NTP Security by Delay-based Detection in Intelligent Defense Systems

A. E. Dinar1, 2, S. Ghouali1, 3, and B. Merabet1
1Faculty of Sciences and Technology, Mustapha Stambouli University, Mascara (29000), Algeria.
2Laboratoire de Sciences et Techniques de l’Eau (LSTE), University of Mascara
3STIC Laboratory, Univ Tlemcen, Algeria.
amina.dinar@univ-mascara.dz

Abstract— Nowadays, computer equipment has hardware or software clocks to which they refer to time stamp files, transactions and emails. The design of a quartz oscillator, such as clocks drift functions like ordinary watches that do not perfectly match. Therefore, it needs networked machines sharing common resources. For instance, UNIX makes command updates key files ensuring that files on which it depends exist and are up-to-date. Also, correlating log messages from several systems becomes very difficult if it does not occur at the same time. This paper focuses mainly on how to detect attacks, trying to predict attacks based on delays caused by this equipment. A server is configured using NTP protocol whose main target is to be implemented in UNIX system, to see how the NTP server is managed with the powerful package Chrony for Ubuntu. The examined results via Python reveal that clients neither will be nor able to make final decisions just after negotiating with servers in several attempts, before or after accepting their clock.

Index Terms— Attacks on Networks; Network Security; Network Time Protocol; Server Synchronization; Python.

I. INTRODUCTION

Deploying firewalls can lead to a false sense of comfort and security if an organization do not perform formal risk analysis, configure firewalls and envisage safety security processes to express its global risk strategy [1]. Important arrangements have to be viewed as applied in switch and firewall to accomplish a viable security approach execution and to avert a few sorts of dangers and assaults [2]. Delays and accumulated-timing jitter impede clock synchronization performance of distributed systems, either in the network, or time stamping procedures of the devices being synchronized [3-6]. In addition, a key goal of system security model is to shield against the expressed classes of dangers and assaults in Layers 2 and 3 of the TCP/IP model [7].

A comprehensive security policy (SP) is needed to provide comparable security to more tailored policies, so as such SP may delay detection of failed attacks, and deterministically and immediately stops successful attempts for any considered attack [8]. Discovery dangers and assaults can be actualized by various strategies, like interruption discovery framework (IDS) for checking systems [9, 10].

Once dangers are recognized, a smart thought is given about the basic system design procedure that needs to take out continuous assaults [11].

The vital security matter with VLAN innovation is a wasteful arrangement and a few arrangements' issues is a prerequisite for the setup technique in exchanging security strategy [12]. The knowledge obtained here may not only be used for synchronizing deployed WLANs but it can be also used to improve the performance of existing wireless and wired networks, which rely on software timestamping.

The present work shows how clients will never be able make final decisions just after a negotiation with servers and in several attempts before actually accepting clocks; the server time is only applied if servers are deemed reliable after negotiations.

The rest of this paper is organized as follows: Section II highlights some previous works related to security and NTP. How to ensure the synchronization of networked equipment is presented in Section III. NTP with Chrony is described in Section IV. The development of our contribution via Python is given in Section V. NTP Chrony comparison task analysis in Section VI. Finally, Section 7 concludes the paper.

II. STATE OF THE ART

The possibility of the security arrangements is developed by using a plethora of security techniques, and they are categorized into several methods, as follows [13-19]:

- Make and fabricate physical security arrangement as a significant point about who is approved and has a consent to set up, uninstall, include or change organize gadgets.
- Portrayal authorizers have the consent to sign in and access to arrange gadgets by immediate access or support port associations and characterize the secret key strategy.
- Change the default password on all network devices and enable secret password for auxiliary port, virtual terminal lines (VTs) ports, and console port.
- To improve the performance of security access to network device, and it is important to specify a minimum password length.
- Restrict the connections via the VTs ports to accept connections with protocols needed, and configure VT timeouts, and adjustment. Also, change the default secret phrase on all system gadgets and empower mystery secret key for assistant port, virtual terminal lines (VTs) ports, and comfort port.
- Confine the associations by means of the VTs ports to acknowledge associations with conventions required, and design VT breaks, and modification of VTs to get just Telnet sessions.
- The AAA server checks the whole association demand, approves the official clients can admit to the system based on their security strategies. Every one of the associations access between VLANs necessary go through AAA server.
• Disabling Unnecessary and Unneeded Services:
  Debilitating IP source directing on the switch, disabling any
  superfluous and unneeded highlights benefits in
  organize gadgets, utilizing IP Security encryption
  (IPSec), SSL, or SSH for all remote access to organize
  gadgets rather than Telnet.

