IEEE 802.1AS Multi-Domain Aggregation for Virtualized Distributed Real-Time Systems

Jan Ruh
TTTech Computertechnik AG
Vienna, Austria
jan.ruh@tttech.com

Wilfried Steiner
TTTech Computertechnik AG
Vienna, Austria
wilfried.steiner@tttech.com

Gerhard Fohler
TU Kaiserslautern
Kaiserslautern, Germany
fohler@eit.uni-kl.de

Abstract

Cyber-physical systems like cars, mobile agriculture machines, or duty vehicles are increasingly connected to public networks. Thus, in addition to their inherent fault-tolerance requirements, there is a pressing need to design these systems cost-effectively to withstand cyber-attacks, i.e., to ensure cyber resiliency. This paper implements a cyber-resilient clock synchronization architecture that features fault-tolerant dependent clocks and a fault-tolerant average (FTA) using standardized protocols (e.g., IEEE 802.1AS) and open-source software (e.g., ACRN hypervisor). Our experiments show how OS diversification can improve the cyber resiliency of the FTA. Furthermore, our $24\,h$ fault injection experiment demonstrates the robustness of the architecture against fail-silent grandmaster clocks and clock synchronization virtual machines. Therefore our architecture can serve as proof of concept to extend commercial offerings.

1 Introduction

Today's cyber-physical systems (CPS) are becoming increasingly connected to meet the advanced requirements of modern applications. Fault-tolerant clock synchronization acts as the heartbeat of many CPS by enabling software and hardware components to have a consistent time view of their environment represented by sensor inputs and the internal state of the CPS. Moreover, for CPS following the time-triggered paradigm, fault-tolerant clock synchronization actively drives the operation of the distributed real-time system. Fault-tolerant clock synchronization is a well-understood problem, both theoretically and practically, that has been investigated for decades [8, 11, 15, 20, 21]. However, the growing complexity and connectivity of CPS challenges established fault hypotheses as it raises the risk of security hazards that mitigate the safe operation of clock synchronization protocols highlighting the need for cyber-resilient clock synchronization architectures.

Treytl et al. [23] provide an overview of attacks on the PTP protocol family, including message manipulation, message delay, message insertion, and Byzantine grandmasters (GMs). Various authors demonstrated attacks on PTP and how to detect and mitigate these attacks [4, 13, 14]. Furthermore, the current IEEE 1588-2019 [1] standard proposes using a voting algorithm to detect faulty GM clocks if more than two redundant time sources are available, such as GPS or multiple PTP domains. Both IEEE 1588-2019 and IEEE 802.1AS [2] emphasize using hot-standby GM clocks to prevent single points of failure. However, the standards do not detail the implementation or configuration of redundancy mechanisms.

As a result, Kyriakakis et al. [10] introduce a high-level fault-tolerant PTP end-system architecture that allows for multi-domain aggregation using the fault-tolerant average (FTA) algorithm. They evaluate their fault-tolerant PTP end-systems for different network topologies utilizing OMNeT++ [24]. Their results indicate that operating PTP with FTA multi-domain aggregation enables the Byzantine fault tolerance of GM clocks. However, the authors' end-system design has drawbacks concerning real-world applicability. Most importantly, they conceptually neglect the problem of (initially) synchronizing GM clocks of different domains with each other. As a result, they limit the use of their multi-domain end-system architecture to PTP clients only. In practice, this limitation prohibits locating PTP GM clocks on physically separated nodes that do not share a common time source, thus breaking the Byzantine fault tolerance of their architecture in real-world systems.

 DOI 10.1109/DSN-S58398.2023.00027

Our work investigates a cyber-resilient clock synchronization architecture that implements the fault-tolerant average (FTA) for IEEE 802.1AS and fail-silent dependent clocks for the ACRN hypervisor. Furthermore, we argue that using feature-rich and accessible operating systems (OSs), such as GNU/Linux, executing on virtualized multicore hardware platforms allows for robustness against random hardware faults and design faults materializing in software vulnerabilities.

2 Cyber-Resilient Clock Synchronization

We propose a two-part architecture to achieve cyber-resilient clock synchronization using commonly available software components. Firstly, we provide $n = f + 1$ redundant clock synchronization virtual machines (VMs) to extend the dependent clock architecture implemented in previous work [19] to tolerate f fail-silent clock synchronization VM without losing synchronization. Furthermore, we present an extension to IEEE 802.1AS that enables multi-domain aggregation based on the FTA enabling Byzantine fault tolerance against malicious GM clocks. The combination of fault-tolerant dependent clocks, IEEE 802.1AS multi-domain aggregation, and virtualized GM clocks with diversified OSs aids the cyber resiliency of our clock synchronization architecture.

2.1 A Fail-Silent Dependent Clock

The dependent clock paradigm [3] utilizes a dedicated VM per node, the so-called *clock synchronization VM* [19], that executes a clock synchronization protocol and shares the derived synchronized time with co-located VMs via a shared memory region. However, previous work did not consider the implications of the dependent clock paradigm for fault tolerance or cyber resiliency. In previous work [19], we implemented a dependent clock for the ACRN hypervisor that exports a synchronized time shared memory `STSHMEM` via a virtual PCI device to co-located VMs. The virtual machine interfaces with virtual PCI device in the hypervisor via a device driver. The driver initializes the `STSHMEM` mapping and derives the synchronized time as a POSIX clock `CLOCK_SYNCTIME` from the shared clock parameters. The clock synchronization achieves a sub-microsecond precision using IEEE 802.1AS.

