Time Synchronization Under Temperature and Distance Variations

Matheus Martins Susin, Lucas Francisco Wanner
Institute of Computing
University of Campinas
Av. Albert Einstein 1251, 13083-852, Brazil
matheus.susin@students.ic.unicamp.br, lucas@ic.unicamp.br

Davi Resner, Antônio Augusto Fröhlich
Software/Hardware Integration Lab
Federal University of Santa Catarina
PO Box 476, 88040-900 – Florianópolis, SC, Brazil
davir@lisha.ufsc.br, guto@lisha.ufsc.br

***Abstract*—The Trustful Space-Time Protocol (TSTP) allows for time synchronization to be performed upon receiving any message from another node in a sensor network, removing the need for explicit synchronization messages. Previous work has shown that TSTP performs well under controlled experimental environments. In this work, we analyze how the quality of synchronization in TSTP is affected when nodes are communicating over varying distances, and across a temperature gradient. The results show that, while distance and packet loss has only a small effect on the quality of the synchronization, high temperatures of 80°C and up can negatively affect it. The results from the temperature variation experiments are incorporated into a network simulator to make its clock drift model more realistic, as well as tied to current node temperature.**

***Index Terms*—time synchronization; clock synchronization; WSN; TSTP**

I. Introduction

Many wireless sensor network applications require time synchronization. Failing to coordinate action may lead to security risks, sensor inaccuracy, and loss of efficiency in power usage, given that duty cycles will be out of sync. The Trustful Space-Time Protocol (TSTP) [1] [2] provides time synchronization by including a timestamp on the header of each message exchanged by its nodes. The sensor nodes are slaves to a unique gateway, and always adjust their clocks to it.

Many factors can negatively (or even positively, as will be shown) influence the quality of time synchronization. We focused on two independent variables, temperature and distance, and analyzed the behavior of the protocol under several different conditions. First, we kept the sensor node on a fixed position, and performed tests with the gateway, or master node, being put in different spots around the room, mimicking differences in latency and possible interference from other electronic devices. We then placed the gateway in a box, the sensor right next to the box, and with the help of a lamp and a digital thermometer, exposed the gateway to different temperatures and took the same measurements.

For the remainder of this work, we describe the experiments in more detail, provide snippets and statistical analyses of the results, and discuss what can be learned from it. The experiments with distance were performed on a room with many electronic devices, several of which were communicating wirelessly, and showed that not only does distance play a small role in the quality of the synchronization, but longer distances sometimes result in slightly better averages. The experiments with temperature showed that, for this particular pair of crystals, the temperature of 70°C applied to the usually slowest one minimized the average offset that was perceived by the protocol.

We also describe how we used the results collected from the temperature variation experiments to enrich the clock drift model of the Castalia [6] framework for the Omnet++ [5] simulator.

II. Materials

The experiments were performed using two EPOSMote 3 (eMote3) devices, both featuring a CC2538 SoC from Texas Instruments, with an ARM Cortex M3 CPU at 32MHz, and IEEE 802.15.4 compliant radio. There is no hardware distinction between sink (gateway) and source (sensor). The gateway's clock is the reference, and is never adjusted in any way. The USB cable is used on the gateway strictly to power it. The sensor node makes the clock offset measurements and outputs them to a USB cable connected to a computer, which simply collects the measurements into a text file.

For the temperature experiments, a polystyrene box (Figures 1, 2) was assembled with a small hole in the lid for the wires that power the halogen lamp. Two holes were made on the side, close to the bottom, one for the thermometer probe, and another one for the cable that powers the eMote3 inside. A significant difference in temperature stability inside the box with the lamp turned on was perceived when the holes were sealed tighter, possibly because the colder air from the room wouldn't cool the metallic probe. The lamp has a dimmer attached to its wiring, which we used to control how much it heats the air inside the box.

