1. TIMING-SYNC PROTOCOL FOR SENSOR NETWORKS (TPSN)

1.1. Introduction

Time synchronization is a method that allows network nodes to synchronize their internal clocks either with each other or with a third source. There are two types of time synchronization: Sender – Receiver (e.g. TPSN, FTSP), and Receiver – Receiver (e.g. RBS). In the former case the nodes exchange timestamped packets, in order to determine the relation between their own internal clocks. In the latter case, all the network nodes synchronize their internal clocks with the clock of an external node, which transmits timestamped packets.

Each scheme has its own advantages and disadvantages, since their performance depends heavily on the type of the MAC layer. More specifically, the Receiver – Receiver approach performs better, when the radio and the driver are in a black box, which is the case in wireless LANs. On the other hand, the Sender – Receiver approach performs better, when there is a strong coupling between the application and the radio, which is the case in sensor networks. This thesis focuses on the first protocol that was proposed for the Sender – Receiver scheme, which is TPSN.

1.2. Protocol description

TPSN is a protocol for instant time synchronization. More specifically, it is used, in order to calculate the propagation delay and the current (instant) clock drift between 2 nodes. The way that TPSN works is the following:

a) At time T1 node A sends a packet to node B

b) At time T2 node B receives the packet. T2 is equal to T1+PROGATION_DELAY+ CLOCK_DRIFT

c) At time T3 node B transmits a packet back to node A

d) At time T4 node A receives the packet. T4 is equal to T3+PROGATION_DELAY- CLOCK_DRIFT

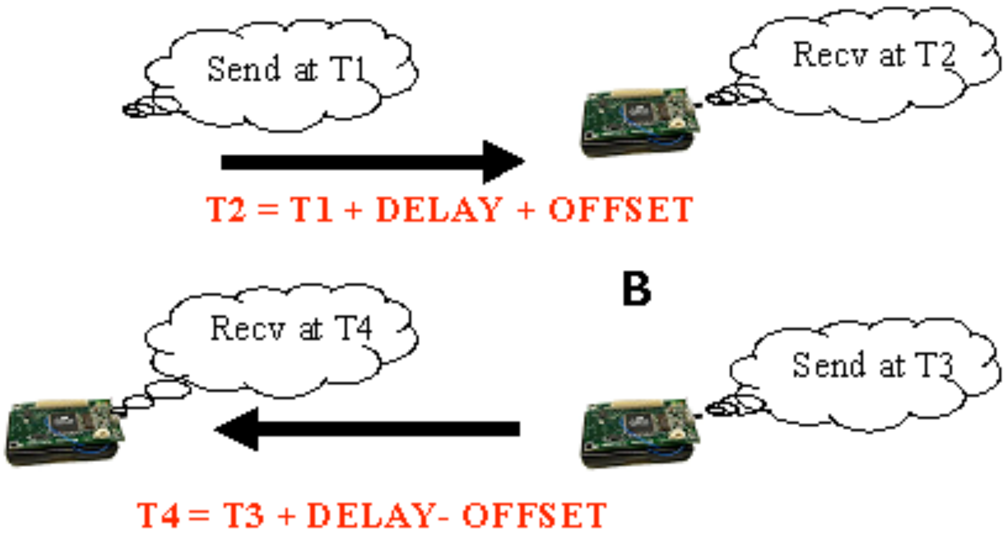

Figure 1.1 TPSN packet exchange

After this packet exchange (Figure 1.1), if node A knows all the times T1 – T4, then it will be able to calculate the above mentioned values using the equations:

$$\text{CLOCK\_DRIFT} = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}$$

$$PROPAGATION\_DELAY = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$

1.3. Implementation

Let’s consider the case, in which node A wants to synchronize its clock with node B. TPSN needs to be running as a module in both nodes (Figure 1.2). In the beginning, the application, which needs to synchronize with node B, sends a message of type MSG_GET_INSTANT_TIMESYNC to the TPSN module and passes a tpsn_t struct. The data of the message contains the following fields:

