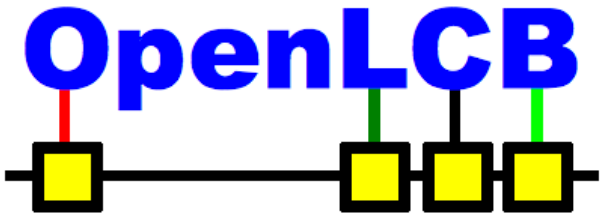


OpenLCB Technical Note	
OpenLCB-CAN Frame Transfer	
Feb 2, 2013	Adopted

1 Introduction

This Technical Note contains informative discussion and background for the corresponding "OpenLCB CAN Frame Transfer Standard". This Technical Note is not normative in any way.

For information on format and presentation conventions, see:

- OpenLCB Common Information Technical Note

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 and no data can result in lost frames; collisions between frames with identical headers that do contain data do not result in well-defined behavior. Both cases must be avoided by design to avoid intermittent problems.
2. 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.
3. It's useful to have a way of addressing a frame to be processed by a specific node, but all CAN nodes receive every frame.
4. 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.
5. 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.

1.1 CAN Specialization for OpenLCB

CAN is adapted to OpenLCB use by a series of specifications which together mitigate the above limitations:

1. 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 2).
2. Providing a destination address that allows a standard method of addressed communications (item 3).
3. Using a distributed algorithm to determine the value for that ID field prevents needing a single "manager" to provide it, thereby handling item 4.
4. The header bit fields discussed below are organized to handle item 5.

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

OpenLCB-CAN is required to work without user intervention; user presets are not required for the CAN frame communications layer to work. The Node ID that's needed for communications is required to be pre-configured into the node.

It's desirable for OpenLCB-CAN nodes to detect duplicate (non-unique) Node IDs, but 100% reliable detection is not required.

1.2 CAN collisions in loaded networks

Because of the CAN priority arbitration 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 of collisions when multiple nodes attempt to send frames with the same header.

For example, 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 many nodes have frames to send, this will result in an unacceptable probability of error, unless the protocols have been designed to have as few header collisions as possible, and absolutely no data collisions.

2 Annotations to the Standard

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

2.1 Introduction

(Nothing to add to the Standard)

2.2 Intended Use

The OpenLCB protocol suite uses 6-byte Node ID values, but this protocol layer could be useful in other contexts too. The Standard is therefore written so that any Node ID length through 7*12 = 84 bits can be used in a straight-forward manner.

2.3 References and Context

Note that the Standard references the Unique ID Standard, but only to the extent that each node is required to have a unique ID as its node ID. The Standard would work equally well with a Unique ID that was allocated via any mechanism, and with any length from 12 through 84 bits. OpenLCB uses a 48-bit unique ID as a node ID, which is assumed for the rest of this Technical Note.

2.4 Frame Format

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

OpenLCB uses a straight-forward transfer of extended-header frames, using no special features, to reduce complexity and increase robustness.

The Standard is silent on the uses for standard CAN (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[1] uses standard frames, for example, and the Microchip bootloader[2] can also use them. Allowing standard frames then allows the use of these bootloaders in parallel with OpenLCB-CAN. Similarly, CBUS[3] uses standard frames, and therefore could coexist with OpenLCB on a CAN segment.

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

RTR frames are not used in the protocol because CAN semantics require a specific use for them, which has been built into some silicon implementations. There are also some arbitration issues, see: CiA Application Note 802 (2005)[4] and http://www.thecanmancan.com/?tag=rtr for more information.

A CAN node can emit an overload frame when “Due to internal conditions, the node is not yet able to begin reception of the next (frame). A node may generate a maximum of two sequential overload frames to delay the start of the next (frame).” Those frames are 17 to 23 extra bit times each.[5] The protocol does not specify transmission of overload frames because not all CAN controller hardware can

[1]See Atmel application note “AVR076: AVR® CAN - 4K Boot Loader” http://www.atmel.com:80/dyn/resources/prod_documents/doc8247.pdf

[2]See Microchip application note “AN247 A CAN Bootloader for PIC18F CAN Microcontrollers” http://ww1.microchip.com/downloads/en/AppNotes/00247a.pdf

[3]See http://www.merg.org.uk/resources/lcb.html

[4]See http://www.can-cia.org/

[5]See http://rs232-rs485.blogspot.com/2009/11/can-bus-message-frames-overload.html

deliberately send them. We require that nodes be able to handle them because some CAN controller hardware will occasionally send them automatically.

