TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Rene Allkivi 135149IASB

PEER-TO-PEER CLOCK SYNCHRONIZATION USING IEEE 802.11

Bachelor’s thesis

Supervisor: Taaniel Uleksin
MSc
Co-supervisor: Andres Rähni
MSc

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rene Allkivi 135149IASB

PEER-TO-PEER KELLADE SÜNKRONISEERIMINE WI-FI VÕRGUS

Bakalureusetöö

Juhendaja:	Taaniel Uleksin
	MSc
Kaasjuhendaja:	Andres Rähni
	MSc

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Rene Allkivi

02.01.2019

Abstract

The goal of this thesis is to develop a system to synchronize Digital Sputnik Voyager lamps in a wi-fi network without using the internet or a dedicated clock server. We research existing methods and protocols with the aim of finding a suitable solution for adoption.

No suitable free and open source software was found, so a hybrid of the two most common protocols, PTP and NTP, was developed.

This thesis is written in English and is 35 pages long, including 8 chapters, 5 figures, and 3 tables.

Annotatsioon

Peer-to-peer kellade sünkroniseerimine Wi-Fi võrgus

Antud töö eesmärgiks on töötada välja süsteem Digital Sputnik Voyager lampide omavaheliseks sünkroniseerimiseks wi-fi võrgus.

Töös antakse ülevaade levinud sünkroniseerimismeetoditest ja -protokollidest, ning võrreldakse saadavalolevaid tarkvaralahendusi et valida neist sobivaim.

Analüüsides selgub, et sobivat vabavaralist lahendust hetkel veel ei eksisteeri. Voyager lampide sünkroniseerimiseks luuakse uus hübriidprotokoll, mis kasutab kombinatsiooni levinud protokollidest NTP ning PTP.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 35 leheküljel, 8 peatükki, 5 joonist, 3 tabelit.

List of abbreviations and terms

NTP	Network Time Protocol
DMX	Digital Multiplex
GPS	Global Positioning System
IP	Internet Protocol
PTP	Precision Time Protocol
SNTP	Simple Network Time Protocol
LAN	Local area network

Table of contents

List of figures

List of tables

1 Introduction

Modern stage and film lights are sophisticated devices that allow users to control their intensity, colors and even beam direction remotely. Remote-controlled lights almost always use a master-slave architecture. The control logic and animation data reside in the master controller which sends data to “dumb” devices by wire over protocols such as DMX512 [1].

Digital Sputnik Voyager lights differ from this traditional scheme in that they are engineered to be controlled wirelessly over wi-fi with computers and smartphones instead of dedicated control surfaces. When using no wires, setting up lamps takes less time and less equipment is needed. Using smartphones causes multiple issues which makes old protocols unsuitable. Firstly, wi-fi cannot guarantee low latency. Indeed, it can be the “weakest link” in a computer network [2]. Secondly, the user might want to use her smartphone for other purposes once everything has been set up.

Figure 1: Digital Sputnik Voyager lights [3]

As an alternative to master-slave architecture, the controlling logic runs on lights. In this new system, the controller sends animation instructions to the lights. For example, to create a strobing light, a single command telling the lamp to toggle on-off every 0.01 seconds can be sent. Once the lamp has received this instruction, it can continue to strobe even if the controller is removed from the network.

DMX works without any synchronization, but because latencies are very low, all devices display the received data immediately. Voyager lights do not have a continuous data stream, however, therefore we need a new way to synchronize the animations.

This thesis aims to research existing synchronization protocols with the aim of adopting an existing solution for use in the Digital Sputnik ecosystem.

2 Requirements

To synchronize animations running on lamps, we need to synchronize their clocks, but we do not need to set them to a time standard such as Coordinated Universal Time (UTC) or International Atomic Time (TAI).

2.1 Network environment

Digital Sputnik does not supply network equipment, and users often use older consumer-grade wireless access points in congested network environments. Sometimes, users configure the lamps before taking them to a movie set in a remote location without a connection. Vast distances can be involved, which means the quality of the connection is sometimes low with many packets being dropped. Therefore, the synchronization must also work with an intermittent connection. Multicast is sometimes not available in wi-fi networks [4], so the system must also work without multicast support. The devices involved in controlling lamps are shown in Figure 2.

