A New Distributed Time Synchronization Protocol for Multihop Wireless Networks*

Roberto Solis†, Vivek S. Borkar*, and P. R. Kumar‡

***Abstract*— A distributed algorithm to achieve accurate time synchronization in large multihop wireless networks is presented. The central idea is to exploit the large number of global constraints that have to be satisfied by a common notion of time in a multihop network. If, at a certain instant, O_{ij} is the clock offset between two neighboring nodes i and j, then for any loop $i_1, i_2, i_3, \ldots, i_n, i_{n+1} = i_1$ in the multihop network, these offsets must satisfy the global constraint $\sum_{k=1}^{n} O_{i_k, i_{k+1}} = 0$. Noisy estimates $\widehat{O}_{ij}$ of O_{ij} are usually arrived at by bilateral exchanges of timestamped messages or local broadcasts. By imposing the large number of global constraints for all the loops in the multihop network, these estimates can be smoothed and made more accurate.**

A fully distributed and asynchronous algorithm which functions by simple local broadcasts is designed. Changing the time reference node for synchronization is also easy, consisting simply of one node switching on adaptation, and another switching it off. Implementation results on a forty node network, and comparative evaluation against a leading algorithm, are presented.

I. Introduction

Several distributed applications over wireless networks require accurate clock synchronization. Examples are closing control loops over wireless networks [1], object localization or tracking in wireless sensor networks [2], and slotted protocols in communication networks [3]. We present a distributed asynchronous algorithm to achieve higher performance clock synchronization in multihop wireless networks, and experimental results comparing it with a leading algorithm.

We improve the accuracy of clock synchronization by exploiting global network-wide constraints satisfied by the very notion of time, and achieve this through a completely asynchronous, distributed algorithm employing only local broadcasts. Suppose O_{ij} is the offset of the clock at node j with respect to the clock at a neighboring node i, at a certain time. An estimate of $\widehat{O}_{ij}$ can be formed by bilateral exchange of timestamped packets between the neighboring nodes i and j. These estimates are, however, noisy and are based only on the particular timestamped packets exchanged between the two neighboring nodes. Now consider any loop $i_1, i_2, \ldots, i_n, i_{n+1} = i_1$, in the multihop network. Then $\sum_{k=1}^{n} O_{i_k i_{k+1}} = 0$. By imposing several such global constraints, one for every loop in the graph, the noisy estimates $\widehat{O}_{ij}$ can be further improved. We provide a distributed asynchronous algorithm that thereby exploits *all* global information, and not just along a rooted tree, as is traditional. It only uses local broadcasts, does not need any global topology knowledge, or any construction of a rooted tree. Further, changing the clock reference node requires only a message to the new node. We also present the results of an implementation over a 40-node Berkeley Motes testbed, comparing it with a leading time synchronization protocol.

II. Related Work

An algorithm suitable for clock synchronization in ad hoc networks is proposed in [4]. Its basic idea is to generate time stamps using unsynchronized local clocks that when passed between devices will be transformed to the local time of the receiving device. The accuracy obtained is in the order of milliseconds. A time translation control time protocol is also developed in [5]. An impossibility result is also established under asymmetric delays. In [6] clock drift between nodes is assumed to be linear, and nodes exchange timestamps to estimate the best linear fit. For a multihop network it is proposed to organize the network as a tree hierarchy, with nodes in layer i synchronizing with nodes in layer i-1. Its performance was tested on an 802.11b multihop ad hoc network, achieving an accuracy of 3 ms over a single hop. In [7] also a hierarchical tree topology of the network is used, with the average accuracy obtained being around 17μs for the synchronization of two motes. In the scheme RBS (Reference Broadcast Synchronization) described in [8], an intermediate node transmits a reference packet and the other nodes record the time at which they receive it. They then exchange this recorded time to find their clocks' difference. The accuracy obtained for two motes is within 11 μs. A multihop extension is also proposed to synchronize at least two groups of nodes, but it relies on effective clustering of the nodes around the broadcast nodes. More recently, in [9], the Flooding Time Synchronization Protocol (FTSP) is proposed which uses MAC layer timestamping capabilities to eliminate several sources of error on the time synchronization process, and linear regression to

*This material is based upon work partially supported by DARPA Contract N66001-06-C-2021, AFOSR under Contract No. F49620-02-1-0217, NSF under Contract Nos. NSF CNS 05-19535, ANI 02-21357 and CCR-0325716, USARO under Contract No. DAAD19-01010-465, DARPA/AFOSR under Contract No. F49620-02-1-0325, DARPA under Contact Nos. N00014-0-1-1-0576. Roberto Solis Robles is studying under the sponsorship of SEP-CONACYT-PROMEP and CSL in the Department of Computer Science, University of Illinois at Urbana-Champaign. The research of Vivek Borkar has been partially supported by Project 2900-IT-1 from the Centre Franco-Indien pour la Promotion de la Recherche Avancee (CEFIPRA).

