Time Synchronization Project Report

Juanxi Li, Wei Da

Algorithm

In our algorithm, each node keeps a *counter*, which is not affected by any external event. It also keeps a variable called *offset* to calibrate the counter to reflect the *local time*, i.e. *local time* = *offset* + *counter*. The initial value of *offset* is 0. The buzzer chirps when *local time* modulo *chirp cycle* is within a small constant called *epsilon*.

The local node broadcasts a time synchronization request when *local time* modulo *request cycle* is within *epsilon*. As shown in Figure 1, when a node receives a request, it will immediately respond with three integers: the time when the request was sent (*t1*), the time when the request was received (*t2*), and the time when the response was sent (*t3*).

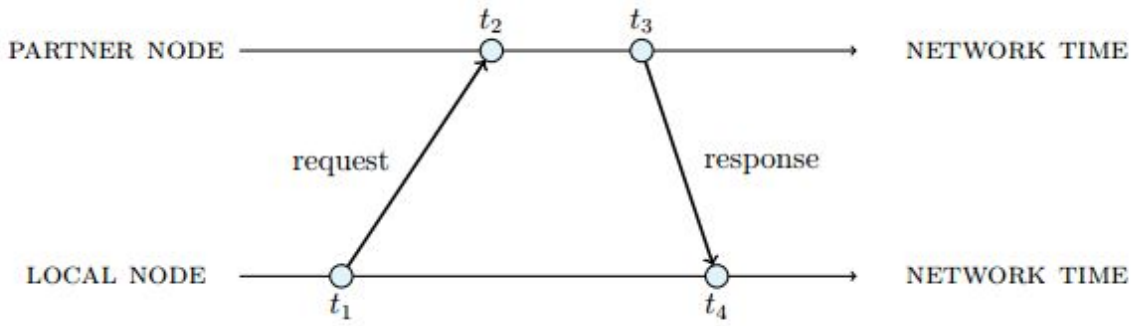

Figure 1

The offset between the local node and a partner node *p* is *offset_p* = (*t3* - *t4* + *t2* - *t1*) / 2. Let *offest_sum* be the sum of all these *offset_p*'s, let *n* be the number of nodes the local node has sent a request to. We then update *offset* using the following equation.

$$offset \mathrel{+}= offest\_sum / (n + 1)$$

It is worth noting that *offest_sum* / (*n* + 1) represents the difference between the local node's *local time* and the time it should be to satisfy time synchronization. Hence, we need to add *offest_sum* / (*n* + 1) to *local time*, i.e., to add it to *offset*. The reason we choose to divide *offest_sum* by *n* + 1 is that we hope the nodes to gradually converge to a common time instead of passing each node's time to another.

Testing

Since the sound of the buzzers is distracting, we disconnected the buzzer and looked at the log instead during testing. We assume that a buzzer is on immediately after a log is printed onto the screen. In this way, if the logs of different motes are printed at the same time, we will know that the buzzers are on simultaneously.

We tested three scenarios.

First, there is only one mote working. In this case, the mote will not adjust its time and will keep beeping periodically until another mote joins the network.

Second, one mote starts and then another joins in. In the beginning, the two buzzers cannot beep simultaneously but after the first round of synchronization, both beeps at the same time.

Third, we added one more mote to the second scenario. It would also be synchronized with others after a short period of time.

Last but not least, we reset any one of the tested nodes to see whether it would break our synchronization.

The final result shows that our algorithm can achieve synchronization among three nodes and be robust for the testing scenarios.

Also, we calculated the difference between synchronized time and real local time. Eventually, it would converge to a stable value which means the node has been synchronized.

To run our App, just build and deploy our project solution on eMotes. Testing with .net Microframework deployment tools and reset each eMote at different time. Please note to reboot with R68B.

Assumptions

Our algorithm requires the system to be synchronous with an upper bound on message delay (in our implementation, we assume it to be 100 milliseconds). Besides, we allow packages dropping and change of node status (died or wake up) during the execution of our program.

We also assume that the delay on a channel between two nodes is symmetrical so that the *offset* is accurate.
One problem we have is that two of our buzzers has low but audible volume.

Future work

We can decrease the rate of request after a node is on for a while since it has become more stable and should be considered to be more accurate. Whereas, a newly joined node should send requests more often to synchronize with other nodes faster. In addition, there may be better ways to estimate the change of *offset* after receiving responses from other nodes. For example, we can assign more weights to nodes that are more reliable.