Written by J. Day, D. Harris, B. Jacobsen, and A. Shepherd

OpenLCB Explanation	
OpenLCB-CAN Frame Transfer	
10/07/10	

1 Introduction

This explanatory note contains informative discussion and background for the corresponding "OpenLCB CAN Frame Transfer Specification". This explanation is not normative in any way.

The Frame Transfer layer of the OpenLCB-CAN stack lives above the CAN Physical Layer, and below the Message Network layer. It is responsible for ensuring the reliable transport of the CAN frames that make up OpenLCB-CAN messages. CAN provides reliable transport between any two nodes on a CAN segment within limitations:

1. Every CAN header must be unique by construction. Collisions between frames with identical headers do not result in well-defined behavior, and must be avoided by design to avoid intermittent problems.
2. CAN does not provide a "link up" or "link down" notification. Nodes may come and go from a CAN segment at any time, and on an individual basis. In general, one can't assert that a particular node is always present, always comes up first, or never comes up first.
3. CAN does not provide any indicator of which node sent a particular frame. Any node can send any frame, and the receiving nodes have no CAN-provided mechanism for identifying the source.
4. On a busy segment, CAN frames are sent in a strict priority order, with the priority determined by the content of the frame headers. If multiple nodes have frames to send at the same time, the highest priority (lowest numerical value header) frame will be sent first. There is no concept of "reserved bandwidth" except through frame priority, and low-priority frames may require significant time to be sent.
5. Nodes must be able to accept frames at the full CAN rate. Hardware filters must be usable to help with that, but even then nodes may receive adjacent frames addressed to them.

Putting a unique source ID field in each frame ensures that the frames are unique (item 1 above), and provides a frame-level indicator of which node sent the frame for the purposes of monitoring, debugging, etc (item 3). Using a distributed algorithm to determine the value for that ID field prevents needing a single "manager" to provide it, thereby handling item 2. The header bit fields discussed below are organized to handle items 4 and 5.

OpenLCB-CAN is required to work with standard CAN components, e.g. bridges and computer adapters must not require customization to operate with OpenLCB. This provides a stringent requirement on protocol design, in that OpenLCB-CAN cannot require specific timing, deliberate creation of error cases or specific error handling.

People will occasionally refer to a restriction against having seven 1 bits in the top 11 bits of the CAN header. There is no such restriction, and the statements are the result of a misunderstanding. There is a restriction against having seven consecutive recessive bits on the CAN segment, but sending seven consecutive 1 bits will not result in this. CAN uses a bit-stuffing technique to prevent that by inserting a dominant bit on the line after the fifth consecutive recessive bit, and removing it at the receiver.

2 Annotations to the Specification

This section provides background information on corresponding sections of the Specification document. It's expected that two documents will be read together.

2.1 Introduction

2.2 References and Context

Note that we reference the Node ID specification, but only to the extent that each node is required to have a unique Node ID. The Specification would work equally well with a Node ID that was allocated via any mechanism, and with any length from 2 to 8 bytes.

2.3 Frame Format

OpenLCB uses a very vanilla transfer of frames, no special features, to reduce complexity and increase robustness.

Standard header frames are also known as CAN 2.0A frames. Extended header frames are also known as CAN 2.0B frames.

The specification is silent on the uses for standard (11-bit header) frames. The proper-operation requirement is so nodes will tolerate them in case they're needed for other purposes. The Atmel bootloader () uses standard frames, for example, and the Microchip bootloader () can use them.

This could also have been phrased as “shall ignore ...”, but the current phrasing is thought to be more exact.

We don't use RTR because CAN semantics require a specific use for it, which has been built into some silicon implementations. There are also some arbitration issues, see: CiA Application Note 802 (2005) and http://www.thecanmancan.com/?tag=rtr for more information.

We don't use overload frames because not all CAN controller hardware can deliberately send them. We require that nodes be able to handle them because some CAN controller hardware will occasionally send them automatically.

Overload frame: “Due to internal conditions, the node is not yet able to begin reception of the next message. A node may generate a maximum of two sequential overload frames to delay the start of the next message.” (That's 17 to 23 extra bit times each) (from http://rs232-rs485.blogspot.com/2009/11/can-bus-message-frames-overload.html) Image: http://3.bp.blogspot.com/_ycHwJEosotY/SvoMDJMGF7I/AAAAAAAAA64/3Ql4_UQnoBg/s1600/Overload%2BFrame.JPG

The CAN format has been allocated on nibble boundaries to make it easier to e.g. read dumps of packets. The header is considered to be right (LSB) aligned.