†Dept. of CS, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. email: rsolis@acm.org.

*Tata Institute of Fundamental Research, School of Technology and Computer Science Homi Bhabha Road, Mumbai 400005, India. email:borkar@mailhost.tifr.res.in.

‡CSL and Dept. of ECE, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. email: prkumar@uiuc.edu.

compensate for the possible drifts in the clocks. A leader or root is elected through message exchanges, and the global time is passed from the root to all nodes through flooding. The average accuracy obtained for a 60-node network (with a maximum of six hops from the root) is 14 μs.

The problem of forming a minimum variance estimate based on using global information is studied in [10], with particular attention paid to the reference broadcast synchronization (RBS) method of [11]. It is shown that the minimum variance estimate satisfies the global property of sum of offsets over loops being zero. A scheme for forming the estimate based on the RBS method is proposed, and an analysis is also carried out. A theoretical analysis of our algorithm is provided in [12].

III. System Model

We suppose that the clock drift at a node follows the linear form: $T_i = \alpha_i t + O_i$, just as in [6], [9], [5] where T_i is the local clock, α_i and O_i are the drift parameters that express the relative speed of the clock and the offset respectively, and t is the real time. Neighboring nodes exchange timestamps to estimate the best-fit offset line between them by using a recursive least squares (RLS) estimation approach.

IV. Pair-wise Time Synchronization between Neighbors

Consider two nodes i and j that can communicate directly, and want to determine their offset O_{ij} and skew α_{ij} by exchanging packets. By the offset $O_{ij}(t)$ we will mean the difference in the two clocks $T_j(t) - T_i(t)$. By the skew α_{ij} we will mean the ratio of the speeds $\frac{\alpha_j}{\alpha_i}$. At time $X(t_k)$, node i sends a packet $p(t_k)$ that includes this timestamp, its latest estimate of its skew with respect to j, $\hat{\alpha}_{ji}(t_k)$, and its latest estimate of the time difference between received and transmitted timestamps for packets from j to i, $\Gamma_{ji}(t_k)$. Packet $p(t_k)$ is received at time $U(t_k)$ by node j. Node j can make a Recursive Least Squares (RLS) estimate $\hat{\alpha}_{ij}(t_k)$:

$$\begin{aligned}\hat{\alpha}_{ij}(t_k) &= \mathrm{argmin}_\alpha \sum_{\mathrm{l}=-\infty}^{\mathrm{k}-1} \lambda^{\mathrm{l}-1-\mathrm{k}} [\,\mathrm{U}(\mathrm{t}_{\mathrm{l}+1}) - \mathrm{U}(\mathrm{t}_\mathrm{l}) \\ &\quad -\alpha\,[X(t_{l+1}) - X(t_l)]\,]^2\,. \end{aligned} \tag{1}$$

Then, using $\bar{\alpha}_{ij}(t_k) = \sqrt{\frac{\hat{\alpha}_{ij}(t_k)}{\hat{\alpha}_{ji}(t_k)}}$, the time at which the current packet should be received can be estimated using a window of N values of $U(t_k)$ and $X(t_k)$:

$$\hat{U}(t_k) = \frac{1}{N} \sum_{l=k-N}^{k-1} \left[U(t_l) + \bar{\alpha}_{ij}(t_k)\,(X(t_k) - X(t_l))\right]. \tag{2}$$

Then, we can determine the difference between the received and transmitted timestamps at nodes j and i, Γ_{ij}. This is the sum of the offset of node j with respect to i, O_{ij}, and the transmission delay of the packet, Δ_{ij}, at time t_k. It is estimated by substracting $X(t_k)$ from $\hat{U}(t_k)$:

$$\Gamma_{ij}(t_k) = O_{ij}(t_k) + \Delta_{ij} = \hat{U}(t_k) - X(t_k). \tag{3}$$

