Covert Message Channels and Attack Vectors for IEEE Precision Time Protocol

Luke Jacobs *, Casimer DeCusatis *, Paul Wojciak **, Clay Kaiser **, and Steve Guendert **

* Marist College, Poughkeepsie, NY
** IBM Corporation, Poughkeepsie, NY USA

Abstract— **The IEEE 1588 standard, known as Precision Time Protocol (PTP), is an emerging candidate for high precision timing and clock distribution networks. We present experimental results from a PTP test bed that demonstrate new types of covert channel communications, which allow PTP protocol to be used for data exfiltration and other network communication that violates the implemented cybersecurity policy. We then expand upon this work to demonstrate two new zero-day vulnerabilities in the PTP protocol, and develop proof-of-concept exploits for these attacks. In one attack, we demonstrate a novel man-in-the-middle (MITM) packet injection exploit against the PTP network that produces large, incorrect timing offsets at PTP timeReceiver nodes. In a second attack, we demonstrate the use of specific meta-data payloads to generate large time Transmitter (i.e. master clock) offsets, and to manipulate not just the clock offset but the actual clock frequency itself. We also investigate proposed mitigation techniques, including the use of NTS secured NTP with PTP concurrently which is suggested by some of our experimental results using Timemaster.**

Keywords—PTP, NTP, timing, cybersecurity

Introduction

1. Introduction

The IEEE 1588 standard, known as Precision Time Protocol (PTP) [1], is an emerging candidate for high precision timing networks, including clock distribution and synchronization. It is a follow-on to the widely used Network Time Protocol (NTP) [2] in applications which require enhanced timing performance. Many enterprise-class data centers, telecommunications back haul systems, cloud service providers, high performance computing applications, and others rely on a timing subsystem that is used to synchronize networked servers and other data processing equipment. Theoretically NTP can achieve timing accuracy of up to 1 ms, although in practice accuracies of tens to hundreds of ms are fairly common. Recently, a need for more accurate time synchronization has emerged. According to recent regulatory requirements in the financial sector [3,4], servers must be synchronized to within 50 - 100 millisecond drift tolerance of the NIST atomic clock, while proposed standards call for tolerances as low as 1 microsecond in some applications. These timing synchronization requirements are significantly lower than a standard NTP implementation can achieve. While some vendors have developed proprietary timing protocols that partially address these needs, they require additional wiring infrastructure that can be costly and problematic to deploy, and are often incompatible with extended distance fiber optic wavelength multiplexing links and software-defined network (SDN) controllers. Thus, there is a strong desire to design future timing networks around an open industry standard clock protocol with improved accuracy, such as the PTP protocol. In a previous paper [5], we described three security vulnerabilities in the current release of PTP, and discussed potential mitigation techniques.

In this paper, we present experimental results from a PTP test bed that demonstrate new types of covert channel communications, which allow the PTP protocol to be used for data exfiltration and other network communications that violate the implemented cybersecurity policy. We then expand upon this work to demonstrate two new zero-day vulnerabilities in the PTP protocol, and develop proof-of-concept exploits for these attacks. In one attack, we demonstrate a novel man-in-the-middle (MITM) packet injection exploit against the PTPv2 network which is used to produce large, incorrect timing offsets at PTP timeReceiver nodes. In a second attack, we demonstrate the use of specific meta-data payloads to generate large timeTransmitter clock offsets (i.e. master clock; note that at the time of this writing, IEEE standard notation for PTP4L systems as used in this testing refers to the master clock, while emerging standards plan to use timeTransmitter notation for this feature; in order to avoid confusion, we have been advised to continue using the master clock notation for this paper). Further, we can manipulate not just the clock offset but the actual clock frequency itself. We also discuss proposed mitigation techniques for these vulnerabilities.

2. Covert channels in PTPv2

A covert channel refers to a communication path used to transfer information between processes that are normally not allowed to communicate with each other under the current cybersecurity policy. Since a covert channel was not designed for communication, it often exhibits low data rates and lacks features normally associated with a communication channel, such as redundancy, retransmission, or error

detection/correction capabilities. Despite these drawbacks, covert channels are widely used to circumvent cybersecurity policies, for example to exfiltrate stolen data or to install and update malware. Ideally covert channel communication is difficult or impossible to detect by other processes, and does not obviously impede normal system operation. Meta-data fields of many popular network protocols can be used as covert channels; prior documented examples include covert channels within DNS, ICMP, NTP, and others [6].