2.2 Software environment

A set-up consists of Voyager lights which include an embedded computer running Ubuntu Linux. A multiplatform controller application running on the user's smartphone or laptop allows her to modify lamps’ settings and presets while also offering monitoring capabilities. For accurate monitoring, the controller application also must be synchronized. Therefore, the solution must be usable as a library in the multiplatform controller software and also run on Ubuntu Linux. The CPU load of the synchronization software on Ubuntu should be <1%

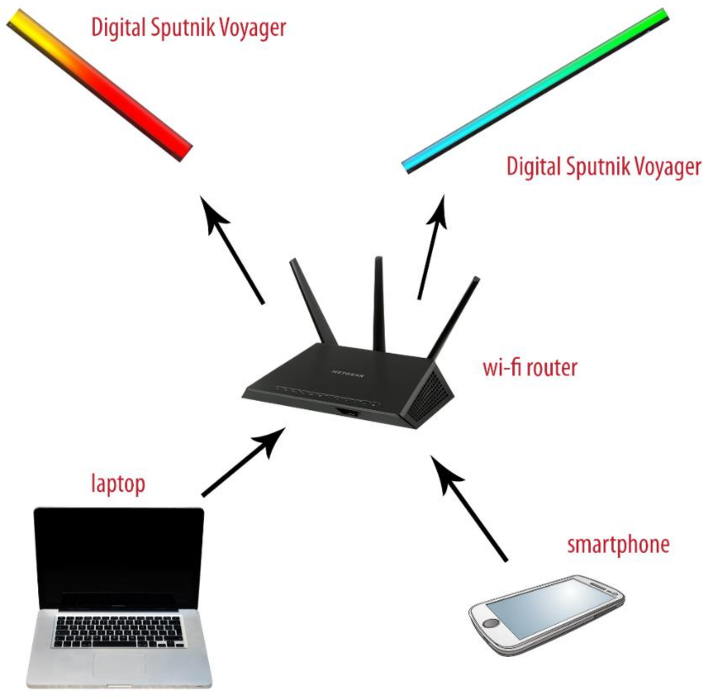

Figure 2: Typical network environment for Voyager lights

2.3 Accuracy

The animations engine runs at 200Hz. Synchronization accuracy should be high enough to satisfy filmmakers. Cinematographers should not be able to notice any deviations when watching slow-motion filmed footage.

When setting up lamps for a scene, quick startup takes priority over initial accuracy. It is important to avoid holding up production because making actors, directors and other crew wait is expensive.

After clocks are fully synchronized, they can be removed from the wi-fi network and must remain in sync for at least three days.

The multiplatform controller application should also be synchronized to the network so that animations can be monitored from a computer screen.

The accuracy requirements were defined to be as follows:

- Initial sync time - <2 seconds
- Fully synced - <60 seconds
- Initial accuracy - <50 ms
- Full accuracy - <5ms
- Controller sync accuracy - <200ms

2.4 Security

It should not be possible for any third-party unauthorized users to modify lamps' settings and animations. However, the lights are meant to be used in a secure wi-fi network, so additional application-level user verification is not necessary.

3 Synchronization methods

Most devices can use either of the two standard options of getting an accurate clock.

The first option is the use of radio time sources such as GPS or shortwave radio broadcasts. GPS offers nano-second level accuracy, but using this signal requires an additional GPS receiver at each node [5]. GPS cannot be used in buildings because of poor reception. Most computers do not require GPS/level accuracy and sync to a time server over the internet [6].

There are thousands of time servers available on the internet [7], including many servers that are themselves directly connected to an atomic clock or GPS receiver [8].

For us, neither radio nor an internet time server is suitable. Voyager lights are often used in remote locations where neither GPS nor internet is available. Therefore, a peer-to-peer solution needs to be devised to allow devices' clocks to sync with each other. Most existing protocols are concerned with syncing devices to a reference clock, not with syncing a cluster of devices amongst themselves [9] [10]. We need to examine the protocols to determine whether they are suitable for use on local area networks not connected to the internet.

3.1 Einstein synchronization

IP-based protocols are all based on the symmetry described in Einstein's 1905 paper "On the electrodynamics of moving bodies [11]."

If we have two clocks at points A and B, and we know that the time light travels from A to B equals the transit time in the opposite direction, the two clocks can be synchronized. If a ray of light is sent from point A at time $\boldsymbol{t_1}$, it arrives at point B at time $\boldsymbol{t_2}$ and is immediately reflected back towards point A where it arrives at time $\boldsymbol{t_3}$. As the transit times are equal, we can deduct the following.

$$t_3 - t_2 = t_2 - t_1$$

If the timestamp $\boldsymbol{t_2}$ can be transmitted or transported to point A, it is possible to find the time differences between the clocks.

