Time Synchronization Writeup

Hung Hua

07/31/2024

1 Problem

You have nodes $N_1, N_2, \ldots, N_m$ on a network with clocks showing times $C_1(t), C_2(t), \ldots, C_m(t)$, respectively, at time t. Events occur at times $t_0, t_1, \ldots, t_N$. What is the difference $|C_i(t_k) - C_j(t_k)|$ within this network, and are we able to keep the difference under 0.0005s (500μs) among all pairs of distinct nodes? Are we able to maintain this difference for a long period (e.g., months) of time? How will the answer change if we consider the (more preferred) time difference $|C_i(t_k) - t_k|$ instead?

In short: How well can we minimize $|C_i(t_k) - C_j(t_k)|$ and $|C_i(t_k) - t_k|$?

2 Preliminary

We provide some preliminary concepts and notation before proceeding.

2.1 Delays

The time delay between a sender (a main/kernel thread) sending a message and the receiver receving a message is dependent on four variables [3, 4]:

1. Send (notating as T_S): Time for message from **MAIN THREAD to NETWORK INTERFACE**.

2. Access (notating as T_A): Time for message from **NETWORK INTERFACE to PROPAGATION MEDIUM**

3. Propagate (notating as T_P): Time for message from **PROPAGATION MEDIUM to NETWORK INTERFACE OF RECEIVER**.

4. Receive (notating as T_R): Time for message from **NETWORK INTERFACE OF RECEIVER to ABILITY TO TIMESTAMP ON RECEIVER**.

Thus, the total time between sender sending a useful message and the receiver being able to timestamp is:

$$T = T_S + T_A + T_P + T_R \tag{1}$$

For example, a slow network will have high T_P, a busy OS will have high T_S, T_R, and T_A (a dedicated microcontroller, which often just does one thing (i.e., executes a single program) will, with access to networking hardware, have a very small T_R, T_A), an occupied kernel or having no high priority access to an interrupt handler to timestamp will induce a high T_R, an unreliable interface card can have a high T_A and T_R, etc.

The variables in (1) are random[1], so they have statistical parameters (e.g., expectation and variance) associated with them. Thus, if needed, we can conduct tests to understand their probabilistic behavior, which will allow us to make some reasonable predictions and even measure the performance of our chosen hardware.

Note that delays with high variance (e.g., a consistenly congested network, a non-RTOS, etc.) is something we should avoid. Even if one of the variables in (1) has high expectation, with a small enough variance we can simply account for it in the software (or hardware) as a simple offset to remove.

2.2 Algorithm

We employ a variant of RBS (Reference Broadcast Synchronization) [3, 4] to assist in the synchronization of multiple clocks of devices within a sensor network. The main benefit of RBS is the fact that the two variables T_S and T_A do not depend on the receiving node. In RBS, there is a broadcaster (preferably a low stratum time server, but this is not a necessity) that periodically sends the receiving nodes a timestamp; the receiving nodes then communicate with one another to correct their respective clock offsets. Our variation of RBS differs in this latter part: the receiving nodes do NOT communicate with one another. Instead, the receiving nodes periodically broadcast their data (and local timestamps) to a processing node that has access to a low stratum time server, after which the data can be assigned absolute time stamps, if needed. See Figure 1.

A rough description of the algorithm is described as follows:

Suppose we have m nodes N_1, N_2, ..., N_m, with clocks $C_1(t), C_2(t), \ldots, C_m(t)$, respectively, at a real-time t. The difference $|C_i(t) - t|$ is ideally 0, as this would imply that there is no time difference

[1]By which we mean each variable is a random variable, so they're real-valued functions defined on some probability space Ω that satisfies a measurability condition w.r.t. to the reals equipped with its Borel algebra. In short, we have a notion of "expectation" and "variance" associated with these variables, implying that we can, if needed, conduct some statistical tests to provide accurate estimates of these values. So, throughout this document I may refer to $T(\omega)$ rather than T to emphasize a sampled delay, where $\omega \in \Omega$.

