Packet-level time synchronization

TEP:	TBA
Group:	Core Working Group
Type:	Documentary
Status:	Draft
TinyOS-Version:	> 2.1
Author:	Miklos Maroti, Janos Sallai
Draft-Created:	15-May-2008
Draft-Version:	1.0
Draft-Modified:	2008-05-15
Draft-Discuss:	TinyOS Developer List <tinyos-devel at mail.millennium.berkeley.edu>

Abstract

This TEP describes a packet-level time synchronization mechanism that allows for sending a time value along with the packet which is automatically converted from the sender's local time to the receiver's local time by the communications stack.

1. Introduction

Time of occurrence of events is often of interest in a sensor network. Maintaining a synchronized UTC or a virtual global time in a sensor network may, however, lead to significant communication overhead and may not always be required by the application.

This TEP describes a packet-level time synchronization mechanism that allows for sending a time value along with the packet which is automatically converted from the sender's local time to the receiver's local time by the communications stack. Packet-level time synchronization is limited to single-hop communication and does not provide synchronized network time. It provides a simple yet powerful abstraction, on top of which it is possible to implement higher-level time synchronization services (e.g. FTSP[6])in a platform-independent way. Packet-level time synchronization is semantically equivalent to the ETA primitives[1].

The rest of this TEP specifies:

- Platform-independent packet-level time synchronization interfaces
- How these interfaces are provided in the HIL
- A guideline how each transceiver's HAL may implement the above interfaces

2. Interface

Packet-level time synchronization is implemented by the communication stack and is exposed through two interfaces, `TimeSyncAMSend` and `TimeSyncPacket`.

The `TimeSyncAMSend` interface allows for sending a time value (e.g. an event timestamp) along with a message. It is parameterized by the precision and width of the time value:

```
interface TimeSyncAMSend<precision_tag, size_type>
{
  command error_t send(am_addr_t addr, mes-
sage_t* msg, uint8_t len, size_type event_time);
  command error_t cancel(message_t* msg);
  event void sendDone(message_t* msg, error_t error);
  command uint8_t maxPayloadLength();
  command void* getPayload(message_t* msg, uint8_t len);
}
```

The `send` command sends a regular message just like `AMSend.send`[2], but it also performs sender-receiver time synchronization. The `event_time` parameter holds the time of some event as expressed in the local clock of the sender. The receiver can obtain the time of this event (expressed in its own local time) via the `TimeSyncPacket` interface.

The rest of the functionality is identical to that of the `AMSend` interface, therefore its description is omitted here. Please refer to[2] for details.

The `TimeSyncPacket` interface, parameterized by a precision tag and width, allows for retrieving a time value that was sent along the received packet:

```
interface TimeSyncPacket<precision_tag, size_type>
{
        command bool isValid(message_t* msg);
        command size_type eventTime(message_t* msg);
}
```

The `isValid` command returns `TRUE` if the value returned by `eventTime` can be trusted. Under certain circumstances the received packet cannot be properly time stamped, so the sender-receiver synchronization cannot be finished on the receiver side. In such case, this command returns `FALSE`. This command MUST be called only on the receiver side and only for messages transmitted via the TimeSyncAMSend interface.

The communications stack MUST guarantee that if the `isValid` command called from within the `receive` event handler returns `TRUE`, then the value returned by the `eventTime` command can be trusted. However, it might be possible that the local clock overflowed more than once or that is was stopped or reset since the packet was received, which causes the `event_time` to be invalid. The `isValid` command MAY return `TRUE` in such situations, and it is the responsibility of the user of the interface to ensure that the clock runs freely from the time of message reception to the time when `eventTime` is called. To avoid this issue, it is recommended that `isValid` and `eventTime` are called from the `receive` event handler.

The `eventTime` command should be called by the receiver of a packet. The time of the synchronization event is returned as expressed in the local clock of the caller. This command MUST BE called only on the receiver side and only for messages transmitted via the `TimeSyncAMSend` interface.

3. HIL requirements

The signature of the platform's ActiveMessageC[5] MUST include:

```
provides interface TimeSyncAMSend<TMilli, uint32_t>;
provides interface TimeSyncPacket<TMilli, uint32_t>;
```

where event times are given in the node's local time, which is available through `HILTimerMilliC.LocalTime`.

The communications stack MAY support timestamp precisions and widths other than `TMilli` and uint32_t, respectively. Also, alternative `TimeSyncAMSend` and `TimeSyncPacket` implementations MAY use clock sources other than `HILTimerMilliC.LocalTime`.

4. Implementation guidelines

Packet-level time synchronization employs the ETA primitives. In this TEP, only the basics of the time synchronization mechanism are described, for details please see[1]. This section presents two possible implementation approaches. The first approach assumes that the payload of the packet is still mutable when the transmission time of the packet (e.g. the timestamp of the SFD interrupt) becomes available. The second approach avoids this assumption and uses the packet timestamping functionality described in TEP[4] to implement packet- level time synchronization.