The reserved first bit (MSB) is likely to be used as a priority-boost bit in the future, e.g. so that a gateway can gain priority access to the CAN segment for various operations that require synchronization or atomicity. CAN encodes “do first” priority as 0 and “do later” as 1, so the specification requires that a 1 bit be sent. To preserve the utility of this, nodes must not require either a 1 nor 0 on receipt.

The order of these fields is chosen to get proper priority and access disambiguation via CAN's standard mechanisms.

Putting the destination alias in the header allows filtering on it with common CAN hardware.

2.4 CAN-specific Control Messages and Interactions

2.5 States

In the Inhibited state, a node can only communicate on the CAN segment to allocate its Node ID alias. This involves sending CIM and RIM messages with a tentative Node ID alias value. Once that process is complete, the node has an assigned Node ID alias and can transfer to the Permitted state, where all communications are possible.

If a node fails, it should restart in the Inhibit state.

You can infer the state of a node from RIM/CIM traffic. CIM means the node is in inhibited state, tough of course you might not know what node it is. If you're keeping a table of associations between Node IDs and aliases, you should remove this entry.

2.6 Control Message Format

RIM, CIM, etc are assigned low numbers to give them higher transmission priority.

2.7 Interactions

Communications protocols are about more than messages. The protocols must also specify the interactions in which the messages are used.

2.7.1 ID Alias assignment

For example, four CIM frames and a RIM:

0x17123FED

0x16456FED

0x15789FED

0x14ABCFED

0x10700FED

For node ID 12 34 56 78 9A BC

… MMM is the message sequence number, 0x7 to 0x4 (to give priority to messages later in the process)

Note that there is no requirement that this alias be consistent from one run to the next. More detail is on a separate page.

CIM/RIM message content and format

The CIM/RIM algorithm for finding aliases is designed to work without any CAN errors. Arbitration conflicts are OK, but deliberate errors are not, because different CAN implementations may deal with them differently. During execution of the algorithm, some frames might have the same header. They therefore have to have the same contents, or else a non-arbitration error can occur.

Although it might be a useful debugging tool, putting the full NID in the data portion of the frames can cause errors. Therefore, we decided not to do that.

2.7.2 ID Alias validation

Regular nodes won't need to use this. It provides a way for a node joining a segment to rapidly learn the Node IDs and Node ID aliases of the reachable nodes. This may be useful for e.g. a gateway that is connecting two segments that are already in operation, or a monitor node who wants to determine that all nodes have come up properly.

In order to acquire the map between CANids and NIDs, a gateway needs to be able to send a CAN frame requiring "everybody reply with your NID". Much like Ethernet gateway mapping, this needs to return the NID of just the specific CAN-attached hardware, not all the NIDs that can be e.g. reached through a gateway, so it needs to be a segment-specific message defined only at the CAN level. It must not be a OpenLCB common message.

2.7.3 ID Alias termination

Regular nodes won't need to use this.

Mapping Reset - indicates that the local alias in the source address field may no longer be associated with the same NID, and mapping tables should be reset.

Gateways will also have to obtain unique aliases for remote nodes they are proxying on to the segment.

Gateways maintain the mapping between remote node's NID and local alias. If they need to break that mapping, e.g. because they need to reuse the local alias due to resource limitations in the mapping tables, they must send a "Mapping Reset" message to force nodes to drop their alias information. *"Mapping Reset" is a CAN-specific message limited to the segment, but all gateways on the segment must act on it.*

Gateways may not be able ensure permanent validity for alias to remote NIDs. For example, if they have a limited routing table, they may need to reuse local alias. Before reusing them, they have to send a "Mapping Reset" frame to drop references to the old NID, followed by a "Initialization Complete" when the alias is allocated to a new NID.

3 Load-Related Frame Synchronization

CAN networks exhibit a behavior called “load-related frame synchronization” (it goes by other names).

Because of the CAN priority structure, the network can run at high utilization rates. Under those circumstances, all nodes attempt to send their highest-priority frame, and the node with the absolute highest priority frame (first dominant bit) wins. All other nodes wait until just after that frame and try again.

This behavior greatly increases the probability that nodes with frames ready to send containing the same header will see a collision.

Consider 5 nodes each of which want to send the high-priority frames 1,...,5, plus one more node A that wants to send the low priority frame 15. If all those nodes attempt to send at once, the frames will arbitrate out and will be sent in the order 1,2,3,4,5,15. If another node, B, also wants to send a frame 15, and starts to send it at any point during that entire sequence, it will end up colliding with the 15 that is being sent by A.

