Worldsensing Assignment

Scalambrin Luca

July 26, 2021

Introduction

In the present document you will find 7 different sections which cover the different points addressed by the assignment. The corresponding code of the implementation can be accessed by this link. In the referenced repository there are four files "`.c`" which correspond to the different entities involved in the simulation.

1 Synchronization protocol

It is a fact that synchronization is a fundamental key for the correct functioning of a network. However, in wireless networks, such as IoT networks, where devices can have reduced capabilities of processing, due to absence of power full hardware, synchronization could be one of the major problems.

In this case, the network to be considered consists of a chain of connected nodes. One of those has an atomic clock, which should be used to synchronize to the others nodes or devices. A protocol to achieve the synchronization objective should allow the exchange of packets between nodes, that make it possible to update or correct the internal clocks of the corresponding nodes, because all of them have an associated drift.

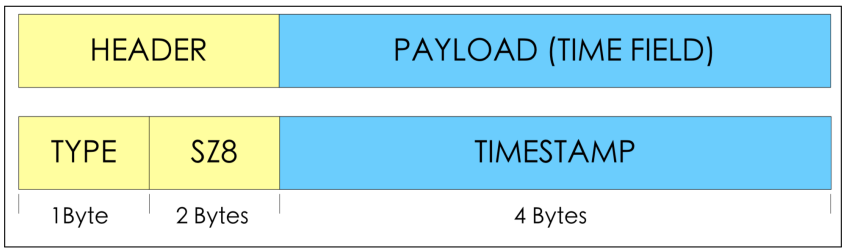

Figure 1: Packets corresponding to the protocol designed for achieving network synchronization.

The proposed protocol consists of an exchange of simple packets, which sometimes will contain a timestamp. Once the server or the time source gets a request from the closest node, it will send a first packet containing just the type "TYPE_SYNC_1", without payload. Then, the server will send a new packet which will be of type "TYPE_SYNC_2" and the payload will have the corresponding timestamp of that instant. On the other side, the node will have to register the time of arrival of the first packet, as well as the time of arrival of the second one and, jointly with the timestamp provided in the payload, the node can update its internal clock by

$$T_{clk} = t_0 + (t_2 - t_1), \tag{1}$$

where t_0 is the timestamp from the reference clock, t_1 is the time at which the node got the SYNC_1 packet and t_0 the time for the SYNC_1 packet.

It is important to mention that this kind of approach has its limitations, but a more complex idea could be more difficult to test and, overall, to implement. However, as a primary approximation this method should work reasonably well.

Figure 1 shows packets used for the protocol. The packets are really simple, they contain a header of 3 bytes and a payload of 4 bytes, being a total of 7 bytes. In this first version of this protocol, there are three different types of messages. The message could be a TYPE_SYN_REQ , TYPE_SYN_1 or a TYPE_SYN_2. Request signals are sent from the node which is connected directly to the atomic clock, requesting a message of synchronization, while response messages are sent from the server to its child, and then this child should propagate the new time to the rest of the network. Figure 3 shows a summary of the sequence mentioned. When the child sends a request, the payload of the message is empty. Then, the server containing the atomic

clock will send the two packets of synchronization, correspondingly. As it was mentioned before, the second of those will contain a timestamp in the payload.