4.1 Approach #1

Several transceivers allow for modifying the contents of a packet after packet transmission is started. Packet-level time synchronization can be implemented very efficiently on such platforms.

Transmitter's story

- When the communications stack services a `TimeSyncAMSend.send` command called with event timestamp `t_e`, it stores `t_e` (e.g. in a map with the pointer of the message_t as key) and sets the designated timestamp field in the packet payload to `0x80000000`.
- When the packet starts being transmitted over the communication medium, a corresponding hardware event is timestamped (e.g. an SFD interrupt). Let us denote this transmission timestamp with `t_tx`. The difference of event timestamp `t_e` and transmit timestamp `t_tx` is written into the designated timestamp field in the payload of the packet (typically into the footer, since the first few bytes might have been transmitted by this time). That is, the information the packet contains at the instance when being sent over the communications medium is the age of the event (i.e. how much time ago the event had occurred).
- If an error occurs with timestamping the transmission or with writing the package payload after transmission has started, then the designated timestamp field in the packet payload will contain `0x80000000`, indicating the error to the receiver.

Receiver's story

- The packet is timestamped with the receiver node's local clock at reception (e.g. with the timestamp of the SFD interrupt). Let us denote the time of reception with `t_rx`. The reception timestamp is stored in the metadata structure of the `message_t`[5].
- When the event time is queried via the `TimeSyncPacket` interface, the `eventTime` command returns the sum of the value stored in the designated timestamp field in packet payload and the reception timestamp, i.e. `e_t- e_tx+e_rx`. This value corresponds to the time of the event in the receiver's local clock.
- The `TimeSyncPacket.isValid` command returns `FALSE` if the time value stored in the payload equals `0x80000000` or if the communications stack failed to timestamp the reception of the packet. Otherwise `TRUE` is returned, which indicates that the value returned by `TimeSyncPacket.eventTime` can be trusted.

4.1 Approach #2

If a particular platform does not support changing the packet contents after the synchronization event (start of transmission, SFD interrupt, etc.) had occured, it is still possible to provide packet-level time synchronization functionality at the cost of some communication overhead. Such an approach can rely on packet timestamping TEP[4] to implement packet-level time synchronization.

Transmitter's story

- When the communications stack services a `TimeSyncAMSend.send` command called with event timestamp `t_e`, it stores `t_e` (e.g. in a map with the pointer of the message_t as key) and sends the packet.
- Transmission of the packet is timestamped using the packet timestamping TEP[4] mechanism. Let us denote this transmission timestamp with `t_tx`. The difference of event timestamp `t_e` and transmit timestamp `t_tx` is sent in an auxilliary packet. That is, the information the auxulary packet contains is the age of the event at the time when the initial packet was transmitted.

Receiver's story

- The packet is timestamped with the receiver node's local clock at reception (e.g. with the timestamp of the SFD interrupt). Let us denote the time of reception with `t_rx`. The reception timestamp is stored in the metadata structure of the `message_t`[5].
- When the auxilliary packet arrives, the time value it carries (`t_e-t_tx`, the age of the event) is stored in a metadata field of the main packet. The auxilliary packet is discarded, and the receive event is signalled with the pointer to the main packet.
- When the event time is queried via the `TimeSyncPacket` interface, the `eventTime` command returns the sum of the value stored in the metadata (age of the event) and the reception timestamp, i.e. `e_t- e_tx+e_rx`. This value corresponds to the time of the event in the receiver's local clock.
- The `TimeSyncPacket.isValid` command returns `FALSE` if the communications stack failed to timestamp the reception of the packet. Otherwise `TRUE` is returned, which indicates that the value returned by `TimeSyncPacket.eventTime` can be trusted.

5. Reference implementation

A reference implementation of the packet-level time synchronization mechanism described in this TEP can be found in `tinyos-2.x/tos/chips/rf230`.

6. Author's Address

Miklos Maroti
Janos Sallai
Institute for Software Integrated Systems
Vanderbilt University
2015 Terrace Place
Nashville, TN 37203
phone: +1 (615) 343-7555

7. Citations

[1] Kusy, B., Dutta, P., Levis, P., Maroti, M., Ledeczi, A., Culler, D., Elapsed Time on Arrival: A simple and versatile primitive for canonical time synchronization services. International Journal of Ad hoc and Ubiquitous Computing, Vol, 2, No. 1, 2006.

[2] TEP 116: Packet protocols

[3] TEP 102: Timers

[4] TEP TBA: Packet timestamping

[5] TEP 111: message_t

[6] Maroti, M., Kusy, B., Simon, G., and Ledeczi, A. 2004. The flooding time synchronization protocol. In Proceedings of the 2nd international Conference on Embedded Networked Sensor Systems (Baltimore, MD, USA, November 03 - 05, 2004). ACM SenSys '04.