In an idealized setting, i.e., when the hypervisor and the hardware platform are design fault-free, ACRN's use of the memory management unit (MMU) to setup the shared memory ensures that all co-located VMs always observe the same clock parameters yielding fail-consistent failure of a dependent clock. In an industrial implementation certified and/or diverse hardware and hypervisors could be used to ensure the isolation property of the hypervisor. Therefore, to tolerate f consistently failing clock synchronization VMs, we require $2f + 1$ redundant clock synchronization VMs. To that end, we extend the dependent clock by introducing a periodically executing monitor in ACRN implementing a voting algorithm to detect clock synchronization VMs providing faulty clock parameters. If the monitor detects a faulty clock synchronization VM, the `STSHMEM` virtual PCI device injects an interrupt into the redundant clock synchronization VMs that is about to take over maintaining the synchronized time.

However, the fail-consistent fault hypothesis is often not feasible since clock synchronization VMs rely on passthrough network interface cards (NICs) to guarantee a reasonably low clock synchronization precision [19] by utilizing HW timestamping. Unfortunately, many modern HW platforms, including those available to us, do not provide the number of NICs needed to host $2f_{des} + 1 \geq 3$ redundant clock synchronization VMs. Therefore, for our experiments, we assume fail-silent clock synchronization VMs reducing the number of required NICs to $f_{des}+1 \geq 2$. Nevertheless, it is straightforward to realize fail-consistent behavior by adding more NICs.

2.2 Shared Memory-based gPTP Multi-Domain Aggregation

The LinuxPTP tool set for clock synchronization in GNU/Linux OS includes two applications. Firstly, the `ptp4l` application executes the actual clock synchronization protocol (e.g., IEEE 1588 or IEEE 802.1AS) synchronizing the NIC's internal clock using `Sync` and `FollowUp` messages. Secondly, the `phc2sys` application synchronizes the Linux kernel's system clock to the NIC.

We extended `ptp4l` so that multiple `ptp4l` instances aggregate the master offsets from M different GM clocks or domains using the FTA. The aggregation of master offsets derived in separate `ptp4l` instances requires a well-defined communication channel between the processes. Therefore, we introduce a user-space shared memory region `FTSHMEM` between the M `ptp4l` instances. Note the difference between the `STSHMEM` and `FTSHMEM` shared memory regions. The guest OS establishes the `FTSTHMEM` between M processes in user space of a VM whereas the hypervisor shares the `STSHMEM` between co-located VMs. Now, the `FTSHMEM` shared memory holds the latest M GM offsets, an array of M booleans indicating whether the corresponding GM clock's offset from the remaining GM clocks is within a configurable threshold, a timestamp `adjust_last` providing when we have last adjusted the NIC's clock frequency, and the state variables of a proportional integral (PI) controller used in LinuxPTP to derive the frequency offsets.

Furthermore, for the FTA algorithm we assume that the M GM clocks of the domains are initially synchronized with a precision Π [7]. We presume the absence of faults for initial synchronization since it represents a separate hard problem [22] that is out of the scope of this work. Therefore, during start-up the nodes of $M - 1$ domains synchronize to an initial domain's GM until their

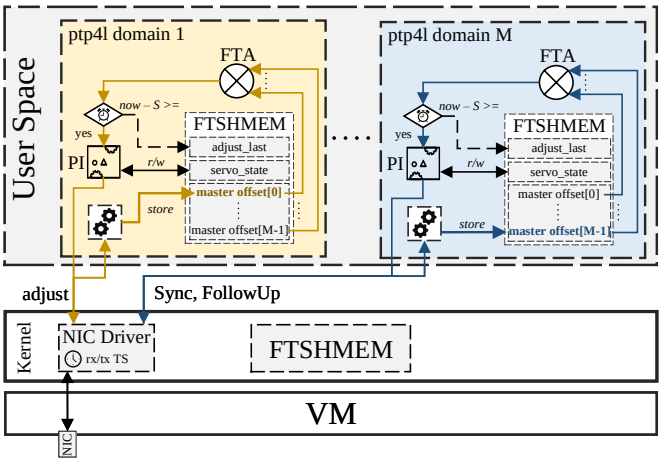


Figure 1: Showing the design of a gPTP multi-domain aggregation using a fault tolerance shared memory (`FTSHMEM`) region between M `ptp4l` instances. Each `ptp4l` instance receives `Sync` and `FollowUp` messages of a distinctive gPTP domain, calculates the GM master offset, and stores it in the `FTSHMEM`. Given the synchronization period S, the first domain for which it is $\texttt{now} + S \geq \texttt{adjust\_last}$ passes the aggregated GM offset to the shared PI controller and forwards the frequency offset to the kernel that adjusts the NIC's clock frequency.