While computer networks use a combination of radio, electrical and light signals, the principle remains the same. In an ideal computer network, all latencies are symmetrical, and the transit time can be computed by measuring round trip time and dividing this by two. Unfortunately, in the real world, the transit times are never symmetrical [12], and additional delays can be caused by scheduling algorithms in the software layer [10].

Sometimes, a time server connected to a reference clock is not available, and one or multiple intermediate proxies are used. The primary goal of a time protocol is to choose the best available time server and combine multiple measurements to estimate errors and achieve higher precision [13]. For example, statistics can be used to estimate the asymmetry of connections. If there appears to be more variability from client to server, we can assume that the latency in this direction is also higher [12].

3.2 IP-based clock synchronization protocols

Next, let’s look at the available synchronization protocols to identify which is more suitable for use with Voyager lights. There are two major protocols for IP networks – Network Time Protocol (NTP) and Precision Time Protocol (PTP).

They are both designed to use a master server directly connected to a reference clock such a GPS probe. All devices synchronize by connecting to this master server via a minimal amount of proxies [10] [9].

There are no reference clocks available when using Voyagers in a remote location so we can disregard many features of the protocols. The following aspects and questions will be considered most important when making the decision:

1. Compatibility with reference-less networks: Can a client be ordered to synchronize to a master server that has not been synced to a reference clock?

2. Peer discovery: Can clocks automatically discover other clocks in the local network?

3. Availability of open source software

3.3 NTP

NTP was first documented in 1985, and the basic functionality remains the same to this day.

Servers and clients use the same NTP message format. When using asymmetric mode, the client initiates by sending a time-stamped packet to the server. The server fills in the packet with the transmit and receive timestamps and sends it back. Servers do not need to store information about the clients and can, therefore, be quite lightweight. The client periodically combines multiple measurements and servers to get a more accurate reading. Of course, clients prefer sources with fewer hops to the reference clock. The periodic correction causes the clock to be shifted instantaneously, which can cause problems with real-time applications. Newer versions of NTP mitigate these problems by making multiple micro-jumps instead of a single correction to the desired value. The system clock's drift rate can also be computed over time and be used to correct the clock's speed [10].

3.3.1 Reference-less networks

The NTP standard presumes a reference clock such as a GPS signal is available. Peers prefer servers that are connected to the reference clock with the least amount of nodes. Clients refuse to sync to servers that have never been synchronized to a reference clock [10]. A workaround has to be used to convince a client into using these reference-less servers.

3.3.2 Peer discovery

Usually, each client is configured manually by inputting a master server address. Broadcast and multicast modes also exist which allow a server to announce its timestamp. Broadcast and multicast messages allow clients to listen in, "discover" the server, and start using it as a master [10].

3.3.3 Software availability

NTP is used to set the clock automatically by many operating systems. Libraries, applications, and thorough documentation exist for most programming languages and platforms [14].

3.4 PTP

PTP was first defined in the **IEEE 1588-2002** standard called "*Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems"* as a higher precision alternative to NTP. Unlike NTP, it is designed from the ground up to be used in local networks. A typical system includes a dedicated master server, while a simple system includes identical clocks which elect a grandmaster (master server) using the *Best master clock algorithm.*

A PTP packet is timestamped in the network interface at the moment of transmission, eliminating errors caused by buffers and software scheduling. For higher accuracy, routing devices supporting transparent PTP mode are used. A transparent device measures the time it takes a packet to travel from its input to output and corrects the PTP packets' timestamps accordingly [9].

3.4.1 Reference-less networks

PTP is designed for synchronization of clocks in a local network and does not presume the presence of a reference clock. It is possible for the user to set the priority of each node manually to force all clients to synchronize to a chosen server [9].

3.4.2 Peer discovery

Each LAN can only have one master at a time. The server-capable devices on a network negotiate to find the best available master server. Once negotiated, all other devices sync themselves to the chosen master [9].

3.4.3 Software availability

PTP is a newer and more specialized solution and fewer libraries are available compared to NTP. It is intended to be used in an environment with a dedicated high-precision hardware clock device. Open source solutions are available only for Linux and BSD systems [15]. While timestamping should be done in hardware, it is also possible to timestamp in the Linux kernel. Software timestamping will, however, cause lower accuracy and reduce the precision advantages compared to NTP [16].

3.5 Comparing NTP and PTP

After initial research, the following table of most important features was compiled comparing the NTP and PTP protocols suitability for peer-to-peer synchronization.

Table 1: Comparison of NTP and PTP protocols

Feature	**NTP**	**PTP**
Timestamping	Software-based.	Hardware-based.
Hardware requirements	None.	PTP-capable network devices are recommended.
Reference-less networks	Needs workarounds.	Supported.
Accuracy	>1 ms.	<1 µs.
Software availability	All platforms, multiple solutions.	Specialized platforms, few solutions.
Peer discovery	None, each client selects which master server to use separately.	Negotiated on local subnet level using best master clock algorithm.

