FTSP+: A MAC Timestamp independent flooding time synchronization protocol

Heder Dorneles Soares[1], **Raphael Pereira de Oliveira Guerra**[1], **Célio Vinicius Neves de Albuquerque**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Av. Gal. Milton Tavares de Souza, São Domingos – Niterói – RJ – Brasil

{hdorneles,rguerra,celio}@ic.uff.br

Abstract. *Wireless sensors networks are distributed system composed of battery-powered nodes with low computational resources. Like in many distributed systems, some applications of WSN require time synchronization among their nodes. In the particular case of WSN, synchronization algorithms must respect the nodes computational constraints. The well known FTSP protocol is famous for achieving nanosecond precision with low overhead. However, it relies on MAC timestamp, a feature not available in all hardware. In this work, we propose MAC timestamp independent version in order to extend and adapt FTSP to work on hardware that do not have MAC timestamp while keeping the low overhead and high synchronization precision. Our results we estimate an average synchronization error of* $1.508\mu s$ *per hop, while adding a corretion message.*

1. Introduction

Wireless Sensor Networks (WSNs) are composed of small computational devices equipped with an antenna for wireless communication, one or more kind of sensors, and a small CPU for simple data processing [Baronti et al. 2007]. These devices are usually called *motes*. Due to the limited radio signal range and energetic constraint, WSNs have restrictions and unique characteristics that differ from traditional networks and distributed systems [Arampatzis et al. 2005].

WSNs have several applications in many diverse fields. Sensors deployment in military applications have always been widespread, so the introduction of motes was a natural incorporation to the advancement of systems already used. Applications enhanced with WSN include tracking enemy and targets [Yang and Sikdar 2003], monitoring of vehicles [Sinopoli et al. 2003], countersniper system [Simon et al. 2004], and surveillance systems [Gui and Mohapatra 2004]. Environmental monitoring also provide opportunities to apply WSN. Our environment has a lot of information that play an important role in our quality of life, such as quality of air, water, sound and solar radiation to which we are exposed directly and affect our health [Oliveira and Rodrigues 2011, Cardell-Oliver et al. 2004]. Increasing interest in green computing has led to concerns with energy consumption of IT facilities [Chong et al. 2014], and WSNs play a strategic role in monitoring and controlling these environments [Zanatta et al.].

Some of those applications require time synchronization mechanisms with good accuracy and scalability, all this complying with their low computational resources and

energy availability [Zanatta et al. , Simon et al. 2004, Yang and Sikdar 2003]. Flooding Time Synchronization Protocol (FTSP) is the most popular time synchronization algorithm for WSN [Maróti et al. 2004]. It is fault-tolerant, achieves high accuracy ($\sim 1,5\mu s$ per hop) utilizing timestamps in low layers of the radio stack, and saves energy using a linear regression technique to compensate clock skews with few exchanges of synchronization messages. However, MAC layer timestamping is not a standardized feature, and hence, not interoperable among different hardware and physical layer protocols. There is an effort from Google to standardize MAC layer timestamping in WSN [Wang and Ahn 2009], but so far there is not much compliance.

Many synchronization protocols use MAC timestamp: some have less accuracy than FTSP, focus at other problems and make more restrictive assumptions [Ganeriwal et al. 2003, van Greunen and Rabaey 2003]; others can achieve better accuracy between distant nodes [Nazemi Gelyan et al. 2007, Lenzen et al. 2009, Sommer and Wattenhofer 2009]. Elson et al. proposed the Reference Broadcast Synchronization [Elson et al. 2002] (RBS) to eliminate uncertainty of the sender without MAC timestamp by removing the sender from the critical path. The idea is that a third party will broadcast a beacon to all receivers. The beacon does not contain any timing information; instead the receivers will compare their clocks to one another to calculate their relative phase offsets. It has $\sim 30\mu s$ error per hop and is independent of MAC timestamp. Ranganathan and Nygard offer a good overview of these protocols [Ranganathan and Nygard 2010].

In this paper, we propose a modified version of FTSP, called FTSP+, in order to work without MAC timestamp. For that, we use application level time-stamping on synchronization messages, which leads to a well-known problem: the time elapsed between time-stamping the packet and gaining the wireless medium reduces synchronization accuracy. We circumvent this problem using medium access interrupt handlers on the sender to measure the time needed to gain the medium, and correction messages to reduce sender side time uncertainty.