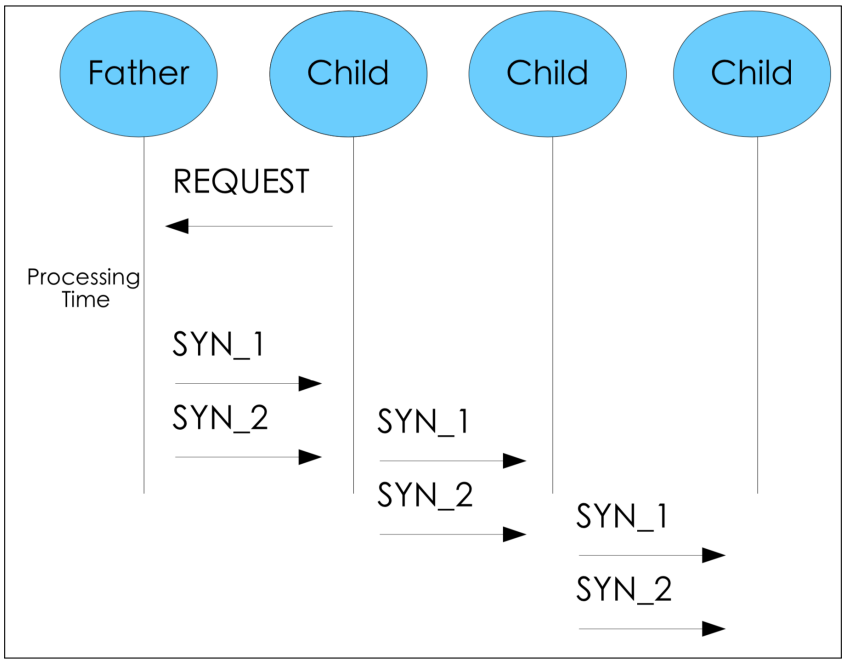

Figure 2: Sequence of messages sent from the closest node from the father or time source to the rest of the network.

Once the child node gets the response from the father, it will be able to recalculate the time for its internal clock as it was stated with eq. (1). The new clock value will be "locked" in a more precise way to the clock of the server. However, this node is responsable for doing the same task, this is, sending the two "SYNC" packets to its child, and therefore the clock corrections for the nodes will propagate through the wireless network. It is important to mention that each time the nodes send the TYPE_SYN_2 packet to their children, the timestamp to be loaded should be of the corresponding node, not from the original atomic clock.

The whole process of the protocol, since a child establishes a connection to the network until request messages are sent from the first node, is shown in Figure 3. The implementation of this protocol was over TCP, therefore the child needs to establish a connection with the father, which is achieved through the use of sockets on each side, father and child. Since packets are of a reduced size, this could be done over UDP as well, but just as a first approximation the implementation tested was over TCP.

Once they are connected, the main server should listen, periodically, to synchronization requests, and so the child which is next to it will send a request, allowing it to synchronize its internal clock. In this implementation, the child must propagate the new time to the rest of the network, but in a future implementation, there could be added a dedicated time for listening in all the network nodes. A simple linear model was used to represent the drift of the clocks, which is not the most appropriate idea. This process is repeated all the time, since drift does not stop in real life.

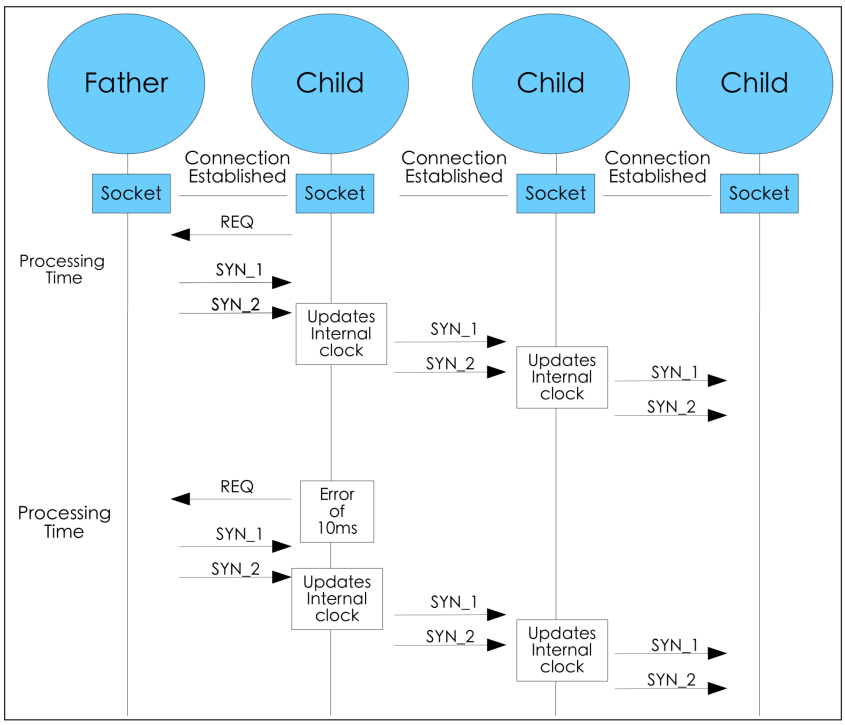

Figure 3: Exchange of messages between the nodes of the network and the time source. The request is generated by the first node in the chain.