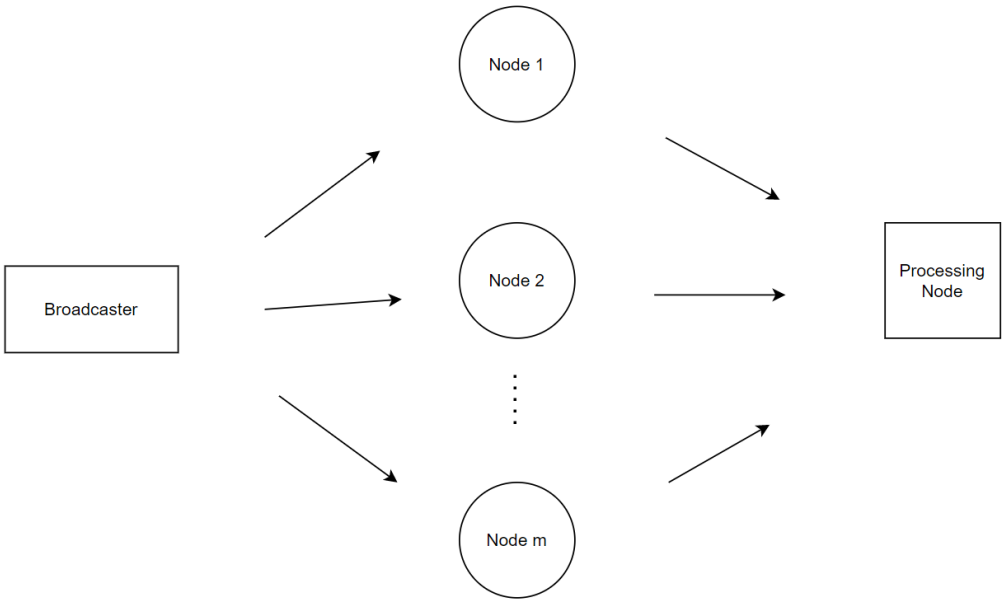


Figure 1: Clock synchronization setup scheme via RBS variant. Note: broadcaster and processing nodes could be on the same piece of hardware.

between node N_i's clock and real-time. However, the difference is most likely non-zero, and it is a goal of clock synchronization to minimize this difference. An easier problem is to minimize $|C_i(t) - C_j(t)|$, which is the more suitable problem given our needs.

With RBS, there is a broadcasting node, N_B, with clock $C_B(t)$ at real-time t. Periodically, this node would broadcast $C_B(t)$ to each node N_i, and each node would update their clocks at time t via[2]

$$C_i(t + T_S + T_A + T_P^{(i)} + T_R^{(i)}) \leftarrow C_B(t) \tag{2}$$

Notice that with this update[3], the real-time difference between the clocks of nodes N_i and N_j at

[2]the superscript on T_P and T_R indicates the dependence of these delays on the receiving node; the lack of a superscript on T_S, T_A indicates that these delays are the same for all receiving nodes.

[3]Fact: Assuming negligible dift, for any $t_1, t_2 \in \mathbb{R}$, $C(t_1 + t_2) = C(t_1) + t_2$. Thus, for any t, $C(t) = C(0) + t$, so clocks are equivalent up to a single reference point (at time 0, for example)

time t is[4]

$$\Delta_{ij}^{\text{real-time}}(t) = |C_i(t) - C_j(t)| = |(T_P^{(i)} - T_P^{(j)}) + (T_R^{(i)} - T_R^{(j)})| \tag{4}$$