Our experimental results show that we only double FTPS synchronization inaccuracy, a great result compared to RBS which is 10 times worse. We also present a quantitative evaluation of medium access and processing delays on TinyOS, an evaluation that we have not found in related work.

There rest of this paper is organized as follows. Section 2 briefly recalls FTSP as needed for this work. Section 3 describes FTSP+. Section 4 brings our experiment results, and finally, Section 5 brings the concluding remarks.

2. Flooding Time Synchronization Protocol

This section briefly describes the Flooding Time Synchronization Protocol (FTSP). For more detailed information, refer to [Maróti et al. 2004].

FTSP is a synchronization protocol for WSN that provides high accuracy, consumes few resources, uses little bandwidth and is fault-tolerant. It elects a node as root to provide the time reference for synchronization; if root failure is detected (using timeouts), another root is elected. Root and synchronized nodes send synchronization messages periodically, and receiving nodes use these messages to synchronize. Therefore, FTSP supports multi-hop networks.

Synchronization messages comprise a *sender timestamp* which is the estimated global time and *rootID* which is the network identifier of the root (where the node with the lowest ID is the chosen root). *seqNum* is a sequence counter that is incremented each synchronization round; this field is used to verify the redundancy of messages [Maróti et al. 2004].

All nodes think they are root when the network starts, so they broadcast synchronization messages to the network. When they receive a synchronization message, they check who has the lowest ID: if the local ID is higher, this node gives up on being root and starts synchronizing. Another important check is the *seqNum*. If it is greater than the local value *highestSeqNum*, it means that this is a new synchronization message and starts the synchronization procedure.

The synchronization procedure consists of computing a linear regression [Elson and Estrin 2003] that will provide the clock skew (used to estimate the global time) in relation to the reference node. The last step is to forward its local (synchronized) time to other nodes.

```
1  event Radio.receive(TimeSyncMsg *msg){
2    if( msg->rootID < myRootID )
3        myRootID = msg->rootID;
4    else if( msg->rootID > myRootID
5      || msg->seqNum <= highestSeqNum )
6        return;
7    highestSeqNum = msg->seqNum;
8    if( myRootID < myID )
9        heartBeats = 0;
10   if( numEntries >= NUMENTRIES_LIMIT
11     && getError(msg) > TIME_ERROR_LIMIT )
12       clearRegressionTable();
13   else
14       addEntryAndEstimateDrift(msg);
15 }
```

Figure 1. FTSP Receive Routine [Maróti et al. 2004]

Figure 1 shows the routine of receiving synchronization messages. Lines 2 and 3 compare the *rootID* of the synchronization message with the local *rootID*. If the message has a lower *rootID*, the node assumes this *rootID* as root. Lines 4 to 7 ignore messages with higher *rootID* and lower *seqNum*. If *seqNum* is higher, local *highestSeqNum* is updated. Lines 8 and 9 makes a root node give up on being root when it has a *rootID* lower than its own ID. In case the *rootID* is larger it is checked whether the *seqNum* is greater or equal to the value of *highestSeqNum*, this check prevents information redundancy because the message will only be used when the *rootID* is less than or equal to *myRootID* and the number greater than the value of *highestSeqNum*. Lines 10 to 15 verify if the time of a message is in disagreement with the earlier estimates of global time, if applicable clear the regression table, if not, accumulate synchronization messages to calculate the linear regression and synchronize.

```
1  event Timer.fired() {
2    ++heartBeats;
3    if( myRootID != myID
4      && heartBeats >= ROOT_TIMEOUT )
5        myRootID = myID;
6    if( numEntries >= NUMENTRIES_LIMIT
7      || myRootID == myID ){
8        msg.rootID = myRootID;
9        msg.seqNum = highestSeqNum;
10       Radio.send(msg);
11   if( myRootID == myID )
12       ++highestSeqNum;
13   }
14 }
```

Figure 2. FTSP Send Routine [Maróti et al. 2004]

Figure 2 shows the sending routine. A node decides to be root because has not received a synchronization message for ROOT_TIMEOUT (lines 3 to 5). A node sends synchronization messages if it is root or has synchronized (lines 6 to 10). If a node is root, it also has to increment its *highestSeqNum*.