  Shutdown inert interfaces on organize gadgets is an
  advantage that dispirit unapproved, utilize additional
  interfaces to add new system associations with arrange
gadgets, and deactivate a few conventions (CDP, ICMP,
finger convention demands, intermediary ARP convention,
NTP) without influence or diminish organized execution to
hardship programmers, and unapproved usage or tempering
[20].

  In case where NTP protocol is activated, it is necessary to
  monitor each time difference, monitor each delay committed
during an unexpected attack, and synchronization between
network equipment; this article focuses on this idea, and
makes some traceability of this NTP protocol, as shown in
Figures 1 to 3.

The chronology of events also allows errors to be traced on
the same millisecond scale: Traceability ensures a backup, or
automatic backup, at night requiring an accuracy of
about ten seconds [31]. This increases the reliability of daily backups;
the time server allows protecting against time deviations
caused by an electrical frequency that is not stable enough,
which varies permanently around 50Hz in Europe and the
synchronization provided by the server in NTP allows
reliable and robust clustering [32-34].

III. HOW TO ENSURE THE SYNCHRONIZATION OF
NETWORKED EQUIPMENT

Since the manual method has its limits, three protocols
have been designed for this purpose [35]:

A. Time Protocol

  It is the oldest (1983), and it is the subject of RFC868. Relying on UDP or TCP, it can be summarized as the servers
  sending a packet containing the time in seconds elapsed since
  January 1, 1900 at 0H. Time Protocol was used by the UNIX
  timed daemon, but its low resolution and the lack of
  specification of transit time compensation mechanisms led to
  the study of a more sophisticated protocol.

B. Simple Network Time Protocol (SNTP)

  Described in RFC 1361, it is a simplified version of NTP,
  without selection mechanisms, and it is used where accuracy
  in the order of the second is sufficient. A SNTP client can
  synchronize to an NTP server (Figure 4), although it was
designed to implement simple clients. SNTP also allows servers to be implemented, but these must be synchronized directly by a time reference.

C. Network Time Protocol (NTP)

It is the subject of RFC1305 and is in its third version. It is more elaborate than the Time Protocol. It allows the creation of networks of NTP entities with multiple redundancies in order to ensure the permanent and reliable synchronization of the machines concerned. The main contribution to the work on NTP is that of David L. Mills from the University of Delaware [30, 36].

 Filtering and selection algorithms and implementation models are defined in NTP. They allow NTP clients to determine the best source of synchronization, eliminate suspicious sources and correct network transit times at any time. Regarding its implementation, one of the main characteristics of an NTP network is its pyramidal structure [37]. Time references synchronize NTP servers that are directly connected to them. These constitute "stratum" 1, they will each synchronize several dozen other servers that will constitute "stratum" 2 and so on up to the terminal clients. This principle makes it possible to distribute the load of the servers well, while maintaining a "distance" to the relatively small reference sources [38, 29].

NTP is therefore a protocol that allows synchronizing the time of different systems through an IP network. Clients synchronize their clocks with servers. These servers synchronize themselves with other servers and so on. This network is organized in layers called strataums [40, 41].

An NTP (Figure 5) server can operate in the following modes:

- **Simple server mode**: It only responds to requests from its clients.
- **Active symmetric mode**: It asks to be synchronized by other servers and announces to them that it can also synchronize them.
- **Passive symmetric mode**: It is the same as active symmetric mode, but on the initiative of other servers.
- **Broadcast mode**: It is intended for local network. It is limited to the distribution of time information to customers who may be either passive or discover the servers with which they will synchronize.

- **Client mode**: It sends requests to one or more servers.

