timeserver's status as *monitoring only* by adjusting the `net_speed` parameter so that none of the NTP clients solicits a time value from the timeserver, (ii) we ensured no entities between the attacker and the target timeserver to avoid any unnecessary casualty by the attack, and (iii) we ignored DB lookup overheads since it only takes a few hundred nanoseconds—the maximum DB entries are $<$ 5K—and does not significantly affect the results. Figure 7 shows the asymmetric delays applied to the target timeserver following the experiment steps.

Impact of Adding 500 ms. Adjusting β is to compensate for the network jitter observed in the previous monitoring process, and thus there is always a gap between α_i and α_{i-1}. In addition, as α is an inference from approximating the offset value, it is also affected by the real offset coming from different clock drifts between the target timeserver and monitoring server. Nonetheless, as shown in the results, we confirmed that the approximation-based delay inference does not significantly affect the overall attack performance. The attacker successfully enforced an asymmetric delay of about 500 ms with an up to $2 * jitter$, relatively small errors compared to `target_delay`, and compensated for the errors in the very next inspection rounds. In a nutshell, the monitoring results intrigued by the attack were settled at an average offset of 245.6 ms (233.6 ms $\sim$ 258.9 ms)—successful time shifting from an average of 0.8 ms under normal conditions—and the step value accordingly could be suppressed from one to an average of 0.017 as shown in Figure 8.

The experiment was conducted for approximately 18 hours, from 07:20 to 01:20 the next day in local time. Figure 9, captured from the NTP Pool Management Web, shows the monitoring results of the target timeserver logged by the genuine monitoring server; the green and red plots indicate the recorded offsets and the blue line draws the target timeserver's score. In a nutshell, with the forged offsets (red dots), we successfully inactivated the target timeserver within 3 hours and 35 minutes (20 $\rightarrow$ 9.8, entering the grey zone). Since then, the score kept decreasing and finally stabilized at 0.7 after 15 hours of entering the grey zone.

7.2 Implications of a Faulty Single Monitor

Next, we demonstrate if a monitoring server with shifted local clock operates without interruption, and analyze what implications it has. The experiment uses one of the monitoring servers we injected into the multi-monitoring server testbed in order not to affect the legitimate monitoring server.

No integrity check at all except the sanity check. The higher stratum timeserver the monitoring server synchronizes with is known. Thus, by hijacking the time synchronization packets, we successfully shifted the local clock of the monitoring server by 250 ms. The next step is to pass the sanity checks. As shown in Table 1, the hard-coded timeserver list for the sanity check is also known. However, we observed the list has changed frequently for the last four months. We thus double-checked the server list by sniffing DNS packets. Table 2 is the list of external timeservers we confirmed. A total of 12 domain names were queried, of which ten were confirmed in the Github code, and two were new domains.

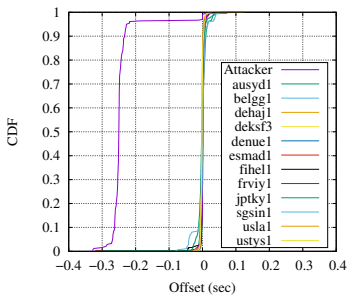

Figure 10: CDF for offsets.

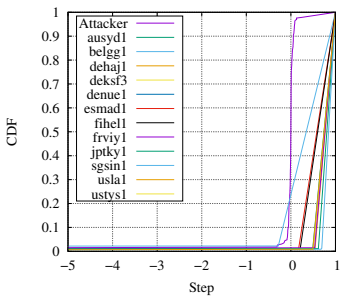

Figure 11: CDF for step values.

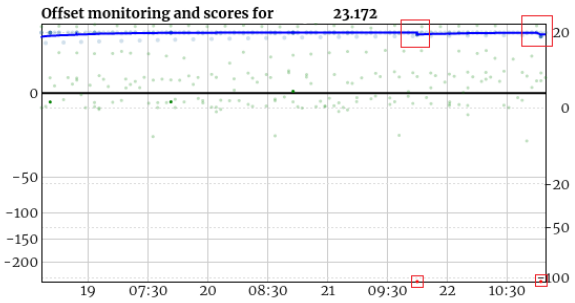

Figure 12: Example of an effected timserver's score.

Table 2: Observed external timeservers via DNS sniffing.

