On CAN Priority, MTI Structure and Software Layout

OpenLCB has several characteristics that have to be balanced:

- Small node implementations: In particular, nodes can't require large amounts of memory to do unbounded buffering of traffic.
- Multiple master and slave nodes: Nodes may receive requests from multiple other nodes at the same time, which have to be successfully handled.
- CAN protocol: CAN arbitration puts strict limits on which nodes can send when.

Consider a node X which receives addressed requests from node A, B and C, and replies with a global message. This could happen as separate exchanges:

req(A→X) →
        ← resp(X→A)
req(B→X) →
        ← resp(X→B)
req(C→X) →
        ← resp(X→C)

where the arrows are shorthand to make the direction of communications clearer, and the (A→X) means that it's from node A to node X. If A, B and C try to send these messages at the same time, CAN arbitration or delays in node X may mean that B and C sends their request before X can send its response:

req(A→X) →
req(B→X) →
req(C→X) →

How is X to handle this? There are several approaches.

1. Provide buffering in X that can remember all the requests, and eventually serve them by sending replies. Alternately, provide buffering of all the responses to A, B and C, so that they can sit in the node until CAN arbitration allows them to be sent. The amount of buffering required is hard to estimate, because there can be multiple protocols taking place in an overlapped manner, and there's no clear limit on the number of requesting nodes.
2. In some cases, such as sending global responses to global requests, multiple replies are not required by the protocol. This is valid:
   req(A) →
   req(B) →
   req(C) →
           ← resp(X)
   The node can merely set a bit that indicates a reply is needed, instead of using a buffer for each request, and send the reply when it eventually can. The race conditions in the code have to be carefully considered, but this method implements the protocol without requiring unbounded buffering. It doesn't work with addressed messages, because the node has to buffer (at least) the reply destination address.
3. Make sure that X can always promptly send its response by ensuring that the frame format gives X's response higher priority than the request. Then X's response can always be sent before the request from B or C arrives and has to be processed. While that response is being sent, CAN

arbitration holds off the next request.

Solutions 2 and 3 are much more desirable than 1 in the long run, because unbounded buffering is just a problem waiting to happen.

Currently (July 5, 2012), our frame format and MTI selection doesn't fully support 2 and 3. Although the 16-bit MTI has an priority field that could help with solution 3 above, there are several problems[1]:

- For directed messages, the MTI priority field for addressed messages is carried in the CAN data segment, where it doesn't effect CAN arbitration.
- Even if they could be moved back into the CAN header, several of the MTIs don't actually have the right ordering.

Short of redesigning everything, what are the options? There are at least two.

1) Use the reserved high-order bit in the frame.

The high-order bit is currently reserved, send as 1 (low priority), accept any. Setting this to 0 on a frame that's being sent allows it to be sent immediately, so long as no other node does the same thing. So a node could implement solution 3 by setting this bit on response frames that need to be sent quickly, with high priority, before the next response comes in.

We'd imagined using this bit for e.g. gateways that need to be able to forward arriving traffic onto CAN to prevent their internal buffers from filling up. That use would just be generalized to allow other nodes to do that too.

The biggest problem in writing a standard-form description for this is to specify when the bit can't be set. It loses its utility if every node sets it every time.

Depending on the internal software structure of the node, it might also be hard to use this. "Set the bit only when needed" requires the node software to know some things that can only be determined at run-time, which requires code to keep track of it.

2) Modify the MTI coding

Right now, there's a field in the CAN header that determines the frame type:

1 The original frame format had the full MTI and source address in the CAN header, which prevented the problems being discussed here. In the difficult conversations around a common frame format and the supposed need for hardware filtering, the destination address was moved into the header and parts of the MTI were moved to the data segment. With that, the MTI ordering became less important and wasn't given enough attention.

Frame Type	Meaning
0	Unaddressed MTI
1	(Reserved)
2	Datagram complete in frame
3	Datagram first frame
4	Datagram middle frame
5	Datagram final frame
6	Addressed MTI other than datagram or stream
7	Stream Data

Table 1: Existing MTI Type Values in the CAN header field

The “reserved value” is a high priority message, only lower than global messages. We could use that for high-priority responses that need to be sendable before the lower-priority responses:

Frame Type	Meaning
0	Unaddressed MTI
1	High-priority addressed MTI other than datagram or stream
2	Datagram complete in frame
3	Datagram first frame
4	Datagram middle frame
5	Datagram final frame
6	Low-priority addressed MTI other than datagram or stream
7	Stream Data

Table 2: Proposed new MTI Type Values

To make this work transparently for MTIs to be defined in the future, the bit assignments inside the 16-bit MTI should be redone:

1) Move the MSB of the MTI priority field out of the 8 bits that go into the data segment, and into the top nibble. That can then be used to decide whether to send it with “1” or “6” in the frame's type value.
2) Revisit the priority coding of e.g. the PIP and SNIP replies so that their MTIs have the right priority order.

The advantage of this is that it'll let properly coded nodes solve the problem. The disadvantages are more numerous:

- It's programming and documentation work at a busy time. But when better to do it?
- It doesn't solve the issue for globals, though solution 2 above will handle those
- It consumes our last type-field expansion code