To synchronize our clocks with our computer network, the most secure and reliable method is to have a dedicated NTP or SNTP server. The architecture in NTPv4 allows a 10x greater time accuracy than the old NTPv3 protocol [43].

The proximity (of the server to the network) provides a minimum latency between the server and our clocks, computers and other equipment.

The implementation of the NTP protocol as well as various drivers used for the connection of time references permit implementing both a simple terminal client and a primary server. The purely NTP part runs on a large number of operating systems: SunOS 4.x, Solaris 2, HP/UX 9.x, Ultrix 4.3, OSF/1, IRIX 4.x, AIX 3.2, A/UX, *BSD, Kali Linux.

Achieving good accuracy depends on how well the messages are identified at:

- **The application level**: UNIX is not a real-time system, it is the least efficient solution, but the easiest to implement.
- **The level of the kernel software queues**: much more precise solution but requires intervention in the kernel.

In our study, our operating system is Linux (Kali), in which we configure the time of our machine and set the system time with timedatectl. This command will display the time information of our system:

```
root@amina-kali:~ # timedatectl
Local time: mar. 2019-08-20 23:33:36 CET
Universal time: mar. 2019-08-20 22:33:36 UTC
RTC time: mar. 2019-08-20 23:33:36
Time zone: Africa/Algiers (CET, +0100)
System clock synchronized: yes
NTP service: inactive
RTC in local TZ: yes
```

- **Warning**: The system is configured to read the RTC time in the local time zone. This mode cannot be fully supported. It will create various problems with time zone changes and daylight saving time adjustments. The RTC time is updated, it relies on external facilities to maintain it. If at all possible, use RTC in UTC by calling:

`timedatectl set-local-rtc 0`
If the clock is not automatically synchronized online, the server time can be configured using set-time:

```
#sudo timedatectl set-time
```

We list the different time zones by list-time zones:

```
#timedatectl list-timezones | grep Algeria
```

The time zone is configured using set-time zone:

```
# sudo timedatectl set-timezone Africa/Algeria
```

One of the largest clusters of public NTP servers is called pool.ntp.org. This one is configured by default in most Linux distributions. Under the latest versions of Linux, the system clock is automatically synchronized in a network. This synchronization is managed by the systemd-timesyned.service service. More information about this service can be accessed by the command:

```
# Systemctl status systemd-timesyned
```

It is therefore possible to synchronize the clock of all the servers on your network by synchronizing each of them with the global NTP network, but as soon as the network grows, it becomes advantageous to have your own NTP server.

There are several other NTP concepts: Stepping, slewing, insane time, drift and jitter.

- **Stepping** is when the time difference between the provider and the consumer is large. Then it will have to make time adjustments very quickly.
- **Slewing** is when the time difference between the provider and the consumer is small, such as less than about 128 milliseconds. Then the NTP protocol is going to adjust the time on the time consumer very gradually.
- **Insane time** is when implementing and maintaining an NTP deployment. This is if the time difference between the provider and the consumer is more than 17 minutes out of sync. Then the ntpd [44] daemon is going to consider time insane, and as a result it is not going to adjust it and that can cause all kinds of problems.
- **Drift** is when NTP measures and corrects for incidental clock frequency errors, which is a fancy way of talking about drift where the system time on one system may not run at exactly the same frequency as the system time on another system.
- **Jitter** refers to the time difference between the time consumer and the time provider since the last polling. Several commands are used to monitor a system working and to closely keep the time synchronized to show the result in Table 1. The first command is:

```
#ntpq -p
```

