Synchronization Service Integrated into Routing Layer in Wireless Sensor Networks

Szymon Fedor and Martin Collier
Research Institute for Networks and Communications Engineering (RINCE),
Dublin City University, Dublin 9, Ireland,
Email: {fedor.szymon, collierm}@eeng.dcu.ie

***Abstract*— The time synchronization problem needs to be considered in a distributed system. In Wireless Sensor Networks (WSNs) this issue must be solved with limited computational, communication and energy resources. Many synchronization protocols exist for WSNs. However, in most cases these protocols are independent entities with specific packets, communication scheme and network hierarchy. This solution is not energy efficient. Because it is very rare for synchronization not to be necessary in WSNs, we advocate integrating the synchronization service into the routing layer. We have implemented this approach in a new synchronization protocol called Routing Integrated Synchronization Service (RISS). Our tests show that RISS is very time and energy efficient and also is characterized by a small overhead. We have compared its performance experimentally to that of the FTSP synchronization protocol and it has proved to offer better time precision than the latter protocol.**

I. Introduction

Wireless Sensor Networks (WSNs) have recently drawn significant research attention because of their range of anticipated applications. The unique nature of WSNs such as limited energy and computational resources, application specificity and limited link capacity pose challenges in the network architecture design. Traditionally, the network stack is divided into hierarchical and independent layers, based on the OSI model of computer networks. Higher layer protocols only make use of the services at the lower layer and are not concerned with the details of how those services are provided. Communication between nonadjacent layers is not allowed, whereas adjacent layers communicate through static interfaces, independent of the individual network constraints and applications. This is in compliance with the principle of modularity in system design.

This layered independent architecture may be unduly restrictive in the WSN context. Indeed, the autonomous operation of the network stack layers can lead to a significant drop of communication performance. For example it has been shown that the LEACH routing scheme may cause a failure of the synchronization protocol when it puts the node into low power mode [3]. Two approaches to improving WSN efficiency by departing from a layered protocol have been considered in the literature. The first approach involves designing a monolithic system for a specific application (*e.g.* the work of Mainwaring *et al.* [6]). Such solutions are generally incompatible with the work of others and so greatly reduce the synergy possible between research efforts [2]. To overcome this problem, researchers have proposed a layered solution with a cross-layer information sharing. In those scenarios, the advantage of granularity of the network stack is still retained. However, some information gathered by one layer can be shared with other layers. This data may include the RSSI value, neighbors table, network time *etc*. Examples of such solutions are [3], [8].

Time information is required by many protocols and applications in WSN. There is thus a need to estimate it efficiently and to share it across the network stack. There are many protocols in the literature trying to solve this problem [5], [7]. In this paper we advocate the cross-layer solution for network synchronization. We also describe how this might be achieved with a minimum communication and processing cost. The rest of the paper is organized as follows. In the next section we argue why time synchronization is needed for WSN and why it is difficult to achieve in those systems. Then we provide a detailed analysis of Routing Integrated Synchronization Service (RISS), our synchronization protocol, in section III. In section IV we present our empirical results of experiments with RISS. At the end, in section V we conclude the paper and suggest further development of RISS.

II. Time synchronization

The time synchronization problem is a standard problem in distributed systems, but especially in WSNs where a common time reference is necessary for duty cycling of nodes and where most applications require time stamping of the samples gathered by sensors. Clock synchronization algorithms face two main problems: time-stamping jitter caused by delays in transmitting a packet and time errors due to the operating differences of hardware. The source of message time-stamping inaccuracy was decomposed and analyzed by Maroti *et al.* in [7]. It includes sending, channel access, transmission, propagation, reception and receive time uncertainty. These errors are random and difficult to predict and eliminate, especially in a network with many communicating nodes. Because they are stochastic, we measure them every time the synchronization is performed and we incorporate the value into the protocol outcome. The latter source of error, clock drifts, may be compensate but it requires a periodic update of time information.