GM offsets fall below a configurable threshold. Once the $M-1$ GM clocks' offset to the initial domain falls below the threshold, the system starts fault-tolerant operation using the FTA to aggregate master offsets.

Next, we must guarantee that each GM clock sends its `Sync` message at approximately the same time bounded by the clock synchronization precision Π. We implement synchronous transmission of `Sync` messages using Linux Earliest-Time-First (ETF) queuing discipline and utilizing the launch time feature supported by some NICs[1].

The M `ptp4l` instances on a node process the time packets their respective gPTP domain received on the NIC. When an `ptp4l` instance i on a node n receives a `Sync` message from GM clock $g_i \in GM = \{g_0, ..., g_{M-1}\}$ during synchronization interval $s \in \mathcal{N}_0$, it retrieves the reception timestamp $tn^n(\texttt{Sync}_s^i)$ from the NIC and uses the *precise origin timestamp* $tn_i^g(\texttt{Sync}_s^i)$, *the correction field* $e_s^{g_i,n}$, and the *rate ratio* $R^{g_i,n}$ received in the `FollowUp` message to calculate the GM offset. The `ptp4l` instances store the M offsets $c_s^{g_i,n}$ for $i = 0, ..., M-1$ to the gPTP GMs g_0 to g_{M-1} in the `FTSHMEM` region. In synchronization interval $s+1$, the first `ptp4l` instance on time-aware system n, for which it is

$$\texttt{adjust\_last} + \texttt{sync\_interval} \leq tn^n(now) \qquad (1)$$

[1]For example the Intel i210 Ethernet Controller [6].

with $tn^n(now)$ being the current time in node n, sorts the master offsets $c_s^{g_i,n}$ for $i = 0, ..., M-1$ in ascending order and calculates the FTA master offset [8]. After aggregation, the `ptp4l` instance updates `adjust_last` with the current time and passes the aggregated master offset c_s together with the aggregated precise origin timestamp to the PI controller to derive the frequency offset that the NIC driver uses to correct the NIC's clock frequency. Figure 1 illustrate the design of our multi-domain aggregation using `ptp4l` in a single synchronization VM where the overall system consists of multiple physical nodes with multiple VMs as shown in Figure 2. Finally, we use LinuxPTP's `phc2sys` to synchronize `CLOCK_SYNCTIME` to the NIC's fault-tolerant global time by deriving the relevant clock parameters and updating the `STSHMEM` of the dependent clock.

For our research prototype, we use diverse Linux kernel versions to prevent a single kernel vulnerability affecting multiple GM clocks. However, for a production system, the degree of diversification has to be justifiable concerning implementation cost and gained cyber resiliency so that in some industries, it is beneficial to use entirely distinct OSs. In fact, a previous study [5] on shared vulnerabilities in OSs, including the BSD, Linux, Solaris, and Windows OS families, suggests that the number of shared vulnerabilities between two OSs is significantly lower than the number of total vulnerabilities of the individual OS. Moreover, their study indicates that even the number of shared vulnerabilities between Linux distributions is considerably smaller than the number of total vulnerabilities in individual distributions.

3 Evaluation

We performed two experiments to demonstrate the fault tolerance and cyber-resilience capabilities of our proposed clock synchronization architecture. Firstly, we demonstrate how reusing the same clock synchronization stack for redundant GM clocks breaks Byzantine fault tolerance. Therefore, we intentionally used an exploitable kernel version on all GM clocks enabling an attacker to take control of more than one GM clocks and to execute a timing attack on the clock synchronization despite the presence of multi-domain aggregation, causing the clock synchronization precision to violate its upper bound. In a second experiment, we validate the robustness of the fault-tolerant clock synchronization against a single fail-silent GM clock and fail-silent clock synchronization VMs by injecting clock synchronization VM faults during a continuous 24 h experiment. Note that there is never more than one faulty clock synchronization VM per node since since this would violate our fail-silent fault assumption. However, it is possible for multiple clock synchronization VMs to fail at the same time across nodes.

3.1 Methodology

3.1.1 Experimental Setup

We ran the experiments on a network of four identical edge computing devices (ECDs), each equipped with an Intel Atom E3950 multicore processor with four cores at $1.594\,GHz$, $8\,GB$ main memory and two Intel I210 NICs. We denote the ECDs with dev_1 to dev_4 and the NICs of dev_x with NIC_1^x and NIC_2^x. Each device's NIC_2 is wired to an integrated Linux-based TSN switch with four external ports. All nodes, including the TSN switches sw_1 to sw_4, executed IEEE 802.1AS using OpenIL's LinuxPTP[2] v1.8 with our multi-domain aggregation feature and external port configuration enabled, meaning that there is no best master clock algorithm (BMCA) picking GM clocks. Instead, we configured four distinct gPTP domains $dom_1, ..dom_4$ with spatially separated GM clocks.