### Table 1: Result of `ntpq`-p Command

<table>
<thead>
<tr>
<th>remote</th>
<th>refid</th>
<th>st</th>
<th>t</th>
<th>when</th>
<th>poll</th>
<th>reach</th>
<th>delay</th>
<th>offset</th>
<th>jitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.kali.pool.n</td>
<td>POOL</td>
<td>16</td>
<td>p</td>
<td>-</td>
<td>64</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1.kali.pool.n</td>
<td>POOL</td>
<td>16</td>
<td>p</td>
<td>-</td>
<td>64</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2.kali.pool.n</td>
<td>POOL</td>
<td>16</td>
<td>p</td>
<td>-</td>
<td>64</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>3.kali.pool.n</td>
<td>POOL</td>
<td>16</td>
<td>p</td>
<td>-</td>
<td>64</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>ntp.kali.com</td>
<td>POOL</td>
<td>16</td>
<td>p</td>
<td>-</td>
<td>64</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>-102.130.49.223</td>
<td>85.199.214.98</td>
<td>2</td>
<td>u</td>
<td>792</td>
<td>1024</td>
<td>337</td>
<td>242.941</td>
<td>-22.544</td>
<td>4.332</td>
</tr>
<tr>
<td>-ns.bitco.co.za</td>
<td>41.78.128.17</td>
<td>3</td>
<td>u</td>
<td>759</td>
<td>1024</td>
<td>377</td>
<td>242.113</td>
<td>-19.735</td>
<td>4.662</td>
</tr>
<tr>
<td>-160.119.238.133</td>
<td>196.21.187.2</td>
<td>2</td>
<td>u</td>
<td>786</td>
<td>1024</td>
<td>377</td>
<td>260.651</td>
<td>-21.193</td>
<td>8.842</td>
</tr>
<tr>
<td>+chilipepper.can</td>
<td>145.238.203.14</td>
<td>2</td>
<td>u</td>
<td>779</td>
<td>1024</td>
<td>377</td>
<td>91.593</td>
<td>-24.107</td>
<td>2.746</td>
</tr>
<tr>
<td>*pugot.canonical</td>
<td>17.253.108.253</td>
<td>2</td>
<td>u</td>
<td>928</td>
<td>1024</td>
<td>377</td>
<td>85.897</td>
<td>-30.369</td>
<td>7.252</td>
</tr>
<tr>
<td>dbn-ntp.mweb.com</td>
<td>194.58.204.148</td>
<td>2</td>
<td>u</td>
<td>791</td>
<td>1024</td>
<td>377</td>
<td>287.697</td>
<td>-20.849</td>
<td>4.199</td>
</tr>
<tr>
<td>+golem.canonical</td>
<td>145.238.203.14</td>
<td>2</td>
<td>u</td>
<td>311</td>
<td>1024</td>
<td>377</td>
<td>93.789</td>
<td>-32.492</td>
<td>6.703</td>
</tr>
</tbody>
</table>

where: **Remote** = Specifies the hostname address of time provider that’s we’re getting time from  
**Refid** = Indicates the type of time reference source that we’re connecting to  
**st** = Specifies the stratum of that time provider  
**when** = Specifies the number of seconds since the last time poll occurred  
**Poll** = Indicates the number or seconds between tow time polls  
**Reach** = Displays whether or not the time server was reached, the last time was pulled, a successful pole increments this field by one, so as we can see these server were hit 377 times  
**Delay** = Indicates how much time in milliseconds that it took for the time provider to respond to the time request that was sent from the local system  
**Offset** = Specifies the time difference between the local system and the time on the time provider (in milliseconds)  
**Jitter** = Indicates the size of the time discrepancies and again this is measured in millisecond [45]

```
#ntptrace
localhost: stratum 3, offset -0.046775, synch distance 0.152070
```

The `ntptrace` command has been used to monitor the time synchronization that specifies the time provider’s stratum, and to list the time offset between the local system and the time provider. Indeed, having your own NTP server allows you to improve synchronization between network servers, reduce traffic due to time synchronizations on the Internet.
connection, keep servers synchronized even in the event of an Internet outage and avoid unnecessary strain on the global NTP network [39].

IV. NTP SERVER WITH CHRONY

Kali Linux uses Chrony software as the default NTP server. This program is installed by the command [46]:

```
#sudo apt-get install chrony
```