For the spatial experiments, our Computer Systems Laboratory was used. It is a single 1335x688cm rectangular

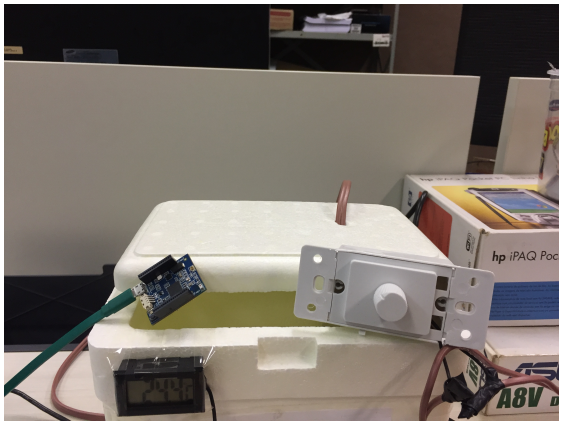

Fig. 1. Exterior of the box, the source node (sensor), the dimmer, and the digital thermometer display.

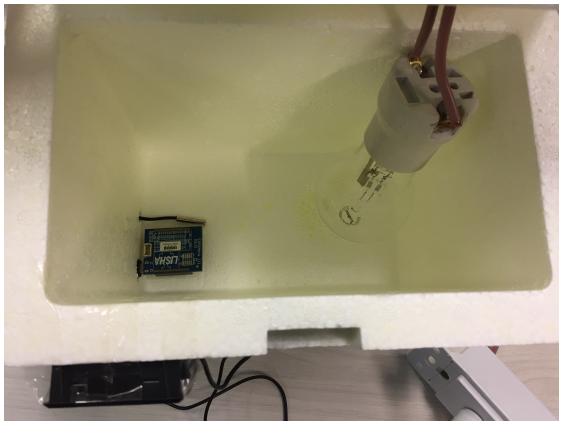

Fig. 2. Interior of the box, the sink node (gateway), the lamp, and the digital thermometer probe.

```
Line 1: TSTP::keep_alive():keep_alive = 0
    x20003bf4 =>{v=0,t=T,co=100,tr=1,s=0,ot
    =133846696,o=(10,10,0),lt=4200498129,l
    =(10,10,0)},st=5
Line 2: TSTP::update: Keep_Alive: {v=0,t=T,co
    =100,tr=0,s=0,ot=131266724,o=(0,0,0),lt
    =4210129171,l=(0,0,0)},st=5
Line 3: TSTP::Timekeeper::update: adjusted timer
     offset by -444
Line 4: TSTP::Timekeeper::update: synchronizing
    with (0,0,0)
Line 5: now() = 134813375
```

Fig. 3. Sample output collected from the sensor node during time synchronization.

room, with multiple electronic devices in use, and several people moving around all day. Data collected during the experiments include an estimation on how many packets have been lost.

III. Methods

In order to analyze the time synchronization accuracy of TSTP, we log synchronization messages received and transmitted in the system. One iteration of the clock synchronization is shown in Figure 3.

The first line shows a message from the sensor to the gateway, at time 4200498129, explicitly requesting a timestamp to synchronize its clock. Line 2 shows the gateway's response, with the corresponding high-accuracy time stamp of 4210129171 extracted from TSTP's message header. The sensor node uses this time stamp, as well as a precise measurement of the local clock when the radio interface receives the message, to adjust its clock via the Timekeeper class (the precise description of TSTP's time synchronization mechanism is shown in previous work [1], and is out of scope for this work). Line 3 shows that the sensor node detected, and corrected, a clock offset of -444 ticks. The fourth line simply informs the geolocation of the node that the clock is being adjusted to, which is used as a way to uniquely identify it. The final line corresponds to the time reported by the system, in units of clock tick count. We know that, since the piezoelectrical crystal has a frequency of 32MHz, each clock tick corresponds to 31.25ns if the crystal is perfectly cut and kept at ideal temperature. IEEE 802.15.4 defines an acceptable clock drift rate as being of, at most, 0.004% of the frequency, or 40 parts per million (ppm). Given that this clock synchronization process happens every 3 seconds, we establish 120 microseconds, or 3840 ticks, as the target maximum deviation.