Factors relevant to the design of WSN synchronization protocols are not only achieving the highest time precision but also addressing limited energy and computation resources. We can classify the existing synchronization protocols as being either independent entity [7] or a service integrated into one of the layers of the network stack [8]. The latter solution is more applicable to WSNs, since the additional communication cost due to the transmission of separate synchronization messages is eliminated. Indeed, for those solutions in most cases the time information is deduced from the timestamp field inserted into

1525-3511/08/$25.00 ©2008 IEEE

regular communication packets and it is less energy demanding than sending additional beacon packets for synchronization purposes. This solution is mainly adapted in the former group of protocols. We can further reduce the energy dissipation due to the synchronization by minimizing the time information added to the packet. In the next section we describe in details how we can achieve the network synchronization.

III. RISS - NEW SYNCHRONIZATION PROTOCOL

Our aim is to provide each node of the WSN with a time reference for the minimum energy cost. The design goals are to:

- minimize the overhead of the protocol
- minimize the awake time of the transceiver
- achieve robust performance of the protocol across a range of values for the duty cycle of the nodes

To attain those goals the solution should explore the characteristics of the typical WSN application.

The main purpose of the WSN is to interact with the environment by sensing or controlling physical parameters [5]. In most application scenarios (*e.g.* monitoring of the environment, habitat, offices, systems, buildings, and industrial sites) nodes perform the computational and transmission tasks periodically. It is rarely the situation that the node waits for an event to happen but even in those cases it must periodically send a beacon message to maintain network connectivity. So because this periodic transmission is a common property of WSN applications we will exploit it to obtain an overall network time reference. The operating principles of RISS are described below.

A. Protocol overview

In general, data is forwarded regularly to the *downstream* nodes (in the direction towards the base station) from the *upstream* nodes (in the reverse direction). To minimize the protocol overhead we propose that a mote synchronizes to the *upstream* mote. It maintains a table with the clock information of every *upstream* neighbor. To construct that table, the *downstream* node uses the periodicity of the operation of the *upstream* neighbor and the time information added to the packet by the sender. So the task of the *upstream* node is to add information to the message that would facilitate the time alignment of nodes. However, we recommend adding not a time-stamp of the sending instant, but rather the time which has elapsed since the last local clock periodic interrupt. Then the receiver after collecting multiple messages can calculate the neighbor clock frequency with respect to its local clock.

The repetitive operation of the protocol overcomes the problem of hardware clock errors. The aim of the time field included in the packet is to minimize the synchronization error due to the time-stamp jitter. How this is done is described below.

B. Time-jitter error minimization

Time-jitter error has multiple origins. These include the time taken to assemble a packet and submit it to the MAC layer

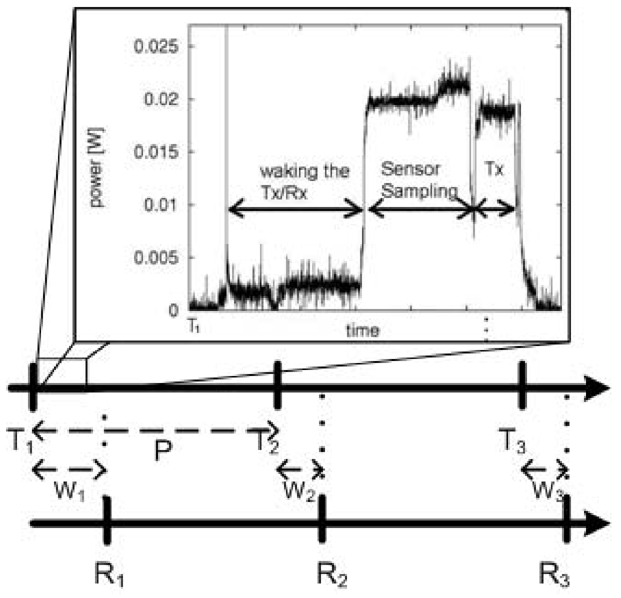

Fig. 1. Time line of the operation of the receiver (bottom) and transmitter (top). Every P seconds the sender wakes-up the transceiver at its local time T_i, samples the sensor, and transmits the SFD of the packet at local time $T_i + W_i$. Sender adds the value W_i to the message and turns off the transceiver. The receiver hears the SFD at the local time R_i.

which is unpredictable, and it depends on the CPU usage. Also the channel access time is random. To synchronize the clocks we propose to measure those random quantities (the values of W_i in figure 1) and send them in a packet to a receiver which will estimate the frequency of the sender clock. Sending just the information about random time delay (W_i) and not the counter value of the transmission ($T_i + W_i$) has two advantages. First of all, this number is of a small range in comparison to the possible clock scope. Thus, sending it requires less energy and packet space. In our experiments we observed that the maximum nondeterministic delay of the transmission was 566 ticks using a 32kHz clock. We need 10 bits to transmit this number instead of 32 bits for sending the time-stamp. This is important in WSN where most of the energy resources are consumed by the transceiver [4]. Also in WSN, the packet space is very limited. For example in Tmote's implementation of the network stack the user has 28 bytes for application data. So saving even 22 bits of the header might have substantial benefit. In the next section we describe how the synchronization is performed in our protocol.