It is important to mention that more complex synchronization systems could be designed, but the implementation could also be more complex, and sometimes it requires more processing capabilities in the nodes.

The code provided has an extra type of messages, which was not mentioned. This type is "TYPE_SYN_CLOCK", and was not mentioned because it is a type of message used for the synchronization of the simulated internal clocks of the nodes, not for the protocol itself. It is impossible to run five programs(one for each node) at the same time, manually. Therefore, with this signal, the clocks from all the nodes will start at times relatively closed.

2 Clock granularity and drift

Considering crystals with a frequency of 32768 [Hz] to source clocks of the devices implies a period of 30.5 [us]. If the crystals have a "drift" of 100ppm, then we can be sure that the clock will generate a signal or "tick" each (30.5 ± 0.000305) [us], this is, the clock does not work perfectly. It has an error of 0.001. This being said, if we define the granularity as the time error present in every period, then granularity is 305 [ps] = 0.305 [ns].

If we consider a set of two devices and we want to ensure their clocks are aligned within 1ms, then the maximum difference between their clocks should be 1ms. We need to find a number N that satisfies $N * G = 1$[ms], where G stands for granularity (if we consider the worst case, where drifts have the same sign, this is, always negative or always positive). Now, we need to synchronize the devices each $N * T$[s] $= N/f$[s] $= 100[s]$, where T stands for the period of the crystal or f for its frequency of oscillation.

3 Implementation of the protocol

The programming language used for implementing the protocol was C, and the code can be found here.

For this implementation, each node of the network consists of a program that simulates the device. The fundamental part of the simulation is the clock running inside each device, which was simulated by a thread. Therefore, each program has two threads, one dedicated for the internal clock of each device and the other dedicated to do the other tasks. For the correct functioning of these simulated clocks, the use of signals was fundamental, in particular signals associated with condition variables.

To simulate a real network, each program must connect to the predecessor and to the next node. There is a principal server which has an atomic clock without drift, and who only has a child (no father). Then, there are 3 nodes which have a father and a child. At the end of the chain, there is an end node, which does not have a child. Due to the topology to be represented, the implementation consists of four different programs, one for

the server or atomic clock, one for the first node, which will generate the requests, one for the internal nodes of the net, and one for the end node.

To generate the internal clock on each of the devices, the syscall "`nanosleep()`" was used. This syscall allows you to pause or sleep the calling thread. In the meantime, the main thread of the program will listen for synchronization packets from the father. The node which is directly connected to the time source will generate the requests for the time source, and then will propagate the synchronization signals. An important assumption here is that the different nodes have the same drift, therefore, if all of them are initiated at the same time, there will be a unique synchronization request that needs to be issued each 60 seconds. However, there are functions that were not developed in the code, at least in this first version, but some of those would allow generating requests from every node in the network.

Due to the difficulty of generating a simulated clock by use of syscalls, the period of the clock was much greater than 30.5 [us]. This problem is due to the implementation, because it is not easy to ensure that a set of code lines takes a defined time(such as "`nanosleep()`"), therefore the resolution used was lesser than that from the real crystals. This allows that variations on the time assigned by the operative system do not affect in a significant way. Testing the implementation could be not a trivial task, because we need to compare the internal simulated clocks from the different nodes, with that of the atomic clock. The main issue here is that we need to have a reference time to measure those differences, and the implementation was not a unique program. At this point, the function that allows you to get a time from the CPU of the laptop which runs the code is called "`gettimeoftheday()`". The return value from this function could be in microseconds, which was very useful for the comparative between the different times of the node clocks.