To parse this output, we wrote a script able to generate a CSV file that contains the "now" value paired with the offset, negative or positive. Since we are interested only in the value of the deviation, and not on which crystal has the highest frequency, the sign of the offset is removed, and internally, statistical calculations always take its absolute value.

Lost packets are registered when two or more "keep_alive" packets are sent in sequence, with no reply in between.

There are two noteworthy cases that may cause great impact on the standard deviation, but not so much on the average offset. The first one is a complete reset on the connection, evidence by a huge setback on the offset, which follows a sequence of lost packets, indicating that the gateway died and recovered at some point. Since the experiments are not concerned with major hardware or software faults, the results in which this happened were discarded.

The other case can be observed when there is a large negative offset, meaning that the sensor had to set its timer back, followed by a large positive offset, meaning that the sensor realized the mistake from the next packet that arrived and corrected it. This is possibly due to a calculation error in the protocol's implementation. Adding

TABLE I
DISTANCE VARIATION

Set	Avg	Stddev	MAD	Loss	> 40ppm
close	287.589	566.313	34.0	0.347%	0.262%
around	246.159	58.296	34.0	0.262%	0.000%
corner	292.011	574.523	42.0	6.195%	0.661%
outside	187.030	829.047	67.0	3.212%	0.716%

both values typically results in an offset value that isn't much larger than the other values around it.

Each experiment runs for 15 to 20 minutes, and consists of logging the adjusted offset every 3 seconds. Part of the experiments are grouped in sets of experiments which share the same location, and the rest are grouped in sets that share the same temperature, depending on what condition is being tested. The total time of a set is always equal to or larger than 60 minutes, resulting in 1200 samples, including losses, and 1116 in the worst case, discounting the losses.

IV. Experimental Results: Distance Variation

In this experiment, the sensor was kept on a fixed position, and the gateway was placed in a different position for each step, as described below.

1) "close" - 10cm from the sensor, with no obstructions;
2) "around" - 1m from the sensor, with a 45cm wide desktop between the nodes;
3) "corner" - 7.4m from the sensor;
4) "outside" - 6.8m from the sensor, outside the room, and running about 8°C hotter than the sensor.

Table I contains the output of our analysis tool. All statistics are computed over the magnitude of the offset. 'Avg' is the mean of all values. The standard deviation is provided for completeness, but due to the outliers mentioned previously, they are not a good measure of the deviation. MAD is the median absolute deviation, a robust measure of variability. The loss is the ratio of keep alive packets that were sent in succession without receiving a response. The final column is the percentage of readings that were above the limit of 40 ppm.

The second experiment had a better average than the first one, with no offset going over the standard, even with a metal box and a running computer between the nodes between. The third experiment has a significantly high loss ratio, but even then, the average offset was not much higher than the previous tests. The final experiment shows that, despite the number of outliers growing, the average offset was the best one among the distance tests, and even when compared to the temperature tests, it remains the second best average.

V. Experimental Results: Temperature Variation

This section presents the results obtained from the experiments in which the gateway was placed inside a polystyrene box. With the lamp on, we allowed the air inside the box to reach a desired temperature, as informed by a digital thermometer, then waited at least 1 hour for the temperature to stabilize, dimming the lamp when necessary. We discarded the experiments in which the deviation between the desired temperature and the one shown on the thermometer surpassed ±1°C at any moment during its execution. Table II shows the results, and Figures 4, 5, and 6 show the corresponding plots.