Each ECD utilized the ACRN hypervisor [12] v2.4 with the fault-tolerant dependent clock to run a privileged service VM and two clock synchronization VMs with a single vCPU and $1\,GB$ of main memory each. The hypervisor-native task monitoring the clock synchronization VMs executed with a period of $125\,ms$. We pinned the vCPUs of each VM to a dedicated processor core. We denote clock synchronization VM i that was running on dev_x with c_i^x and the set of all clock synchronization VMs with $\mathcal{C}$. Note that clock synchronization requires access to a dedicated physical NIC to minimize timestamping jitter [18]. As a result, our configuration is limited to two clock synchronization VMs per ECD, restricting the failure assumption of the dependent clock to be fail-silent. We illustrate the network topology in Figure 2. We configured clock synchronization VMs $GM := \{c_1^1, ..., c_1^4\}$ to act as GM clocks for gPTP domains $dom_1, ..., dom_4$ respectively. We provided a static port configuration for all gPTP domains that allow for a redundant path between all virtual and physical nodes of the test network.

3.1.2 Clock Synchronization Precision Measurement

We measured clock synchronization precision by sending a multicast measurement packet p_s every second $s \in \mathcal{N}$ from a dedicated clock synchronization VM $c_{measure}$, which acted as a measurement VM, to the remaining clock synchronization VMs $c_i^x \in \mathcal{C} \setminus \{c_{measure}\}$. The measurement VM sent the multicast packet on a specified virtual LAN (VLAN) to enforce that the packets take known paths from the source node to the destination nodes in the mesh network. Upon reception the clock synchronization VMs serving measurement requests created timestamps $tn^{c_i^x}(rx_{p_s})$ using `CLOCK_SYNCTIME` and returned them to the measurement VM. From the reception timestamps, we calculated the measured precision during measurement interval s as:

$$\Pi_s^* = \max_{\forall c,c' \in \mathcal{C} \setminus \{c_1^m, c_2^m\}} (|tn^c(rx_{p_s}) - tn^{c'}(rx_{p_s})|) \quad (2)$$

In our setup, a clock synchronization VM c_2^m acts as the measurement VM. Therefore, the number of hops to the remaining clock synchronization VMs $\mathcal{C}$ equals three except for the path to clock synchronization VM c_1^m, which equals two, as packets only had to pass the internal TSN switch sw_m of ECD dev_m to reach c_1^m. Asymmetric paths result in an increased measurement error γ [19]. Therefore, we omitted clock synchronization VM c_1^m for clock synchronization precision measurement resulting in symmetric path latencies from the measurement VM to all remaining destination clock synchronization VMs $CS := \mathcal{C} \setminus \{c_1^m\}$, thus minimizing the measurement error.

Let $E_{n,n'}$ be a network path from node n to node n' and $E_{c_2^1}$ the set of all paths from the measurement node c_2^1 to clock synchronization VMs $c \in CS$. For each experiment, we determined the measurement error

$$\gamma = \max_{\forall E_{n,n'} \in E_{c_2^1}} (d_{max}^{n,n'}) - \min_{\forall E_{n,n'} \in E_{c_2^1}} (d_{min}^{n,n'}) \quad (3)$$

utilizing data on the network latency between adjacent nodes extracted from `ptp4l`.

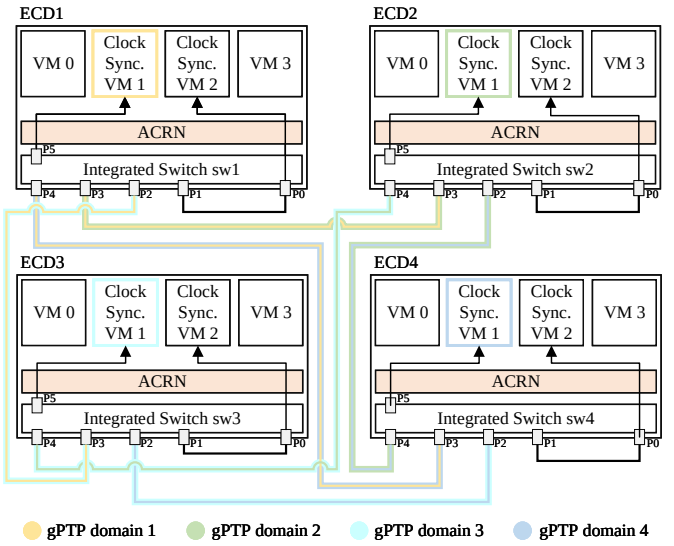
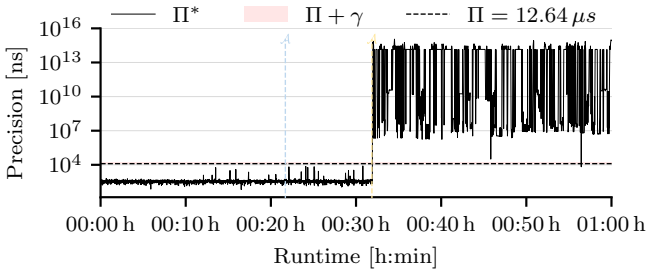
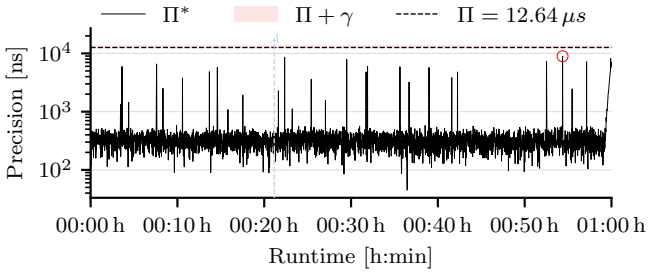


Figure 2: Showing the network topology and node setup of our experimental virtualized distributed real-time system. The switches of the four devices form a mesh network ensuring redundant data paths. Each device hosts the GM clock of one gPTP domain and a redundant clock synchronization VM. The network ports of each gPTP domain are configured statically with IEEE 802.1AS external port configuration.

