Wireless Sensor Networks: A Perspective to Time Synchronization

Neeraj Jain
Computer Science and Engineering, Mewar University, Chittorgarh, India
Email: njain108@gmail.com

Dr. Manju Mandot
Associate Professor J.R.N.Vidyapeeth, Udaipur, India
Email: manju.mandot@gmail.com

Abstract- Wireless Sensor Networks is getting lot of attention by the research community. It consists of small devices distributed over geographical areas and each device has sensing, computing and communicating components. Time synchronization targets to equalizing the local times for all nodes in the network. It has been observed from the literature survey that all the nodes suffer from a problem named clock drift. This problem causes clock difference among nodes as time goes because the processors do not run exactly at the same speed. To overcome this problem a number of techniques have been proposed. TPSN (Timing-Sync Protocol for Sensor Networks) is one of the effective protocols proposed to synchronize sensor networks. In this paper, we propose improvements over TPSN to synchronize nodes in a wireless sensor network more effectively with a lower message complexity and higher precision.

Keywords: Wireless Sensor Network, Time Synchronization, Clustering, RBS, TPSN

I INTRODUCTION

As the advances in technology have enabled the development of tiny, low power devices capable of performing sensing and communication tasks, sensor networks emerged and received high attention of many researchers. Sensor networks are special type of ad-hoc networks, where wireless devices (usually referred as nodes in the network) get together and spontaneously form a network without the need for any infrastructure. Because of the lack of infrastructure, such as routers in traditional networks, nodes in an ad-hoc network cooperate for communication, by forwarding each other’s packets for delivery from a source to its destination. Such special type of ad-hoc networks, sensor networks have limited energy sources, high density of node deployment, cheap and unreliable sensor nodes. As in all distributed systems, time synchronization is an important component of a sensor network. Time synchronization in a computer network aims at providing a common time scale for local clocks of nodes in the network. Since all hardware clocks are imperfect, local clocks of nodes may drift away from each other in time, hence observed time or durations of time intervals may differ for each node in the network. For the time critical applications running on WSN, clock drift problem should be reduced to a reasonable level or completely eliminated if possible. However, for many applications or networking protocols, it is required that a common view of time exists and is available to all nodes in the network at any particular instant.

A network which has two nodes may have two clocks labeled with same frequencies, and these clocks will not be running at exactly the same speed due to various reasons. This minute difference increases with the time and after a while if there is no interference to synchronize these nodes, clocks will have significant differences. Other than clock drift, some nodes can be added to the network later or some nodes might be started later then other nodes in the network. In such cases, there would be a fixed clock difference offset. This problem which needs to be solved. [1]

Sensor nodes are very tiny instruments and running with a limited energy so it is not easy to synchronize nodes effectively because of energy consumption. As they have not much processing power so any complex algorithm cannot run on WSN. [2]

A number of time synchronization protocols exists due to varying requirements, such as degree of mobility or precision. Time synchronization procedure basically is a message exchange containing the timestamp and the measurement of delay. There are three basic solutions for time synchronization in sensor networks [3]:

1) Receiver-Receiver Based Synchronization
2) Sender-Receiver Based Synchronization
3) Delay Measurement Time Synchronization

Receiver-Receiver based synchronization algorithms commonly use one-way message exchange such as in the Reference Broadcast Synchronization (RBS) [4]. On the other hand, two-way message exchange is used in Sender-Receiver based synchronization protocols, such as the Timing-sync Protocol for Sensor Networks (TPSN) [5]. There are also some synchronization protocols based on one-way message exchange as well as the measurement of delay. An example of such a protocol is Delay Measurement Time Synchronization
II BACKGROUND

Ganeriwal et.al. proposed a network-wide time synchronization protocol for sensor networks, which they call Timing-Sync Protocol for Sensor Networks (TPSN) [8]. Their protocol works in two phases: “level discovery phase” and “synchronization phase”. The aim of the first phase is to create a hierarchical topology in the network, where each node is assigned a level. Only one node is assigned level 0, called the root node. In the second phase, a node of level i synchronizes to a node of level i-1. At the end of the synchronization phase, all nodes are synchronized to the root node and the network-wide synchronization is achieved. TPSN is a Sender-Receiver based time synchronization protocol for WSN. The two main steps to synchronize the network are described as follows:-

