

Packet Link Layer

TEP: 127
Group: Core Working Group
Type: Documentary
Status: Draft
TinyOS-Version: 2.x
Author: David Moss, Philip Levis

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 the behavior and interfaces of the TinyOS 2.x Packet Link Layer. The Packet Link Layer is an optional radio stack layer responsible for the reliable delivery of a packet from node to node. Packet Link provides error correction functionality found in Layer 2 of the OSI model¹.

1. Introduction

In wireless networks, packets are regularly dropped between point to point communications due to interference or RF range. Correcting these dropped packets requires the transmitter to first recognize that the packet was not acknowledged, and then retransmit that packet.

Retransmitting the packet adds its own set of issues. Specifically, the receiver will receive duplicate packets due to its own dropped acknowledgements. Logic is therefore required to allow receiver to recognize and dump duplicate packets, but only after the acknowledgement has been sent back to the transmitter.

With these two pieces of functionality: the ability for a transmitter to keep retrying until it gets an acknowledgement and the ability for a receiver to accept only a single copy of the packet, a Packet Link Layer can be formed to reliably deliver a single packet from point to point.

2. Design Considerations

2.1 Time spent retrying a delivery

Some networks have different timing characteristics than others. Consider a person carrying a wireless device while walking in and out of RF range of another node. In this case, the device may want to retry the transmission a few times over the course of a few seconds before declaring the delivery unsuccessful. A robust Packet Link Layer should allow the developer to specify the amount of time spent retrying as well as the number of retries to send.

2.2 Setting the packet sequence number

To detect duplicate packets, a sequence number byte can be used within the packet to verify against previously received packets. If the source address and sequence number of a newly received packet matches that of a previously received packet, then the newly received packet is a duplicate and may be dumped. To prevent the packet sequence number byte from changing on each retransmission, the byte should be set at or above the Packet Link Layer and never changed below.

2.3 False Acknowledgements

The low levels of a radio stack or the radio chip itself are typically responsible for generating packet acknowledgements. It has been observed in some platforms that the radio chip will generate false auto-acknowledgements. This occurs when the radio receives the packet and sends an acknowledgement but the microcontroller never gets the message due to some error. In this case, the transmitter would believe the packet got to the destination successfully but the receiver would have no knowledge that a packet was received. Software initiated acknowledgements prevent this issue by removing the possibility of false acknowledgements.

3. Interface

3.1 PacketLink Interface

The TEP proposes this PacketLink interface::

```
interface PacketLink {
    command void setRetries(message_t *msg, uint16_t maxRetries);
    command void setRetryDelay(message_t *msg, uint16_t retryDelay);
    command uint16_t getRetries(message_t *msg);
    command uint16_t getRetryDelay(message_t *msg);
    command bool wasDelivered(message_t *msg);
}
```

PacketLinks interface functions:

```
setRetries(message_t *msg, uint16_t maxRetries)
```

- Sets the maximum number of times to retry the transmission of the message

```
setRetryDelay(message_t *msg, uint16_t retryDelay)
```

- Set the delay, in milliseconds, between each retransmission

```
getRetries(message_t *msg)
```

- Returns the maximum number of retries configured for the given message

```
getRetryDelay(message_t *msg)
```

- Returns the delay, in milliseconds, between each retransmission for the given message

```
wasDelivered(message_t *msg)
```

- This command may be called after the sendDone() event is signaled. It is the equivalent of `PacketAcknowledgements.wasAcked(message_t *msg)`, so is only a helper function to make the PacketLink interface easier to use.

3.2 Expected Behavior

The PacketLink interface is accessed by the application layer before the packet is passed to the radio stack for transmission. The application layer will call `setRetries(...)` and `setRetryDelay(...)`, passing in the message it is about to send.

The interface MUST configure metadata for the packet to specify the maximum number of retries and the amount of delay between each retry. When the send process reaches the Packet Link Layer, it MUST automatically check the packet's metadata and retry sending the packet as previously configured.

For example, to configure a packet to be retried a maximum of 50 times over the next 5 seconds, the developer would execute the following commands before sending the packet::

```
call PacketLink.setRetries(msg, 50);
call PacketLink.setRetryDelay(msg, 100);
```

By placing a 100 ms delay between each retry and allowing up to 50 retries maximum, the maximum amount of time the message will be transmitted is (50 * 100) ms, or 5000 ms.

When transmitting a packet configured for reliable delivery to the broadcast address, the Packet Link Layer SHOULD allow the packet to be retried. This will let the transmitter know that at least one node received the message.

4. Author's Address

David Moss
Rincon Research Corporation
101 N. Wilmot, Suite 101
Tucson, AZ 85750

phone - +1 520 519 3138
email ? dmm@rincon.com

Philip Levis
358 Gates Hall
Stanford University
Stanford, CA 94305-9030

phone - +1 650 725 9046
email - pal@cs.stanford.edu

5. Citations

¹ "OSI model" http://en.wikipedia.org/wiki/OSI_model