Porting IEEE 802.15.4 to TinyOS

Buffer management
TinyOS has no dynamic memory allocation, so management of primitives is left up to the application programmer. This also means that any assignment of buffer space for 802.15.4 primitives must happen at the top side of the interface.

External structures / message payload
On some of the primitives, variable length members exist. This includes the data request and data indication primitives, where the message payload (msdu) must be specified. Either the variable length members can be allocated directly in the primitives, or they can be allocated externally and pointed to by the primitive members.

If the message payload is allocated directly in the primitives, problems may arise if trying to build a high level stack upon the 802.15.4: Either the message payload must be copied in to place when generating a request primitive, or the high level stack must account for the surrounding members of the request primitive. Similar restrictions apply for indications.

If the message payload is allocated externally, the above problems can be handled. However, in the case of data indications, a problem arises: As all buffer management happens at application level, the application needs to specify where the MAC should put the next msdu. One solution is to let the application fill out the msdu pointer of the indication primitive before passing it to the MAC. This is an implicit requirement, though, and may confuse application programmers.

Request and confirm primitives
Requests are initiated at the application side of the interface. All requests, except SYNC, give rise to a confirm event. The application must provide the 802.15.4 with enough buffer space for both the request and confirm primitives. However, if a request is always followed by a confirm, it makes sense to use buffer sharing between request and confirm primitives. This can be implemented by creating unions of the request-confirm pairs. A confirm primitive is then provided in the original request buffer. In the event of SYNC request, a confirm event must be generated, but the contents of the buffer passed is not important.

Buffer sharing has the advantage of less buffer memory usage. However, it may imply more buffer copying, because the contents of a request primitive is overwritten by the confirm primitive.

Indication primitives
Indication events are initiated at the MAC side of the interface. Thus, the buffer for
the primitive must be assigned prior to the indication event. TinyOS has a tradition for *buffer swapping* where the initial buffer is allocated at the MAC side. Whenever the event occurs, the application is expected to return a pointer for the next buffer to use.

Buffer swapping can be applied to the indication interface. However some penalties are implied:

- The indication event is expected to happen in an asynchronous context. The application should return a new pointer as soon as possible to not miss subsequent indication events. Should an application fail to return from the indication event in good time, events may be lost even though buffer space is still ample.
- 802.15.4 has 8 different indication primitives of varying size. Either the MAC should keep buffers ready for each of the 8 possible indications, or it should apply buffer sharing for a single buffer large enough to hold the largest primitive. In the first case, memory usage is higher than necessary. In the latter case, the application must always return a buffer large enough for the largest indication primitive, no matter what indication event was signalled. This implicit requirement is potentially confusing for the application programmer. Also, the latter case rules out using pointers for external structures in the primitives, as discussed previously.

An alternative to buffer swapping is to define an indication primitive request event that is signalled prior to the actual event. The application is then expected to provide a buffer large enough for the indication to be signalled. This saves the buffer swapping penalties outlined above, but complicates the interface a bit. Note also that this solution does not rule out application initialization of external structure pointers.

**Response primitives**

Responses do not have any event counterpart. However, the application must know when it is safe to use the response primitive buffer space for other purposes. Thus, a response done event is introduced to pass the buffer back to the application.