Aside from the pathological case at 65°C, where nearly

TABLE II
TEMPERATURE VARIATION

Set	Avg	Stddev	MAD	Loss	> 40ppm
45°C	553.965	908.989	40.5	0.350%	0.175%
50°C	587.801	608.963	38.0	0.393%	0.000%
55°C	578.098	271.469	37.0	0.350%	0.000%
60°C	549.115	82.640	39.0	0.394%	0.000%
65°C	521.931	1258.136	38.0	0.262%	0.962%
70°C	156.602	76.421	34.0	0.262%	0.000%
75°C	228.379	66.873	41.0	0.262%	0.000%
80°C	687.415	564.640	49.0	0.262%	0.000%
85°C	1356.602	635.996	57.5	0.350%	0.350%
90°C	1975.607	443.361	71.0	0.131%	0.262%
95°C	2979.806	314.979	87.0	0.790%	0.707%

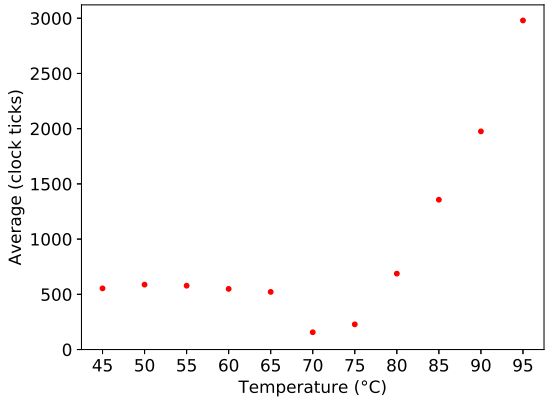

Fig. 4. Temperature vs. average offset.

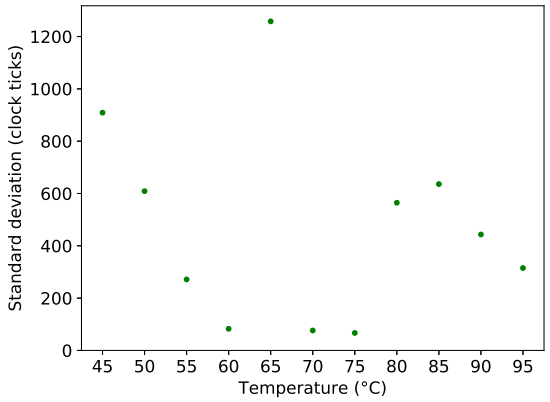

Fig. 5. Temperature vs. standard deviation of the offset.

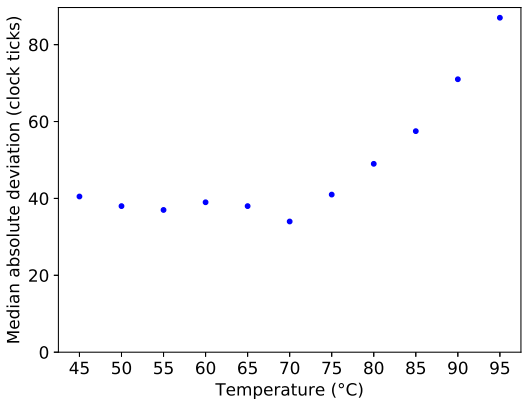

Fig. 6. Temperature vs. median absolute deviation of the offset.

```
simtime_t TimerService::getClock() {
       return simTime() * timerDrift;
}
```

Fig. 7. Castalia's static clock drift example. simTime() is an Omnet++ function that returns the actual simulation time. timerDrift is the clock drift ϕ.

1% of the adjustments were above the goal, we can see that the system remained within the specification up until 95°C, where the average was already high, the packet loss was tremendous, the MAD was the highest one yet. There is a valley between 65°C and 75°C where the synchronization was excellent. The curve of the MAD seems to follow that of the average, while the standard deviation proved itself to be quite erratic, being heavily influenced by the outliers.

VI. Simulation Model: Temperature Variation

With the collected data, we are able to implement a simulation model tying clock drifts to node temperature. We describe the process in this section. We use the OMNET++ [5] (version 4.6) simulator with the Castalia (version 3.3) framework [6]. Castalia simulates clock drifts in a simplistic way, with a fixed floating-point value close to 1 that is multiplied to every time interval calculation. We changed this fixed value to the result of a normal distribution with the mean and standard deviation tied to the temperature, according to the results we collected shown in Section V.