C. Detailed RISS description

As mentioned previously the *downstream* node estimates the frequency of its *upstream* neighbors. So, for example the base station has to evaluate the clocks of the nodes which are reporting data directly to it. The task of the *upstream* motes is to perform some periodic operation (*e.g.* awake the transceiver) at time T_i and send a packet (figure 1). The frequency of the cyclical task ($1/P$) is known to the *downstream* receiver and it may be decided before the deployment of the network or changed operationally and reported to all nodes. In this case the value travels in the *upstream* direction so before reaching a node it is learned by the *downstream* neighbor.

It takes the time W_i for the node to actually transmit the packet. This waiting time is inserted into the message just after the transmission of the SFD. When the receiver captures the SFD of the packet it samples the local clock and saves the value (R_i) for the further evaluation of the senders clock frequency. A *downstream* node needs to collect several packets from a single neighbor in order to synchronize with its clock. In the next paragraph we describe how the receiver synchronizes with its neighbors.

D. Receiver operation

The *downstream* node records the information about the last Q packets from a mote in order to estimate its clock frequency. This data is stored in a $Q \times l$ array where l is the number of *upstream* neighbors. For every packet received, the node must save the corresponding W_i and R_i values. After that, the mote estimates the clock frequency of the most recent transmitter from the set of W_i and R_i values stored in the table using the method below. This calculation is performed every time a packet is received. We have developed a new technique of sender's clock frequency estimation because the linear regression method generally used for such purposes is time demanding (see section IV-A) due to the division of large integers.

We call the frequency of the transmitter clock expressed at the receiver $F_{r\to t}$. For every packet sent with an inter-packet period equal to P we can write:

$$R_i - R_{i-1} = F_{r\to t} \times (P - W_{i-1} + W_i) \tag{1}$$

or

$$F_{r\to t} = \frac{R_i - R_{i-1}}{P + (W_i - W_{i-1})} \tag{2}$$

We want to avoid the division operation in equation 2 for computational efficiency. Firstly, the approximation

$$F_{r\to t} \simeq \frac{(R_i - R_{i-1}) - k \times (W_i - W_{i-1})}{P} \tag{3}$$

is exact when $k = \frac{R_i - R_{i-1}}{P + W_i - W_{i-1}}$. Since $R_i - R_{i-1} \simeq P$ (*i.e.* the interval between successive received packets is approximately equal to the period of the cyclical task) and $W_i - W_{i-1} \ll P$, it follows that $k \simeq 1$. Hence we write

$$F_{r\to t} \simeq \frac{(R_i - R_{i-1}) - (W_i - W_{i-1})}{P} \tag{4}$$

Secondly, we calculate only the numerator of that expression. Defining $F'_{r\to t} = P \times F_{r\to t}$ and averaging that value over the Q most recently received packets we obtain:

$$\overline{F'_{r\to t}} = \frac{\sum_{i=c}^{c-Q+1} [R_i - R_{i-1} - W_i + W_{i-1}]}{Q} \tag{5}$$

The obtained value of the neighbor clock frequency ($\overline{F'_{r\to t}}$) can be used for various purposes. In WSN there are two main applications: duty cycling of nodes and estimation of a common time reference. We show in the next section how this can be done.

E. Duty cycling of the node

Duty cycling is one of the most common techniques used in WSN to save energy resources of the motes. It is necessary that the communicating motes have the same time reference and wake up at the same instance to communicate. We can use the frequency of sender's clock estimated with the RISS protocol and synchronize the duty cycling of the nodes in the following way.

