Packet timestamping

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>

Note

This memo documents a part of TinyOS for the TinyOS Community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited. This memo is in full compliance with TEP 1.

Abstract

This TEP describes a mechanism that provides access to the time of transmission and time of reception of a packet. The local clocks of the sender and recipient are used to timestamp the transmission and reception of the packet, respectively.

1. Introduction

Time of packet sending and reception is often of interest in sensor network applications. Typically, neither the time of invocation of the send command, nor the time of signaling of the sendDone event can be used to estimate, without significant jitter, the time when the packet was transmitted. Similarly, the time of occurrence of the receive event cannot be used to reliably estimate the time of reception.

A straightforward way of message timestamping is to use the start-of-frame delimiter interrupt, commonly exposed by packet-oriented radio transceivers. This approach was taken by the CC2420 radio stack in TinyOS 1.x: the SFD interrupt handler was exposed by the radio stack as an asynchronous event. This solution was problematic, because higher- level application components that wired the interface containing this event could break the timing of radio stack due to excessive computation in interrupt context.

This TEP overcomes this issue by providing a standardized, platform- independent interface to access packet timestamps without exposing timing critical and/or hardware-specific events. Also, this TEP does not prescribe how packet timestamping should be implemented: it only describes the interfaces and the required functionality (semantics).

2. The PacketTimeStamp interface

This TEP specifies a standard interface (`PacketTimeStamp`) to access the packet transmission and packet reception times. The sender and the receiver use unsynchronized clocks to timestamp packets. The precision and width of timestamps is specified as interface parameters `precision_tag` and `size_type`:

```
interface PacketTimeStamp<precision_tag, size_type>
{
        async command bool isValid(message_t* msg);
        async command size_type timestamp(message_t* msg);
        async command void clear(message_t* msg);
        async command void set(message_t* msg, size_type value);
}
```

The `timestamp` command of the `PacketTimeStamp` interface is an accessor to the the timestamp. The `timestamp` command returns the time of transmission after a sendDone event, and the time of reception after a receive event.

In some cases, it is not possible to timestamp certain packets (e.g. under very heavy traffic multiple interrupts can occur before they could be serviced, and even if capture registers are used, it is not possible to get the time stamp for the first or last unserviced event). The `PacketTimeStamp` interface contains the `isValid` command to query if the packet timestamp is valid.

The communications stack MUST guarantee that if the `isValid` command called from within the `sendDone` or `receive` event handler returns `TRUE`, then the value returned by the `timestamp` 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 timestamped, which causes the value returned by the `timestamp` command 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 `timestamp` is called. To avoid this issue, it is recommended that `isValid` and `timestamp` are called from the `receive` or `sendDone` event handler.

The clear command invalidates the timestamp: after clear is called, `isValid` will return `FALSE`. A `set` command is also included to allow for changing the timestamp associated with the message. After the `set` command is called, `isValid` will return TRUE.

The communications stack guarantees that the transmission timestamp and the reception timestamp that belong to the same packet transmission always correspond to the same physical phenomenon, i.e. to the same instance of physical time. This TEP does not prescribe what synchronization event the communications stack should use. For example, the communications stack may chose to timestamps hardware events that correspond to the start of transmission/reception of the packet, signaled a start-of-frame delimiter (SFD) interrupt. The SFD interrupt occurs at the same time on the transmitter and the receiver (assuming that the signal propagation delay is negligible). Alternatively, on a byte oriented radio, the timestamp may correspond to the average of the transmission times of bytes, as described in[2].

3. HIL requirements

The signature of the platform's ActiveMessageC[3] MUST include:

```
provides interface PacketTimeStamp<TMilli,uint32_t>;
```

where timestamps are given in the node's local time, which is available through `HILTimerMilliC.LocalTime`[4].

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

4. Implementation

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

5. 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

6. Citations

[1] TEP 111: message_t

[2] 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.

[3] TEP 116: Packet protocols

[4] TEP 102: Timers