Then, after some defined period of time, each node will broadcast their timestamps (along with any associated data, such as sensor data gathered by a Data Acquisition Unit) to some post-processing node N_P that has access to a low stratum time server. Suppose N_P obtains data

$$D_i = [(d_0, C_i(t_0)), (d_1, C_i(t_1)), \ldots, (d_{N_i}, C_i(t_{N_i}))] \tag{5}$$

from node N_i at time t. Then, N_P stores this as a 3-tuple $(C_P(t), C_i(t - T^{(i)}), D_i)$. Since N_P is a low-stratum time server, it uses its more accurate clock $C_P(t)$ as the time reference, and transforms each datum $(d_*, C_i(t_*))$ in the dataset D_i via[5]

$$d_* \leftarrow d_* \tag{6}$$

$$C_i(t_*) \leftarrow C_P(t) + C_i(t_*) - C_i(t - T^{(i)}) = C_P(t) + C_B(t_*) - C_B(t) + T^{(i)} \tag{7}$$

Thus, the time difference between nodes N_i and N_j during the post-processing stage for any datum d_* at time t_* would then be

$$\Delta_{ij}^{\text{post-time}}(t_*) = |T^{(i)} - T^{(j)}| \tag{8}$$

$$= |(T_S^{(i)} - T_S^{(j)}) + (T_A^{(i)} - T_A^{(j)}) + (T_P^{i\to P} - T_P^{j\to P})| \tag{9}$$

Therefore, under the assumption of good, reasonable hardware with negligible random errors and frequent broadcasts to eliminate drift, the time-difference between nodes N_i and N_j would be

$$\Delta_{ij}^{\text{real-time}} = |(T_P^{(i)} - T_P^{(j)}) + (T_R^{(i)} - T_R^{(j)})| \tag{10}$$

$$\Delta_{ij}^{\text{post-time}} = |(T_S^{(i)} - T_S^{(j)}) + (T_A^{(i)} - T_A^{(j)}) + (T_P^{(i\to P)} - T_P^{(i\to P)})| \tag{11}$$

Although it may seem like we're simply trading a set of errors for another when considering the differences between real-time and post-time, the main benefit of post-time difference is that the difference $|C_i(t) - t|$ is minimized, as opposed to the minimization of $|C_i(t) - C_j(t)|$ during real-time.

[4] whereas if there was no broadcaster and we relied on inter-nodal communications, the time difference would've been more in line with

$$\Delta_{ij}^{\text{real-time}}(t) = |(T_S^{(i)} - T_S^{(j)}) + (T_A^{(i)} - T_A^{(j)}) + (T_P^{(i)} - T_P^{(j)}) + (T_R^{(i)} - T_R^{(j)})| \tag{3}$$

[5] the datum d_* could be sensor data array, event triggers, etc.

3 Test Results (Part 1)

This section outlines the results we gathered from conducting an experiment using

3.1 Nodes

The representation of the nodes from Figure 1 are:

1. **One Laptop** (represents both N_B, N_P)
2. **Two RPI4s** (one reperesents N_1, the other N_2)

A third RPI4 is used, but it's not repesented as a node. This third RPI4 acts as (simulates) the valve and MCC switch (they serve as the event triggers in this experiment). This third RPI4 has a HAT that provides a (2-channel) 16-bit DAC, so it was able to simulate both the stem thrust from the valve (channel 1) and motor current from the MCC (channel 2).

See the results in Figures 10 and 11. Now, please note that the time difference is given as statisical parameter (the average), and it represents the real-time difference $\Delta_{12}^{\text{real-time}}$, as opposed to the post-time difference.

Again, the errors shown are mainly due to the random error

$$\Delta_{12}^{\text{real-time}} = |(T_P^{(1)} - T_P^{(2)}) + (T_R^{(1)} - T_R^{(2)})| \tag{12}$$

We can often assume that the propagation delay is negligible [3], so (12) can be approximated as

$$\Delta_{12}^{\text{real-time}} \approx |T_R^{(1)} - T_R^{(2)}| \tag{13}$$

Since the RPI4s do not use a real-time OS, the error (13) can be relatively large.[6]

In the final product, having a dedicated submodule whose sole purpose is to acquire broadcasts from a broadcaster node and updating its relative time clock will reduce (13) considerably, allowing for far better results.

[6] So, for example, the OS dedicates time for other services to grab hold of the available time slices; timestamping isn't one of its priorities.