When the *downstream* node captured a sufficient number of packets from its *upstream* neighbor to estimate its clock frequency, it calculates the arriving time of the next packet and goes to sleep. After reception of packet i, we can expect that the arrival timestamp of packet $i+1$ is:

$$R_{i+1} = R_i + P \times F_{r\to t} + F_{r\to t} \times (W_{i+1} - W_i) \tag{6}$$

However, we cannot predict the term W_{i+1} which is stochastic. Thus we propose to replace the difference $(W_{i+1} - W_i)$ by a minimum of that value over the last Q packets $\min_{a\epsilon(0:Q-1)}(W_{i-a} - W_{i-a-1})$. We do that to minimize the packet loss due to the transceiver waking up after the packets' arrival. In order to reduce the error of the neighbor clock frequency calculation we propose to combine the results of multiple estimates. Also, the term $\min_{a\epsilon(0:Q-1)}(W_{i-a} - W_{i-1-a})$ corresponds to a small value so we can neglect its multiplication by $F_{r\to t}$ which is very close to one. So equation 6 becomes:

$$R_{i+1} = R_i + P \times \overline{F_{r\to t}} + \min_{a\epsilon(0:Q-1)}(W_{i-a} - W_{i-1-a}) \tag{7}$$

If we predict the arrival time of the packet with equation 7, a packet loss may still occur because of an extended sleep of the transceiver. For this reason we diminish the estimated time R_{i+1} by a constant C which we determine empirically (as described in section IV-B.2). Also, we can reduce the execution time of equation 7 and replace $P \times \overline{F_{r\to t}}$ with the value of $\overline{F'_{r\to t}}$ which will save time by avoiding division and multiplication by P. So finally, the estimated time of the next arriving packet calculated more efficiently than with equation 7 as:

$$R_{i+1} = R_i + \overline{F'_{r\to t}} + \min_{a\epsilon(0:Q-1)}(W_{i-a} - W_{i-1-a}) - C \tag{8}$$

where $\overline{F'_{r\to t}}$ is obtained using equation 5.

When a receiver has multiple *upstream* neighbors, it must keep a record of future arrivals from all of them. Whenever it receives a packet, it predicts the arrival time of the next message from the same sender and goes to sleep until the projected time of the next message.

F. Estimation of event time correlation

Another major purpose of obtaining a common network time reference in WSN is the ability to timestamp events. In most WSN applications the user must know the time of sensed event. In section III-D we describe how every node can estimate the frequency of its *upstream* neighbors with RISS. So if the receiver knows the time elapsed between the sensor reading and the SFD of the packet, it is able to express that period in its local clock units and by consequence precisely estimate the time of acquisition of the data sample. Thus we propose to insert into the packet a field which is updated by every node on the communication chain between the source of the packet and the base station. Initially, the mote records the time elapsed between an event observed (it can be sampling of the sensor, a packet reception *etc.*) and the periodical clock interrupt (T_i) described in section III-C in that field. Then the receiver computes in its local clock units the period between the event observed by the sender and the SFD of the received packet. The instance I of the observed event can be obtained with the following equation:

$$I = R_i - \frac{\overline{F'_{r\to t}} \times [W_i + E]}{P} \tag{9}$$

where E is the time value sent by the source node and $\overline{F'_{r\to t}}$ is the frequency of the sender clock estimated at the receiver (see section III-D).

If the receiver of the packet is not the sink, it forwards the data to the *downstream* neighbor. But before the transmission of the packet, it must update the time elapsed since the initial sensor sampling. Then the receiver will collect the SFD of the packet and repeat the update of the time field before the transmission. This operation continues until the message reaches the base station. In the next section we describe how the synchronization method can be integrated into the network stack.

G. *Integration of the method into routing layer*

We mentioned in section II that the time synchronization should be integrated into one of the network stack layers and then the time information should be accessible by other layers. We think the best option is to incorporate the synchronization service into the routing protocol. The main reason for that is the fact that information about the network architecture is managed by this layer. In the synchronization method we propose, every node must first discover its *upstream* and *downstream* neighbors. This information can be deduced form the routing tables. Besides, we propose that if the duty cycling causes the lost of a packet than the receiver does not go to sleep until the arrival of the next message. This event can be discovered by comparison of the sequence number of received packets. This value is included into most of the routing protocols headers so we can use it for the duty cycling control. For those reasons we think that incorporation of our synchronization method into the network layer will require a minimum additional data transmission and permits an easier use of the information already possessed by the node.