Source	IPs	Hard-coded
ntp.nist.jp	133.243.238.163, 133.243.238.244, 61.205.120.130, 133.243.238.164, 133.243.238.243	✓
defra1-ntp-002.aaplimg.com	17.253.54.253	✓
uklon5-ntp-001.aaplimg.com	17.253.34.125	✓
time.fu-berlin.de	130.133.1.10	✓
ntp.ripe.net	193.0.0.229	✓
ntp.inet.tele.dk	193.162.159.194	
ntp1.net.berkeley.edu	169.229.128.134	✓
uslax1-ntp-001.aaplimg.com	17.253.26.125	✓
time.apple.com	17.253.82.253, 17.253.82.125, 17.253.114.125	✓
ntp.stupi.se	192.36.143.150, 192.36.143.151, 192.36.143.153, 192.36.143.234	✓
tock.ucla.edu	164.67.62.212, 164.67.62.199	✓
jptyo5-ntp-001.aaplimg.com	17.253.68.125	

According to the sanity check algorithm (see, Equation (2)), a sanity check can be easily avoided if five out of 12 timeservers are subjected to have an offset < 3.5 ms. We also hijacked the sanity check packets toward the timeservers with known domains and successfully passed the sanity check.

In this demonstration, the most important finding is that there is no additional hidden verification procedure over the integrity of the monitoring server's local clock other than the sanity check. The monitoring server that passed the sanity check was able to pull the job list from the management server, perform inspections with the shifted local clock, and register the incorrect monitoring results to the management server.

One failure for all. Our monitoring server operates with shifted local clock for two hours, from 21:09 to 22:10 in local time. A total of 234 timeservers were inspected during the experiment. The inspection frequency was hour basis—a monitoring server would have an inspection cycle of either one hour or 13 minutes, according to the management server's job scheduling. Therefore, a timeserver was inspected up to two times by our monitoring server during the period. Figure 10 plots the CDF of the offset values measured for the timeservers by our monitoring server as well as other (active) monitoring servers. The average offset we reported was 242.5 ms (STD: 49.2 ms), while the other monitoring servers reported 0.1 ms (STD: 9.4 ms). From our reports, 12 timeservers were recorded with offsets < 10 ms; five were *offset* $= 0$ due to I/O timeout. Considering that the remaining seven showed similar offsets in the other monitoring servers' reports, we suspect that the attack failure was caused by arbitrary network delays.

The offsets reported are subsequently reflected as step values. As shown in Figure 11, the step values derived from the other monitoring servers' reports converge to one, whereas our reports drive the step value to nearly zero. We confirmed this from the management web interface. Figure 12 shows the history of monitoring results of one of the inspectees. As highlighted with the red boxes, the gaze of the monitoring server whose local clock is shifted looking at other timeservers is indeed reflected. It is important to note that this experimental setup does not affect the live timeserver's operation as other monitoring servers quickly compensate for the score drop. However, in the current NTP pool with a single monitoring server, it implies that an attack on the monitoring server's local clock can affect all the timeservers in the pool and, consequently, all the underlying NTP clients.

7.3 Injecting a Few Monitors is Enough

We now analyze how resilient multi-monitoring environments are against malicious monitoring servers. To this end, two monitoring servers we injected repeatedly filed the lowest step value, -5 (i.e., I/O timeout), for one of our timeservers. Recall that the target timeserver is configured for *monitoring only*, not appearing in DNS queries. The experiment was conducted from 11:30 am to 3:00 pm the next day. Figure 13 shows the offset result reported by multiple monitors for the target timeserver and the corresponding score value change. While continuously reporting I/O timeouts, we observed no warnings (e.g., email notification) or implicit actions (e.g., ignoring reports) from the management system.

After filing an I/O timeout (red dot), we immediately observed a sharp drop in the score (blue line). However, due to the "*server OK*" report (green dot) from the other normal monitors, the score value gradually ramped up and soon recovered to almost the original score value even before the next attack, unlike our anticipation. In order to understand this phenomenon, we carefully investigated the leading cause, and

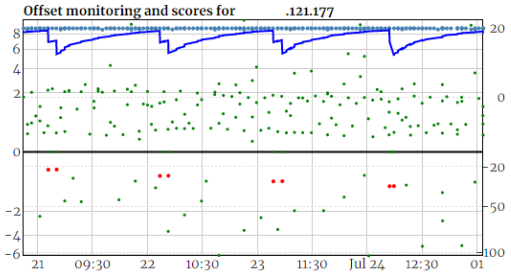

Figure 13: Reported offsets and the corresponding score with two attackers.