[2] `https://github.com/openil/linuxptp`



(a) Clock sync. precision, identical Linux kernel version



(b) Clock sync. precision, diverse Linux kernel versions

Figure 3: Showing the measured clock synchronization precision for the 1 h cyber-resilience experiment. The attacker executed a root exploit in virtual GMs c_4^1 (in color blue) and c_1^1 (in color yellow) at times 00:21:42 h and 00:31:52 h as indicated by the dashed lines.

We acknowledge that measured precision does not include any clock synchronization VM hosted on dev_m due to our goal of minimizing the measurement error γ when setting up the measurement VLAN. However, we want to emphasize that the device dev_m hosting the measurement VM c_2^m has been chosen arbitrarily so that our selection did not affect the measured clock synchronization precision.

3.1.3 Upper Bound on Clock Synchronization Precision

We calculated the upper bound on clock synchronization precision by instantiating the convergence function $\Pi(N, f, \mathcal{E}, \Gamma) = u(N, f)(\mathcal{E} + \Gamma)$ introduced by Kopetz and Ochsenreiter [8]. We determined the drift offset $\Gamma = 2r_{max} \cdot S = 1.25\,\mu s$ by using expected maximum drift rate $r_{max} = 5\,ppm$ referenced in the literature [2] and the synchronization period $S = 125\,ms$. The reading error $\mathcal{E} = d_{max} - d_{min}$ equals the difference between the maximum d_{max} and minimum latency d_{min} of any two nodes in the network. Therefore, for each experiment, we measured the latency between all nodes in our network using `ptp4l` to determine d_{min} and d_{max} and derive an estimate of the reading error.

As a result, for our experiments with $N = |GM| = 4$, we expect the following inequality to hold for $f \leq 1$:

$$\forall s : \Pi_s^* - \gamma \leq \Pi(N, f, \mathcal{E}, \Gamma) = 2(\mathcal{E} + \Gamma) \tag{4}$$

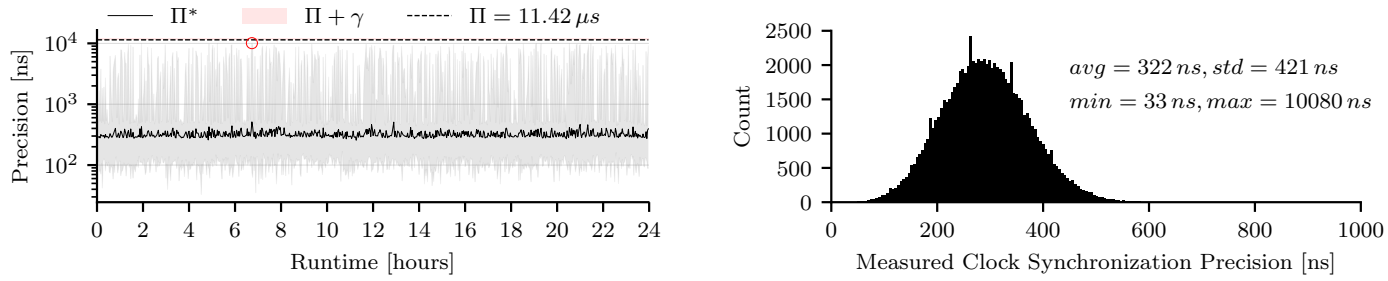
3.2 Cyber-Resilience Experiments

For our first experiment, we measured minimum and maximum path delays of $d_{min} = 4120\,ns$ and $d_{max} = 9188\,ns$ yielding a reading error of $\mathcal{E} = 5068\,ns$ and with it an upper bound of clock synchronization precision of $\Pi = 2(\mathcal{E} + \Gamma) = 12.636\,\mu s$. Furthermore, we found a measurement error of $\gamma = 1313\,ns$ for the symmetric paths from the measurement node c_2^1 to the remaining clock synchronization VMs CS. Next, we configured all virtual GM clocks to use the same exploitable Linux kernel v4.19.1. For our realistic attack scenario, we presumed an attacker $\mathcal{A}$ that has restricted user credentials for at least two virtual GM clocks $c_1^i, c_1^j \in GM$ of our experimental virtualized distributed real-time system. The attacker utilizes an exploit [3] for CVE-2018-18955 to gain root access to two virtual GM clocks c_1^i and c_1^j.