However, we need to first determine the value of Δ_{ij} so that we can estimate the value of $O_{ij}(t_k)$. In order to achieve this, packets containing relevant information about previously received packets must be sent back from node j to node i. Accounting for the change in offset between $\hat{U}(t_k)$ and S due to skew, $\Gamma_{ij}(S)$ is then computed as follows:

$$\Gamma_{ij}(S) = \Gamma_{ij}(t_k) + \left(S - \hat{U}(t_k)\right) \frac{\bar{\alpha}_{ij}(t_k) - 1}{\bar{\alpha}_{ij}(t_k)}. \tag{4}$$

Symmetrizing, transmission delay and offset are estimated:

$$\hat{\Delta}_{ji}(t_k) = \frac{\Gamma_{ji}(t_k) + \Gamma_{ij}(t_k)}{2}, \tag{5}$$

$$\hat{O}_{ji}(t_k) = \Gamma_{ji}(t_k) - \hat{\Delta}_{ji}(t_k). \tag{6}$$

V. Multi-hop Time Synchronization

Just for simplicity of presentation, suppose that there are n nodes all having clocks running at exactly the same speed, except that they have different offsets. With node 1 chosen as the reference, let z_i denote the amount that node i's clock is ahead of node 1. So $z_1 = 0$. Thus, if $t_i(t)$ denotes the time at clock i at real time t, then $t_i(t) = t_1(t) + z_i$ for all t. Let $x_{ij} := z_i - z_j$ be the offset between nodes i and j. Let N_i denote the neighbors of node i, and $|N_i|$ the number.

Assume that through experimentation as in Section IV, we obtain estimates $\hat{x}_{ij}$ of $x_{ij} = (z_i - z_j)$ for $j \in N_i$ for all i. The estimates will not be exactly equal to the true values. We will also suppose that

$$\hat{x}_{ji} \;=\; -\hat{x}_{ij} \text{ for } j \in N_i \text{ for all } i,$$

so that nodes i and j have talked to each other in arriving at this estimate (using the algorithm in Section IV).

The problem is this: We want to obtain estimates $v_i = \hat{z}_i$ of the offsets with respect to the reference node.

A. Formulation

Note that the true x_{ij}'s satisfy the global constraint

$$x_{i_1 i_2} + x_{i_2 i_3} + \cdots + x_{i_m i_1} \;=\; 0,$$

for every loop $\ell = ((i_1, i_2), (i_2, i_3), \ldots, (i_m, i_1))$ in the multihop network. However, estimates $\hat{x}_{ij}$ arrived at through only bilateral transactions need not satisfy such constraints.

By enforcing the large number of such constraints, one for each loop in the multi-hop network, the estimates $\{\hat{x}_{ij}\}$ can be smoothed and improved. In turn, this will lead to better time synchronization with respect to any chosen reference code. Moreover, we would like to perform such constrained estimation over the ad hoc network with the nodes acting in a completely distributed asynchronous manner. We will now develop an algorithm where each node only needs to asynchronously broadcast a few numbers. Each node updates its numbers through an extremely simple formula based on each received broadcast. This then will be shown to lead to an estimate of the offset with respect to any node which acts as a reference node.

Nodes will not need to know which is the reference node, or the topology of the network. No tree rooted at

the reference node needs to be constructed. The manner of changing from one reference node to another is also easy. The old reference node simply starts adapting, while the new one stops. The rest of the network automatically adapts to this change, and need not be explicitly informed of such reference handoffs.

Consider the network in Figure 1. Let A be the Node $\times$

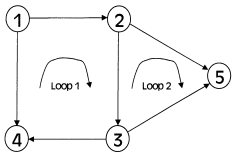

Fig. 1. Example of a network.

Arc matrix, called the incidence matrix:

A =

	(1,2)	(2,3)	(3,4)	(1,4)	(2,5)	(3,5)
1	+1	0	0	+1	0	0
2	-1	+1	0	0	+1	0
3	0	-1	+1	0	0	+1
4	0	0	-1	-1	0	0
5	0	0	0	0	-1	-1

where in the row corresponding to node i, we have an entry $+1$ for all arcs of the form $(i,*)$, an entry -1 for all arcs of the form $(*,i)$, and 0 otherwise. Let v_i denote the estimate of z_i. Then the estimate of the offset between nodes i and j is $v_i - v_j$, which is the inner product, $\langle (ij)$-th column of A, vector $v\rangle$. Hence, written as a vector of estimates of arc offsets, it as $A^T v$.