While PTP offers better accuracy and peer discovery, there are no open source implementations available for use in our multiplatform controller application. Lack of libraries leaves us with no option than to go with NTP. Furthermore, when developing in-house an application level library, it is unknown what the accuracy would be without making further research.

The biggest shortcoming of NTP is related to peer discovery. For easy testability and maintainability, it would be preferable to use a single master server at a time, which is precisely the PTP way of doing things.

Why not use NTP broadcast or multicast modes? These are both ways of masters to announce their presence in the network. If all devices are configured as broadcast or multicast servers, every peer attempts to synchronize to all other devices. In a network of tens of peers, this would cause a lot of traffic and slow synchronization. It is also not possible in these modes for a peer to be a server and client at the same time, as this would cause a feedback loop, which is detected by NTP causing the client to ignore the server. With all devices being servers at the same time, it would be impossible for synchronization to take place.

The decision was made to develop a PTP-inspired master selection algorithm to be used with NTP synchronization.

4 Existing NTP software packages

In the following chapter, NTP libraries and applications will be compared.

Using the same library on all platforms would simplify the development and testing. Initial research divided the available software packages into the following two categories:

- Portable SNTP (Simple NTP) client-only libraries with minimal, meant to be used for application-level synchronization when the system clock cannot be trusted
- UNIX software implementing a large subset of NTP

For the first category, the following open source solutions were considered:

- GuerrillaNtp
- Arduino-libraries NTPClient
- Ntplib
- Nettime
- Ntpclient

For the second category, there are fewer options [17]:

- ntp (reference implementation)
- chrony
- openntpd

Unfortunately, all portable implementations turned out to only implement SNTP, which is a small subset of the full protocol [10]. SNTP uses a single measurement to determine the clock. This solution achieves lower accuracy and is not suitable for a high-jitter environment.

Because it is not possible to use any of the options from the second category in the multiplatform controller software, we decided to use different software for the controller and lamp. The controller software is written in C#, therefore it would also be most convenient to use a C# NTP library. We chose to use GuerrillaNtp, because it was the only one to be written in C#. High precision is not required for the controller application, so a simple library is sufficient.

On the Voyager side, however, high precision is required and therefore a more sophisticated software package needs to be used.

4.1 Comparison of UNIX software solutions

Chrony makers have compiled a thorough comparison of all three options. We eliminated openntpd from the contest right away because it does not support changing the polling interval [18]. Without adjustable polling, it’s impossible to achieve a fast initial synchronization time.

Let’s look at the accuracies of chrony and the ntp reference implementation in a simulated Linux environment. The measurements were made using *clknetsim.* Mean values and deviations were obtained by running 100 simulations. Table 2 shows the accuracy in a stable network environment [18]. In the second test, shown on table 3, the network was available to clients only for 30 continuous minutes in a 24 hour period. A similar use case to the second test would be the configuration of lamps indoors before taking them outdoors to be used on a set without a wi-fi network.

Chrony was more accurate in all scenarios except for the permanent network connection with 1000 μs jitter, where the difference was less than 5 percent. With the intermittent network connection, differences were colossal, with Chrony being an order of magnitude more accurate.

Moving on to the feature set, while ntp is a full implementation of the NTP protocol [18], we are not planning to use any of the additional features such as manycast and multicast modes, or autokey. Chrony has been designed to be used in a network where access to the master server is intermittent, while ntp is an older solution that goes back to times when all devices used wired internet. Furthermore, ntp is also more resource-intensive, using more memory and CPU time [18].

Table 2: NTP accuracy with permanent network connection and stable clock [18]

Network jitter [µs]	**Chrony [µs]**	**Ntp reference implementation [µs]**
10	35 ± 8	234 ± 46
100	109 ± 14	256 ± 50
1000	475 ± 93	454 ± 94
10 000	1603 ± 447	3665 ± 651

Table 3: NTP accuracy with intermittent network connection [18]

Network jitter [µs]	**Chrony [µs]**	**Ntp reference implementation [µs]**
10	7273 ± 1744	608803 ± 510468
100	9528 ± 1895	580679 ± 481379
1000	10706 ± 2521	1115961 ± 733914
10 000	26105 ± 70408	897703 ± 847901

Chrony claims to synchronize the clock faster and with higher accuracy, adapts to changes in the clock’s frequency [18]. This would be useful when immersing the waterproof Voyager lamps in water, causing the temperature to drop.