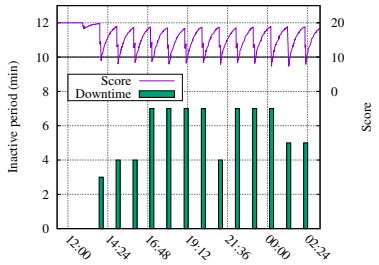

Figure 14: Recovery time to escape the grey zone (>10).

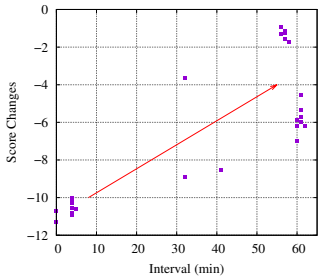

Figure 15: Score changes per attack interval.

found it was due to the different influence of each monitoring server on the monitoring result. Specifically, amongst the total 13 monitoring servers, seven have a monitoring frequency of $12 \sim 13$ minutes, and the remaining six (including two monitoring servers used in the attack) have approximately an hour frequency. That is, considering the step formula's biased sensitivity, an attacker's monitoring server with an hour monitoring frequency is equivalent to that of a normal monitoring server with 12 minutes; arithmetically, it is two to 39 influence metric. Indeed, according to the monitoring log, more than 40 logs from the normal monitoring servers were filed during an hour interval. Given that each normal monitoring report increases the score by $step - 0.05 * score$, it takes 15 reports from 0 to over 10. Therefore, to be successful in the attack, we conclude that the attacker needs four low-frequency monitoring servers or two high-frequency monitoring servers.

Although the current attack configuration was unable to settle the target timeserver's score in the grey zone, we observed great potential. Figure 14 illustrates how long the target timeserver's score stayed in the grey zone. With two I/O timeouts, the score went from *max_score* to $8.3 \sim 8.5$. Furthermore, it takes about $4 \sim 7$ minutes (i.e., time to get filed with four normal reports) to recover the score over 10. Considering that the default `ntpd` synchronization frequency is between 64 and 1024 seconds and the most clients are relying on resolvers with a 150 s TTL for the DNS records [24, 35], we expect that a significant number of NTP clients will not be able to synchronize with the timeserver, and it is an impressive result for only two monitoring servers.

When only a small amount of attack power is available, the effectiveness of the attack would more depend on the concentration of available resources. With filing several negative monitoring results at once, the score can be directly dropped to the grey zone before it is recovered. Figure 15 plots the score's dropping amount (i.e., score before $attack_1$ - score after $attack_2$) for different attack intervals. When the two malicious reports are perfectly synchronized, it results in the maximum attack effectiveness of -11.7. On the other hand, as the two malicious reports occurred sporadically, the attack achieved only half of the maximum effectiveness (i.e., -6). Even worse, the score drops a little ($-1 \sim -2$) if too many normal reports are filed before $attack_2$, restoring the score to a higher value than before $attack_1$. A viable approach is to fail in sanity check intentionally. When a sanity check fails, the monitoring server awaits two minutes to attempt another sanity check. Although the attacker cannot precisely control the monitoring schedule, it is possible for the attacker to postpone the scheduled monitoring tasks and perform a synchronized attack using multiple monitoring servers.

Table 3: Meantime a timeserver's administrator gets notified after its score drops below `BAD_SERVER_THRESHOLD`.

Server	Monitor type	Meantime to notify (Score)
TS_1 (x.x.23.171)	Single	1h 53m (-20.4)
TS_2 (y.y.121.177)	Multi	12m (-42.2)
TS_3 (z.z.23.172)	Multi	14m (-35.2)