Thus we consider the least squares problem:

$$\text{Min}_v \|A^T v - \hat{x}\|^2.$$

The solution of this optimization problem is

$$A(A^T v - \hat{x}) \quad = \quad 0,$$

or

$$AA^T v \quad = \quad A\hat{x}. \tag{7}$$

Now let us consider the i-th row of A. It corresponds to node i. It has a ± 1 for every incident arc. Now the j-th column of A^T similarly has ± 1 for every arc incident to j. Hence $(AA^T)_{ii} =$ # of neighbors of $i = |N_i|$. Also,

$$\begin{aligned}(AA^T)_{ij} &= -1 \text{ if } j \in N_i \\ &= 0 \text{ if } j \not\in N_i.\end{aligned}$$

Thus the i-th entry of $AA^T v$ is

$$|N_i| v_i - \sum_{j \in N_i} v_j.$$

Also, the i-th entry of $A\hat{x}$ is the sum of terms of the form $+1$ times $\hat{x}_{i*}$, or (-1) times $\hat{x}_{*i}$, which in either case is $\hat{x}_{i*}$. Thus the i-th entry of $A\hat{x}$ is

$$\sum_{j \in N_i} \hat{x}_{ij}.$$

Hence (7) can be written as:

$$|N_i| v_i - \sum_{j \in N_i} v_j \quad = \quad \sum_{j \in N_i} \hat{x}_{ij} \text{ for all } i. \tag{8}$$

Note that this is deficient by at least one rank. So we set

$$v_1 \quad = \quad 0.$$

This corresponds to choosing node 1 as the reference node.

We will now consider the problem:

$$\text{Min}_v \sum_i \left(|N_i| v_i - \sum_{j \in N_i} v_j - \sum_{j \in N_i} \hat{x}_{ij} \right)^2. \tag{9}$$

We will solve this by coordinate descent since that will provide a fully distributed algorithm, in addition to being asynchronous. At the m-th iterate, let $v(m)$ be the estimate. We can take the initial iterate as

$$v(0) \quad = \quad 0.$$

Also, since node 1 is the reference, we also have

$$v_1(m) \quad = \quad 0 \quad \text{for all} \quad m \geq 0.$$

We will minimize over $v_i(k)$ perturbing it by δ_i to minimize the objective function (9), while keeping $v_j(k)$ invariant for $j \neq i$. Let

$$\begin{aligned} e_j \quad &:= \quad |N_j| v_j - \sum_{k \in N_j} (v_k + \hat{x}_{jk}) \\ &=: \quad \text{“Reported error of node j”}. \end{aligned}$$

Then

$$\left(|N_i|^2 + |N_i| \right) \delta_i + |N_i| e_i - \sum_{j \in N_i} e_j \quad = \quad 0.$$

So

$$\begin{aligned} \delta_i \quad &= \quad \frac{\sum_{j \in N_i} e_j - |N_i| e_i}{|N_i|^2 + |N_i|} \\ &= \quad \frac{1}{|N_i| + 1} \left[\frac{1}{|N_i|} \sum_{j \in N_i} (e_j - e_i) \right]. \end{aligned}$$

So the distributed algorithm is very simple: At each iterate, some node, say node i, changes its v_i to $v_i + \delta_i$, where

$$\begin{aligned} \delta_i \quad &:= \quad \frac{1}{|N_i| + 1} \left[\frac{1}{|N_i|} \sum_{j \in N_i} (e_j - e_i) \right] \\ &= \quad \frac{1}{|N_i| + 1} \left[\text{Average of } (e_j - e_i) \right. \\ &\qquad \left. \text{reported by its neighbors } j \right]. \end{aligned}$$

Thus, sporadically, each node i broadcasts (i, v_i, e_i). From this each node can calculate:

$$e_j \quad = \quad |N_j| v_j - \sum_{i \in N_j} (v_i + \hat{x}_{ji}).$$

It then adjusts its v_j to $v_j + \delta_j$. Then it rebroadcasts e_j, etc.

Now, if the clocks do not run at the same speed, then the product of the skews around a loop must be 1. So we can simply substitute $log(\hat{\alpha}_{ji})$ for $\hat{x}_{ji}$ and use the same approach.

