Rate-based Synchronous Diffusion Algorithm Sensor Networks

Danilo Burbano
University of Neuchâtel
danilo.burbano@unine.ch

José Luis Carrera
University of Neuchâtel
jose.carrera@unine.ch

Didier Aeberhard
University of Freibourg
dieder.aeberhard@unifr.ch

Thomas Rouvinez
University of Freibourg
thomas.rouvinez@unifri.ch

***Abstract*—Sensor networks applications often require global ime synchronization between numerous sensors. In this paper we implemented the Rate-Based Synchronous Diffusion Algorithm as described in [RE1] on TelosB Motes with Contiki. We report the development procedure and the results obtained on the TARWIS testbed from the University of Bern, Switzerland.**

***Index Terms*— time synchronization, mobile reference node, TARWIS tesbed, Rate-Based Synchronous Diffusion, synchronization initiator.**

I. INTRODUCTION

GLOBAL synchronization is a crucial issue to deal with in sensor network applications. This work relies on the Rate-Based Synchronous Diffusion method toachieve global synchronization within a sensor network. The implementation ofthe method in this work was divided in two general phases:

- Neighbours Discovery.
- Convergence Phase.

The Main goal of this project is to achieve a global synchronization between the 40 nodes in a sensor network. The algorithm solution was developed with Contiki operating system and its performance was tested using TARWIS testbed management architecture.

II. NEIGHBOURS DISCOVERY

Defining the Neighbour Tables is important and one of the first steps in the initialization of the sensor network. In this work a deterministic neighbor discovery method is used.

A. Discovery

In order to acquire the Neighbour Table, each node transmits broadcast messages according to a schedule defined by the user. This broadcast contains the id of the node and its clock ticks when is send. When a broadcast packet is received, the entry is checked to find out if it is already present in the Neighbour Table or not. If not, the offset between both nodes is computed and added to the Table with the node id. The Ratio Signal Strength Intensity (RRSI) is also computed and added to the Table. The user can schedule the broadcast with 3 different parameters:

(1) the time the node waits after started to begin broadcasting,
(2) the time the node waits after sending the broadcast, and
(3) the number of times the broadcast is resend

Note that the time the node waits after sending a broadcast will be tuned in Section 5 and called broadcast interval.

III. CONVERGENCE PHASE (CLOCK SYNCHRONIZATION)

We are using an internal synchronization mechanism. This means that each node knows how the clock of its neighbours works and then translates from one clock time to another. In order to reach such behavior on each node, a fully localized diffusion based method is implemented in which each node exchanges and updates information locally with its neighbours.

A. Principle

The Rate-Based Synchronous Diffusion is a fully distributed and localized method to synchronize locally the nodes without a global synchronization initiator. This diffusion method achieves global synchronization by spreading the local synchronization information to the whole system and then each node in the overall network agrees to change its clock readings to a consensus value. In the end the times of each node will converge to a global common time.

B. Unicast

Each node will iterate over all its neighbours and will determine the offset between the clocks using unicast messages with a modified Round-Trip (RTT) Synchronization schemes. The original scheme uses four different clock times when the message is sent form the neighbor node and when it is received from the node we want to synchronize the time. So that the offset of those clock times is more accurate. Then the node will adapt its time by a factor r to the neighbours node time. Finally after a non-deterministic number of rounds of diffusion the clock in each sensor will have the same value. Note that the factor r will be tuned in the Section 5 and called r-value.

IV. METHODS

The functions for sending/receiving unicast messages are described in this section as it is a modified version of the RTT synchronization. A unicast message contains four parameters:

(1) a boolean value to know if the message is completeor not,
(2) a clock time value, and
(3) a node id.

The function **send_uc** prepares a message to be sent to a node from the Neighbour Table. The message contains the id of the node (so that the other node knows who the sender is) and the boolean value false to know the message is not complete.

```
static void send_uc(){
 //Write the id of the current node
 tmRoot.originator = node_id;
 tmRoot.completed = false;
 //Prepare the unicast packet to be sent.
 packetbuf_copyfrom(&tmRoot, sizeof(tmRoot));
 //Send
 unicast_send(&uc, &addr);
 }
}
```

The function **recv_uc** is separated in two parts. The first part concerns the messages that are completed and the second otherwise.