7.4 Flying under the Radar

E-mail notification system. The grey zone of the current monitoring system provides an optimal environment for such low-profile attacks. It suspends a decision on temporary status abnormalities of the timeservers prior to immediate actions. With the pending status, administrators of the pool and target timeservers are unaware of such abnormality. Indeed, throughout the experiment, we received no notification from the pool while keeping a timeserver's score in the grey zone. We then deliberately lower the timeserver's score below `BAD_SERVER_THRESHOLD` (-15) in order to understand the pool's email notification system. According to our understanding, the notification system consists of two emails, i.e., an initial notification and a follow-up notification. We were notified after a considerable amount of time (see, Table 3) had passed since `BAD_SERVER_THRESHOLD`. The follow-up notification came 4 hours and 46 minutes after the first notification. Another follow-up has not been observed. Our findings from this are: (i) the administrators of timeservers would not be tipped as long as their timeservers reside in the grey zone, and thus (ii) taking immediate action upon such abnormality will be difficult (even when the score goes below `BAD_SERVER_THRESHOLD`).

8 Mitigation

So far, we have dealt with the inherent vulnerabilities of the current NTP pool monitoring system and practical attack approaches to exploit them. We now discuss the functional and design requirements to address these vulnerabilities. In addition, a new monitoring architecture for more secure and reliable NTP pool management from a short-term and a long-term perspectives is outlined.

8.1 Securing the Monitor's Operations

For better security of the NTP pool monitoring system, the first to consider is the integrity of the time synchronization chain from the external time source, monitoring server, to timeservers, even in the presence of such strategic attackers (see, ① and ② in Figure 4).

Embedded, trustworthy reference clock. The monitoring server performs time synchronization with external timeservers. Usually, the external timeservers are stratum-1 servers that are directly synchronized with highly precise reference clocks, e.g., GNSSes or atomic clocks, meaning that the monitoring server's local clock would also achieve sufficient accuracy as stratum-2 timeservers. Nevertheless, this is held only under ideal network conditions. As seen in Section 5.2, such a strategic attack can easily shift the monitoring server's local clock. NTP authentication is rarely applied in practice, and even if applied, a delay attack is always possible. The state-of-the-art NTP security techniques using multiple time sources are only suitable against malicious time sources, and their incentives are not clear against hijacking attacks on the monitoring servers prefix; all packets are affected by the attack, and thus outliers are hardly distinguishable.

One of the most acceptable mitigations is to reposition the monitoring server as stratum-1. That is, the monitoring server would receive time inputs directly from an atomic clock or GNSS. Atomic clocks are considered the most accurate time source. Atomic clocks are designed to measure the precise length of a second based on the oscillations of a certain atom (e.g., 9B oscillations of a caesium-133 atom), which is used as a frequency standard for the timekeeping element. GNSS is another feasible choice that distributes precise time information. Given that it is easy to find inexpensive atomic clocks on the market, the stratum-1 position is no longer the property of institutions with large budgets. As the only health-check system in the pool is responsible for the time of millions of distributed systems, having such a reliable and secure reference clock is essential. Further, besides the time accuracy, it would dramatically reduce the chance of an attacker shifting the monitoring server's local clock.

RTT-offset correlation. All Internet services and applications, including NTP, inherit the security characteristics of the underlying packet-switching infrastructure. Due to the lack of forwarding transparency and limited routing control in the current Internet, protocol-specific security amends demonstrated a limited improvement. In addition, such design changes require a wide adoption to be effective, but deployability, especially with voluntary participants, is challenging. Then, what is the most effective and viable way to secure the monitoring process without requiring changes from the underlying infrastructure or protocol design? We consider introducing heuristics for monitoring anomaly detection.

We go back to the basics. The NTP accuracy is RTT sensitive. Abusing the monitoring system requires a relatively higher delay than the RTT expected in usual NTP communication. Luckily, such an unusual RTT can be easily recognized [38]. The monitoring system may trace expected RTTs considering the geographical proximity of the two communicating systems and set, e.g., per-timeserver `MAXDIST` with an additional 10% of the expected RTT. For a long-distance RTT like 650 ms, its 10% only causes $32 \sim 33$ ms of offset, which is a tolerable margin for NTP considering, de facto, NTP users do not expect very high precision. Another approach is to detect the change in RTT. The NTP pool monitoring system readily reports RTTs for each timeserver. It can detect anomalies in RTT when a significant instability other than common jitter appears. In addition, with the future multi-monitor system, if possible, it can determine whether the RTT abnormality is a local or global characteristic. Timeservers having locale similarity may be grouped and conduct correlation analysis to locate the root cause of the abnormality.