Then, we configure Chrony by editing the file /etc/chrony/chrony.conf. In this configuration file, there is a certain amount of information, such as:

```
#Welcome to the chrony configuration file. See chrony.conf(5) for
#more information about usable directives.
pool 2.debian.pool.ntp.org iburst
#This directive specify the location of the file containing ID/key
#pairs for NTP authentication.
keyfile /etc/chrony/chrony.keys
#This directive specify the file into which chronyd will store the
#rate information.
rfcfile /var/lib/chrony/chrony.drift
#Uncomment the following line to turn logging on log tracking
#measurements statistics
logdir /var/log/chrony
#Stop bad estimates upsetting machine clock.
maxupdateskew 100.0
#This directive enables kernel synchronisation (every 11 minutes)
of the real-time clock. Note that it can't be used along with the
#rtsfile' directive.
Rtcsync
#Step the system clock instead of slewing it if the adjustment is
#larger than one second, but only in the first three clock updates.
Makestep 1 3

The line beginning with pool indicates the address of the NTP servers (or groups of servers more precisely) to be used and the maximum number of resources to be used. A priori, we can continue to use the default selection.

Drift file indicates the file used to record the time drift of the server from the pool. It allows you to resynchronize the clock faster.

By default, Chrony does not allow customers to synchronize with this time service. The clients' network must be authorized by the directive to editing the following line at the end of the file. The address of our network, for example allow 192.168.0/24. We can launch Chrony and activate it when the server starts:

```
#sudo systemctl enable chrony
# sudo systemctl start chrony
```

By default, Chrony listens on UDP port 123 (default port for the NTP service). This port on the firewall need to be opened so that clients can synchronize.

As Chrony is now in charge of synchronizing our system clock, we disable systemd-timesyncd by:

```
$sudo systemctl stop ntp
```

Chrony provides a command line interface to query and manage Chrony: chronyc. We can therefore display the servers with which we are synchronized by the command:

```
sudo chronyc sources
```

<table>
<thead>
<tr>
<th>MS Name / IP Address</th>
<th>Stratum</th>
<th>Poll</th>
<th>Reach</th>
<th>Last Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>^ apollo.slash.co.za</td>
<td>2</td>
<td>10</td>
<td>377</td>
<td>+3157us</td>
</tr>
<tr>
<td>^ ntp4.inx.net.za</td>
<td>2</td>
<td>10</td>
<td>377</td>
<td>+4423us</td>
</tr>
<tr>
<td>^ ntp2.inx.net.za</td>
<td>2</td>
<td>10</td>
<td>377</td>
<td>+348us</td>
</tr>
<tr>
<td>^ ns5.btc.bw</td>
<td>2</td>
<td>10</td>
<td>377</td>
<td>+8697us</td>
</tr>
</tbody>
</table>

The server that starts with ^ is the current time source. Those starting with + are used to calculate an average time and those starting with ^- are not currently used.

V. NTP TIME RESYNCHRONIZATION VIA PYTHON

A. Features of Our System

In order to be able to summarize our system, we are going to summarize it in points [47-52]:

- The NTP system consists of a network of primary and secondary time servers, clients and interconnecting transmission paths.
- System should create a hierarchy of servers, which are self-organizing in nature. For example, there are servers which come under a class Stratum 1, Stratum 2,... Stratum 0 is the reference clocks i.e. our primary source for synchronization. Here, Stratum n+1 server synchronizes its time with stratum n. If any server synchronizes with another server with stratum x, then it becomes stratum x+1.
- NTP client sends request to NTP server to query the time after some designated interval. The NTP server should reply to this request with its own timestamp.
- Synchronization between two servers should give some timing guarantees, meaning the system should bound to some duration of time. If you take any two pair of nodes in the system than they must be in some max allowed time drift with respect to each other.
- System should be scalable. System should handle dynamic addition of new nodes.
- System should be fault tolerant. It should tolerate removal or crash of nodes. It should handle spurious clocks, drop of packets, duplication of packets etc.
- When receiving reply from the server, NTP client shall do some corrections on time to be applied based on statistical metrics e.g. Round trip time RTT, server processing time.
- To ensure reliability, client should negotiate with the server multiple times before actually accepting its clock. The server time is applied only if the server is found to be reliable after the negotiations.
- Client shall reject highly deviated values or outliers if received from server. Client assumes that deviations beyond some bound could not happen if the NTP is working correctly in the first place.
- After calculating the correct time, client should decide
how to apply the time, client does not directly apply the time to itself. Rather it sees the difference and according to that it fastens or slows its clock till some time.

- System shall accept some parameters from the user’s, for example maximum drift allowed or resynchronization time.
- Server can negotiate time with its neighbors (server at the same stratum level) to ensure servers reliability or in case of temporary failure of server.

B. Tools

i. Python

Python is an interpreter, multi-paradigm, cross-platform programming language. It supports structured, functional and object-oriented imperative programming. It has strong dynamic typing, automatic garbage collection memory management, and an exception handling system, making it similar to Perl, Ruby, Smalltalk, and Tcl. The Python language is licensed under a free license similar to the BSD license and runs on most computer platforms, from smartphones to mainframes, from Windows to UNIX, including GNU/Linux, MacOS, Android, iOS, and can also be translated into Java or .NET.

ii. Python-ntplib

This module provides a simple interface for querying NTP servers with Python. It also provides utility functions for translating NTP data into text. It is written in Python only, and depends only on standard modules.

iii. NumPy

It is an extension of the Python programming language, designed to manipulate multidimensional matrices or arrays as well as mathematical functions operating on these arrays. More precisely, this free and open source software library provides multiple functions allowing to directly create an array from a file or on the contrary to save an array in a file, and to manipulate vectors, matrices and polynomials. NumPy is the basis of SciPy, a grouping of Python libraries around scientific computation.

C. Implementation

Algorithm 1

Clock Selection