Particularly during layout startup, when lots of nodes have frames to send, this can result in cascades of errors that prevent proper operation unless the protocols have been designed to have as few header collisions as possible, and no data collisions.

4 Throughput and Temporarily Deaf Nodes

OpenLCB requires that CAN-attached nodes be able to handle the full frame rate on the CAN bus. There is no guarantee that frames for a given node will arrive with non-zero time between then.

Some nodes are not always able to process frames. For example, the node may cease processing for a short time while writing non-volatile memory. Higher-level protocols, e.g. configuration write datagrams, have mechanisms built in to prevent overrun while writing configuration memory, but there could be multiple activities going on in parallel that will result in frames arriving while the node is not processing.

Short outages can be covered by CAN hardware buffering, so long as the node will eventually catch up even at full arrival rate. Outages long enough that frames are lost due to e.g. buffer overflow require node to broadcast that it's back up if hardware detected frame lost. This is because tehre could have been frames of some form that modified the OpenLCB node state, which are now lost (e.g. drop alias, CIM/RIM in absence, or even higher level events)

29-bit extended header and 8-byte payload results in a total transmission taking about 135 bit times.

125Kbps with max-length extended header frames is about 900 frames/second. CAN is very good at running at 100% utilization, so long as nodes can keep up and a proper set of priorities is in place.

CAN is designed to run at 100% usage without problem, and it's routine for CAN segments in other contexts to run at 100% for extended times. Therefore, the CAN network does not require any inter-frame spacing, limitations on bandwidth usage, etc; the segment can be allowed to run at 100%.

The real limitation is whether the nodes themselves can handle the full rate of frame arrival. There is (currently) no mechanism in OpenLCB to throttle the rate of frames arriving at a node, nor is it easy to create one in a CAN network. The stream and datagram protocol have been designed to allow a node to efficiently use a very limited amount of buffer memory, but don't provide mechanisms to limit the arrival of frames.

For long term reliability, a node must be able to completely process an entire CAN frame in the time it takes to receive the next one, which may be as little as 42 (?) bit times.

Eventually may have to allow use of Overload Frames to throttle for e.g. NVRAM writing.

Bit 0	Bits 1	Bits 2-16	Bits 17-28
Priority: 0 high, 1 low	Frame Type 1: OpenLCB Message 0: CAN Control Message	Variable Field	Source NID Alias
0x1000,0000	0x0800,0000	0x07FF,F000	0x0000,0FFF
Solo top bit	Top bit of 6th nibble from right	3 bits lower bits, then three nibbles	Right-most three nibbles

The "Variable field" has different meanings depending on the "Frame Type" field. CAN control messages are identified with a Frame Type bit of 0, to ensure high priority.

5 On the choice of a 12-bit alias length

How big should the NodeID alias field be on CAN links?

- Smaller (fewer bits) allows more payload and/or simpler coding of other parts of messages.
- Larger allows more unique nodes to be accessed.

Issues

Addressing

The Node ID size limits the number of unique nodes that can take part in communications on the CAN segment. Because Node ID aliases are assigned independently on each CAN segment, the only issue is how many different nodes are involved, not which ones they are or what pattern(s) are available in their address numbers.

For electrical/timing reasons, a segment itself can only support a maximum of about 50 nodes, but communication also involves nodes off the segment. For example, chaining N CAN segments via bridges will generally make up to 50*N nodes available. We have a use case of chaining small numbers of CAN segments together, implying the need to address 500 or 1000 nodes.

As another example, a very large network might connect many CAN segments via TCP/IP or other networked connections. Each CAN-attached node might need to communicate with a larger number of others spread across that network.

(CAN backbone case)

In some cases, want 2 aliases (source and destination), plus a few more bits, in 29 bits total. 29-(2*12) = 5 is a pretty compelling choice.

Collisions during Allocation

(not talking CAN arbitration here, but two nodes trying to use same alias) The allocation process resolves collisions (attempt to use an already allocated Node ID alias), but that takes time. Minimizing the number of collisions is good. If the address space is just the size of the number of things you want to address, the collisions get hard /slow to resolve across multiple nodes. This points to having a factor of 2, at least, between the number of nodes to be addressed and the size of the address space.

(Mostly talking about separate nodes here, not just one big interface node: Points to CAN backbone case)

Size

Reducing size most important (so far) in the stream protocol.

Events and Datagrams can be transmitted with a 16-bit alias size.