5 Integration

Before starting work on the discovery protocol, the chosen software packages were tested in various environments to validate their accordance with the defined functional and non-functional requirements.

5.1 Chrony

Pre-compiled Chrony packages were available from the main Ubuntu repository [19]. Unfortunately, the package provided with Ubuntu (2.1.1 from June 2015) was an older version which was more than two years old. Meanwhile, many enhancements had taken place in Chrony, most importantly "estimation and correction of asymmetric network jitter." Packages with the newest version were compiled for use on the Voyager lights.

Chrony works by directly adjusting the system clock and the animation engine running on the Voyager lights is synchronized to the system clock, so no additional interfacing was required to make the two cooperate.

First tests were made without any effort being placed on configuration and all servers being added to clients manually via the built-in command-line interface. The local directive was used to convince clients that the server is serving valid time, despite the server never having been connected to a reference clock. With the default configuration, the following issues became apparent.

- Initial sync taking too long, over 10 seconds
- Server randomly failing NTP validity checks and being regarded as "falseticker"
- Client not stepping the clock, when the deviation is small (a few seconds)

Thankfully, Chrony offers a wide assortment of configurable preferences. The aforementioned issues were fixable one-by-one.

First, to speed up initial sync *initstepslew* directive was used to force rapid polling and clock stepping on start-up.

The trust parameter was added to force clients into using the specified servers. This made the client forego any validity checks and always assume that the source is always true. Unfortunately, bypassing all checks brought up multiple issues a few weeks later. In some sporadic cases, overloading a wi-fi access point caused very high latencies of over 10 seconds. With the *trust* parameter, the client continued to trust the servers even though the accuracy of measurements with such high latencies is very low, causing the lights to desynchronize. Replacing the *trust* parameter with a manually set maximum round trip time of 1 second made the client ignore the packets with ultra-high latencies.

Last, to force more aggressive clock stepping, the *makestep* directive was used to reduce the maximum deviation before the clock is stepped to 0.05 seconds.

After making the changes in preferences, initial synchronization speed was less than 2 seconds.

5.2 GuerrillaNtp client

GuerrillaNtp provides a C# class that can be used to get the offset of the host system's time from server time [20]. There is no built-in polling method, so it is up to the host application to poll when necessary. Querying the offset worked on the first try, we compared the output to the clock on the host server manually. No effort was made to measure the accuracy of GuerillaNTP because the latency of a wi-fi connection is almost always smaller than the required synchronization accuracy.

5.3 GuerrillaNtp server

As an additional exercise, server functionality was implemented into GuerrillaNtp. In the future, a need to synchronize the clocks to real-time could arise, and for this purpose synchronizing to the user's laptop or smartphone is a simple solution.

6 Discovery protocol

Because none of the available NTP solutions supported the protocol's broadcast option, we had two options:

- Implement NTP broadcast client functionality into GuerrillaNtp and Chrony
- Implement discovery separately

At first, we researched the feasibility of implementing NTP broadcast functionality. Unfortunately, obstacles came up quickly. Because the system is a peer-to-peer system, every device is necessarily also a server. When using broadcast mode, this would mean that all devices would have to broadcast time stamps all the time. When all lamps on the network are equal, clients will use all available devices as masters. With tens of devices, it is impossible to predict what the outcome would be, but it is certain that quick synchronization would not be guaranteed. Using one device at a time as the master would be more effective.

Looking back at the protocol comparison, we discovered that the PTP master selection algorithm is perfect for use in a local network. Therefore the decision was made to develop discovery software based on the PTP discovery mechanism while using NTP for actual clock synchronization.

This could be achieved via a discovery script that handles master negotiations and passes the current server's IP address to GuerrillaNtp and chrony. The hierarchy is depicted in Figure 3.