3. FTSP+

In any distributed time synchronization technique, nodes have to tell each other their local time. Figure 3 depicts this scenario. The sending node stores its local time $t1'$ in the synchronization message at time $t1$ and orders the message to be sent. Due to medium random access uncertainties, the sending node only gets access to the medium at time $t2$ and starts sending me message. $t2 - t1$ is called *medium access time*. The message propagates over the medium for an interval of time tp until it reaches the receiver radio at time $t3$. tp is the *propagation time*. Due to interrupt handling policies and packet header processing, the receiver node timestamps the message receipt at time $t4$ with its local time $t4''$. $t4 - t3$ is the *processing time*.

Synchronization inaccuracy happens because the receiving node thinks that at time $t4$ the sender has time $t1'$ and the receiver has time $t4''$. We can see in Figure 3 that this is not true. At time $t4$, the sending node has time $t4' = t1' +$ *medium access time* $+$ *propagation time* $+$ *processing time*. MAC layer time-stamping makes *medium access time* and *processing time* equal zero. Since *propagation time* is negligible ($\sim 1\mu s$) [Maróti et al. 2004], some synchronization policies — including FTSP — can achieve very good accuracy. However, without MAC layer time-stamping, these times are non-negligible and have to be calculated.

FTSP+ calculates *medium access time* using an interrupt handler to time stamp the moment that the node gets medium access to send the synchronization message. Although medium access is granted at time $t2'$, *medium access timestamp* equals $t2' + \delta$, where δ is the overhead to process the interrupt handler.

The sender sends a correction message with content $t2' + \delta - t1'$ so the receiver

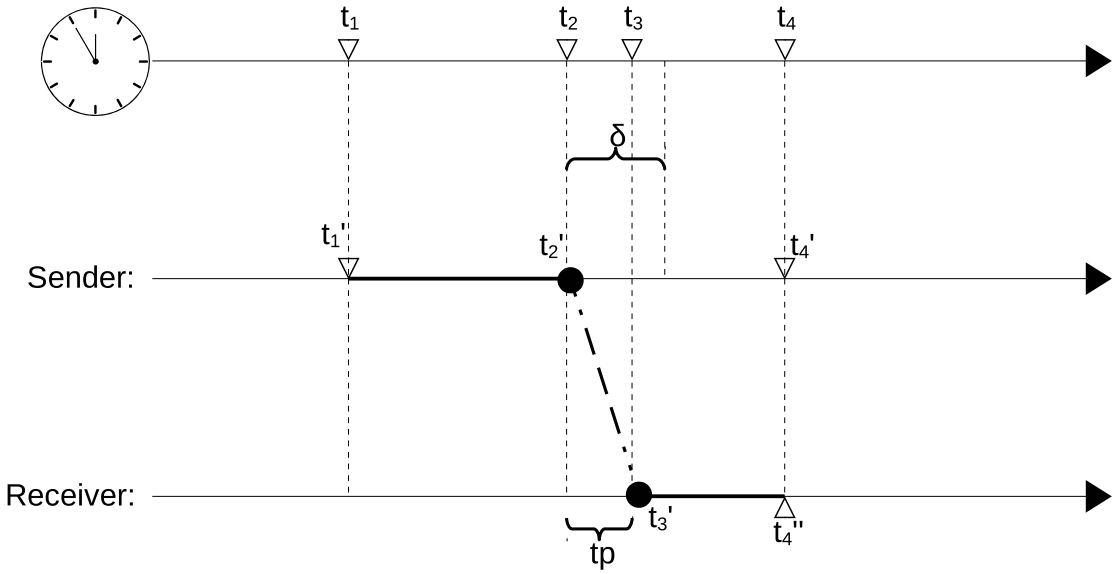

Figure 3. Syncronization steps.

can estimate $t4'$. Let us call this estimation $t\bar{4}'$. The receiver calculates $t\bar{4}'$ as in Equation (1).

$$\begin{aligned} t\bar{4}' &= t1' - (t2' + \delta - t1') \\ t\bar{4}' &= t1' + \textit{medium access time} + \delta \end{aligned} \tag{1}$$

The difference between $t4'$ and $t\bar{4}'$, which is the estimation error, is:

$$t4' - t\bar{4}' = \textit{propagation time} + \textit{processing time} - \delta \tag{2}$$

Since *processing time* and δ are interrupt handler processing latencies, they tend to cancel out each other. We investigate their values in our experiments in Section 4. Remember that *propagation time* is negligible.