```
for each server
    correctness_interval=[] #for each server
    jitterlist=[] #select jitter relative to each cadidate in truechimers each element is a list.
    lowpoint=[] #lowest point of correctness interval.
    midpoint=[] #mid-point of correctness interval.
    highpoint=[] #highest point of correctness interval.
    selectjitter=[] #root mean squared of each element in jitterlist.
    temp=[] #lowest and mid highest point in sorted order.
    currentlowpoint=[] #lower end of intersection interval.
    currenthighpoint=[] #higher end of inter section interval.
    minclock=2 #we keep at least 2 clocks as threshold.
```

Input:

```
bufferSize = 256
host = '10.14.1.153'
port = 7845
```

Output:

```
adjustThreshold = 120000000
resyncInterval = 64
numPolls = 5
```

Algorithm 2

Clustering

```
Jitter relative to the i-th candidate is calculated as follows.
Computed as the root mean square (RMS) of the di(j) as j ranges from 1 to n.
Relative jitter for each server
Computed as the root mean square (RMS)
Clustering algorithm to generate final servers
End
```

Algorithm 3

Clock Combining

```
Calculate normalization constant
Finding final offset value
End
```

Notes:

```
def getSinceEpoch()
    current = datetime.datetime.now()
    epoch = datetime.datetime.utcfromtimestamp(0)
    diff = current - epoch
    microseconds = (diff.days * 24 * 60 * 60 +
                    diff.seconds) * 1000000 +
                    diff.microseconds
    # microsecond = (diff.days * 24 * 60 * 60 +
                    diff.seconds) + diff.microseconds/1000000
    tempoffset = ((T2 - T1) + (T3 - T4)) / 2
    tempdelay = (T4 - T1) - (T3 - T2)