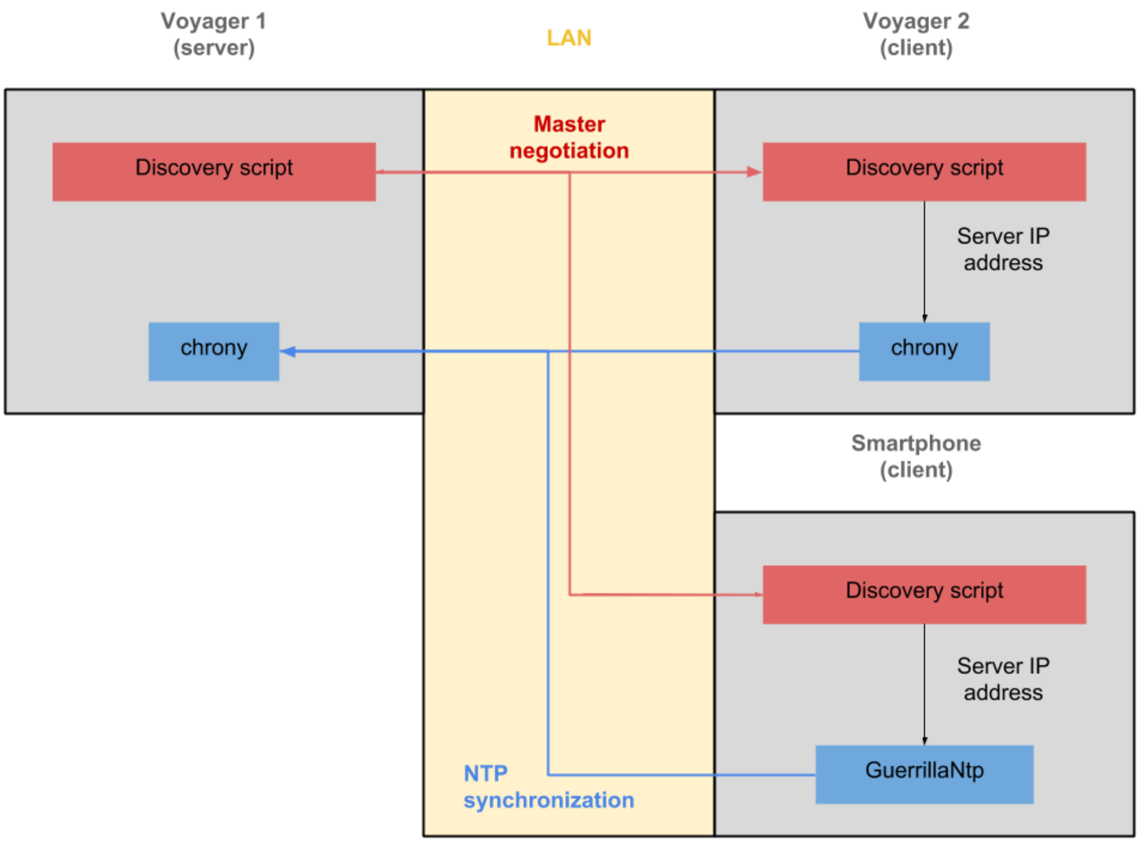

Figure 3: Communications of discovery scripts and NTP applications

6.1 PTP discovery mechanism

Although PTP servers use multicast for announcing, multicast is often blocked on wi-fi networks. Broadcasting was chosen instead to make the system compatible with more networks.

PTP uses *Announce* messages to select the master. The master "advertises" itself by sending *Announce* packets at regular intervals. Whenever a clock joins a LAN, it listens for announces, and if it receives none with a higher priority, it starts sending out *Announce* packets itself. Whenever a current master receives a higher priority *Announce* packet, it accepts the new master and stops announcing itself announcing itself. The protocol does not take into account the quality of the network connection of nodes, only the contents of the *Announce* packet are considered [9].

6.2 Discovery implementation

First tests were carried out without any synchronization capabilities. A Python script was written closely following the PTP announce packet principles. The announce interval was initially set to 2 seconds and timeout to 8 seconds. This script was installed on Voyager lamps and the logs were monitored. In a network of three lamps, everything worked perfectly. When the number of devices was increased to ten, random master switching started taking place. The culprit was traced to dropped packets. When the consequent announces of the master were lost for some reason, all other devices started announcing their presence. Decreasing the announce interval to 1 second and sending out an additional duplicate packet fixed this issue.

6.2.1 Chrony

Next, the discovery had to be set up to add the current master server to Chrony. Thankfully. Chrony has a command-line interface that can be used to modify various parameters while the daemon is running.

Every time a new master server is selected, the script first removes the current server (if there is one), and then adds a new server.

6.2.2 GuerrillaNtp

Once the Chrony-based solution was working, an identical C# library also had to be implemented. For ease of use, the decision was made to separate the whole system into three parts:

1. GuerrillaNtp client
2. Discovery service
3. Voyager NTP service, which passes the discovered server IP to GuerrillaNtp automatically

7 Validation

Before releasing the synchronization to clients, in-house testing was conducted to validate the final system's compliance with the requirements.

7.1 Estimated accuracy

First, two lights were synchronized and set to display a chase[1] animation. Next, a high-speed camera was used to film the two lights side-by-side at 240 frames per second. In the resulting frames shown in Figure 4, lights are set up vertically.