As can be seen in Figure 4, an FTSP+ receiver has routines to handle two different types of income messages: `Radio.receiveSyncMsg(SyncMsg *msg)` handles synchronization messages and `Radio.receiveCorrectionMsg(CorrectionMsg *msg)` handles correction messages. The *correction time*, expressed in Equation (3), is the information that is transmitted to the receiver to apply the correction to previous messages.

$$\textit{correction time} = t2' - t1' + \delta \tag{3}$$

The `Radio.receiveSyncMsg(SyncMsg *msg)` differs from FTSP receive only for lines 13 and 14. These lines store synchronization messages in a list to wait for the correction messages if MAC layer time-stamping is not available.

```
event Radio.receiveSyncMsg(SyncMsg *msg){
  if( msg->rootID < myRootID )
      myRootID = msg->rootID;
  else if( msg->rootID > myRootID
    || msg->seqNum <= highestSeqNum )
      return;
  highestSeqNum = msg->seqNum;
  if( myRootID < myID )
      heartBeats = 0;
  if( numEntries >= NUMENTRIES_LIMIT
    && getError(msg) > TIME_ERROR_LIMIT )
      clearRegressionTable();
  else if (MAC_Time==false){
      addToWaitCorrectionMsgList(msg);
  }else{
      addEntryAndEstimateDrift(msg);
  }
}

event Radio.receiveCorrectionMsg(CorrectionMsg *msg){
  applyCorrection(msg);
  addEntryAndEstimateDrift(msg);
}
```

Figure 4. Receiver algorithm

The event `Radio.receiveCorrectionMsg(CorrectionMsg *msg)` receives the message with the correction value, use the function `applyCorrection(msg)` that applies the correction and removes from the wait list the synchronization message that matches the incoming correction message, corrects the timestamp and calls function `addEntryAndEstimateDrift(msg)`.

```
1  event Timer.fired() {
2    ++heartBeats;
3    if( myRootID != myID
4      && heartBeats >= ROOT_TIMEOUT )
5        myRootID = myID;
6    if( numEntries >= NUMENTRIES_LIMIT
7      || myRootID == myID ){
8        msg.rootID = myRootID;
9        msg.seqNum = highestSeqNum;
10       initialTime = call getLocalTime();
11       msg.timestamp = initialTime;
12       Radio.send(msg);
13   if( myRootID == myID )
14       ++highestSeqNum;
15   }
16 }
17
18 event sendDone(SyncMsg *msg, error_t error){
19    finalTime = call getLocalTime();
20    correctionMsg.correction = finalTime-initialTime;
21    correctionMsg.seqNum = msg.seqNum;
22    Radio.send(correctionMsg);
23 }
```

Figure 5. Sender algorithm

As can be seen in Figure 5, an FTSP+ sender has two routines: `Timer.fired()` periodically sends synchronization messages (like in FTSP) and `sendDone(message_t *msg, error_t error)` sends the correction message.

The function `Timer.fired()` differs from FTSP only for lines 10 and 11, which collects the timestamp at application level. `sendDone(message_t *msg, error_t error)` handles the interrupt when wireless medium access is granted to collect the medium access timestamp by compute the time that has passed between `send/sendDone` and send out the correction message, this technique is previously discussed by [Sousa et al. 2014] that is a simple technique for local delay estimation in WSN.

Lines 20-21 show how the correction time is computed. `seqNum` is used to identify which message is being adjusted, `finalTime` and `initialTime` refers to times $t2$ and $t1$ in Equation (3).

4. Evaluation

In this section, we describe our experiments to assess FTSP+'s accuracy. We have implemented FTSP+ on TinyOS 2.1.2 [Levis et al. 2005] and ran our tests on Micaz motes [MEMSIC 2015]. Micaz motes do support MAC layer timestamping, but we need this information to measure our correction accuracy. For synchronization, we overwrite MAC timestamp with our application layer timestamp.

We use *jiffys* as time unit because this is the time basis of TinyOS and represents the time between 2 clock ticks. Our experiments results correspond to 10 minutes experiments with synchronization messages being sent every 3 seconds, with 2 nodes communicating and a base station connected to an computer via serial port for record the messages.