To get the full payload (8 bytes) for streaming requires getting the rest of the protocol control, including two aliases for source and destination, in the 29 bit header. Allowing 1 bit for priority/extension, 1 for protocol control, 3 for MTI and stream control (all very bare minima) leads to a 12-bit node ID alias length.

(Need to discuss the case of two segments being joined, to discover they've both arbitrated the same aliases, including with gateways on one or both)

Discuss the use of repeaters, bridges and gateways w.r.t. Timing; ref physical layer. Possibility of reordering.

- Nodes must handle full rate messages, specifically including messages not addressed to them. (The datagram and stream protocols provide ways for nodes to indicate whether or not they have sufficient buffering for the next transmission, triggering retries as needed) For long term reliability, a node must be able to completely process an entire CAN frame in the time it takes to receive the next one, which may be as little as 64 bit times, or about 500 microseconds.

All traffic on a CAN segment uses aliases for both Source ID (always) and Destination ID (when the full NID isn't known).

If/when a full NID is needed, it can be obtained by sending a "Verify Node ID Number (Addressed)" message with an appropriate Destination Node ID in local alias form. The reply will eventually come back with the Source ID in local alias form and carrying the full NID in the message content.

From Node ID Alias Allocation Algorithm doc

OpenLCB nodes on CAN segments do CAN arbitration by using their unique Node ID alias (NIDa) as part of the CAN header. This page documents a proposed algorithm for assigning unique Node ID aliases.

Introduction

This method of assigning aliases and checking that they are unique is necessary because of the characteristics and limitations oFromf CAN.

Each node has 48-bit unique id, and each wants a 12-bit local alias. Each assigns itself a potential 12-bit alias. The problem comes in determining whether this alias is in use by another node.

If a node sends out an check message containing just the alias, then it could expect that another node to complain if it has the same alias. This would work, except in the (admittedly unlikely) case where both nodes send out the identical check message simultaneously. Neither would recognize a conflict and both would consider that they own the same alias. Therefore, a method needs to be found that guarantees that the check message(s) are unique, even if they are sent simultaneously.

(timing of serialization of low-priority CAN nodes at network startup)

(can store starting point, not just alias, for later use)

Unfortunately, limitations of CAN will not allow the full 48-bit NodeID to be included in the check message. Doing so would violate CAN rules as it could not be wholly contained in the header, and packets with identical headers are not allowed to have differing data-parts.

Therefore, the CIM/RIM method splits the NodeID into byte-sized parts, sends out 4 check messages (CIM = Check Id Messages, RIM = Reserved Id Message), each consisting of the alias and one quarter (12 bits) of the 48 bit id. Only if all of the CIMs *do not* conflict with another node (ie no RIM is received) does it guarantee that the alias is not in use by another node. Even if two nodes send out the identical packets simultaneously, at least one of these will differ due to a differing ID-part between the two nodes.

Requirements

- Must not require any user intervention.
- Must be possible to map the assigned alias to a Node Sequence Number and vice-versa.
- It is not required to come up with the same alias every time a node or segment is powered up.
- Must always converge to an alias, regardless of when or in what order nodes come up.
- Another node watching the process must be able to monitor how the decision was reached and what the result was.
- It is desirable to detect duplicate (non-unique) NIDs, but 100% reliable detection is not required.

Algorithm

Start with the six-byte unique Node ID in each node, with the bytes labelled N1 through N6.

Initialize a specific full-sequence 48bit PRNG with the 48 Node ID.

A:

Take the next element from the PRNG . Derive the 12-bit value for TCA, the tentative alias. If TCA is zero, repeat.

Start listening for CAN frames containing Check ID messages (CIMs) and Reserved ID Messages (RIMs). A CIM consists of an extended header with a 12 bit Source NIDa field, an 3 bit sequence field, a 12-bit TestN field, and other bits to denote that this is a Check ID Message. There is no other variable information in the frame. Having NIDa, Nn and TestN in the message ensure that CAN arbitration will work without error. An RIM is the same, except that it can be identified as being a different message for diagnostic purposes; it means the same thing upon receipt.

Send four CIMs with 0 through 3 in the Nn field, the 12-bit quarters of the NodeID in the TestN field, and TCA in the Source NIDa field. During this process, if a CIM or RIM is received that contains the TCA value in its CI field, TCA does not belong to this node. Repeat from (A).

Keep listening until your CAN hardware says your last CIM message has been sent and the CAN link is idle, plus 500 msec. If you cannot detect that condition, wait 1 second.

If no CIM or RIM with this TCA value in the CI field is received before the end of the previous step, you've successfully allocated TCA as this node's source NIDa. Send an RIM with TCA in the CI field and go into normal operation.