In this paper, we investigate several approaches to implementing covert channels using PTP. In particular, the accuracy of clock synchronization over packet networks is highly sensitive to delay jitter in the underlying network, which dramatically affects clock accuracy. To address this, PTP defines transparent clocks (TCs), i.e. switches and routers that improve end-to-end clock accuracy by updating a "correction field" in the PTP packet header. A transparent clock is a PTP node with more than one port which doesn't participate in the Best TimeTransmitter Clock Algorithm (BTCA), such as a stateless switch. PTP data frames are forwarded through a TC, and their resident time is added to the correction field in the frame header. The correction field thus contains the latency caused by the current TC, and can be used to compensate for delays due to queuing, processing time, and propagation delays. PTP does not specify how the information from the correction field is to be used to accomplish clock synchronization; this is determined by other timing profile rules [7]. The correlation field is a 64-bit integer, but the PTP standard does not specify whether the length or data type should be validated; there is also no standard approach to determining whether the correlation field is in use or not. There is no corresponding field in the NTP protocol. Since it is possible to modify the contents of the correction field without significantly impacting normal PTP protocol operation, this field is a good potential candidate for use as a covert channel.

3. **Experimental Results**

Our PTP testbed spans two environments, namely the Marist College enterprise computing research lab (ECRL) and the IBM Poughkeepsie New York Z Systems test floor. Both environments run standardized PTPv2 (i.e. PTP4L), and testing in two environments helps insure that our results will generalize to other configurations. Since our results are related to fundamental parameters implemented in PTPv2, it is likely that the results shown here are representative of a typical timing network and would also apply to down-level versions of PTP. All tests were conducted using Intel x86 based servers, specifically multiple identical IBM System X servers (x3550 M3) each equipped with Intel X540-AT2 PCI NICs. The IBM environment is a "yellow zone" security configuration, containing x86 PTP clients and servers running Ubuntu Linux 18.04.2. This is interconnected to the Marist campus via an IBM 8264 switch and approximately 30 miles of AT&T network infrastructure, using the Cisco AnyConnect client. The Marist ECRL includes x86 PTP clients and servers running Ubuntu Linux 17.1. Across this environment we implemented PTPv2 (the LinuxPTP package). A grandmaster PTP timeTransmitter is routed through an IBM/Lenovo G8264 PTP compatible switch to a pair of listeners, one of which serves as our covert channel attack node. The attack node attempts to send corrupted PTP protocol packets which contain exfiltrated data, with minimal disruption to the functioning timing network. Note that the attack node does not need to be running PTP in order to function as an attack node.

To evaluate behavior of the covert channel, we wrote an original script in Python 3.6 which creates spoofed delay_request messages and sends them to the grandmaster clock for processing (normally the grandmaster will reply with a delay_response packet). In this relatively low data rate covert channel, 8 bytes of exfiltrated data is inserted into the packet header correction field, and optionally another 8 bytes of exfiltrated data is inserted into the clock identification field (this is comparable to data rates achieved by other covert channel protocols discussed previously). For testing purposes, we created a large text file of hexadecimal data for exfiltration. Our script reads 16 bytes at a time from this file, creates the spoofed delay_request packets, and sends them to the grandmaster at time intervals which mimic normal timing signal operation. Optionally, we can also sniff incoming packets to predict the next sequence ID value for our spoofed packets, in order to avoid issues with potential packet collisions. A sample packet trace captured with WireShark 3.4.6 illustrating a spoofed packet header is shown in figure 1. Spoofed packets and their exfiltrated data can be extracted at the grand master node.

```
⊟ Precision Time Protocol (IEEE1588)
  ⊟ 0000 .... = transportSpecific: 0x00
      ...0 .... = V1 Compatibility: False
    .... 0001 = messageId: Delay_Req Message (0x01)
    .... 0010 = versionPTP: 2
    messageLength: 44
    subdomainNumber: 0
  ⊟ flags: 0x0000
      0... .... .... .... = PTP_SECURITY: False
      .0.. .... .... .... = PTP profile Specific 2: False
      ..0. .... .... .... = PTP profile Specific 1: False
      .... .0.. .... .... = PTP_UNICAST: False
      .... ..0. .... .... = PTP_TWO_STEP: False
      .... ...0 .... .... = PTP_ALTERNATE_MASTER: False
      .... .... ..0. .... = FREQUENCY_TRACEABLE: False
      .... .... ...0 .... = TIME_TRACEABLE: False
      .... .... .... 0... = PTP_TIMESCALE: False
      .... .... .... .0.. = PTP_UTC_REASONABLE: False
      .... .... .... ..0. = PTP_LI_59: False
      .... .... .... ...0 = PTP_LI_61: False
  ⊟ correction: 59345.000000 nanoseconds
      correction: Ns: 59345 nanoseconds
      correctionSubNs: 0.000000 nanoseconds
    ClockIdentity: 0x001d9cfffeb1acfe
    SourcePortID: 1
    sequenceId: 15638
    control: Delay_Req Message (1)
    logMessagePeriod: 127
    originTimestamp (seconds): 1436270274
    originTimestamp (nanoseconds): 26902220
```