8.2 Resilience against Injected Monitors

New scoring algorithm. The current scoring algorithm equally weights each monitoring report. Upon receiving a report, an offset is immediately reflected in the final score of the corresponding timeserver. This allows the pool management system to react instantly to a status change of timeservers. On the other hand, it also increases the sensitivity to faulty reports. Even with injecting a small number of monitoring servers, scores of target timeservers can be significantly lowered, putting them in an inactive state. Considering the case in which all (voluntary) monitoring servers cannot be completely trustworthy, we need to be able to classify such faulty reports and exclude them from scoring (③ in Figure 4).

Chronos [8] inspires us. Chronos' trust assumption is that not all NTP servers are trustworthy. Given the trust model, an NTP client requests time input from multiple timeservers, excludes outliers, and calculates the most statistically accurate time value. The same approximate agreement approach can be applied to the multi-monitoring system. When multiple voluntary monitoring servers participate in the scoring process, the pool cannot underestimate the possibility of faulty monitoring inputs. Driven by this, the management server collects monitored offsets from the monitoring servers, excludes outliers, and calculates the correct score.

Algorithm 2 shows an example of the modified approx-

Algorithm 2: Modified approximate agreement algorithm for the step formula with multiple monitoring servers.

Input: M = $\{m_1, m_2, ..., m_n\}$, t
Output: s_t
O = *collectOffset*(M, t) // gather results for a timeserver t from M
P = *pruneOutlier*(O, d) // prune d lowest and highest values
if $max(P) - min(P) \leq 2\omega$ **then**
    $s_t = stepFormula(avg(P))$
else
    O = *collectOffset*(trusted(M), t) // trusted monitors only
    $s_t = stepFormula(avg(O))$
end
return s_t

imate agreement algorithm for the step formula under the multi-monitor environment. The pool collects inspection results O for a target timeserver t from the registered n monitors, $M = \{m_1, m_2, ..., m_n\}$. Out of the n offsets, the d lowest and highest offsets are excluded from the monitoring samples. As discussed in §6 and §7.3, the biased step range yields a meaningful proportion of injected monitoring servers at around $15 \sim 20\%$. Therefore, the default value of $d = \frac{n}{5}$ would be practical. Later, it is examined that the maximum difference between all possible pairs of two surviving offsets does not exceed $2w$, where w is the max tolerable jitter for RTT_t. Considering the empirical analysis of asymmetric network delay (§3.2), we set $w = 20$ ms as default. When the condition is met, a new step value will be derived from the average offset of surviving samples $avg(P)$ and updated to the new score for t. The algorithm makes the monitoring process resilient to various invasive attack scenarios:

- **The attacker controls $< d$ monitors.** The attacker controls a small subset of monitoring servers. Considering that the attacker's goal is to lower the target timeservers score, all reports from the monitoring servers under the attacker's control will have a higher offset value than honest monitoring results. Consequently, they always fall into top/bottom d samples, and will be discarded by the pruning process.

- **The attacker controls $< n - d$ monitors.** In this case, the attacker controls enough monitoring servers to survive the pruning process. Thus, now, the attacker aims to let the algorithm discard as many true offsets as possible. Nonetheless, as the attacker reports fewer than $n - d$ offset values, at least $d + 1$ true offsets will also be reported. It means that, even if the algorithm discards top and bottom d offsets, at least one true offset value survives at the top/bottom of the remaining samples $n - 2d$. Since the algorithm checks if the maximum distance between all pairs of surviving offsets is within $2w$, to pass the check, all the attacker's reports must have offsets within $\pm 2w$ from the true offset. Given that $w = 20$, $avg(P)$ would be < 0.04 s; hence, the final step value would be $+1$ following Algorithm 1. Therefore, the attack is ineffective. Note that, to discard all the true offsets, the $d + 1$ true offsets need to be split and appear in both top and bottom d. This implies that all the attacker's reports in $n - 2d$ must be between the true offsets in the top and bottom d. Consequently, no attack impact occurs.

- **The attacker controls $> n - d$ monitors.** The attacker dominates the monitoring system with a sufficiently large number of injected monitoring servers. In this attack scenario, true offsets are discarded, and all the surviving $n - 2d$ samples are from the attacker's monitoring servers. This implies that the attack is successful.