A. Pair-wise Synchronization

Let’s have 2 nodes i and j which have been synchronized and node i starts the synchronization. Here are the steps to synchronize i to node j:

1. Node i creates a synchronization pulse packet and hands over packet to the operating system and network stack for transmission.
2. Just before transmission, the packet is time-stamped with T1 and delivered to the medium. This prevents uncertainties of network stack and medium access delay.
3. After propagation delay and packet transmission time, the packet will be delivered to node j. Then it is time-stamped with T2 immediately to prevent uncertainty again.
4. Node j then prepares a synchronization acknowledgment packet and hands it over to the operating system and network stack.
5. Like in step 2, just before transmission, the packet is time-stamped with T3 and delivered.
6. Node i receives the packet and timestamps again with T4 for last time.
7. Finally, node i calculates the clock offset and fixes its clock.

The calculation is simple and can be computed very fast by the nodes. Let’s say there is an offset O and it needs to be calculated when reply package comes. Assuming there is no timestamping uncertainties, the formula is just like below:

\[ O = \frac{(T_2 - T_3) - (T_4 - T_3)}{2} \]  \hspace{1cm} (1.1)

After computing the offset, the clock difference is known between node i and node j. It depends on the implementation to change local clock using the offset or keep the offset in a separate place and use it when needed.

B Network-wide Synchronization

TPSN builds a spanning tree where each node knows the level of itself and the parent. Level 0 is assigned to one node which is named as root node. This node has the responsibility to build tree by triggering level discovery phase. To start the tree construction, root node sends a level discovery packet with its level 0 in it. When all the one hop neighbors receive this packet, they set their level to 1, parent to 0 and send another level discovery packet with the level 1 in it. However, before sending another packet, each node waits for random time to avoid collisions. This process is done for all other nodes. In summary, when a node receives a level discovery packet, it sets its level to the level one more in the packet and sets the sender node as its parent.

After a period of time, if a node could not get any level discovery packet, it sends a level request packet. This also occurs when sync requests fail. When its neighbors receive this level request message, they reply the message with their level in it. Then the node sets its level to the minimum received level + 1 and sets it parent to the sender of minimum level.
Synchronization is done for each node periodically. Each node sends a sync pulse packet to its parent periodically to be synchronized. When a node cannot get a synchronization acknowledgment packet from its parent, that may mean that its parent is dead therefore it sends a level request packet to set a new level itself [9].

III IMPROVEMENTS

In order to analyze improvements done over TPSN, a simulation of a wireless network with 50 nodes is prepared using ns2simulator. All nodes are located in a 1,000 by 1,000 units area randomly using uniform distribution. To generate clock drifts for nodes, normal distribution is used with parameters $\mu=1$ and $\sigma^2=0.1$. This will be used as a reference result before synchronization is done. Simulation duration will be 10,000 seconds and periodic synchronization interval will be 100,000 seconds.

A. Reference TPSN Results

In this phase, TPSN is directly applied to the network. The improvement in synchronization can be seen easily. However, there is still a considerable clock difference on the graph. Decreasing the interval will improve the synchronization but it will also increase the message load on the system. Therefore, we will look into ways to increase precision by not increasing message traffic too much.

B. Clustered Results

In this method, a spanning tree which is clustered using a level depth 4 will be used. This means node level 0-3 will be in a cluster, level 4-7 will be in another cluster and so on. This clustering algorithm simply assigns a group number to a node using its level and the depth number 4 is arbitrarily selected. Any other depth value or clustering algorithms could also be used. As an exception, root node will be left alone as another cluster at the top. Clustering itself is not enough for an improvement in synchronization. Because TPSN has only one hop message traffic, there needs to be a change in behavior between cluster synchronization. For clusters, root nodes of the clusters will be used in synchronization which means child cluster root sends a message for synchronization to parent root leaf. That leaf transfers the request to its parent and others also do this until the message reaches the parent cluster root node. That parent cluster root node then sends the time information back using the same path to the requested node in the child cluster. For the child and root node, there is no change in the TPSN behavior. For intermediate nodes, it is more reasonable to update intermediate node clocks also because there will be no message overhead. However, this needs temporary package modifications at intermediate levels.