A. Varying Clock Drift

Castalia provides a TimerService API to manage time with a clock drift model. Clock drift ϕ is modeled as a constant floating point number determined at initialization time. It is defined as $\phi = 1 + normal(0, \sigma)$, where $(0, \sigma)$ are the mean and standard deviation of a normal distribution. σ is determined by the user as a simulation parameter. The value of the normal distribution is truncated to at most ± 3 standard deviations from the mean. ϕ is then multiplied to each time interval calculation, such as in the example code shown in Fig. 13, which returns the current clock value.

```
simtime_t TimerService::getClock() {
    simtime_t now = simTime();
    lastRead += (now - lastReadTime)
                * getTimerDrift();
    lastReadTime = now;
    return lastRead;
}
```

Fig. 8. Dynamic clock drift application example. lastRead and lastReadTime are attributes initialized at 0.

```
40 0 0
45 5.770468e-06 9.468635e-06
50 6.122927e-06 6.343364e-06
55 6.021854e-06 2.827802e-06
60 5.719947e-06 8.608333e-07
65 5.436781e-06 1.310558e-05
70 1.631270e-06 7.960520e-07
75 2.378947e-06 6.965937e-07
80 7.160572e-06 5.881666e-06
85 1.413127e-05 6.624958e-06
90 2.057923e-05 4.618343e-06
95 3.103964e-05 3.281031e-06
```

Fig. 9. Temperature model file. The columns represent, respectively, a temperature, its mean clock drift and standard deviation of the clock drift.

This is a simplistic model of clock drift. If the frequency offset were a static value, two given clocks would always drift by the same value for a given period of time, which is not what we observed in the experiments shown in Sections IV, V. The considerable change in the offsets suggests that clock drift is not a constant value.

To incorporate a dynamic clock drift (one that varies over time) into Castalia, we changed the timerDrift attribute to a method call. We also need to make sure that the clock will never "go backwards" (i.e. in any two calls to getClock(), the second one must never return a value smaller than the first one). The resulting code is shown in Figure 8. Next we show how we tie the getTimerDrift() method to the current temperature.

B. Temperature Model

We incorporate a temperature parameter to Castalia's ResourceManager module and tie the clock drift to it. Temperature is initialized with a user-configurable simulation parameter, and may be changed in runtime by any class using the appropriate getter and setter methods.

To determine how the temperature will influence the timer drift, we look at the simulation results shown in Table II. In those experiments, the offset measurements were taken every 3 seconds, or $96 * 10^6$ clock ticks at a 32MHz frequency. We divide each average and standard

```
double TimerService::getTimerDrift() {
  return timerDrift +
    resourceManager->getTemperatureDrift();
}

double ResourceManager::getTemperatureDrift(){
  double t = temperature;
  auto x = temperatureModel.lower_bound(t);
  return normal(x->mean, x->stdDev);
}
```

Fig. 10. Tying the clock drift to the temperature model. lower_bound(t) is a C++ STL function that returns the first element in the set which does not compare less than t.

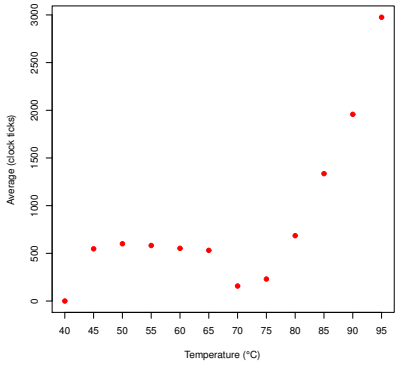

Fig. 11. Temperature vs. average offset for the simulation experiment.

deviation by the clock tick value to find the proportional drifts. We save these values in a plain text file, shown in Figure 9. At initialization time, we parse this file into a temperatureModel data structure, which is a set of <temperature, mean, standard deviation> tuples indexed by the temperature. The clock drift ϕ is then defined as the static clock drift defined by Castalia plus the value of a normal distribution with mean and standard deviation taken from the model entry corresponding to the current temperature. The resulting code is shown in Figure 10.