To successfully lower the target timeserver's score, an attacker must appear en masse in the monitoring server pool (e.g., more than 80% when $d = \frac{n}{5}$). We however argue that such a case is not practical. The NTP pool's administrator can ensure that at least $d + 1$ monitoring servers are under the administrator's direct control, such that fewer than $n - d$ voluntary monitoring servers can participate in the scoring process.

One extreme case not yet covered is that the attacker intentionally reports faulty offsets that violate the "*within* $2w$" rule to sabotage the scoring algorithm. When the offset-distance checkup fails, the algorithm ignores all the participating monitoring servers, only collects monitoring reports from the trusted monitors (i.e., the pool's administrator controls), and calculates a new step value.

8.3 Towards Better Security for the NTP Pool

Responsible monitoring servers. Only reliable and trustworthy participants should be able to join as the monitoring server operator. The monitoring system is a critical part of the management. Participants wishing to operate a monitoring server thus need to prove their qualifications. The participants prove their legitimacy through a strict verification process, e.g., verifying an institutional or organizational email address. They must also meet sufficient performance requirements for a correct monitoring server operation, e.g., the presence of a highly precise reference clock. Such high requirements increase the reliability of the monitoring system operation while providing source accountability for malicious monitoring results.

Adding unpredictability in monitoring. The monitoring system must introduce unpredictability to prevent a sophisticated attacker with a high-level understanding of the current pool operation. First, we anticipate the anonymity of the monitoring server. A malicious timeserver can distinguish the monitoring packets through the source IP of the incoming NTP request. If the source IP becomes indistinguishable by leveraging an anonymous communication system, e.g., Tor [41], the malicious timeserver will not be able to distinguish the monitoring packets. Second, the monitoring system needs irregularity. Even if the monitoring server's IP is anonymized,

the attacker's timeserver can circumvent the inspection by inferring the next monitoring cycle and giving an honest answer to all NTP requests within the expected monitoring window. Introducing randomness to the monitor scheduling will thus make the prediction difficult, improving resilience to the sophisticated attacker.

Long-term visions for secure monitoring. Various research on NTP security has been conducted to improve the integrity of NTP communication, source authentication, and malicious time source detection. Nevertheless, the delay attacks still remain a powerful attack against NTP. This is mostly, if not all, due to the lack of routing control and lack of transparency of the current hot-potato routing environment underneath. In fact, this is not only NTP's problem; many Internet services are suffering from the same issue. Fortunately, various studies for improving routing security are being actively conducted, and among them, path-based future Internet architectures that pursue a new design of Internet routing from a clean slate attract attention [4, 42]. In path-based routing, two communicating endpoints determine the routing path based on the network information given by the network's control plane, and the data plane ensures packet forwarding according to the path information carried in the packet header. This allows the communicating entities firm control over the forwarding path, ensuring communication via a symmetric path, hijack prevention, and dynamic rerouting to bypass on-path attackers.

9 Related Work

We review related work in the areas of secure and reliable time synchronization which includes NTP pool mechanisms.

Secure time synchronization. Malhotra et al. [18] show how vulnerable unauthenticated NTP is to on-path and off-path time-shifting attacks. They exploit the fact that NTP clients accept any time shift when first initialized. They also demonstrate that NTP payloads can be overwritten by an off-path attacker via an IPv4 fragmentation attack. Annessi et al. [1] investigate the challenges of source authentication for broadcast time synchronization. NTPv3 [20] applies packet authentication with a pre-shared symmetric key, which needs to be done out-of-band. In NTPv4, a PKI-based authentication mechanism is introduced [21]. Authenticated Network Time Synchronization (ANTP) [9] minimizes server-side cryptographic operations by using symmetric cryptography for subsequent synchronization processes once the client is authenticated. Secure Time Synchronization (STS) [23] offloads the authentication to a third party (i.e., Authorization Server), reducing the server's authentication overhead and enabling mutual authentication between the server and client. Adopting cryptographic primitives ensures secure time synchronization in the presence of network attackers. Due to the difference in requirements for synchronization precision between time synchronization and the cryptographic systems, however, delay attacks still remain unsolved.