If the message is complete, the new time of the node is directly computed using the following formula:

*newTime = clocktime()*r_(clocktime()-tmSlave.time)*

And the new clock time is set. Using this formula is slightly different from the RTT sync one. In fact, here we only take the two times of the different node to do the offset.

Otherwise, the message is filled up with the id and clock time of the node, set completed and sent back to the originator node.

```
static void recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{
 //Read the message
 packetbuf_copyto(&tmSlave);
 //If the hardware is not waiting for a message, update the time
 if(tmSlave.completed){
  newTime = clock_time() - r*(clock_time() - tmSlave.time);
  clock_set(newTime);
 }else{
  //Directly get the time when a runicast message is received
  tmSlave.completed = true;
  tmSlave.originator = node_id;
  tmSlave.time = clock_time();
  //...then copy the message and send it back...
  packetbuf_copyfrom(&tmSlave, sizeof(tmSlave));
  unicast_send(&uc, from);
 }
}
```

V. EXPERIMENTS

The first part of the experiments was done using the Telos nodes. Two parameters were tuned:
(1) the *r-value* of the syncing algorithm, and
(2) the broadcast interval

The *r-value* was set to 0.5 and will be tuned afterwards. The broadcast interval was set to at least 5 seconds in order to let the nodes discover each other. The second part consisted in testing our algorithm on the TARWIS platform.

A. Broadcast interval and r-value tuning

The tuning of the broadcast interval is crucial to avoid congestion and packets loss while the r-value tuning will increase the syncing convergence.

Three values were chosen for the broadcast interval: 5, 10 and 15 seconds. Using an interval of 5 seconds was not enough as the nodes were able to sync only for a short period of time before receiving wrong values. With a 10 seconds interval, the results were far better and the node kept on synchronizing during the whole period. With the 15 seconds interval, the results were worst than with 10 seconds.

In fact at the end of the period, the nodes were not synchronized as nicely as before.

For those reasons, an interval of 10 seconds was kept for the rest of the tests.

As the r-value has to be between 0 and 1, the value 0.25 and 0.5 were chosen.

In fact as the synchronization is made between 2 nodes, values higher than 0.5 mirror the value lower than 0.5. The value 0.5 gave a better convergence between all the nodes after the test period.

Figure 1 shows the results with a r-value of 0.5 and an interval of 10 seconds.

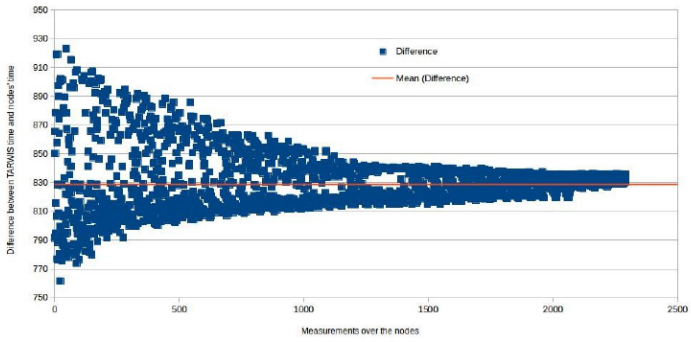


Fig.1. Graph of the difference of time between a node and the TARWIS time through time with a r-value of 0.5 and a broadcast intervals of 10 seconds.

B. General tuning

The r-value and the broadcast interval are not the only tunable parameters.

Firstly, a different protocol (X-MAC) for sending/receiving packets was tested.

Secondly, as we modified the RTT synchronization algorithm, we also tested the original one.

The X-MAC protocol was used with the same parameters as before. As expected, the nodes take more time to converge. This is due to the fact that the nodes are not constantly checking for an incoming message.

The genuine RTT synchronization is also tested with the same parameters. As the time is measured four times and the offset is calculated differently, the results show that the synchronization is converging slower but more constantly.

Figures 2 and 3 show the result graph of those two tuning.