Data type	Field name	Description
unsigned char	mod_id	Module id of the application
unsigned char	msg_type	Return message type, which will be used to inform the application of the response
unsigned short	node_id	Id of the node, with whom time synchronization is requested
int	clock_drift	Variable, in which the clock drift will be stored, after TPSN finishes

Table 1.1 Format of tpsn_t struct

The whole data structure is stored in a buffer and will be used, in order to send the reply to the application. Afterwards, at time T1, the TPSN module sends a TPSN request packet to the node, which was specified by the application. This packet needs to be timestamped by the SOS network stack; therefore, its type is MSG_TIMESTAMP. The packet’s contents are:

Data type	Field name	Description
unsigned int	time[0]	Time at the parent (filled by the network stack at the parent). It corresponds to the value of T1,

		which was described above.
unsigned int	time[1]	Time at the child (filled by the network stack at the child). It corresponds to the value of T2, which was described above.
unsigned char	type	Set to TPSN_REQUEST
unsigned char	seq_no	Sequence number of the requests

Table 1.2 TPSN REQUEST packet fields

The node receives the TPSN request at time T2 and sends back to the previous node a TPSN reply at time T3. The type of this packet is also MSG_TIMESTAMP and its format is:

Data type	Field name	Description
unsigned int	time[0]	Time at the parent (filled by the network stack at the parent). It corresponds to the value of T3, which was described above
unsigned int	time[1]	Time at the child (filled by the network stack at the child). It corresponds to the value of T4, which was described above.
unsigned char	type	Set to TPSN_REPLY
unsigned char	seq_no	Sequence number of the response. It needs to have the same value as the sequence number of the corresponding request.
unsigned int	previous_time[0]	It is set equal to T1, which was obtained from the corresponding TPSN request packet.
unsigned int	previous_time[1]	It is set equal to T2, which was obtained from the corresponding TPSN request packet.

Table 1.3 TPSN REPLY packet fields

Finally, the initial node receives the TPSN reply at time T4 and it is able to calculate the clock drift and the propagation delay using the equations:

$$\text{Propagation delay} = \frac{(\text{previous\_time}[1] - \text{previous\_time}[0]) + (\text{time}[1] - \text{time}[0])}{2}$$

$$\text{Clock drift} = \frac{(\text{previous\_time}[1] - \text{previous\_time}[0]) - (\text{time}[1] - \text{time}[0])}{2}$$