Figure 1 – WireShark trace of a spoofed packet header

We tested several different attack node and spoof packet configurations to determine the impact on PTPv2. First, when the attack node is not running PTP, the master node responds normally to the delay_request message with a delay_response message. There is no impact on the timing network, and we were able to iterate our script thousands of times to exfiltrate significant amounts of data undetected. Next, we tested with the attack node running PTP. In this case, if we use non-colliding packet sequence IDs as discussed earlier, the effect is the same as if the attack node was not running PTP (i.e. the master node responds normally to the delay_request message with a delay_response message and there is no other detectable impact on the timing network). This use case with non-colliding sequence IDs is the default implementation for many enterprise class PTP profiles, which are susceptible to this form of data exfiltration. Since enterprise class PTP profiles are all unicast, it's not possible to intentionally configure colliding sequence IDs. Next, we tested an attack node running PTP with colliding packet sequence IDs, to determine if there was any advantage to reconfiguring the default enterprise class PTP profile. As before, the master node responds normally to the delay_request message. In this case, however, there is a noticeable difference; the data in the correction field is reflected by the raw delay value in the PTP output. This suggested that it might be possible to use a covert channel to manipulate operation of the timing network. Further testing showed that if we use colliding packet sequence IDs and set tsproc_mode to "raw", the clock offset is now computed by taking into account the raw output of the correction field. In other words, we were able to produce large, incorrect offsets in the master clock value (on the order of minutes to hundreds of minutes or greater) under these conditions.

4. **Correction Field MITM Attack**

Based on this zero-day vulnerability, we developed a new man-in-the-middle (MITM) packet injection exploit against the PTPv2 network. The configuration for the correction field MITM attack is shown in figure 2, in which a timeReceiver node is connected through a boundary clock. By intercepting packets before they leave the boundary node and injecting large data values into the correction field, we should be able to produce large, incorrect clock offsets at the timeReceiver node.

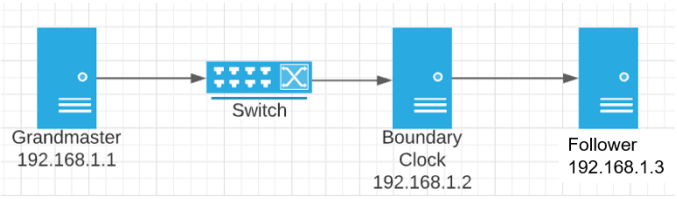

Figure 2 – Correction field MITM attack configuration

Our first attempts involved sniffing for incoming sync_followup messages, copying those packets, inserting spoofed data into the correction field, then resending the packets to the timeReceiver node. This approach didn't work, since the timeReceiver node was receiving the authentic sync_followup messages before our spoofed packets, and was calculating its timing offset from the real data; upon receiving our spoofed copy, the timeReceiver node ignored the spoofed packet. Our second attempt involved sniffing for the sync messages instead; once again, the authentic packets were received before our spoofed packets, and there was no effect. While it may be possible to address this race condition and cause the spoofed packets to be processed before the authentic packets, we instead opted for a different approach.