4 Conclusion (Part 1)

With the consideration of the 5 outliers, the average time difference resulted in **367 us** with a sample standard deviation of around **658 us**.

Without the 5 outliers, the average time difference is **160 us** with a sample standard deviation of around **75 us**. Therefore, 95 % of the time (assuming the distribution is Gaussian), the time difference between the clocks of Node N_1 and N_2 will be between **10 us** and **310 us**.

Our nodes N_1 and N_2 in this setup are RPI4s, so each node are full fledged Operating Systems with services running in the background hogging computing resources, causing the intermittent outliers to enter our dataset. In the final product, we can rely on the ability to timestamp at the network interface level and we will not have unnecessary processes running in the background, which will yield far better results than the ones shown in this experiment.

5 Test Results (Part 2)

We conducted an experiment similiar to the experiment conducted in Part 1 described above. The differences that affected the results the most are:

- Incorporated the the use of hardware timestamping (as opposed to software timestamping) with wired ethernet and PTP.
- Used BeagleBone Black boards rather than the Raspberry PI 4s as the nodes.
- Incorporated the usage of PRU submodules.

The hardware timestamping (as opposed to software timestamping) improved the results drastically, as the timestamping is performed by the network interface, bypassing the heavy overhead typical of timestamping at the software level.

The BeagleBone Black board uses the Sitara AM3358AZCZ100 processor from Texas Instruments; this processor includes (in addition its 1 GHz core processor) two 200 MHz microcontrollers dedicated to real-time applications (these microcontrollers are referred to as Programmable Real-time Units, or PRUs, for short). The existence of these two PRUs allow, in total, three processors to execute code in parallel; thus, with one PRU executing code to collect analog data and the other executing code for the collection (and association) of timestamps provided by the kernel (this collection is done via shared memory) allows the kernel to focus on less time-sensitive tasks (e.g., hardware resource management, exposing communication ports for remote access and firmware programming, etc.). Thus, even though the BeagleBone Black board does not use a real-time OS (similar to the RPI4), it has PRUs that can be dedicated for real-time tasks (unsimilar to the RPI4).

For this test, each node collected around 3700 timestamps, each spaced 1 second apart. Some statistical information about these 3700 timestamps is shown in Figure 2.

number of samples	3675
average time difference	87.1 ns
sample std of time difference	64.2 ns
average + 3*(sample std)	279.7 ns

Figure 2: statistical results for Part 2 of our experiment using BeagleBone Black

6 Conclusion (Part 2)

We have two nodes (node N_1 with a clock $C_1(t)$ and node N_2 with a clock $C_2(t)$) on a local network whose clocks we'd like synchronized to within a bound of 500 microseconds; that is, we'd like

$$|C_1(t) - C_2(t)| \leq 500,000 \text{ ns} \tag{14}$$

for all t. The results provided in the previous section indicate that this has been achieved.

We'd like to note that the results are primarily due to three important factors: (1) hardware timestamping with wired ethernet and PTP, (2) the BeagleBone Black boards, and (3) the PRUs located within the Sitara AM3358AZCZ100 processor. Due to these three factors, we have collected data that indicates

$$|C_1(t) - C_2(t)| \leq 279.7 \text{ ns} \tag{15}$$

for many t. The upperbound in (15) is 0.056% of the upperbound in (14).

Assuming that the distribution from which the results are gathered is Gaussian, we may assume that over 99% of the time, the time difference between the two DAU nodes will be within 279.7 ns of each other. With our time difference budget of 0.005 s = 0.5 ms = 500 μs = 500,000 ns, we can safely say our result of **279.7 ns** is well below our **500,000 ns** budget.