1. Chain Synchronization: Instead of leaving intermediate node clocks unaffected, it is possible to update all the nodes on the path between cluster roots. To understand the working of chain synchronization we take Node $i$ is the child, requesting synchronization. Node $j$ is the intermediate node and node $k$ is the root to be synchronized. When node $i$ starts the synchronization, it asks for the time to its parent, which is node $j$. Node $j$ answers this request packet with acknowledge packet, however, using the chain synchronization, forwards the synchronization request to node $k$.

Before forwarding the request packet, node $j$ has three important jobs to do.

(i) Store requester node’s address: Because TPSN tree structure is directed and only the children know their parent. Therefore node $j$ stores node $i$ as requester node.

(ii) Store requester’s sent time: This is crucial for TPSN offset calculation and it will be used when sending acknowledge packet to node $i$.

(iii) Replace requester’s sent time in the packet: To update intermediate node $j$’s clock, new TPSN packet must be prepared. Therefore, requester’s sent time information is replaced with the node $j$’s sent time as required by TPSN.

When node $k$ receives the forwarded packet, it is just a TPSN packet from node $j$. So, node $k$ replies request with its time information. When node $j$ receives the packet, this packet can be used to update node $j$’s clock and so, node $j$ updates it clock. However, after updating the clock, node $j$ replaces requester’s sent time information with the original one, which is stored before forwarding the request and forwards acknowledge packet to node $i$. With this packet, node $i$ updates its clock correctly.

The assumption here is that TPSN only uses 4 timestamps to synchronize the clocks so the time that passes on the parent node logically is not important. It can be seen that node $j$’s request forwarding and acknowledge waiting times can be considered as node $j$’s internal process. In this way, what is done here is actually similar to TPSN functionality.

C. Chain Synchronization Results

Now the chain synchronization is performed for cluster root synchronization. This operation can be done for all nodes but if it is implemented in the same way as done for clusters, nodes with low level will be too busy for synchronization, especially the root. That might reduce precision because they cannot answer synchronization requests when doing another chain synchronization. It is
possible to do chain synchronization for all request by implementation but that’s not wanted because it will also increase message traffic a lot. Therefore, a simple solution may prevent this problem. If a node is in chain synchronization state and cannot answer child node’s chain synchronization request, then it can reply request just like in TPSN. No chain request is required. TPSN like behaviour is kept here when an intermediate node is busy with chain synching. The message overhead here is minimum. Such results may be obtained with reference TPSN with reduced interval

D. Adaptive Interval Results

Some nodes have a bigger drift and some nodes have smaller one. So, it is not an optimized solution to use same intervals for all the nodes. Therefore, we gradually increase or decrease interval using the last synchronization results. 1% can be a reasonable step value. Results can be momentarily big or small so gradual changes is more reliable at this point. After some time, there will be a network which is optimized for message load. It is obvious that, if the expected value for clock difference is kept small, than it will increase the messages required. Thus acceptable average clock difference should be carefully selected. As a result, the total message count is reduced to the same value as available in reference TPSN.

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WSN - COMPARISON TABLE</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Reference</td>
</tr>
<tr>
<td>Clustered</td>
</tr>
<tr>
<td>Chained</td>
</tr>
<tr>
<td>Adaptive</td>
</tr>
</tbody>
</table>

CONCLUSION

As TPSN produces a simple and effective way of time synchronization in the WSNs, the improvement methods to TPSN proposed in this paper provides reasonable precision improvements with low message overheads. Table I provides a comparison of all of these methods.

First improvement was making network hierarchy clustered and doing chain synchronization between cluster nodes. As seen in the table, both average clock difference and the standard deviation have decreased significantly. But the message traffic has increased slightly which is expected. Second enhancement was applying chain synchronization to all nodes. In this method, all nodes try to chain synchronize at first. If it was not possible to synchronize in chain, parent node replies the request as similar to what reference TPSN does. Now, the synchronization was improved again significantly with a small message overhead using the clustered TPSN.

At last we removed extra message overhead. However, it was shown that it also enhanced the synchronization results. The reason behind this might be the balanced message traffic. The choice of adaptive interval offset parameter is very important and any improper value can degrade network performance. Thus we can say that all the methods above provided significant improvements over TPSN and they can be implemented easily using TPSN, or any sender-receiver based synchronization protocol.

REFERENCES