Our third attempt avoids making copies of the packets, and instead directly manipulates iptables, using Python-iptables 1.0.0. This produces significant non-intuitive results. For example, if we insert a very large value in the correction field, PTP4L will stop generating master offset messages completely. This makes it impossible to determine the effect on the master clock offset. There is apparently no explanation for this behavior in the PTPv2 operating specifications. Further, if we insert a very small value in the correction field, PTP4L will consistently generate negative delay messages and computations. The full ramifications of using only negative delay messages are unclear, since this condition is not addressed in the PTPv2 operating specifications. Example results showing small and large correction field values are shown in figure 3. We note that phc2sys behaves normally during both attacks. In yet another example, we show that there are certain iptables rules that can prevent PTP communication altogether. When these rules were in place, the timeReceiver node did not receive any messages from the boundary clock, and therefore defaulted into acting as its own master clock.

```
ptp4l[620564.317]: negative delay      -112
ptp4l[620564.317]: delay = (t2 - t3) * rr + (t4 - t1)
ptp4l[620564.317]: t2 - t3 = -186474202174
ptp4l[620564.317]: t4 - t1 = +186474201949
ptp4l[620564.317]: rr = 1.000000000
ptp4l[620564.317]: delay   filtered       107   raw       -112
ptp4l[620564.810]: port 1: delay timeout
ptp4l[620564.810]: negative delay      -140
ptp4l[620564.810]: delay = (t2 - t3) * rr + (t4 - t1)
ptp4l[620564.810]: t2 - t3 = -186967257005
ptp4l[620564.810]: t4 - t1 = +186967256724
ptp4l[620564.810]: rr = 1.000000000
ptp4l[620564.810]: delay   filtered       105   raw       -140
ptp4l[620566.391]: port 1: delay timeout
ptp4l[620566.391]: negative delay      -228
ptp4l[620566.392]: delay = (t2 - t3) * rr + (t4 - t1)
ptp4l[620566.392]: t2 - t3 = -188548883971
ptp4l[620566.392]: t4 - t1 = +188548883515
ptp4l[620566.392]: rr = 1.000000000
ptp4l[620566.392]: delay   filtered        51   raw       -228
ptp4l[620566.475]: port 1: delay timeout
ptp4l[620566.475]: negative delay      -232
ptp4l[620566.475]: delay = (t2 - t3) * rr + (t4 - t1)
ptp4l[620566.475]: t2 - t3 = -188632874595
ptp4l[620566.475]: t4 - t1 = +188632874130
ptp4l[620566.475]: rr = 1.000000000
ptp4l[620566.476]: delay   filtered       -28   raw       -232
ptp4l[619229.939]: port 1: delay timeout
ptp4l[619229.940]: delay   filtered      2342   raw       2342
ptp4l[619230.942]: port 1: delay timeout
ptp4l[619230.942]: delay   filtered      2342   raw       2345
ptp4l[619231.046]: port 1: delay timeout
ptp4l[619231.046]: delay   filtered      2342   raw       2345
ptp4l[619232.838]: port 1: delay timeout
ptp4l[619232.838]: delay   filtered      2342   raw       2347
ptp4l[619233.331]: port 1: delay timeout
ptp4l[619233.331]: delay   filtered      2343   raw       2348
ptp4l[619234.991]: port 1: delay timeout
ptp4l[619234.991]: delay   filtered      2344   raw       2350
ptp4l[619235.260]: port 1: delay timeout
ptp4l[619235.261]: delay   filtered      2345   raw       2351
ptp4l[619236.752]: port 1: delay timeout
ptp4l[619236.753]: delay   filtered      2346   raw       2358
ptp4l[619237.465]: port 1: delay timeout
ptp4l[619237.465]: delay   filtered      2347   raw       2360
ptp4l[619238.362]: port 1: delay timeout
ptp4l[619238.363]: delay   filtered      2349   raw       2360
ptp4l[619239.516]: port 1: delay timeout
ptp4l[619239.517]: delay   filtered      2350   raw       2359
ptp4l[619239.922]: port 1: delay timeout
```

Figure 3 – sample trace showing small correction fields causing negative delay (top) and large correction delays causing timeouts (bottom)

5. Clock Frequency Attack

We further attempted using the sync_followup field for covert channel exfiltration. This turned out to be unsuitable, since spoofing data into the sync_followup field results in large offsets to the master clock, which are easily detected by the timing network. However, the resulting offsets had the additional unanticipated effect of changing the clock frequency itself. This suggested another new zero-day vulnerability against timing networks; the corresponding exploit is called the clock frequency attack. In this attack, sync_followup packets are spoofed with large amounts of data inserted into the correction field. This has two effects; first, it causes a large, incorrect clock offset, and second, it causes the clock frequency to exceed its maximum allowed value. Under these conditions, the PTP clock servo algorithm is unable to synchronize back to the master clock. Further, the clock's master offset value continues to drift even after the attack has concluded.