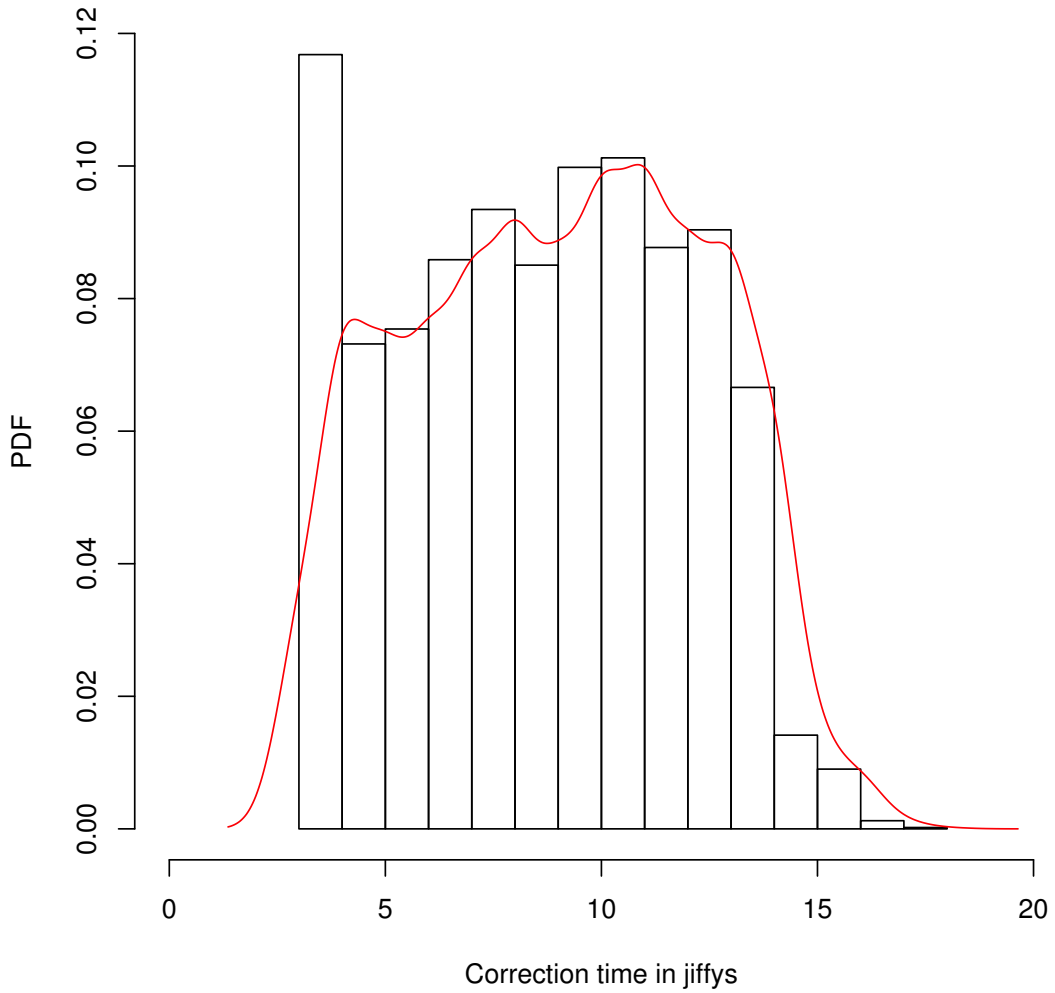

Figure 6. Histogram and P.D.F. of correction times.

Our first experiment measures the probability distribution of correction times (recall from Section 3 that it is $t2 - t1 + \delta$). As can be seen in Figure 6, correction time