7 Test Results (Part 3)

For this part of the test, we're repeating experiments similar to Part 2 above except for one difference: Software PTP will be experimented upon, in addition to Hardware PTP. The reason why we're considering Software PTP even though Hardware PTP will, with high probability, always outperform Software PTP, is due to the potential hardware constraints at Nuclear Power Plants. When performing Hardware PTP, all of the networking related hardware (this includes the NIC at the PTP masters and clients, switches within the network, routers, etc) will have to be PTP compliant - the experiment in Part 2 assumed PTP compliant hardware, which is a requirement that most likely will not be met at Nuclear Power Plants. Without PTP compliant hardware, we'll have to rely on a less performant variant of PTP, called Software PTP, where the big difference is that the timestamps are generated at the higher end of the network stack (farther away from the NIC).

This part of the experiment is broken into three parts, each consisting of a slightly different experimental configuration. As mentioned above, we are primarily comparing Hardware PTP and Software PTP, but we are also considering the use of a Virtual Local Area Network (VLAN) for the Software PTP experiments in order to limit the effects of network traffic. For a short summary of the configurational details and results, please see Figure 3 below. By "PTP Network" we are referring to a network with PTP-compliant hardware.

Config Type	PTP Type	VLAN?	PTP Network?[7]	Network Traffic	$P(X \leq 0.5 \text{ ms})$
1	Hardware	No	Yes	Arbitrary[8]	≈ 1
2	Software	No	No	Heavy[9]	≈ 0.31
3	Software	Yes	No	CN Traffic[10]	≈ 1

Figure 3: Experiment Configurations. Ordered by date of experiment[11]. Probabilities based on KDE estimate X of dataset collected from corresponding experiment.

7.1 Experimental Configuration 1

See Figure 3, Configuration Type 1, for the basic configurations of this experiment. For the basic experimental setup, see Figure 4, and for the probability distribution of the dataset collected, see Figure 5.

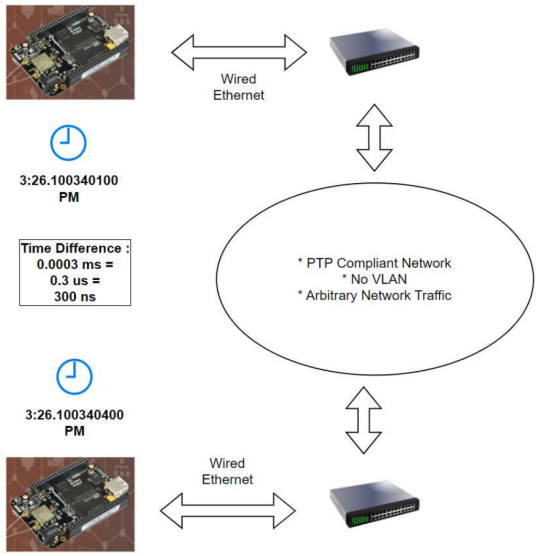
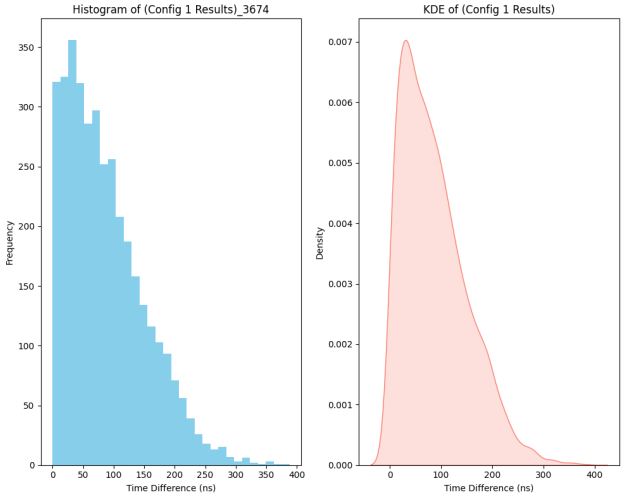


Figure 4: Experiment 1 Setup



(a) Histogram and KDE estimate of results. Non-zero probabilities for negative values in KDE due to smoothing, and should be ignored.

x	$P(X \leq x)$
9.2	0.075
32.5	0.227
55.7	0.383
102.2	0.645
171.9	0.882
241.6	0.978
288.1	0.993
311.4	0.996

(b) CDF of KDE estimate X. First column contains the time difference in nanoseconds. Second column contains the probabilities.