We decided to pick virtual GMs c_1^1 and c_4^1 as attack targets for demonstration purposes. The attacker $\mathcal{A}$ executed the root exploits in virtual GM c_4^1 at time 00:21:42 h and in virtual GM c_1^1 at time 00:31:52 h into the experiment. After gaining root access, the attacker replaced the benign `ptp4l` instances with malicious instances for both virtual GMs. The malicious `ptp4l` instances distribute faulty `preciseOriginTimestamps` that are offset by $-24\mu s$ for gPTP domains dom_1 and dom_4. In Figure 3a, we illustrate the measured clock synchronization precision in a setup that used identical Linux kernel versions for the virtual GMs as it faced an attacker $\mathcal{A}$. We can see that the FTA successfully masks the initial attack on virtual GM c_4^1 that resulted in the distribution of malicious `preciseOriginTimestamps` at time 00:21:42 h. However, as soon as the attacker takes over the second GM c_1^1 at time 00:31:52 h, we can observe how the measured clock synchronization precision violates the calculated upper bound and the nodes lose synchronization.

In contrast, Figure 3b shows the measured clock synchronization precision of a second experiment using the same attacker $\mathcal{A}$ but diversifying the used Linux kernel version so only virtual GM c_4^1 used the exploitable Linux kernel v4.19.1. Again, we observe how the Byzantine fault tolerance successfully masks the initial attack on virtual GM c_4^1. However, the attacker's attempt to also exploit GM c_1^1 is unsuccessful, thus demonstrating the effectiveness of explicitly identifying and enforcing distinct software stacks to harden Byzantine fault tolerance against cyber-attacks.

[3] The used exploit 47164 can be found at `https://github.com/offensive-security/exploitdb-bin-sploits/raw/master/bin-sploits/47164.zip`



(a) Clock sync. precision over 24 h

(b) Distribution of values

Figure 4: Showing the results of the 24 h fault injection experiment. Figure 4a illustrates the measured clock synchronization precision $\Pi 1^*$, the measurement error γ, and the upper bound of clock synchronization precision Π in the presence of faults over the course of 24 h on a logarithmic scale. We measured the clock synchronization precision once per second. We have aggregated intervals of 120 sec and plotted the average, the minimum, and the maximum of our the data points. Figure 4b visualizes the corresponding distribution of the measured clock synchronization precision.

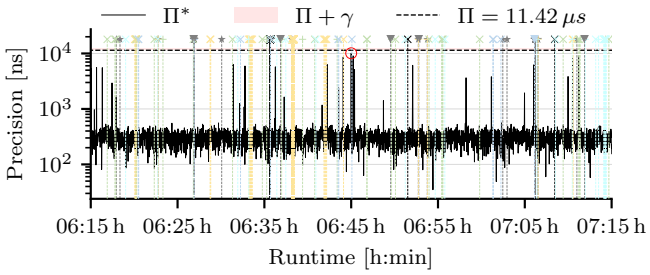


Figure 5: Showing the measured clock synchronization precision for an 1 h interval of the 24 h fault injection experiment. Colored arrows with dashed lines indicate GM clock and gray arrows with dashed lines redundant clock synchronization VM failures. Furthermore, gray stars with a dashed line mark a clock synchronization VM taking over maintaining `CLOCK_SYNCTIME`. Grandmaster clock events are color-coded in accordance with the colors used for gPTP domains in Figure 2. Therefore, GM clock events of dom_1 are encoded yellow, GM clock events of dom_2 are encoded green, GM clock events of dom_3 are encoded turquois, and GM clock events of dom_4 are encoded blue.

3.3 Fault Injection Experiments

In a second experiment, we ran fault injection experiments for 24 h to validate the robustness of our gPTP multi-domain aggregation against fail-silent GM clocks and clock synchronization VMs. To that end, we demonstrate how the fault-tolerant dependent clock of the hypervisor prevents a node from losing synchronization in case of a silent failure of an active clock synchronization VM by issuing the redundant clock synchronization VM to takeover maintaining `CLOCK_SYNCTIME`.

For the fault injection, we ran a python tool in the service VM of each ECD. The fault injection tool triggered periodic sequential shutdowns of the GM clocks hosted on each ECD with a period of 1 h, i.e., 24 fail-silent GM clocks during the experiment. In the case of redundant clock synchronization VMs, which are not GM clocks, the fault injection tool randomly triggered shutdowns with a minimum frequency of one and a maximum frequency of 12 failures per hour per node, i.e., at a maximum one fail-silent clock synchronization VM that is not a GM clock every five minutes. In summary, during our experiment, we observed 94 random fail-silent clock synchronization VMs, 48 of which were grandmaster clock failures. Note that the fault injection tool avoided injecting faults to both clock synchronization VMs of a node simultaneously since this would have violated our fault hypothesis. However, up to four clock synchronization VMs can fail simultaneously on separate nodes. Finally, unintended protocol or software faults resulting from the software stack could occur independently at any time. For example, we occasionally observed missed transmission deadlines of `Sync` packets or timeouts when `ptp4l` attempted to retrieve transmission timestamps from the Linux kernel.

In this setup we derived an upper bound of clock synchronization precision of $\Pi = 2(\mathcal{E} + \Gamma) = 11.42\,\mu s$ and a measurement error of $\gamma = 856\,ns$. Note, that differences in the upper bound of clock synchronization precision between experiments stem from varying minimum and maximum network latency measurements.