Reliable time synchronization. The NTP Pool Project [30] provides a centralized access to over 4,500 NTP timeservers (as of Oct. 2022) distributed across the globe for millions of users, servers, and applications. Rytilahti et al. [35] provide in-depth analyses on the NTP pool ecosystem and discuss potential vulnerabilities. Through active probing, they discovered that the NTP pool is highly dependent on a small number of timeservers. Moura et al. [24] explore the NTP pool's client-server mapping characteristics and reveal its biased server selection. Chronos [8] suggests an approximate agreement algorithm where a client gathers time samples from multiple NTP timeservers in the pool, prunes outliers, and averages the remaining samples, achieving reliable time synchronization in the presence of Byzantine attackers. Ananke [32] further improves Chronos by allowing the clients to verify the time information from normal timeservers with the pool-guaranteed stratum 1 timeservers.

10 Conclusion

The NTP pool that started as a small project has become an indispensable infrastructure for setting the time on millions of modern Internet devices. Unfortunately, the security of the NTP pool has obtained relatively little attention. In this paper, we examined the structural vulnerabilities of the current NTP pool monitoring system and the potential vulnerabilities of the multi-monitoring system under a pilot test. We demonstrated that: (i) the adaptive delay attack can manipulate the pool's monitoring process and deactivate healthy timeservers without being caught, (ii) shifting the monitoring server's local clock affects most of the inspection results (95%) under its monitoring, and (iii) only two injected time servers can deactivate timeservers for $4 \sim 7$ minutes. We further discussed the short-term and long-term design directions for enhancing the security of the NTP pool monitoring system from the strategic attacks. We envision that the security improvement plan will enhance the inspection accuracy of the NTP pool monitoring system, and increase the resistance against the injected malicious timeserver and monitoring server attacks. To the best of our knowledge, this paper is the first work that raises a fundamental question about the NTP pool's security for ensuring the integrity of timeservers in the pool.

Acknowledgments

We would like to thank the anonymous reviewers and shepherd for their insightful feedback and valuable suggestions. We gratefully acknowledge support from ETH Zurich, and from the Zurich Information Security and Privacy Center (ZISC). This work was also supported by IITP grant funded by the MSIT, Korea (No.2022-0-00411, IITP-2022-2021-0-01810, IITP-2023-2020-0-01819).

References

[1] R. Annessi, J. Fabini, and T. Zseby, "It's About Time: Securing Broadcast Time Synchronization with Data Origin Authentication," in *Proceeings of the International Conference on Computer Communication and Networks (ICCCN)*, 2017.

[2] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[3] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain Validation++ for MITM-Resilient PKI," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[4] L. Chuat, M. Legner, D. Basin, D. Hausheer, S. Hitz, P. Müller, and A. Perrig, *The Complete Guide to SCION. From Design Principles to Formal Verification*. Springer, 2022.

[5] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "A Longitudinal,End-to-End View of the DNSSEC Ecosystem," in *Proceedings of the USENIX Security Symposium*, 2017.

[6] T. Dai, P. Jeitner, H. Shulman, and M. Waidner, "The Hijackers Guide To The Galaxy: Off-Path Taking Over Internet Resources," in *Proceedings of the USENIX Security Symposium*, 2021.

[7] W. de Vries, J. J. Santanna, A. Sperotto, and A. Pras, "How Asymmetric is the Internet?" in *Proceedings of the IFIP International Conference on Autonomous Infrastructure, Management and Security*, 2015.

[8] O. Deutsch, N. R. Schiff, D. Dolev, and M. Schapira, "Preventing (Network) Time Travel with Chronos," in *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2018.

[9] B. Dowling, D. Stebila, and G. Zaverucha, "Authenticated Network Time Synchronization," in *Proceedings of the USENIX Security Symposium*, 2016.

[10] R. Exel, "Mitigation of Asymmetric Link Delays in IEEE 1588 Clock Synchronization Systems," *IEEE Communications Letters*, vol. 18, no. 3, pp. 507–510, 2014.

[11] D. Franke, D. Sibold, K. Teichel, M. Dansarie, and R. Sundblad, "Network Time Security for the Network Time Protocol," RFC 8915, IETF, Sep. 2020. [Online]. Available: https://www.ietf.org/rfc/rfc8915.txt

[12] B. Haberman and D. Mills, "Network Time Protocol Version 4: Autokey Specification," RFC 5906 (Informational), IETF, Jun. 2010. [Online]. Available: https://www.ietf.org/rfc/rfc5906.txt

[13] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker, "On Routing Asymmetry in the Internet," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, 2005.