Figure 5: Experiment 1 Results

7.2 Experimental Configuration 2

See Figure 3, Configuration Type 2, for the basic configurations of this experiment. For the basic experimental setup, see Figure 6, and for the probability distribution of the dataset collected, see Figure 7b.

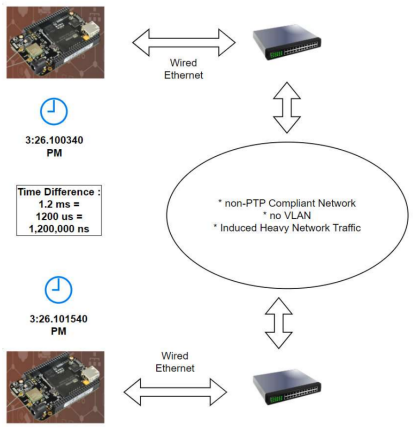
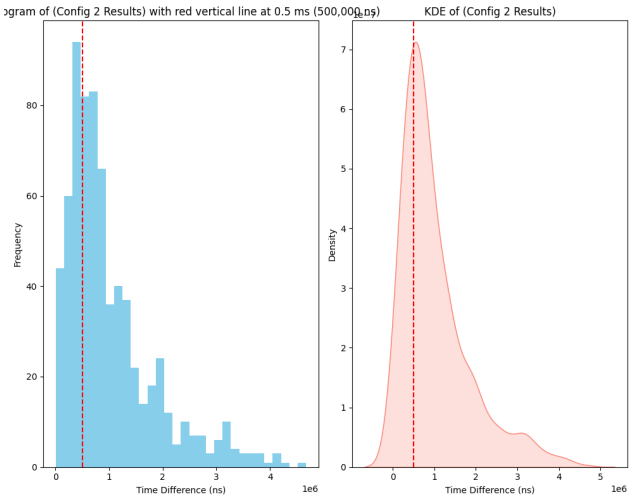


Figure 6: Experiment 2 Setup



(a) Histogram and KDE estimate of results. Non-zero probabilities for negative values in KDE due to smoothing, and should be ignored.

x	$P(X \leq x)$
4996	0.043
523506	0.316
1042016	0.632
1560526	0.796
2079036	0.8849
2597546	0.931
3634566	0.984
4671586	0.999

(b) CDF of KDE estimate X. First column contains the time difference in nanoseconds. Second column contains the probabilities.

Figure 7: Experiment 2 Results

7.3 Experimental Configuration 3

See Figure 3, Configuration Type 2, for the basic configurations of this experiment. For the basic experimental setup, see Figure 8, and for the probability distribution of the dataset collected, see Figure 9.

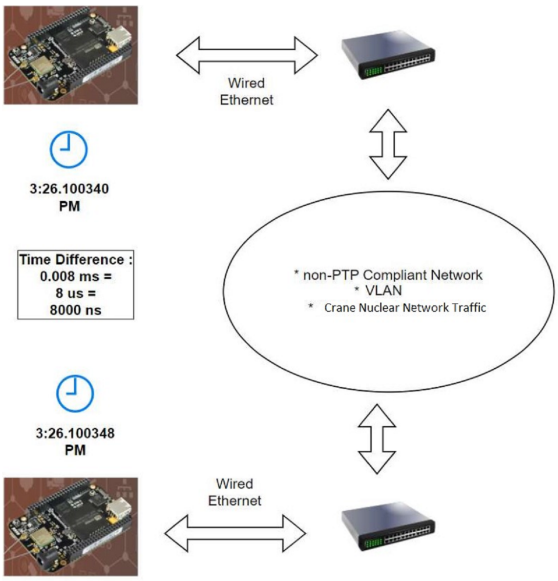
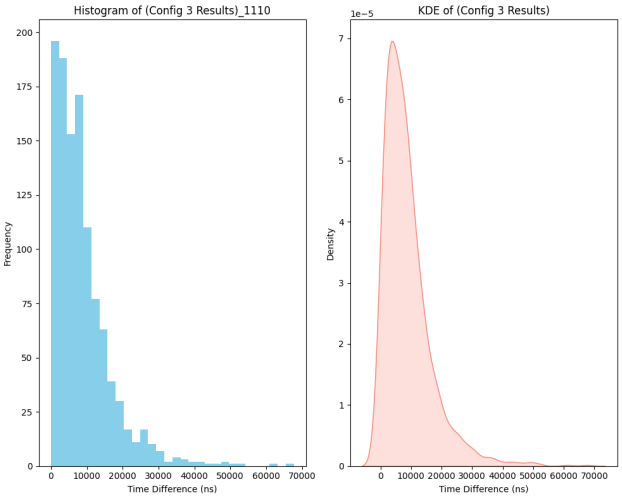