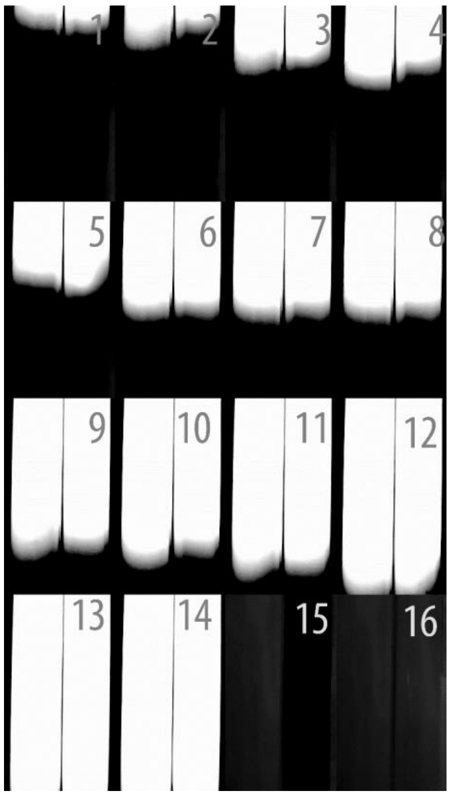

Figure 4: Frames from video of two Voyager lights

[1] chase - an electrical application where strings of adjacent light bulbs cycle on and off frequently to give the illusion of lights moving along the string [21]

The accuracy set up in the requirements was <5 ms. The time between frames is 1/240 seconds = 4.2 ms. Identical pixels are illuminated on both lamps in all frames, which translates to an accuracy of at least 4.2 ms. The desired goal was achieved, and no further testing was deemed necessary.

7.2 Initial sync delay and accuracy

A Python script was written to measure the initial sync delay. The script started the synchronization application and checked the clock every 1 ms until a clock correction was detected. A master clock was available and continuously announcing on the network. This test was repeated 10 times, the results are shown in Figure 5.

The mean value of the initial sync delay was calculated to be 1.05±0.34 with 95% confidence. This is well within the required maximum initial sync delay of 2 seconds.

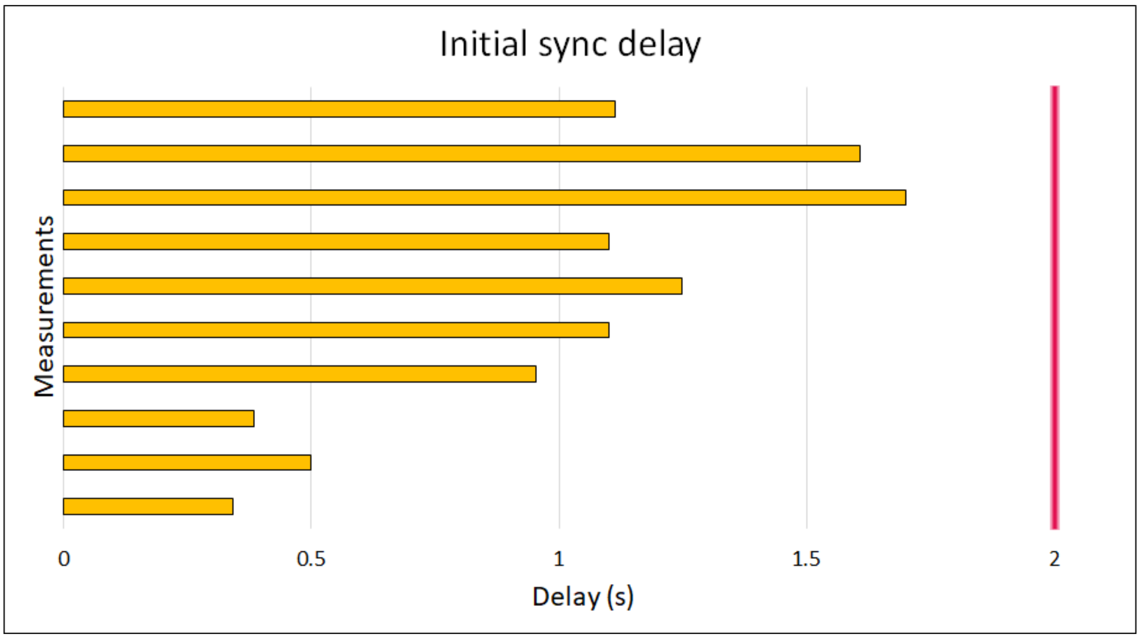

Figure 5: Initial sync delay measurements

The initial sync accuracy was not measured, but visual observation of animations confirmed that the clocks, and therefore the animations, were indeed in sync within 2 seconds.