C. Simulation Results

We replicated the temperature variation experiment described in Section V on the simulator using the temperature model described in this section. We deploy two nodes 1m apart, running TSTP's clock synchronization every 3s. The simulations run for 20min (simulated time), each with the reference node at 40°C and the synchronizing node with varying temperature, from 40°C to 95°C (step 5°C). Temperatures are set at the start of each simulation, and stay constant for the entire run. We set Castalia's static clock drift to 0, so that nodes at a temperature less than or equal to 40°C are always perfectly synchronized. The results from the simulations are shown in Figures 11 and 12. We can see that the results are very similar to the results observed in real hardware, shown in Figures 4 and 5. Moreover, the baseline of both nodes at 40°C correctly shows 0 clock offset, which indicates that the implemented offset detection and compensation algorithm is correct. The differences that do exist between Figures 4, 5 and 11, 12 might arise from three main factors: 1) the dynamic clock drift in real hardware might not follow the implemented normal distribution; 2) the real experiments present outliers and packet losses due to environmental and implementation issues, which are not present in the simulator; 3) intrinsic variance.

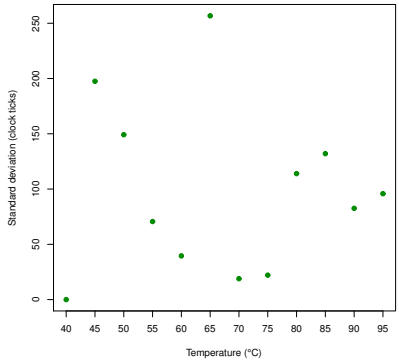

Fig. 12. Temperature vs. standard deviation of the offset for the simulation experiment.

VII. Conclusion

The protocol performed within expectations. We were able to show that TSTP functions within the specifications even in temperatures as high as 95°C. These results show that the protocol performs even under strenuous temperature conditions, which is important given that the eMote3 hardware is being used in a variety of cyber-physical systems. At an update period of 3 seconds, the goal of 40ppm is achieved when the offset is smaller than 3840, and the mean was around 2979. Higher temperatures could not only damage the hardware, but cause TSTP to no longer provide the expected time synchronization quality.

Figure 13 shows the temperature vs. frequency plot of various AT-cut crystals. Each plot is labeled according to the angle of the cut. It suggests that the difference in frequency between the crystal in each node will increase with the temperature, and so will the number of offsets above the goal of 3840.

What the temperature vs. standard deviation plot (Figure 5) suggests is that this is not a good measure of deviation for this problem, mostly because it is greatly affected by the seemingly random outliers, discussed in Section III. Further investigation on what causes these pairs of outliers is required. Since they appear in pairs, and usually result in an offset that is larger than 3848, a

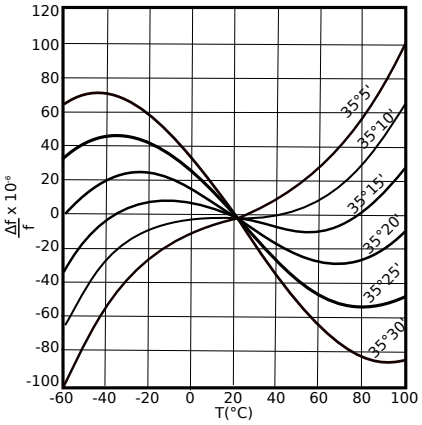

Fig. 13. Temperature vs. frequency plot of various AT-cut crystals. Figure based on [3].

good start is that their rate is approximately equal to half of the "Over 40ppm" rate for temperatures between 40°C and 60°C, and seem to spike at 65°C. Therefore, the user of the network should be made aware, and still decide if it is wise to choose to tolerate these deviations that happen less than 1% of the time. It is possible that one of the units is simply faulty, in which case performing these experiments again to test them before deployment is recommended.