- If that message fails with an error occurring during the data portion of the message, it's possible that two nodes have the same NodeID, a fatal error. To prevent the OpenLCB being taken down, this node should stop trying to allocate an alias and signal the error to the user via some non-CAN method, e.g. LEDs, emitting smoke, etc.
- If any other error occurs, repeat from A.

While in normal operation, if you receive a CIM with your source NIDa, send a RIM, which will cause the other node to back off and try something else.

It should not be possible to receive an RIM with your allocated source NIDa in normal operation. If you do, log an error (on whatever it is they log it on there) and restart the process at (A).

Proposed alias derivation

The 12 bit alias has to be derived from a 48-bit sequence number in a way that keeps as much entropy as possible.

Sequentially numbered nodes will have initial seeds that differ only in the low bits of the PRNG output. For shift-register PRNGs, this can present a problem, as all bytes of the result are shifting together. Combining bytes (or even 12-bit chunks) via XOR or add operations results in reduced entropy.

The OpenLCB proposed PRNG (next section) uses add operations with a constant term to avoid this problem, so treating the 48-bit PRNG output as four 12-but values, which are then XORed, works well.

Proposed PRNG

The proposed 48-bit PRNG is from “A 48-Bit Pseudo-Random Number Generator”, Heidi G. Kuehn, Communications of the ACM, Volume 4, Issue 8 (August 1961), pages 350-352.

$x_{i+1} = (2^9+1)\, x_i + c$

where c = 29,741,096,258,473 or 0x1B0CA37A4BA9.The paper actually describes a PRNG that uses signed arithmetic, but for our purposes the 48th “sign” bit isn't important. *(verify that about sign) (This is a byte+bit shift and add, so it's quick to calculate)*

Note that, unlike some other PNRGs, this one can generate a zero result, and can accept a zero seed.

Discussion

CAN arbitration reliably avoids collisions between frames with unique headers. It does not guarantee arbitration between frames with identical headers and no data; if the timing is right, they may overlay each other such that only one frame appears to have been sent. It is very difficult to ensure that two nodes send initialization packets only at different times. In addition, CAN will eventually signal an error if two packets with the same header and different data payloads collide, but not all CAN interface hardware provides reliable indications about why that error occurred. *(lock step issue at startup)*

At the highest level, this algorithm is broadcasting the complete Node ID and tentative alias to see if any other node is checking the same tentative alias. If two nodes have taken the same tentative alias, at least one of the four packets used in that broadcast will not be identical between the two nodes, because their six-byte Node IDs are different in at least one bit. CAN will successfully arbitrate this, and the other node will receive the frame, causing it to back off.

CAN transmissions are not atomic operations; you can receive a frame between the time you tell the hardware to send a frame and the time that frame is actually sent. It's therefore possible for both nodes contending for the same alias value to back off and try again. Using the pseudo-random number generator as a sequence number makes it likely that the next attempt will be made using different alias values in the two nodes. The use of the sequence generator also makes the arbitration process faster in the case of sequential Node ID values. People like to use small and sequential numbers for things, and the sequence generator maps those to very different values that are less likely to collide during arbitration.

The delays at the end of the algorithm are to ensure that higher latency nodes, such as software nodes working through USB convertors, can reply to CID messages from nodes that come up on a working link. OpenLCB is a soft-real-time system, and software that's interacting with it needs to have response times of a couple hundred milliseconds or better to be reliable. The algorithm provides a wait of a few times that to enable those programs to take part.

6 CAN Controller Filters

This coding has been generated such that simple nodes can use three hardware filters to select only frames that are of interest to them.

Purpose	Mask	Required Value
CAN-specific control messages	0x0800,0000	0x0000,0000
OpenLCB format addressed to this node	0x0C00,0FFF	0x0C00,0nnn
Other simple OpenLCB format MTIs (See later doc)	0x0F00,0000	0x0800,0000

(but the simple-MTI stuff is at Message level)

7 Additional References

B. Gaujal and N. Navet, “Fault confinement mechanisms on CAN: analysis and improvements”, IEEE Transactions on Vehicular Technology, pp.1103-1113, 54(3), May 2005 (An interesting analysis of how CAN nodes react to error rates on the CAN bus by e.g. taking themselves offline, etc)

(Take some from physical layer doc)

“A 48-Bit Pseudo-Random Number Generator”, Heidi G. Kuehn, Communications of the ACM, Volume 4, Issue 8 (August 1961), pages 350-352.

Table of Contents