One can actually use the simpler recursion to solve the optimization problem:

$$v_i = \frac{\sum_{j \in N_i} v_j + \sum_{j \in N_i} \hat{x}_{ij}}{|N_i|} \text{ for all } i. \quad (10)$$

However, at the time of writing of this paper, due to software errors, we have not finished implementing this algorithm and so results are not provided.

VI. Simulation Results

Before implementing the algorithms discussed, simulations were performed to verify if the behavior of the algorithms was as expected.

First, we simulated the pair-wise synchronization procedure with different values of λ and N (forgetting factor for the RLS algorithm and window size to estimate $U(t)$ respectively). We found through simulation with various values of λ and N that a value of $\lambda = 0.999$ and $N = 10$ provide the best estimation accuracy without requiring too much memory space.

Then, the skew of one of the nodes simulated was modified frequently within the ranges ($\pm 100ppm$) of the physical oscillator [13] found on the experimental testbed used during the course of the simulations in order to determine if the calculations of skew and offset adapt to the changes. We can see in Figure 2 that the skew estimate indeed adapts to the real skew as it changes. Also, in Figure 3 we see that the difference between the real and estimated offsets between the two nodes is below 2 μs.

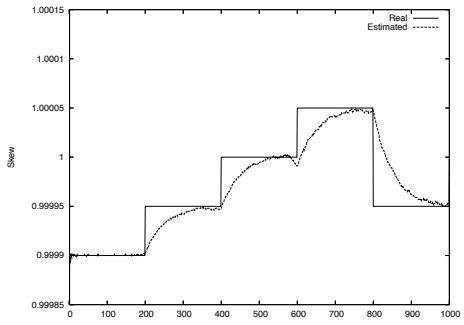

Fig. 2. Simulation results of pair-wise synchronization: Skew estimation.

VII. Implementation on Berkeley Motes

The Berkeley motes used in the implementation of the algorithms described in the previous sections were the MICA and MICA2 motes; see [14], [15], [16].

The communication stack on the Berkeley motes provides an understanding of the sources of error in the time synchronization process.

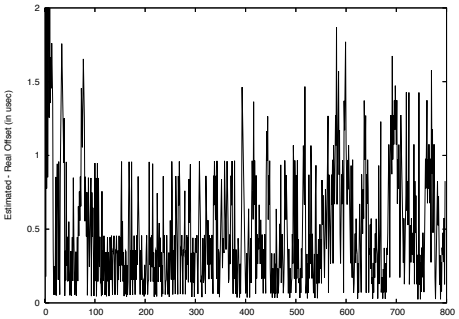

Fig. 3. Simulation results of pair-wise synchronization: Offset estimation.

- The message to be sent is constructed at the application layer and is then passed to the lower layers for its transmission. The time that it takes to construct the message and pass it down to the MAC layer is the *send time*. This is the time period starting at the point the Time Synchronization application calls sendMsg(),
- Once in the MAC layer, some time must elapse before the message is actually transmitted over the medium, since it has to wait until it has been determined that the medium is idle. This is the *access time*. This is the time period starting at the reception of the message at the RadioCRCPacket component,
- Once the bits have been received at the receiver, the message is reconstructed and passed up to the application layer. The time taken to do all this is the *receive time*. This is the time period starting at the reception of the first byte at the RadioCRCPacket component from the Radio Module,

Therefore, in order to reduce the sources of error in our implementation, we have modified the MAC layer of TinyOS to allow the timestamping of a packet at the time the first byte is sent, instead of doing it at the application layer. We also record the time at which the first byte is received by the MAC layer at the receiver side for later use by the application layer. In this way, we attempt to mitigate the send time, access time and receive time as sources of error, leaving only the propagation time, which is negligible in the case of the broadcast medium used in the Berkeley motes.

VIII. Pair-wise synchronization

For this first stage of testing, three motes were used for the implementation, the sender and receiver as described before, and a third mote connected to a PC through the serial port acting as a data collector or gateway. This latter mote continuously senses the medium and communicates the data being transmitted to the PC.

In order to get better accuracy, the motes' clocks were modified to generate a lower granularity clock, which is triggered by a crystal oscillator. The frequency of such crystals was modified in the MICA motes to 500 KHz. For this, the clock timer had to be changed since the original

timer (Timer 0) uses an 8-bit register to maintain the clock counter. This would overflow at this frequency and we use instead Timer 1 which has a 16-bit clock counter. These changes were problematic since Timer 1 is already used for other purposes by TinyOS, and the conflicts raised had to be solved in order for TinyOS and our application to work correctly. For MICA2 motes, a component to use a higher frequency of 921.6 KHz was made available by Maroti et al [17].