getSinceEpoch Example:
```

```
2020-8-8 13:51:35 GMT

Epoch timestamp: 1596894695
Timestamp in milliseconds: 1596894695000
```

Date and time (GMT): Saturday 8 August 2020 13:51:35

`
Minimum delay offset 12644380.0

Received T3: 1595879104520046
getSinceEpoch returning time as 1595879104520131
Monday 27 July 2020 19:45:04:520
getSinceEpoch returning time as 1595879104520237
Monday 27 July 2020 19:45:04:520
getSinceEpoch returning time as 1595879104522207
Monday 27 July 2020 19:45:04:522
Received T1: 1595879104519783
Monday 27 July 2020 19:45:04:519
Received T2: 15958791117162088
Monday 27 July 2020 19:45:17:162
Received T3: 1595879111685678
Monday 27 July 2020 19:45:17:162
Received T4: 1595879104522207
Monday 27 July 2020 19:45:04:522

Minimum delay offset 12642851.5
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879104531409
Monday 27 July 2020 19:45:04:531
Received T1: 1595879104520046
Monday 27 July 2020 19:45:04:520
Received T2: 15958791117162088
Monday 27 July 2020 19:45:17:162
Received T3: 1595879111685678
Monday 27 July 2020 19:45:17:162
Received T4: 1595879104519783
Monday 27 July 2020 19:45:04:519

Minimum delay offset 12642851.5
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879104531879
Monday 27 July 2020 19:45:04:531
Received T1: 1595879104520131
Monday 27 July 2020 19:45:04:520
Received T2: 1595879111717054
Monday 27 July 2020 19:45:17:169
Received T3: 159587911169560
Monday 27 July 2020 19:45:17:169
Received T4: 1595879104531879
Monday 27 July 2020 19:45:04:531

Minimum delay offset 12643544.0
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879104532081
Monday 27 July 2020 19:45:04:532
Received T1: 1595879104520187
Monday 27 July 2020 19:45:04:520
Received T2: 159587917170524
Monday 27 July 2020 19:45:17:170
Received T3: 159587917170601
Monday 27 July 2020 19:45:17:170
Received T4: 1595879104532081
Monday 27 July 2020 19:45:04:532

Minimum delay offset 12644401.5
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879104532187
Monday 27 July 2020 19:45:04:532
Received T1: 1595879104520237
Monday 27 July 2020 19:45:04:520
Received T2: 159587917170583
Monday 27 July 2020 19:45:17:170
Received T3: 159587917170601
Monday 27 July 2020 19:45:17:170
Received T4: 1595879104532187
Monday 27 July 2020 19:45:04:532

Minimum delay offset 12644380.0
Waiting to receive data at peerThread
Printing lists in pollProc
[12641125.0, 12642851.5, 12643544.0, 12644401.5, 12644380.0]
Exception caught min() arg is an empty sequence
Minimum delay offset -36321545.0
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879106528274
Monday 27 July 2020 19:45:05:528
Received T1: 1595879106523784
Monday 27 July 2020 19:45:05:527
Received T2: 1595879106523784
Monday 27 July 2020 19:45:05:523
Received T3: 1595879106523784
Monday 27 July 2020 19:45:05:523
Received T4: 1595879106523784
Monday 27 July 2020 19:45:05:523

Minimum delay offset -36320460.5
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879106528245
Monday 27 July 2020 19:45:06:528
Received T1: 1595879106528245
Monday 27 July 2020 19:45:06:528
Received T2: 1595879106528245
Monday 27 July 2020 19:45:06:523
Received T3: 1595879106528245
Monday 27 July 2020 19:45:06:523
Received T4: 1595879106528245
Monday 27 July 2020 19:45:06:523

Minimum delay offset -36321606.5
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879106523784
Monday 27 July 2020 19:45:05:527
Received T1: 1595879106523784
Monday 27 July 2020 19:45:05:527
Received T2: 1595879106523784
Monday 27 July 2020 19:45:05:523
Received T3: 1595879106523784
Monday 27 July 2020 19:45:05:523
Received T4: 1595879106523784
Monday 27 July 2020 19:45:05:523

Minimum delay offset -36320460.5
Waiting to receive data at peerThread
getSinceEpoch returning time as 1595879106528245
Monday 27 July 2020 19:45:06:528
Received T1: 1595879106528245
Monday 27 July 2020 19:45:06:528
Received T2: 1595879106528245
Monday 27 July 2020 19:45:06:523
Received T3: 1595879106528245
Monday 27 July 2020 19:45:06:523
Received T4: 1595879106528245
Monday 27 July 2020 19:45:06:523
from these results, we can clearly see that our NTP client sends a request to the NTP server to ask for the time after a certain designated interval; when a response is received, i.e. an acknowledgement of receipt, the NTP client has to make certain corrections within the time limits to be applied based on measures such as the RTT ‘Round Trip Time’ and the server processing time.

After calculating and deciding the accurate time, clients apply time differences and depending on this, they accelerate or slow down their clocks to certain times. Such delays and resynchronizations predict clearly unexpected attacks from strangers. Worth noting that it is mandatory for intelligent defense systems to counter unknown attackers via assessing the delays using NTP.

VI. NTP CHRONY COMPARISON TASKS