7.3 CPU and memory use

The CPU utilization was monitored by running the Linux ps (process status) application every 5 seconds and piping the output to a file. The CPU utilization shown by ps was 0% at all times for both the discovery and chrony processes. This was satisfactory and no further measurements were made.

8 Summary

The purpose of this study was to develop or adopt a suitable solution to synchronize Digital Sputnik Voyager lights in wi-fi network. We compared the existing software and discovered that there are no existing free and open source solutions for time reference-less peer to peer synchronization in wi-fi networks.

The two most popular internet protocol based time synchronization based are Network Time Protocol (NTP) and Precision Time Protocol (PTP). PTP is a nearly perfect solution for our use case, but its peer discovery does not work in networks where multicast has been blocked, and there are very few PTP implementations available. No open source PTP applications were found for operating systems other than Linux.

As a solution, a hybrid of NTP and PTP was developed, which uses NTP for synchronization and a subset of PTP for peer discovery. Ready-made NTP applications, chrony and GuerrillaNtp, were used for the synchronization. For the peer discovery, applications were developed from scratch in Python and C#.

References

[1] "Stage-lighting with DMX512 protocol," *Elektor Electronics,* pp. 52-55, February 1998.

[2] . P. Changhua, Z. Youjian, C. Guo, T. Ruming, M. Yuan, M. Minghua, L. Ken and P. Dan, "WiFi can be the weakest link of round trip network latency in the wild," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, San Francisco, 2016.

[3] "Digital Sputnik," Digital Sputnik, 2018. [Online]. Available: https://www.digitalsputnik.com/. [Accessed 10 December 2018].

[4] C. P. M. McBride, "Internet Engineering Task Force," April 2018. [Online]. Available: https://tools.ietf.org/id/draft-mcbride-mboned-wifi-mcast-problem-statement-01.html. [Accessed 10 December 2018].

[5] R. Kim, T. Nagayama, H. Jo and B. F. Spencer Jr., "Preliminary study of low-cost GPS receivers for time synchronization of wireless sensor networks," *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems*, 2012.

[6] M. Lombardi, "Computer Time Synchronization," National Institute of Standards and Technology, 1993.

[7] "All NTP Pool Servers," NTP Pool Project, 2018. [Online]. Available: https://www.ntppool.org/zone. [Accessed 15 December 2018].

[8] "Stratum One Time Servers," Network Time Foundation, 2018. [Online]. Available: https://support.ntp.org/bin/view/Servers/StratumOneTimeServers. [Accessed 20 December 2018].

[9] *Precision Time Protocol, IEEE 1588-2008,* IEEE, 2008.

[10] *Simple Network Time Protocol (SNTP) Version 4, RFC 4330,* 2006.

[11] A. Einstein, "Zur Elektrodynamik bewegter Körper," *Annalen der Physik,* vol. 322, no. 10, pp. 891-921, 1905.

[12] N. Simanic, R. Exel, P. Loschmidt, T. Bigler and N. Kero, "Compensation of asymmetrical latency for ethernet clock synchronization," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, Munich, 2011.

[13] "What is NTP?," Network Time Foundation, 2018. [Online]. Available: http://www.ntp.org/ntpfaq/NTP-s-def.htm. [Accessed 12 December 2018].

[14] *Network Time Protocol Version 4: Protocol and Algorithms Specification, RCF standard 5905,* 2010.

[15] "List of PTP implementations," [Online]. Available: https://en.wikipedia.org/wiki/List_of_PTP_implementations.

[16] EndRun Technologies, “Precision Time Protocol whitepaper,” EndRun Technologies, [Online]. Available: https://www.endruntechnologies.com/pdf/PTP-1588.pdf.

[17] “Network Time Protocol,” Wikipedia, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Network_Time_Protocol#Software_implementations. [Accessed 13 December 2018].

[18] “chrony homepage,” Miroslav Lichvar, 2018. [Online]. Available: https://chrony.tuxfamily.org/comparison.html. [Accessed 21 December 2018].

[19] “Ubuntu - details of package chrony in xenial,” Ubuntu, 2018. [Online]. Available: https://packages.ubuntu.com/xenial/admin/chrony. [Accessed 29 December 2018].

[20] R. Važan , “GuerrillaNTP,” 2018. [Online]. Available: https://guerrillantp.machinezoo.com/. [Accessed 10 December 2018].

[21] “Chase (lighting),” Wikipedia, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Chase_(lighting). [Accessed 15 December 2018].