Figures 4a and 4b show the results of the fault injection experiment. For clarity, we aggregated and plotted the average (black line), minimum, and maximum (gray area) of the measured clock synchronization precision for intervals of 120 $secs$ on a logarithmic scale. We achieve an average clock synchronization precision of $322 \pm 421\,ns$ throughout 24 h. The measured clock synchronization precision depicts frequent spikes with a maximum value of 10.08 μs

at time 06:45:49 h indicated by a red circle that lies within the upper bound of precision Π and the measurement error $\Pi + \gamma = 12.28\,\mu s$. The frequent spikes, despite remaining within the bounds of clock synchronization precision and the measurement error, raise the question of their origin and accordingly means to eliminate them. Previous work on clock synchronization using the `ntpd` [17] and the `ptpd` service for Linux [16] emphasized their disposition to instability due to the feedback control of the parameters of the Linux kernel's software clocks such as `CLOCK_REALTIME` or our implementation of `CLOCK_SYNCTIME`. In [18], the author utilized memory-mapped IO granting VMs direct access to the clock register of a NIC to bypass the feedback control of `CLOCK_SYNCTIME`. The author achieved an improved robustness of the synchronized time in exchange for reduced isolation due to the NIC's internal state being exposed to co-located VMs. Furthermore, Ridoux et al. [16, 17] proposed a feed-forward robust absolute and difference clock (RADclock) and demonstrated its advantages compared to the status quo. Since then, the Linux kernel and PTP clock synchronization protocol family evolved, and new implementations have been published, such as LinuxPTP, including hardware timestamp support. Nonetheless, we cannot rule out that measured precision's instability stems from the feedback-based operation of the clocks. Unfortunately, however, the RADclock never found its way into the upstream Linux kernel impeding its use as a dependent clock for our purposes. Therefore, to test the hypothesis of a feed-forward `CLOCK_SYNCTIME` solving the instability in measured clock synchronization using LinuxPTP requires a from-scratch prototype implementation for a current version of the Linux kernel, which was out-of-scope of this work, leaving it to future work.

In Figure 5, we illustrate the time interval from 6:15:49 h to 7:15:49 h that includes the occurrence of the maximum measured clock synchronization precision indicated by a red circle. Furthermore, we plotted the failure of clock synchronization VMs, redundant clock synchronization VMs taking over maintaining `CLOCK_SYNCTIME`, and `ptp4l` transient SW stack faults. We represent clock synchronization VM failures as triangles, redundant clock synchronization VMs taking over maintaining `CLOCK_SYNCTIME` as stars, and `ptp4l` application faults as crosses. We color-coded events corresponding to a specific gPTP domain's GM clock according to the colors used in Figure 2. The transient application faults in the plotted time interval include timeouts `tx_timeout` of `ptp4l` attempting to retrieve the transmission hardware timestamp from the Linux kernel, `Sync` packet transmission deadline misses, and invalid `Sync` packet transmission deadlines passed to the kernel. For all `ptp4l` instances throughout the $24\,h$ experiment, we observed 2992 transmission timestamp timeout faults and 347 transmission deadline misses in total. The issue of transmission timestamp timeouts with the Intel i210 NIC used in our experiments, despite a sufficient configured timeout of $5\,ms$ and alternating Linux kernel versions, seems to originate in the Linux kernel's `igb` driver. However, we could not find the root cause of this fault. Note, that this issue in the existing implementation does not limit the applicability of our architecture yet it further demonstrates its effectiveness in masking faulty behavior.

4 Conclusion and Future Work

We provided an implementation of our cyber-resilient architecture utilizing open-source software. We implemented a fault-tolerant dependent clock for the ACRN hypervisor, enabling fail-silent clock synchronization VMs. Moreover, we extended OpenIL's LinuxPTP by multi-domain aggregation utilizing an FTA. Finally, we evaluated our cyber-resilient clock synchronization architecture by performing a simulated cyber-attack and a $24\,h$ fault injection experiment. The simulated cyber-attack on the fault-tolerant clock synchronization conclusively demonstrated the problem of inadequate fault containment units by reusing software stacks even in the presence of an FTA and the $24\,h$ fault injection experiment showed the robustness of the provided architecture against fail-silent clock synchronization VMs.

However, relying on several feature-rich OS stack to implement cyber-resilient clock synchronization imposes significant implementation overhead. Therefore, for future work, we identify unikernels, such as Unikraft [9], as an exciting extension to our cyber-resilient clock synchronization architecture since they enable simple, open-source software stacks that reduce design faults due to their minimal code base. Additionally, they combine predominant performance concerning runtime overhead and boot times with a small memory footprint aiding failure recovery.

Acknowledgment

The research presented throughout this paper has partially received funding from the project AI4CSM. AI4CSM receives funding within the Electronic Components and Systems For European Leadership Joint Undertaking (ESCEL JU) in collaboration with the European Union's Horizon2020 Framework Programme and National Authorities, under grant agreement n° 101007326.

References

[1] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)*, pages 1–499, 2020.

[2] Ieee standard for local and metropolitan area networks–timing and synchronization for time-

sensitive applications. *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pages 1–421, 2020.