Although MICA motes were initially used for the implementation of the pair-wise synchronization, due to the lack of wireless reprogramming and support in newer versions of TinyOS, the subsequent experimentations were only performed on MICA2 motes.

With the implementation on MICA2 motes, we obtained an average accuracy of below $2\mu s$ (with a worst-case accuracy of $6\mu s$) as can be seen on Figure 4. This is better than the accuracy obtained in related works, with the exception of [9]. It has similar results as ours since we use the same timestamping technique to eliminate most of the sources of error that appear when we try to synchronize nodes in a network through message exchanges [18].

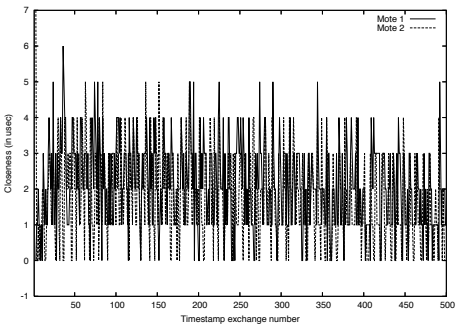

Fig. 4. Accuracy of estimated to real time between 2 neighboring node.

IX. Multihop time synchronization

The algorithm was implemented on a 40-node network where different grid topologies were enforced by software. That is, although all nodes operated on the same cell, packets were filtered out according to the multihop topology desired, i.e., as in MAC filtering.

As with the implementation of the pair-wise synchronization algorithm, we had a mote acting as a gateway to the PC that continuously collected the packets transmitted by the motes in the network. Each node sends a time synchronization message to its neighbors, with a time interval randomly selected between 25 and 45 seconds so we can reduce the probability of collisions. The gateway mote requests the reference time each mote has estimated, every 15 seconds. To avoid collisions and errors in the measurements, instead of each mote responding to the queries made by the gateway mote immediately, they store the information regarding their estimates of the time at the reference, at the time each query is received on the external 512 KBytes flash EEPROM. The complete information was collected once the experiment was finished, so we could determine the accuracy of the algorithm by comparing the estimate the reference mote had made at each of the queries, which are identified using a sequence number, with the estimate that each of the other motes had at that same query.

For a particular grid of 4x10 motes, which results in a 9-hop network, we show the minimum, maximum and average accuracy obtained between the estimated time and the actual time at the reference across the whole network in Figure 5. The variant shown is that where the averaging is done over neighbors at a fewer number of hops from the reference node.

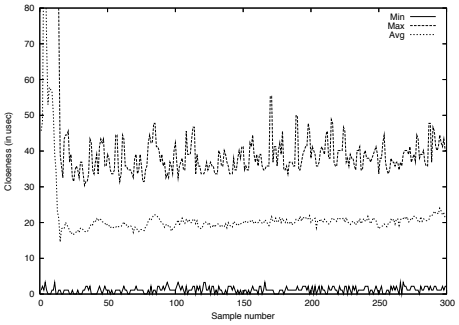

Fig. 5. Average closeness of estimated to real time in a 40-node network.

Such results are quite similar to the ones reported for FTSP in [9]. So taking into account that leading research groups on sensor networks, such as the ones in the University of Virginia and Ohio State, use FTSP, we set to compare the behavior of this algorithm with FTSP, by running it on the exact same topology. FTSP nodes send time synchronization messages every 30 seconds, and our gateway mote queries the global time that every mote has estimated every 15 seconds.

The minimum, maximum and average accuracy obtained between the estimated time and the actual time at the reference across the whole network through FTSP are shown in Figure 6.

Also plotted in Figure 7 is the behavior of the average accuracy as well as the standard deviation for each of the 9 hops in the network. We can observe that our algorithm provides better accuracy, and the estimates have smaller standard deviation, than the estimates obtained using FTSP in the same network. The statistics were computed starting at query 100 so they are not affected by the initial state, which can not be compared exactly since our algorithm starts with the reference node already determined, while FTSP goes through an election process to determine what node is the reference, i.e., the node with smaller ID.