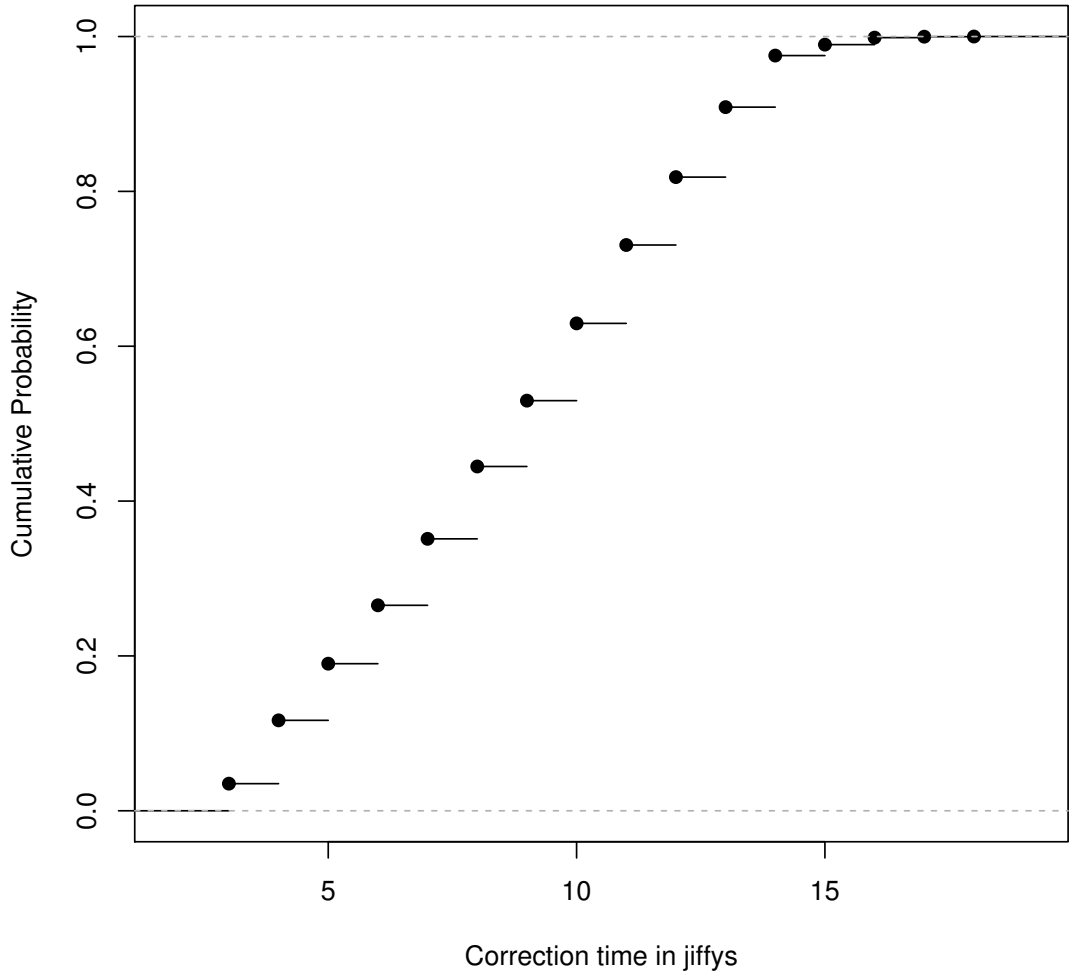

Figure 7. C.D.F. of correction times.

	mean (μs)	**std. deviation**
processing time	0.87 ± 0.0095	0.33
δ	0.88 ± 0.0093	0.33
correction time	9.01 ± 0.093	3.32
medium access	8.13 ± 0.093	3.31
$\mathbf{t4' - \bar{t4'}}$	-0.0047 ± 0.013	0.47

Table 1. Mean and standard deviation

varies mostly from 5 to 13 jiffys with an uniform distribution. We can see in Figure 7, which plots the cumulative density function of the correction times, that in about 80% of cases correction time is above 5 jiffys. This is a significant overhead and source of inaccuracy that FTSP+ is able to measure and compensate for. For our scenario the average

correction time delay is 9.01[1] jiffys as we can see in Table 1 where we have the mean with confidence interval of 95% and the standard deviation.

Delay in jiffys	0	1	2	3	4	5
Frequency	606	4271	0	1	1	1

Table 2. Delay frequency of receiver *processing time*.

Delay in jiffys	0	1	2	3	4	5
Frequency	581	4297	0	0	1	1

Table 3. Delay frequency for δ.

Our second experiment measures the probability distribution of δ and *processing time*. Recall from Section 3 that they are the radio interrupt processing time in the sender and the receiver, respectively. We can see in Table 2 and 3 that their values mostly range between 0 and 1 jiffy, with extremely rare cases that do not go beyond 5 jiffys in our experiments. Their averages are 0.87 and 0.88 jiffys, which can be considered equal. Therefore, those delays, that FSTP+ is not able to calculate and compensate for, are very small and do not compromise synchronization accuracy significantly.