The next section presents the results of the experiments that we carried out in order to confirm the advantages of our solution.

IV. Experimental results

The purpose of our experiments is quadruple:

- Decide whether it is better to use linear regression or our method (section III-D) to estimate clock frequency
- Optimize parameters of the RISS protocol
- Verify robustness the RIIS protocol
- Compare RISS with other synchronization and duty cycling methods in WSN

We implemented RISS in TinyOS-2 and used Tmote Sky motes for experiments. Each Tmote sky node has an 8MHz TI MSP430 microcontroller and a 2.4GHz, 250kbps IEEE 802.15.4 Chipcon wireless transceiver. We built a network composed of four nodes and a single base station. Each node wakes up every 10s, transmits the sensor reading to the sink and goes to sleep. The sink collects 1000 packets from every mote. The time of next wake up is estimated with the RISS protocol. The base station also turns off the transceiver when the channel is free.

A. *Comparison of linear regression and our method*

Initially we compared the computing time of both methods. The results are showed in table I.

	linear regression	our method
time execution	75.6ms	3.6ms

TABLE I

Time Execution of RISS with linear regression and our method

Our method is much faster than linear regression for the similar time precision. The processing inefficiency may be a drawback especially if the frequency estimated is needed to duty cycle nodes. A long calculation may significantly reduce the efficiency of that service. Thus we continued our experiments with our method in order to optimize it.

B. *Optimization of RISS parameters*

The performance of the duty cycle service depends on many parameters. We want to determine empirically their optimal values.

1) Optimization of Q: The first series of experiments permits us to estimate the optimal number of past packets (called Q) needed to predict the next packet arrival time (see equation 5). The execution time of the protocol increases when Q is large. However, when Q is large, our estimation is more accurate. There is a trade-off between time execution and precision. In order to compensate the inaccuracy of the arrival prediction and eventual packet omission we propose to wake up the transceiver earlier than estimated. We measure the average awake time of the sink transceiver in function of the Q.

The result of our test is shown in figure 2(a). The graph has a local minimum when Q equals 8. It means that, for this hardware configuration, the receivers should estimate the arrival of the next packet on the base of the timestamps of 8 previous messages. Then the average time spent on listening