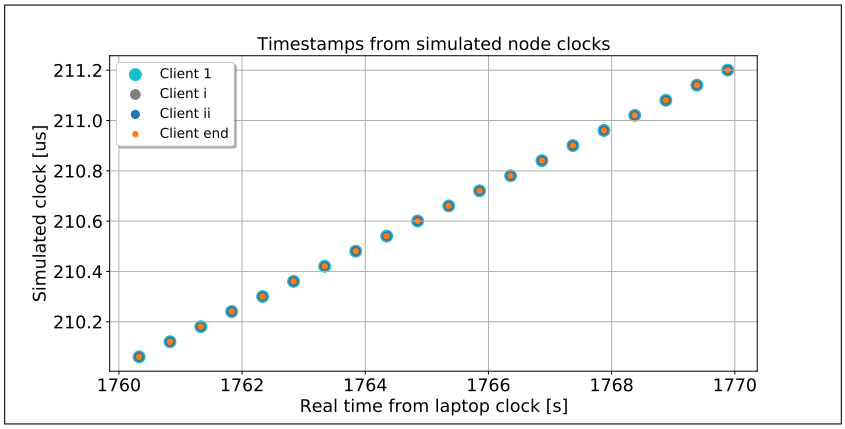

Figure 4: Clocks generated as a function of the real time from the CPU used. The four different clocks correspond to each of the nodes that build the chain topology.

To test the accuracy of the clocks generated by the threads and signals employed, a graph of the different clocks could be very useful. Figure 4 shows the different clocks which run inside each of the nodes of the network. The simulated clock as a function of the real time from the processor of the laptop where simulations were executed exhibits that the clocks were generated with a reasonable accuracy.

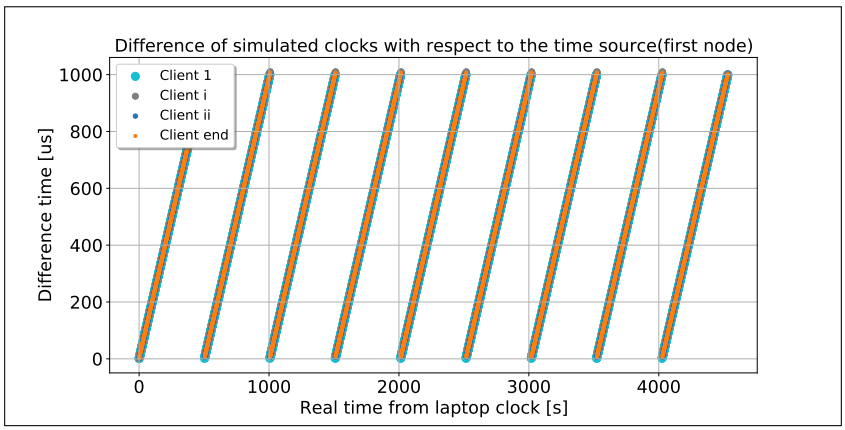

Figure 5: Difference between internal clocks from the nodes and the atomic clock, as a function of time. It is important to note that there is an important overlap in the difference time between the different clocks and the time source.

Figure 5 shows the discrepancy in time between the clock of the different nodes and the atomic clock or time source of the network, where a drift rate of 1ms every 60 seconds was considered. For this simulation, every second from real time corresponds with 0.12 seconds of the simulation, therefore every 1000 seconds from the x axis correspond with 120 seconds from the simulation. Consequently, this graph shows that effectively there is a drift rate of 1000 us or 1 ms every 60 seconds of simulation. Then, it could be concluded that, at least for the time simulated, the protocol met all the requirements and it seems to work in the correct way.

Then, it could be concluded that, at least for the time simulated, the protocol met all the requirements and it seems to work in the correct way.

This does not mean that the protocol and the implementation are perfect, but in this first approximation, the results were fairly good. However, more tests should be done to ensure the correct functioning of the protocol for an extended period of time.

4 Test plan

Testing a protocol for achieving synchronization over a network is not easy, at all. Prototyping is a widely used method, which may provide excellent evaluation for the design. However, it requires a lot of time to be built and, sometimes, the costs could be really high. This is the reason why simulations and models, jointly, are a better way for approaching tests. In the implementation provided, the model used for the drift was the simplest possible, which does not consider any kind of variability and takes for granted that the drift rate is a deterministic value. For a more realistic consideration, the model considered for the drift rate should incorporate other variables, because there are a lot of parameters that could influence the crystals, such as temperature of the environment.

In addition, there are extra points that should be considered for a better evaluation of the protocol. In particular, all the parameters associated with the process of transmission were considered as deterministic, which in real life could differ in an important way. Some points to be considered could be, for example, the time used to build the message and submit the request to the MAC layer and its delay, the time spent in transmitting the message at the physical layer, the time of propagation, which depends directly on the distance between the nodes, the time that it takes to process the packet at the application layer that will depend on the processing capabilities of the nodes, and so on.