We calculate an estimation of the error and compare our results with other protocols (some use MAC timestamp, some do not) and summarize the results in Table 4. Knowing MAC timestamping has an inherent 1 jiffy variation in accuracy and that FTSP has $1.5\mu s$ error per hop, we use a simple rule of three to estimate FTSP+ synchronization error per hop. FTSP and FSTP+ differ only for their timestamping technique, and as can be seen in Table 1 FTSP+ timestamping is on average 0.0047 jiffy higher than MAC timestamping ($t4' - \bar{t4}'$). Therefore, our rule of three is given in Equation 4.

$$\frac{1.5\mu s}{\text{FTSP+ sync error}} = \frac{1\textit{jiffys}}{(0.0047+1)\textit{jiffys}} \tag{4}$$

We also estimate what would be the error of FTSP without MAC timestamping (we call it mFTSP in the table), in which case timestamping is on average 9 jiffys higher than MAC timestamping ($medium\ access\ delay + processing\ delay = 8.13 + 0,87 = 9$). Equation 5 shows the rule of three for this estimation. As can be seen, RBS has a synchronization error about 10 times bigger than FTSP+.

$$\frac{1.5\mu s}{\text{mFTSP+ sync error}} = \frac{1\textit{jiffys}}{(9+1)\textit{jiffys}} \tag{5}$$

In TinyOS 1.x, where FTSP was first implemented and tested [Maróti et al. 2004], jiffys are in microsecond resolution by default. A jiffy in TinyOS 2.1.2 is approximately $1ms$, but there are compiler *TMilli* options to allow microsecond resolution. In future

[1]This value is scenario-dependent while it depends upon the scenario contention level.

[1]This value is scenario-dependent while it depends upon the scenario contention level.

[2]Values estimated based on average error in jiffys.

[3]Based in [Ranganathan and Nygard 2010].

MAC Timestamping		App Timestamping		
FTSP	**PulseSync**	**FTSP+**	**mFTSP**	**RBS**
$1.5\mu s$ [3]	$1.5\mu s$ [3]	$1.508\mu s$ [2]	$15\mu s$ [1 2]	$29\mu s$ [3]

Table 4. Average synchronization error per hop.

work, we want to use jiffys with microsecond resolution and measure the synchronization error per hop using a setup with GPIO pins for tracking events in an external and stable accuracy clock that record every exchanged messages.

4.1. Energy Consumption

The energy consumption was analyzed in experiments using the resources available on the Testbed [Lim et al. 2015]. Running the previous experiments we find the following result listed in Table 5.

FTSP	99.04 mJ
FTSP+	99.80 mJ

Table 5. Mean of consumption

The values represent the power consumption of CPU and radio communication of network (include the root and client node). We find close values in the tests, they are also related to synchronization frequency used on the network, for each synchronization message of FTSP the FTSP+ send another for correction, causing an extra consumption.

5. Conclusion

In this work, we presented a modified version of Flooding Time Synchronization Protocol to work without MAC layer timestamping. To keep accuracy as high as possible, we use radio interrupts to measure the instant nodes get medium access. Senders use correction messages in addition to synchronization messages in order to compensate their synchronization timestamps with medium access delay.

In our experiments, we ran tests on Micaz motes running TinyOS 2.1.2 to measure correction times, processing times and medium access time. We showed that medium access time is the main source of synchronization error and quantified it in a real testbed. We also showed that processing times are very small, on average 0.87 jiffys.

In future work, we want to precisely measure the synchronization error using a high accuracy external clock to measure timing error between synchronization events among the network motes. We also intend to consolidate the power consumption tests for different scenarios and see how much the variation in synchronization parameters affect in energy expenditure of WSN.

References

[Arampatzis et al. 2005] Arampatzis, T., Lygeros, J., and Manesis, S. (2005). A survey of applications of wireless sensors and wireless sensor networks. In *IEEE International Symposium on Intelligent Control, Mediterrean Conference on Control and Automation*, pages 719–724. IEEE.

[Baronti et al. 2007] Baronti, P., Pillai, P., Chook, V. W. C., Chessa, S., Gotta, A., and Hu, Y. F. (2007). Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Comput. Commun.*, pages 1655–1695.

[Cardell-Oliver et al. 2004] Cardell-Oliver, R., Smettem, K., Kranz, M., and Mayer, K. (2004). Field testing a wireless sensor network for reactive environmental monitoring [soil moisture measurement]. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004*, pages 7–12.