[14] P. Jeitner, H. Shulman, and M. Waidner, "The Impact of DNS Insecurity on Time," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.

[15] M. Lévesque and D. Tipper, "Improving the PTP Synchronization Accuracy under Asymmetric Delay Conditions," in *Proceedings of the IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2015.

[16] London Economics (LE), "The Economic Impact on the UK of a Disruption to GNSS," https://www.gov.uk/government/publications/the-economic-impact-on-the-uk-of-a-disruption-to-gnss, 2017.

[17] M. Luckie, R. Beverly, R. Koga, K. Keys, J. A. Kroll, and K. Claffy, "Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[18] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the Network Time Protocol," in *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2016.

[19] K. Man, Z. Qian, Z. Wang, X. Zheng, Y. Huang, and H. Duan, "DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[20] D. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305 (Draft Standard), IETF, Mar. 1992, obsoleted by RFC 5905. [Online]. Available: https://www.ietf.org/rfc/rfc1305.txt

[21] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905, IETF, Jun. 2010. [Online]. Available: https://www.ietf.org/rfc/rfc5905.txt

[22] D. L. Mills, *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space, Second Edition*. CRC Press, Inc., 2010.

[23] F. Mkacher, X. Bestel, and A. Duda, "Secure Time Synchronization Protocol," in *Proceedings of the IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2018.

[24] G. C. Moura, M. Davids, C. Schutijser, and C. Hesselman, "Diving into the NTP Pool," Technical Report. SIDN Labs, Arnhem, The Netherlands., Tech. Rep., 2021.

[25] C. D. Murta, P. R. Torres Jr, and P. Mohapatra, "QRPp1-4: Characterizing Quality of Time and Topology in a Time Synchronization Network," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, 2006.

[26] NTP Pool Project, "Beta Monitoring Operators/Systems," https://community.ntppool.org/t/beta-monitoring-operators-systems.

[27] NTP Pool Project, "Monitor Repository," https://builds.ntppool.dev/repo/.

[28] NTP Pool Project, "NTP Pool Management-Beta," https://manage-beta.grundclock.com/manage/monitors/new.

[29] NTP Pool Project, "NTP Pool Monitor Codebase," https://github.com/ntppool/monitor.

[30] NTP Pool Project, "NTP Pool," https://www.ntppool.org/, 2022.

[31] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao, "A Measurement Study of Internet Delay Asymmetry," in *Proceedings of the International Conference on Passive and Active Network Measurement*, 2008.

[32] Y. Perry, N. Rozen-Schiff, and M. Schapira, "A Devil of a Time: How Vulnerable is NTP to Malicious Timeservers?" in *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2021.

[33] D. Reilly, H. Stenn, and D. Sibold, "Network Time Protocol Best Current Practices," *Work in Progress, draft-ietf-ntp-bcp-00*, 2017.

[34] RIPE Network Coordination Centre, "RIPEstat," https://stat.ripe.net/app/launchpad/, 2022.

[35] T. Rytilahti, D. Tatang, J. Köpper, and T. Holz, "Masters of Time: An Overview of the NTP Ecosystem," in *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[36] Z. Sarker, C. Perkins, V. Singh, and M. A. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control," RFC 8888, IETF, Jan. 2021. [Online]. Available: https://www.ietf.org/rfc/rfc8888.txt

[37] N. R. Schiff, D. Dolev, T. Mizrahi, and M. Schapira, "A Secure Selection and Filtering Mechanism for the Network Time Protocol with Chronos," https://datatracker.ietf.org/doc/html/draft-ietf-ntp-chronos-04, 2022.

[38] S. Sengupta, H. Kim, and J. Rexford, "Continuous In-Network Round-Trip Time Monitoring," in *Proceedings of the ACM SIGCOMM Conference*, 2022.

[39] D. Sibold, S. Roettger, and K. Teichel, "Network Time Security," *Internet Engineering Task Force, Internet-Draft draft-ietf-ntp-network-time-security-nn*, 2016.

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

[41] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The Second-generation Onion Router," in *Proceedings of the USENIX Security Symposium*, 2004.

[42] X. Yang, D. Clark, and A. W. Berger, "NIRA: A New Inter-Domain Routing Architecture," *IEEE/ACM Transactions on Networking*, 2007.