X. Conclusions

We have presented a clock synchronization protocol that (i) is tailored to wireless networks, (ii) can exploit the broad-

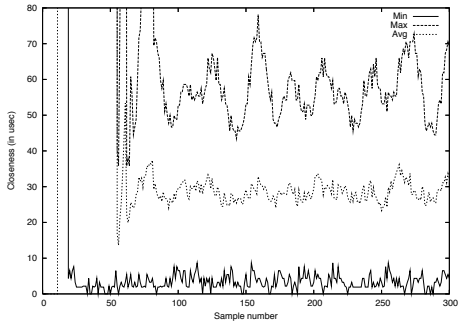

Fig. 6. Experimental results of FTSP on the 40-node network

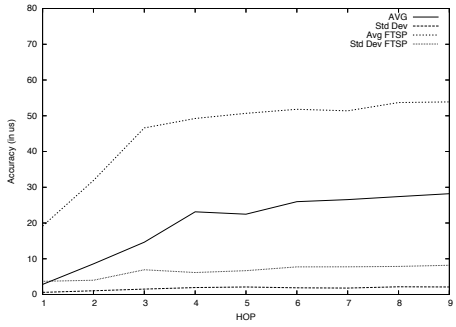

Fig. 7. Comparison to FTSP

cast nature of the wireless medium, (iii) is fully distributed, (iv) uses asynchronous messages, and (v) requires no topological constructions such as rooted trees. The algorithm exploits global constraints required for a common estimate of time, to improve accuracy in large multihop networks. It has been implemented on Berkeley Mica2 motes testbed. We have shown that as the number of hops grows our algorithm presents a better comparative behavior as one expects from the construction of the algorithm.

REFERENCES

[1] Scott Graham and P. R. Kumar, "The Convergence of Control, Communication, and Computation," *Proceedings of PWC 2003: Personal Wireless Communication, Lecture Notes in Computer Science*, vol. 2775/2003, Springer-Verlag, Heidelberg, Germany, pp. 458–475, 2003.

[2] Kurt Plarre and P. R. Kumar, "Object Tracking by Scattered Directional Sensors," *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 3123–3128, Seville, Spain, Dec. 12–15, 2005.

[3] R. Rozovsky and P. R. Kumar, "SEEDEX: A MAC Protocol for Ad Hoc Networks," *Proceedings of The 2001 ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2001*, Long Beach, CA, pp. 67–75, October 4-5, 2001.

[4] K. Römer, "Time synchronization in ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing*. 2001, pp. 173–182, ACM Press.

[5] Scott Graham and P. R. Kumar, "Time in general-purpose control systems: The control time protocol and an experimental evaluation," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Bahamas, Dec. 2004, pp. 4004–4009.

[6] M. Sichitiu and C. Veerarittiphan, "Simple, accurate time synchronization for wireless sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 16–20 2003.

[7] S. Adlakha S. Ganeriwal, R. Kumar and M. B. Srivastava, "Network-wide time synchronization in sensor networks," Tech. Rep., UCLA, 2003.

[8] J. Elson and K. Römer, "Wireless sensor networks: A new regime for time synchronization," Tech. Rep., UCLA, July 2002.

[9] Miklos Maroti, Gyula Simon, Branislav Kusy, and Akos Ledeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, Baltimore, MD, USA, Nov. 2004, pp. 39–49.

[10] R. Karp, J. Elson, D. Estrin, and S. Shenker, "Optimal and global time synchronization in sensornets," Center for Embedded Networked Sensing, University of California, Los Angeles, Tech. Rep., April 2003.

[11] J. Elson, L. Girod and D. Estrin, "Finegrained network time synchronization using reference broadcasts," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, USA, Dec. 2002, pp. 147-163.

[12] Arvind Giridhar and P. R. Kumar, "Distributed Clock Synchronization over Wireless Networks: Algorithms and Analysis," *Proceedings of the 45th Conference on Decision and Control*, San Diego, Dec. 13–15, 2006.

[13] "Ecs oscillators, inc," http://www.ecsxtal.com.

[14] Jason Hill, Robert Szewczyk, Alec Woo, Hollar Hollar, David Culler, and Kristofer Pister, "System architecture directions for networked sensors," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 93–104, Nov. 2000.

[15] "MICA2 Motes," http://www.xbow.com/Products/productsdetails.aspx?sid=72.

[16] Jason Hill and David Culler, "A wireless embedded sensor architecture for system-level optimization," Tech. Rep., UC Berkeley, 2001.

[17] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi, "The flooding time synchronization protocol," Tech. Rep., Vanderbilt, 2004.

[18] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Transactions on Computers*, vol. C-36, no. 8, pp. 933–940, 1987.