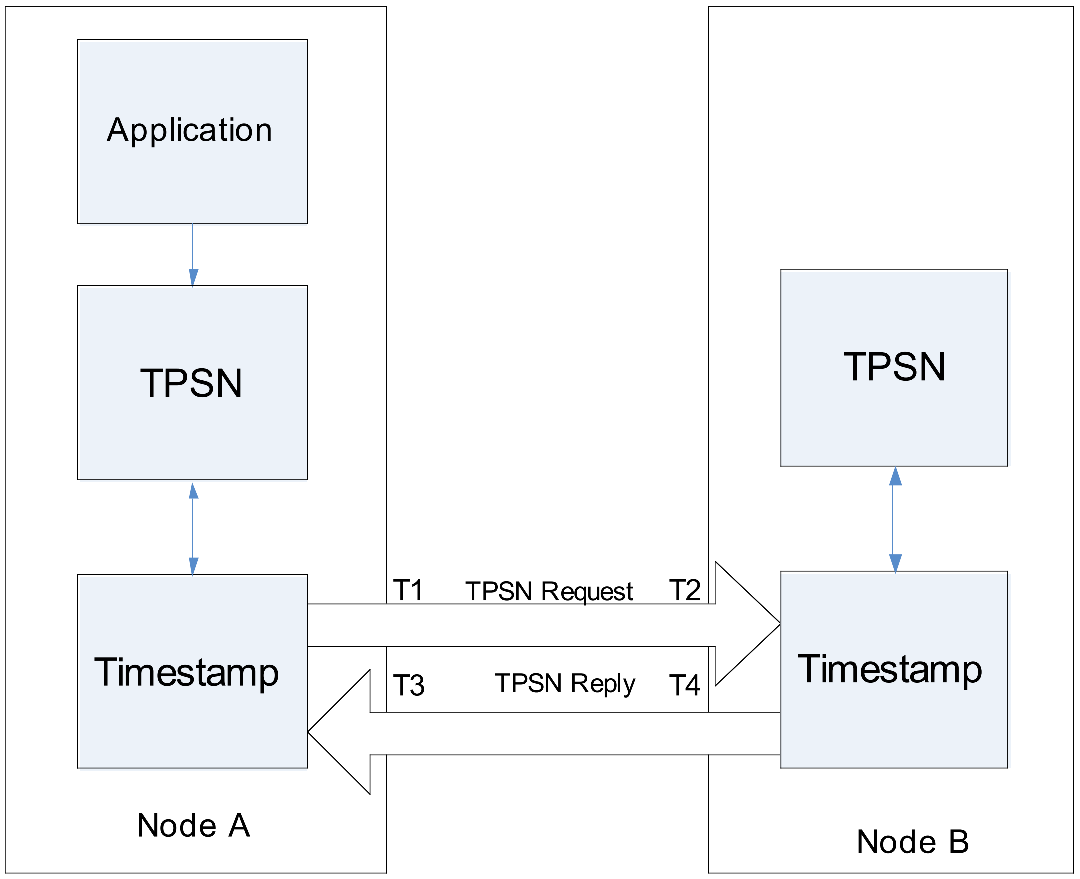

Figure 1.2 TPSN Architecture

1.4. TPSN internal buffer

As it was mentioned above, all the data structures with the time synchronization requests that are sent by various modules are stored in a buffer. The TPSN module uses this buffer, in order to pass the reply back to the requesting module. The data is removed from the buffer in a random order (i.e. when a TPSN reply is received) and we also need to take care of the case, where the buffer is full and we need to overwrite one of the previous entries. In order to satisfy both of these requirements we have implemented a modified version of a LRU (Least Recently Used) algorithm. Each position of the buffer has a counter, which is initialized to BUFFER_SIZE (by default set equal to 7).

Whenever a new request is inserted, the counter from all the inserted items is decremented. A counter with value equal to zero corresponds to the oldest item. If the new request is a duplicate of an old one, then the old request is overwritten and the counter of the corresponding position is to BUFFER_SIZE. This way each application can send a request for time synchronization and start a time-out timer. If the timer expires, before an answer is received, then the application can retransmit the same request, without adding additional burden to the TPSN module. If there is no empty position in the buffer and the request is not a duplicate one, then the oldest item of the buffer is overwritten and the corresponding module is notified using the message type that it requested. However, in the data that is returned, the msg_type is set to REPLY_REQUEST_OVERWRITTEN and the clock_drift is set to 0.

In addition, when a TPSN reply is received, the buffer is searched, in order to find the corresponding request. If the request is not found, then the reply is discarded, otherwise the request is extracted from the buffer. However, in order to be consistent with the scenario of the addition to the buffer, we also need to increment all the counters that were smaller than the counter of the request that is about to be removed from the buffer. This way, we'll always be able to write a non-duplicate packet either to a free position or to an occupied position, which has a counter equal to 0.

Finally, it should be noted that all the buffers that are passed to the TPSN module are owned by the corresponding applications. The TPSN module does not free any of the buffers, therefore the application needs to take care of any possible memory leaks.

1.5. Fail-safe mechanism

In order to increase the security of the protocol, we have added a sanity check on the value of the propagation delay. More specifically, according to statistical measurements, the values of the propagation delay for the mica2 motes follow a Gaussian variation, therefore the true delay will be within the window [AVG_PROPAGATION_DELAY – SIGMA, AVG_PROPAGATION_DELAY + SIGMA] with a confidence of 99.97%, where AVG_PROPAGATION_DELAY is approximately equal to 1230µsec and SIGMA is equal to 0.32µsec. Therefore, in order to avoid various kinds of attacks, if the propagation delay is not within that window, the value is discarded. The application is notified using the message type that it requested, however in the data that is returned the msg_type is set to REPLY_INVALID_PROPAGATION_DELAY and the clock_drift is set to 0.