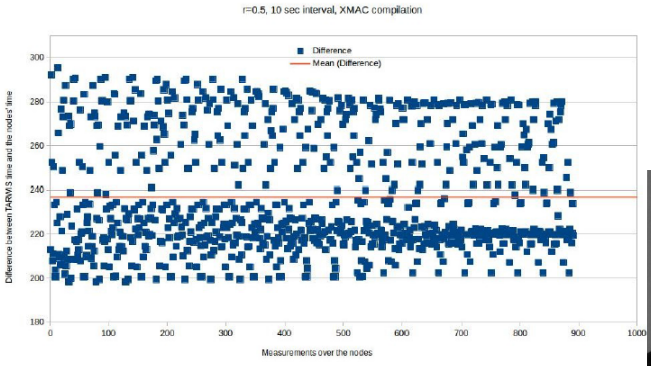
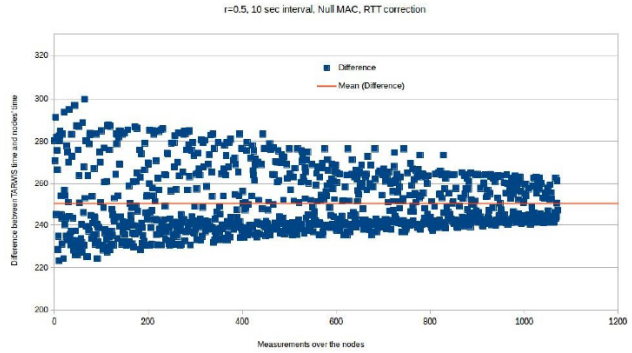


Fig.2. Graph of the results using X-MAC as compilation parameter protocol.

VI. IMPROVEMENTS

Even if the synchronization works well, there is still room for improvements.

Firstly, we were unable to let a node synchronize its clock further than 2^{16} clock ticks. In fact, it seems that a cast is wrongly implemented in our code but we were unable to find it. Secondly, with more time we could have made more tests between our modified RTT and the genuine one in order to see more meaningful differences.



Fig

VII. CONCLUSION

Isolating the optimal interval for the neighbour discovery phase in real world scenarios is a challenge. Indeed not only we need to consider energy efficient mechanisms but also intelligent algorithms for establishing and adapting appropriate broadcasting intervals. The reason behind it lies in impacting as little as possible the synchronization phase.

Determining neighbour relationship between two nodes is a challenging process too. The environment conditions of the network can suddenly change during the working time, influencing parameters like RSSI or round trip time which are used to define neighbour tables.

REFERENCES

[1] Li Qun and Daniela Rus, Global clock synchronization in sensor networks, IEEE Transactions on Computers, Vol. 2, 2006, pp. 214-226.

Danilo Burbano Acuña received his B.S. degree in systems engineering from National Polytechnic School, Quito, Ecuador, in 2006. From 2007 to 2013 he was a system analyst and software developer in three different enterprises. He is currently pursuing the M.S. degree in computer science at the University of Bern, Neuchâtel and Freibourg in

hor of a paper about Green Computing and coauthor of a paper about the Internet of Things and Urban Innovation. His research interest includes the development and deployment of wireless sensor networks, and applications of machine learning.

José Luis Carrera V was born in Quito, Ecuador. He received the B.S. in systems engineering from National Polytechnic School of Ecuador in 2005 and his M.S. in Communication and Technologies of Information Management degree from National Polytechnic School in 2012.

From 2005 to 2011, he was a geo-

tem analyst in National Geographic Institute of Ecuador where he worked in different projects with national scope. From 2011 to 2013, he was Professor in the Computer Sciences Engineering Department in National Polytechnic School of Ecuador.

Currently, he is pursuing another M.S. degree in Computer Science and a Specialization in Distributed Systems in a joint master program at the University of Bern, Neuchâtel and Freibourg in Switzerland.

His research interest areas include distributed systems with wireless sensor networks, mobile communications and pattern recognition and machine learning for human activity.

Thomas Rouvinez received his B.S degree in computer sciences from the University of Freibourg, Switzerland. He is currently pursuing the M.S degree in computer science at the University of Bern, Neuchâtel and Freibourg in Switzerland. His research

nced networking.

Didier Aeberhard, received the B.S degree in computer from the University of Freibourg, Switzerland. Currently he is pursuing a M.S. degree in Computer Science and a Specialization in Distributed Systems in a joint master program at the University of Bern, Neuchâtel and Freibourg in Switzerland.
His research interest areas include advanced networking and machine learning.