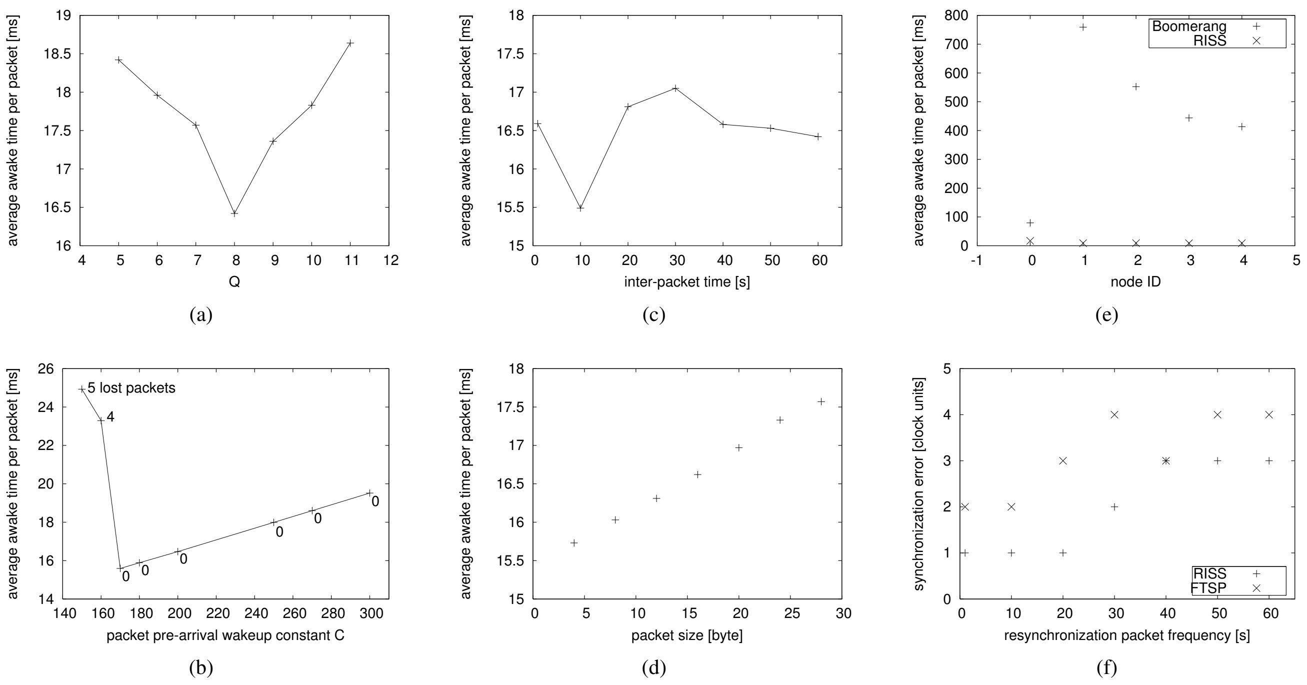

Fig. 2. Results of experiments with RISS protocol. Average transceiver awake time as a function of Q, the number of past packets processed (2(a)); C, the pre-awake constant (2(b)); inter-packet time (2(c)) and packet size (2(d)). Comparison of RISS performance with Boomerang (2(e)) and FTSP (2(f)).

of the channel, processing the packet and turning off the transceiver is minimum and equals 16.42ms.

2) Optimization of C: Also, we carried out a series of tests to optimize the value of constant C in equation 8. The energy efficiency drops with the value of C but if we set the constant C too small, a packet loss may occur. This failure of reception disables duty cycling until the arrival of the next packet and is energy inefficient. So, there is a trade off between setting C to a low value which may increase the energy draining because of the missed packets and a larger value of C that extends the awake time for every packet. We are looking for an optimal value of C. We measure the average awake time and number of packets lost as a function of the C value. The results of this experiment are shown in figure 2(b).

For values of C greater than 170, the average transceiver awake time increases linearly. We did not observe any packet loss in those circumstances. However, the decrease of the C constant below 170 causes an abrupt increase of energy inefficiency. This is due to the packets missed by the receiver. The duty cycling is then disabled until the arrival of the next message. We observed already 4 packets missed when C equals 160. We deduce that duty cycling is optimal when no packet is lost. Even if we have to increase awake time for every packet, this cost is negligible in comparison to the energy lost caused by a lost message. The minimum value of C which prevents packet loss is 170 for this hardware configuration. It means the transceiver is powered up $170/32768kHz = 5.19ms$ earlier than predicted in order to compensate for the stochastic component of the packet arrival time.

C. Sensitivity of RISS

In this series of tests we want to verify the influence of application specific parameters on the performance of RISS. We test how the energy efficiency varies with the packet size. We also want to test whether the RISS protocol is sensitive to the duty cycle period.

1) Energy efficiency of RISS in function of the packet size: We observe how the packet size influences the performance of the duty cycling service. We set up the protocol parameters to the optimal values previously determined. We vary the size of the packet sent by nodes and we record the average awake time of the sink. The time processing increases linearly with the number of bytes of the message (see figure 2(d)). For the minimum packet size of 4 bytes we measured awake time of 15.73ms and for the Tmote's maximum packet size of 28 bytes we evaluated this value to be 17.57ms. We deduce that in average the transceiver power up time increases by 0.0784ms for an additional byte in a packet. The CC2420 transceiver has a transmission rate of 250kbps, so it takes 0.032ms to transmit 1 byte. The difference 0.0784-0.032=0.0464ms is due to the processing by the duty cycling protocol of an additional byte.

2) Energy efficiency of RISS in function of the duty cycle period: We also want to determine whether the energy efficiency of the duty cycling depends on the inter-packet period. This time we modify the frequency of packets and analyze the variation of the energy efficiency. Our observations are plotted in figure 2(c).

The protocol energy efficiency is pretty stable for different values of packet frequency and node wakes up for about 15.5-17ms in average. Obviously if the inter-packet time becomes too large, the awake time increases because of the lack of precision in time synchronization. However, in real applications enlarging this value is unreasonable because it may lead to the lost of network connectivity and lack of information about the neighbor nodes.