[3] T. Broomhead, L. Cremean, J. Ridoux, and D. Veitch. Virtualize everything but time. In *Proc. of the 9th USENIX Symp. on Operating Systems Design and Implementation, OSDI 2010, Vancouver, BC, Canada*, pages 451–464, Oct. 2010.

[4] C. DeCusatis, R. M. Lynch, W. Kluge, J. Houston, P. A. Wojciak, and S. Guendert. Impact of cyber-attacks on precision time protocol. *IEEE Trans. Instrum. Meas.*, 69(5):2172–2181, 2020.

[5] M. Garcia, A. N. Bessani, I. Gashi, N. F. Neves, and R. R. Obelheiro. Analysis of operating system diversity for intrusion tolerance. *Softw. Pract. Exp.*, 44(6):735–770, 2014.

[6] Intel Cooperation. *Intel Ethernet Controller I210 Datasheet*, January 2021.

[7] H. Kopetz. Fault containment and error detection in the time-triggered architecture. In *6th International Symposium on Autonomous Decentralized Systems (ISADS 2003), 9-11 April 2003, Pisa, Italy*, pages 139–146. IEEE Computer Society, 2003.

[8] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Trans. Computers*, C-36(8):933–940, 1987.

[9] S. Kuenzer, V. Badoiu, H. Lefeuvre, S. Santhanam, A. Jung, G. Gain, C. Soldani, C. Lupu, S. Teodorescu, C. Raducanu, C. Banu, L. Mathy, R. Deaconescu, C. Raiciu, and F. Huici. Unikraft: fast, specialized unikernels the easy way. In A. Barbalace, P. Bhatotia, L. Alvisi, and C. Cadar, editors, *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021*, pages 376–394. ACM, 2021.

[10] E. Kyriakakis, K. Tange, N. Reusch, E. O. Zaballa, X. Fafoutis, M. Schoeberl, and N. Dragoni. Fault-tolerant clock synchronization using precise time protocol multi-domain aggregation. In *24th IEEE International Symposium on Real-Time Distributed Computing, ISORC 2021, Daegu, South Korea, June 1-3, 2021*, pages 114–122. IEEE, 2021.

[11] L. Lamport and P. M. Melliar-Smith. Byzantine clock synchronization. *ACM SIGOPS Oper. Syst. Rev.*, 20(3):10–16, 1986.

[12] H. Li, X. Xu, J. Ren, and Y. Dong. ACRN: a big little hypervisor for iot development. In J. B. Sartor, M. Naik, and C. Rossbach, editors, *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2019, Providence, RI, USA, April 14, 2019*, pages 31–44. ACM, 2019.

[13] E. Lisova, M. Gutiérrez, W. Steiner, E. Uhlemann, J. Åkerberg, R. Dobrin, and M. Björkman. Protecting clock synchronization: Adversary detection through network monitoring. *Journal of Electrical and Computer Engineering*, 2016, 2016.

[14] E. Lisova, E. Uhlemann, W. Steiner, J. Åkerberg, and M. Björkman. Risk evaluation of an ARP poisoning attack on clock synchronization for industrial applications. In *IEEE International Conference on Industrial Technology, ICIT 2016, Taipei, Taiwan, March 14-17, 2016*, pages 872–878. IEEE, 2016.

[15] P. Ramanathan, K. G. Shin, and R. W. Butler. Fault-tolerant clock synchronization in distributed systems. *Computer*, 23(10):33–42, 1990.

[16] J. Ridoux and D. Veitch. The cost of variability. In *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 29–36. IEEE, 2008.

[17] J. Ridoux, D. Veitch, and T. Broomhead. The case for feed-forward clock synchronization. *IEEE/ACM Trans. Netw.*, 20(1):231–242, 2012.

[18] J. Ruh. Towards a robust mmio-based synchronized clock for virtualized edge computing devices. In *26th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2021, Vasteras, Sweden, September 7-10, 2021*, pages 1–8. IEEE, 2021.

[19] J. Ruh, W. Steiner, and G. Fohler. Clock synchronization in virtualized distributed real-time systems using IEEE 802.1as and ACRN. *IEEE Access*, 9:126075–126094, 2021.

[20] F. B. Schneider. A paradigm for reliable clock synchronization. Technical report, CORNELL UNIV ITHACA NY DEPT OF COMPUTER SCIENCE, 1986.

[21] W. Steiner and B. Dutertre. The ttethernet synchronisation protocols and their formal verification. *Int. J. Crit. Comput.-Based Syst.*, 4(3):280–300, Dec. 2013.

[22] W. Steiner and H. Kopetz. The startup problem in fault-tolerant time-triggered communication. In *2006 International Conference on Dependable Systems and Networks (DSN 2006), 25-28 June 2006, Philadelphia, Pennsylvania, USA, Proceedings*, pages 35–44. IEEE Computer Society, 2006.

[23] A. Treytl, G. Gaderer, B. Hirschler, and R. Cohen. Traps and pitfalls in secure clock synchronization. In *2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 18–24. IEEE, 2007.

[24] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In S. Molnár, J. R. Heath, O. Dalle, and G. A. Wainer, editors, *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, SimuTools 2008, Marseille, France, March 3-7, 2008*, page 60. ICST/ACM, 2008.