Results of this attack on the master clock offset during the attack are shown in figure 4, which demonstrates the significant clock offsets that can be achieved. The master clock drift after the attack has completed (i.e. the master clock fallout) is shown in figure 5. Sample logs during the attack show artificially high values of PTP4L, master offset, and path delay. The s2 frequency value is locked at -nan (not a number), indicating that the clock servo algorithm is unable to maintain clock synchronization during the attack. Similarly, the s0 clock servo is shown to be locked at -nan during and after the attack, while repetitive clockcheck messages are generated. Further, the PTP hardware clock offset (derived from the phc2sys parameter as before) and the corresponding system hardware logs clearly show that we have successfully altered the clock frequency, not just the clock offset or delay, and that the system is unable to recover for some time after the attack is complete. As noted previously for other attacks, there is apparently no explanation for this behavior in the PTPv2 operating specifications.

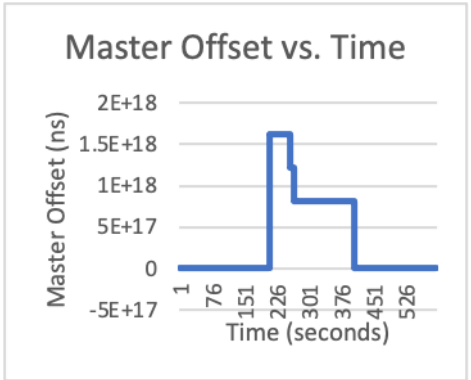

Figure 4 – Master clock offset during attack

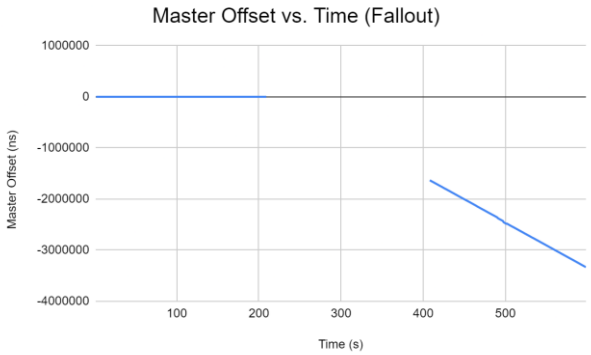

Figure 5 – Master Clock fallout

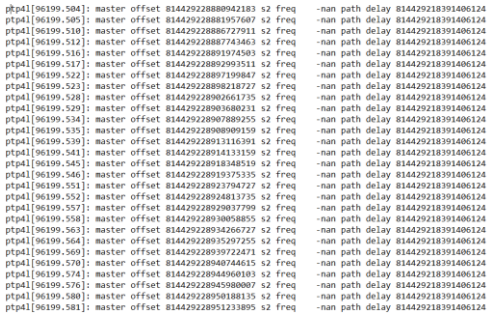

Figure 6 – Sample clock frequency attack logs (-nan results)

6. Proposed Mitigation

While the root cause of the vulnerabilities leading to the correction field MITM attack and the clock frequency attack remain unknown, we can suggest possible mitigations based on our test results. Using the optional AUTHENTICATION TLV for PTP would in principle address the lack of authentication which allows both of the attacks proposed in this paper to succeed. Further enhancement comes from the use of proper key management, such as the proposed Network Time Security (NTS) for PTP. There are two primary reasons to consider NTS for PTP. The first involves reuse of existing network timing infrastructure. Most clients will continue to use both NTP and PTP in their data and telecom centers for the foreseeable future, so there are benefits to providing a common security infrastructure for both protocols. Further, TLS is already present on many devices, since any device with an HTTPS management console already supports TLS and can support NTS. Second, NTS for PTP offers highly flexible configuration options. The TLS cipher selection is adaptable, and implementations of both traditional TLS and hybrid post-quantum handshake protocols are becoming available. NTS can be configured in a unicast-based or group-based (multicast) approach, depending on the application security requirements. Resilient architecture supporting both PTP instances with backup NTP instances on a particular end device (such as an enterprise-class server) has already been proposed for large data centers, and would make it easy to support this type of authentication.