D. Comparison of RISS with other protocols

The aim of the last set of experiments is to prove the advantage of our solution over other existing methods. We compare both uses of RISS: duty cycling of the nodes and events timestamping.

1) Comparison of RISS for duty cycling: To compare the duty cycling performance of RISS we chose as a reference the scheduling protocol which is a part of Boomerang, Moteiv's distribution of TinyOS [1]. In the first stage we deployed Boomerang on the network of four motes and a base station. The *lowpower* coefficient is set to 1%, possibly most efficient value for Boomerang. We measure the average awake time of every mote. In the second phase of the test we installed the RISS protocol on the same network. The comparative results are plotted in figure 2(e).

Each node stays asleep for longer when using duty cycling with RISS. The average awake time per node equals 449.6ms with Boomerang and 10.6ms with our method. The node 0 is the base station. We can observe that with Boomerang it consumes less energy than other nodes. With our protocol we observe an inverse situation. It is due to the different approach toward the synchronization. In Boomerang it is the *upstream* mote which adapts to the duty cycling operation of its *downstream* neighbors. We implemented the opposite solution in RISS. While the difference of average awake time between the sink and least efficient node is significant in the case of Boomerang (679.98ms), with our protocol it is very small (8.06ms).

2) Comparison of RISS for events timestamping: We test the timestamp precision of RISS by measuring the difference between protocol's estimate of an event time and its actual instance. The node computes with RISS the event time on the base of the information sent by a neighbor. To do this we propose the following experiment scenario. A network is composed of a single node and a base station. The sink collects periodically sensor samples from the mote. It also estimates sender's clock frequency with the RISS protocol. The event is simulated by a packet broadcasted by a third node. The two motes which hear the message, record its timestamp. When the sensor reading is sent to the sink, the transmitter adds to the packet the time which elapsed from the simulated event to the SFD of the message. Then, the base station uses that value to calculate the event time and compares the result with its own timestamp of the event. For every test we vary the frequency of the sensor packets and we measure the maximal error of the event time estimation at the base station. We compare the precision of RISS with that of FTSP [7] which we tested with the same scenario. Our observations are showed in figure 2(f).

On that graph we plotted the error of event time estimation with FTSP and RISS. This inaccuracy is expressed in the 32kHz clock units. The exactness of the method depends on the update of the synchronization between nodes. This is done by the exchange of the beacon packets in case of FTSP and by addition of the time information to the application packets. We observed the smallest synchronization error of $1/32768 \approx 30.5\mu s$ when the time information is updated every second. We can also ascertain that RISS estimates time of the event more precisely than FTSP. For almost all experiments the error was less than one clock unit.

V. Conclusion

Most WSN applications require timestamping. Also, the duty cycling of the motes is a common technique to extend the network lifetime. These operations need time synchronization which may use a lot of resources if incautiously implemented. However, limited capabilities of WSN devices oblige researchers to design a very energy efficient protocol. We proposed a protocol called RISS which integrates the synchronization service into the routing layer in WSN. It uses only 10 bits of every packet to send time information and takes about 16ms to calculate each neighbor's clock frequency using typical motes. For such a small cost we achieved a higher performance than FTSP. Also the duty cycling service using our method turned out to be more efficient than the Boomerang implementation of scheduling. It has been demonstrated empirically on a network composing 5 motes that RISS features good time precision and energy efficiency. Future work will address the issues involved in scaling RISS for deployment in a large network (*e.g.* with more than hundred nodes).

References

[1] Boomerang. Moteiv corporation. www.moteiv.com/software.

[2] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, and J. Hui. Towards a sensor network architecture: lowering the waistline. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 24–24, Berkeley, CA, USA, 2005.

[3] S. Fedor and M. Collier. Cross-layer routing with data delivery guarantee in wireless sensor networks. In *Proc. of the ACM Workshop on Real-World Wireless Sensor Networks (REALWSN 2006)*, pages 19–23, Uppsala, Sweden, 2006. ACM SIGMOBILE.

[4] S. Fedor and M. Collier. On the problem of energy efficiency of multi-hop vs one-hop routing in wireless sensor networks. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, Niagara Falls, Canada, 2007. IEEE Computer Society.

[5] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.

[6] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, New York, USA, 2002.

[7] M. Maróti, B. Kusy, G. Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM Press.

[8] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, USA, 2005.