[Chong et al. 2014] Chong, F., Heck, M., Ranganathan, P., Saleh, A., and Wassel, H. (2014). Data center energy efficiency:improving energy efficiency in data centers beyond technology scaling. *Design Test, IEEE*, 31(1):93–104.

[Elson et al. 2002] Elson, J., Girod, L., and Estrin, D. (2002). Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163.

[Elson and Estrin 2003] Elson, J. E. and Estrin, D. (2003). *Time synchronization in wireless sensor networks*. PhD thesis, University of California, Los Angeles.

[Ganeriwal et al. 2003] Ganeriwal, S., Kumar, R., and Srivastava, M. B. (2003). Timing-sync protocol for sensor networks. In *1st International Conference on Embedded Networked Sensor Systems*, pages 138–149. ACM.

[Gui and Mohapatra 2004] Gui, C. and Mohapatra, P. (2004). Power conservation and quality of surveillance in target tracking sensor networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, pages 129–143. ACM.

[Lenzen et al. 2009] Lenzen, C., Sommer, P., and Wattenhofer, R. (2009). Optimal clock synchronization in networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 225–238. ACM.

[Levis et al. 2005] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., et al. (2005). Tinyos: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer.

[Lim et al. 2015] Lim, R., Maag, B., Dissler, B., Beutel, J., and Thiele, L. (2015). A testbed for fine-grained tracing of time sensitive behavior in wireless sensor networks. In *Local Computer Networks Workshops*.

[Maróti et al. 2004] Maróti, M., Kusy, B., Simon, G., and Lédeczi, Á. (2004). The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM.

[MEMSIC 2015] MEMSIC (2015). Micaz datasheet. `http://www.memsic.com`.

[Nazemi Gelyan et al. 2007] Nazemi Gelyan, S., Eghbali, A., Roustapoor, L., Yahyavi Firouz Abadi, S., and Dehghan, M. (2007). Sltp: Scalable lightweight time synchronization protocol for wireless sensor network. In *Mobile Ad-Hoc and Sensor Networks*, volume 4864, pages 536–547. Springer Berlin.

[Oliveira and Rodrigues 2011] Oliveira, L. M. and Rodrigues, J. J. (2011). Wireless sensor networks: a survey on environmental monitoring. *Journal of communications*, 6(2):143–151.

[Ranganathan and Nygard 2010] Ranganathan, P. and Nygard, K. (2010). Time synchronization in wireless sensor networks: a survey. *International journal of UbiComp (IJU)*, 1(2):92–102.

[Simon et al. 2004] Simon, G., Maróti, M., Lédeczi, Á., Balogh, G., Kusy, B., Nádas, A., Pap, G., Sallai, J., and Frampton, K. (2004). Sensor network-based countersniper system. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12. ACM.

[Sinopoli et al. 2003] Sinopoli, B., Sharp, C., Schenato, L., Schaffert, S., and Sastry, S. S. (2003). Distributed control applications within sensor networks. *Proceedings of the IEEE*, 91(8):1235–1246.

[Sommer and Wattenhofer 2009] Sommer, P. and Wattenhofer, R. (2009). Gradient clock synchronization in wireless sensor networks. In *International Conference on Information Processing in Sensor Networks*, pages 37–48.

[Sousa et al. 2014] Sousa, C., Carrano, R. C., Magalhaes, L., and Albuquerque, C. V. (2014). Stele: A simple technique for local delay estimation in wsn. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–6. IEEE.

[van Greunen and Rabaey 2003] van Greunen, J. and Rabaey, J. (2003). Lightweight time synchronization for sensor networks. In *Proceedings of the 2Nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 11–19. ACM.

[Wang and Ahn 2009] Wang, S. and Ahn, T. (2009). Mac layer timestamping approach for emerging wireless sensor platform and communication architecture. US Patent App. 12/213,286.

[Yang and Sikdar 2003] Yang, H. and Sikdar, B. (2003). A protocol for tracking mobile targets using sensor networks. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 71–81. IEEE.

[Zanatta et al.] Zanatta, G., Bottari, G. D., Guerra, R., and Leite, J. C. B. Building a WSN infrastructure with COTS components for the thermal monitoring of datacenters. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea*.