Even with significant outliers in the measurements, we were able to use the data to build a simulation model that takes into account node temperature to determine clock drift. We replicate the experiments on a simulator and show that the clock synchronization algorithm shows the same tendencies as in the real experiments. A better characterization of averages and standard deviations could in principle be applied to the same model without any alteration.

When experiments were performed in different locations in the test environment, it became clear that, with this particular hardware, nodes should be kept reasonably close. A server running inside a metallic case between the two nodes did not seem to impact the quality of the synchronization, but distances of 6.8m and 7.4m did. When operating 7m away from its sink, the node was also receiving direct sunlight, and while the average of the offsets was good, perhaps due to the difference in temperature somewhat compensating for differences in the angles of the cuts on the crystal, the other statistics show that the deviation was high, and that the antennae were having difficulties communicating correctly. In a real scenario, this could lead to more power being drained, due to the fact that several attempts to make the packets reach the other side will need to be made, and the signal will need to be strengthened.

VIII. Future Work

There are two main issues with how the experiments were performed that can be dealt with in future experiments and in attempts to replicate these results.

The first one of them is the fact that many elements were not precisely verified. The amount of electronic devices running at once when the spatial experiments were performed was not constant, nor was the radio bands which they used. For future experiments, it would be advantageous to have similar nodes communication in the same channel and at an adjustable rate to make sure that the interference is known. Also, the experiments required supervision by a human, who had to watch the thermometer and dim the light accordingly. Automating the process, and perhaps adding multiple thermometers for a more precise measure, are improvements that can be taken into account.

Another issue is that we were trusting the timestamps themselves to measure the offset, but the real offset was never measured. With a function that allows an interruption to take place when the clock reaches a specified value, with microsecond precision, we can send a pulse to an I/O pin, and use an oscilloscope to verify the real offset. This provides an even better baseline to compare results.

For future improvements that can use this experiment, e.g., as a means to verify the quality of future changes to the protocol, it is important to note that TSTP is not temperature-compensated, but the hardware that it runs on has support for external thermometers, and the protocol headers allow the nodes to report their current temperature. This information can be used to introduce temperature-compensated time synchronization. Schmid's work [4] provides insight on how to design such a protocol. It is also worth noting that temperature compensated crystal oscillators (TCXO), despite being widely available commercially, are power hungry and costly alternatives [7].

Acknowledgements

This work was supported in part by CNPq and FAPESP.

References

[1] D. Resner, A. A. Fröhlich, L. F. Wanner, "Speculative Precision Time Protocol: submicrosecond clock synchronization for the IoT," in Intl. Conference on Emerging Technologies and Factory Automation (ETFA), 2016.

[2] D. Resner, A. A. Fröhlich, "TSTP MAC: A Foundation for the Trustful Space-Time Protocol", in Intl. Conference on Embedded and Ubiquitous Computing (EUC), 2016.

[3] W. Zhou, "Time, Frequency measurement and control technology", Xidian University Press, 2006.

[4] T. Chid, Z. Charbiwala, R. Shea, M. B. Srivastava, "Temperature Compensated Time Synchronization," IEEE Embedded Systems Letters, vol. 1, no. 2, August 2009.

[5] OpenSim. "OMNeT++ - Objective Modular Network Testbed in C++", 2017. Available: https://omnetpp.org

[6] A. Boulis. "Castalia A simulator for Wireless Sensor Networks and Body Area Networks," 2017. Available: https://github.com/boulis/Castalia

[7] T. Schmid, J. Friedman, Z. Charbiwala, Y. H. Cho, M. B. Srivastava, "Low-Power High-Accuracy Timing Systems for Efficient Duty Cycling," in Intl. Symposium on Low Power Electronics & Design, 2008.