We have conducted preliminary testing using Timemaster, a program that is part of the LinuxPTP package (which is also parent to the PTP4L and phc2sys packages we have used throughout these tests). Timemaster runs on PTP timeReceiver nodes and configures PTP4L and phc2sys to operate as a reference clock (PHC) for NTP packages such as chrony or ntpd which in turn use the NTP false-ticker algorithm to pick from both PTP and NTP sources in which to synchronize the system clock. We used chrony (available under GNU GPL license), an implementation of NTP which can synchronize the system clock with NTP servers, other reference clocks, or manual input. Chrony includes a daemon that can be started at boot time (called chronyd) and a command line interface which can be used to monitor performance and change operating parameters (called chronyc). Using PTP4L and chrony, the grandmaster operates as both a PTP timeTransmitter and NTP server, providing time data to our timeReceiver (and NTP client) node, while another timeReceiver launches the same spoofing attacks. Initially, PTP is selected as the most accurate time source, and NTP is advertised as the next most accurate time source, according to the chronyd logs. We attempted the correction field MITM and clock frequency attacks using this configuration. A successful covert channel exploitation will not produce a PTP offset, and therefore chronyd will not select another time source, so this attack succeeds. A covert channel attack that exploits a PTP4L configuration with timestamp processing set to raw, as discussed previously, does create a PTP offset and prompts chronyd to mark the PTP source as 'false-ticker'. Thus, we suggest a combination of NTP and PTP to mitigate this type of attack under at least some operating conditions.

7. **Conclusions**

Similar to many other networking protocols, it is possible to implement covert communication channels using the IEEE 1588 PTP protocol. Specifically, we demonstrate undetected covert communication using 16 bytes from the correction field and clock ID field in the header of a delay_request message. The master node responds normally to such messages, regardless of whether we use colliding sequence IDs. We further observed that when configured in raw data mode, we could induce large, incorrect offsets in the computed master clock. Similar effects were observed when using sync_followup packets, which are not suitable for covert channels as a result. However, these effects form the basis for two new zero-day proof-of-concept exploits. We demonstrate a MITM packet injection attack using the correction field, which can use specific size payloads to either stop the system from generating PTP4L offset messages or force negative delay messages and computations (phc2sys seems unaffected during these attacks). We also demonstrate a clock frequency attack, which both causes large clock offsets and manipulates the clock frequency such that it exceeds the maximum allowed value. In this case, the clock servo algorithm is unable to synchronize the clock back to the master, and the master offset continues to drift uncontrollably even after the attack is complete. We investigate mitigation techniques using NTS for PTP, although experimental testing with Timemaster and chrony suggest that this will may not completely mitigate the proposed attacks as the correction field remains mutable by intermediary nodes (and open to MITM attacks) after authentication of the timeTransmitter and timeReceiver.

REFERENCES

[1] IEEE 1588-2008, IEEE Instrumentation and Measurement Society. TC-9 Sensor Technology, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems", 2008.

[2] D. Mills, J. Martin (Editor), J. Brubank, and W. Kasch, "Network Time Protocol version 4: Protocol and Algorithm Specification", RFC 5905, (June 2010) https://tools.ietf.org/html/rfc5905 (last accessed July 11, 2018)

[3] Financial Industries Regulatory Authority (FINRA) on SEC notice 16-23 (July 2016) http://www.finra.org/industry/notices/16-23 (last accessed July 11, 2018)

[4] MiFID Regulatory Technical Standard 25 Annex from the European Commission report (July 6, 2016) http://ec.europa.eu/finance/securities/docs/isd/mifid/rts/160607-rts-25-annex_en.pdf (last accessed July 11, 2018)

[5] C. DeCusatis, R. Lynch, W. Kludge, J. Houston, P. Wojciak, and S. Guendert, "Impact of cyberattacks on precision time protocol", IEEE Transactions on Instrumentation and Measurement, vol. 69, no. 5, p. 2172-2181 (May 2020)

[6] T. N. Tsapakis, "Alternative communication channel over NTP", Virus Bulletin, pp. 1-8, April 2019 https://www.virusbulletin.com/virusbulletin/2019/04/alternative-communication-channel-over-ntp/ (last accessed December 21, 2021)

[7] G.Garnet, "IEEE 1588 v2 Tutorial", Proc. ISPCS 2008, Ann Arbor, Michigan https://www.ieee802.org/1/files/public/docs2008/as-garner-1588v2-summary-0908.pdf (last accessed March 4, 2022)