5 Scalability of the system

Scalability is an important feature that should be provided by the simulation, because it is always necessary to take the protocol to its limits. In this implementation, it could be possible to test a system with more than 3 internal nodes between the server with the atomic clock and the end node. However, if the new topology to analyze was something with $N >> 1$ nodes and $M >> 1$ time sources, then a new design would be needed.

Having more than an atomic clock in the system will be a key to synchronizing a topology with many nodes. If the topology has an atomic clock each L nodes, being $L << N$ and $L << M$, then each child will have an atomic clock on each side of the chain connection. This implies that the nodes could send requests to both sides, and therefore, the node should know which atomic clock is the nearest. For this new implementation, packets from fathers must have a new header field which provides a measurement of the closeness to its atomic

clock of reference, consequently the child should keep sending requests to that father which is the closest to its time source.

The existence of many nodes will imply an increase in demand for the medium to transmit, which is always an important issue in wireless networks. This means that more collisions will take place and a further analysis of packet size vs collision will be necessary, because in these scenarios with Dense Low Power WANs could be useful to execute an aggressive fragmentation to enhance the performance of the system. Nevertheless, this approach will work for networks with great size, because for small networks it is not useful to use packet fragmentation, and, in the meantime there is a trade off that needs to be analyzed, between goodput and energy efficiency.

6 Mesh Topology

In a mesh topology where nodes have many fathers it is completely necessary to identify the father which has the best clock accuracy in comparison with the atomic clock of the network. To achieve this, a similar approach as the one mentioned in the previous section could be implemented, this is, incorporating a new header field which allows to identify the father whose proximity from an atomic clock is lesser.

However, if the network is really dense, then a lot of packets could be lost. If the second signal, from the proposed protocol, of synchronization is lost, the father will not know it. For this reason, a possible approach requires nodes to register all the packets that are available, jointly with the corresponding timestamps. Here, a new header is needed, which needs to carry an identification of the father or server nodes. With these elements, the nodes will be allowed to update its internal clock once they get all the corresponding information from a specific father.

As stated in the previous section, in this case the aggressive fragmentation technique could have a more relevant impact over the performance of the system, because this topology implies a denser network, which generates a more collisioned mediums.

Another important problem originated by this topology when considering a server time, or node with an atomic clock, is that assuming a symmetrical network delay does not work correctly as a result of the great number of competing nodes, if we consider CSMA/CA as an access method. In addition, if there is only one time source, implementing protocols such as NTP will generate a traffic hot spot or "bottleneck" over the gateway that is used to communicate with the NTP server.

7 Possible Applications

The number of applications that require time synchronization and could be built on top of the implementation is not low.

A possible application will be for the measurement of the temperature and pressure in places where connectivity is an important issue, like tunnels that are found in the underground. At one end of these tunnels, the atomic clock will take place, because at that point there may be availability for connection to the internet or another source of information. Then, this server will be used by the rest of the nodes to synchronize the system, being the set of nodes inside the construction. Another possible example of this type of structure could be in mines. In addition, not only pressure and temperature could be important parameters, but others could be really needed on critical infrastructure.

An important aspect to be aware of is that the access time and transmission time and other factors that do not remain constant should be estimated by the devices. Therefore, extra code should be implemented in the node, which will be complementary for the protocol. A possible approach would be to consider the PDF (probability density function) of some of the delays under analysis, and adjust its values following some model.

Rails and tunnels are widely spread over the world and so is the necessity for controlling and monitoring them. However, to generate a functional product there are a series of steps that need to be done. First of all, the testing part for the synchronization protocol should be taken with more caution. Then, to offer a better product, it is important to identify other similar products on the market and work really hard specifically on those aspects that are not contemplated, because that difference could have a huge impact.

Reproduction of results

All the implementation was designed and run on a MACBook pro 2015 model A1502. The platform used for coding in C was VS Code and Spyder was used for the scripting to obtain the plots. All those files and more details about the implementation could be found on this repository.

It is important to mention that the operative system used for the coding and processing process was MAC OS, it was not Linux. Therefore, the code may need some adjustment for working on a Linux environment. Hopefully those changes will not be much.

.