NTP underpins the Auto key convention (RFC 5906) to validate servers with open key cryptography. Note that the convention has been demonstrated to be unreliable and it will be presumably supplanted with a usage of the NTS. Specifically, NTP has been ported to even more working frameworks, as it incorporates an enormous number of drivers for different equipment reference timekeepers.

Chrony requires different projects to give reference time by means of the SHM or SOCK interface, and it can perform helpfully in a situation where access to time reference is irregular.

NTP needs normal surveying of the reference to function better; it can, as a rule synchronize the clock quicker and with more time precision. It rapidly adjusts to unexpected changes clock (for example because of changes in the temperature of the precious stone oscillator); he can perform well not withstanding when the system is clogged for longer timeframes.

Chrony bolsters equipment times tamping on Linux, which permits very exact synchronization on neighborhood systems, it offers help to work out the addition or misfortune pace of the continuous clock, for example the clock that keeps up when the PC is killed. It can utilize this information when the framework boots to set the framework time from a redressed adaptation of the ongoing clock. These continuous clock offices are just accessible on Linux, up until now [46, 53].

VII. CONCLUSION, PERSPECTIVES AND ADVICES

In this paper, we have analyzed various delays and jitter sources involved in software time stamping-based clock synchronization over WLAN. We have tried to quantify the delay and the jitter not only from time stamping and WLAN chipset. For any hubs on the Internet, neighborhood clock (for example framework clock) is imperative to record time stamps and agreeably work with different hubs. The nearby clock deferrals and continues from the right time because of different variables. Accordingly, the nearby clock is commonly balanced and synchronized with the standard time by arranged time synchronization administration.

At the present time, all PC gear has equipment or programming clock to which reference is set aside a few minutes stamp records, exchanges, messages, etc. This clock, albeit structured around a quartz oscillator, floats like any normal watch. This is significantly additionally lowering when machines are arranged and share basic assets, for example, document frameworks. For instance, some advanced apparatuses such as the UNIX make direction, based their work on contrasting record change dates. Therefore, the relationship of log messages from a few frameworks turns out to be exceptionally troublesome in the event that they are not simultaneously. In this article, we concentrate on this subject by designing a Server utilizing the NTP convention since the primary focus of the NTP execution is UNIX frameworks, to be progressively unequivocal; we see the administration of the NTP Server with the Chrony tool. On a local network, the use of broadcast mode makes it possible to simplify the configuration of clients. Distribute the load well by setting up as many layers as necessary, in particular so as not to overload the public reference servers. Soon, we will use versions of xntpd: only the redacted versions of the DES are exportable from the US, they carry the word export in their name and are sometimes several numbers late compared to the current version, as xntpd continues to evolve rapidly. Our research has led us to study in the near future, how to manage system delays for
synchronizing cloud computing with heterogeneous/homogeneous servers, where each server can have a different average service rate. In future, we plan to develop an NTP client that implements both the TTL/HL reflecting method and high-precision time stamping. As a perspective, we want to propose an event detection module for attacks, utilizing the information from network time synchronization service.

**NOMENCLATURE**

| AAA | Authentication, Authorization and Accounting |
| ARP | Address Resolution Protocol |
| CDP | Cisco Discovery Protocol |
| DDoS | Distributed Denial of Service |
| DES | Data Encryption Standard |
| GPS | Global Positioning System |
| HTTP | Hypertext Transfer Protocol |
| ICMP | Internet Control Message Protocol |
| IDS | Intrusion recognition framework |
| IP | Internet Protocol |
| NTP | Network Time Protocol |
| NTS | Network Time Security |
| PC | Personal computer |
| PTP | Precision Time Protocol |
| RFC | Request for comments |
| SNTP | Simple Network Time Protocol |
| SSH | Secure Shell |
| SSL | Secure Sockets Layer |
| SYN | Synchronous |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| TTL | Time To Live |
| UDP | User Datagram Protocol |
| VLAN | Virtual Local Area Network |
| VT’s | Virtual terminal lines |

**REFERENCES**


[29] Li X ZH. “Research on the Network Time Synchronization System Based on IEEE1588”, National Time Service Center, Chinese Academy of Sciences.2011.