Figure 8: Experiment 3 Setup



(a) Histogram and KDE estimate of results. Non-zero probabilities for negative values in KDE due to smoothing, and should be ignored.

x	$P(X \leq x)$
78.5	0.062
2476.8	0.194
6074.3	0.439
7273.5	0.516
10870.9	0.707
20464.3	0.922
32455.9	0.980
67709	0.999

(b) CDF of KDE estimate X. First column contains the time difference in nanoseconds. Second column contains the probabilities.

Figure 9: Experiment 3 Results

8 Conclusion (Part 3)

Given the results shown above, it is clear that Software PTP is not as performant as Hardware PTP, and in fact without a VLAN provided, Software PTP will not allow us to meet the time synchronization requirement of 0.5 ms. However, with a VLAN, Software PTP improves, and as the results indicate, the improvements can allow us to comfortably meet the time synchroniztion requirement. However, Software PTP is very dependent on the network on which it is running, so although this experiment shows that Software PTP within a VLAN suffices within the Crane Nuclear network, it doesn't prove that it's possible within an arbitrary network. Therefore, as a next step, we will consult a few Nuclear Power Plants and gather their network infrastructure details and run similar experiments within a virtual network configured to (as closely as we can) match a network state similar to a network with the network state governed by the details they provide.

References

[1] skeeng02-data-eng-Signature-Data.

[2] https://peps.python.org/pep-0564/annex-clocks-resolution-in-python.

[3] Ill-Keun Rhee, Jaehan Lee, Jangsub Kim, Erchin Serpedin, and Yik-Chung Wu. Clock synchronization in wireless sensor networks: An overview. 2009.

[4] Fikret Sivrikaya and Bülent Yener. Time synchronization in sensor networks: A survey.

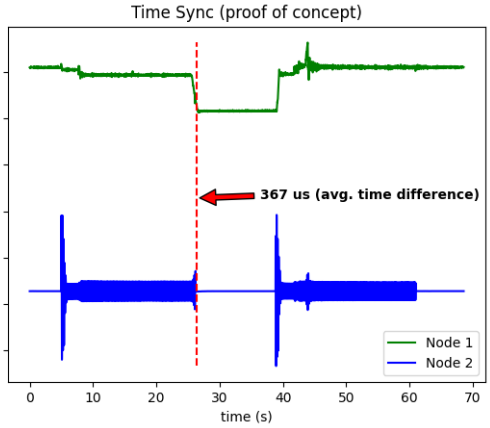


Figure 10: Main Result. Time difference is the average. Our time difference budget is 0.5 ms = 500 us. Stem thrust data (from the valve) and motor current data (from the MCC) gathered from [1]. Resolution of the timestamps is 84 ns [2].

sample	time difference (us)
1	218
2	148
3	**1493**
4	163
.	.
.	.
.	.
49	218
50	223
average	**367**

Figure 11: Time difference between the clocks on our two nodes N_1 and N_2. Dataset consists of 50 samples with 5 outliers (e.g., time differences exceeding 1 ms). Sample outlier in red. Resolution of the timestamps is 84 ns [2].