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.

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

The reserved most-significant bit (MSB) is likely to be used as a priority-boost bit in the future, for example 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 Standard requires that a 1 bit be sent. To preserve the utility of this, nodes must not require either a 1 nor 0 on receipt. It is therefore not available as a protocol expansion bit.

The second-most-significant bit (0x0800,0000) when 0 indicates that the frame is for local control on the CAN segment. Those frames should never leave the CAN segment. When 1, it indicates that the frame is carrying OpenLCB message traffic, which can leave the CAN segment to proceed to the rest of the network; see the message-level documentation for more information on that.

The order of the header fields was chosen to get proper priority and access disambiguation via CAN's standard mechanisms.

2.5 States

In the Inhibited state, a node can only communicate on the CAN segment to allocate its Node ID alias(es). This involves sending CID and RID frames 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 Inhibited state. In general, this state transition is not visible to other nodes until the failed node starts communicating.

Designers of nodes that maintain alias pools, for example in gateways, need to carefully consider to what extent the state applies to the entire node, and to what extent it applies to each individual alias in the gateway pool. Anything that transmits OpenLCB frames on CAN needs to have reserved an alias and have transitioned to Permitted state by transmitting an Alias Map Definition frame using that particular alias. It can acquire and stock-pile other aliases for later use, putting them into Permitted state either immediately or at some later time.

2.6 CAN-specific Control Frames and Interactions

2.6.1 Control Frame Format

RID, CID, etc frames are assigned low numbers to give them higher transmission priority.

The coding is done to allow up to 7 separate CID sequence numbers, for use by protocols with longer node IDs.

2.6.2 Interactions

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

2.6.2.1 Reserving a Node ID Alias

At the highest level, this algorithm is broadcasting the tentative alias to see if any other node is checking the same tentative alias. If it is in use, the algorithm tries additional ones until it finds one that's available. The following considerations had to be taken into account when designing the alias reservation algorithm.

CAN arbitration reliably avoids collisions between frames with unique headers. It does not guarantee non-overlapping transmission from separate nodes of 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 always send initialization packets only at different times, particularly given the load-related frame synchronization issue described earlier. 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.

If a node sends out a check frame 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 frame simultaneously. Neither would recognize a conflict and both would consider that they own the same alias. Therefore, a method needed to be found that guaranteed that the check frame(s) are unique, even if they are sent simultaneously.

The CID/RID algorithm for finding aliases is designed to work with CAN arbitration, but without causing any CAN errors. During execution of the algorithm, some frames might have the same header. They therefore have to have the same contents, including null contents, or else a non-arbitration error can occur.

If two nodes have taken the same tentative alias, at least one of the packets used in the broadcast will not be identical between the two nodes, because their Node IDs are different in at least one bit. CAN will successfully arbitrate this, and one node will receive the frame sent by the other node, causing it to back off.

CAN transmissions are not atomic operations; a node can receive a frame between the time it tells 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. The mechanism for generating the next alias value to try must generate different values for different nodes to prevent this from becoming a permanent lock-step error.

The delay at the end of the algorithm is to ensure that higher latency nodes, such as software nodes working through USB convertors, can reply to CID frames 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 tens of milliseconds or better to be reliable. The algorithm provides a wait of a few times that to enable those programs to take part.

Nodes may assign different alias values each time they are powered up. There is no requirement that they always reserve and use the same one, nor is it possible to guarantee that they will.

The sequence number MMM is sent in descending order, 0x7, 0x6, …, to give priority on the CAN segment to frames later in the process. This lets some nodes complete as early as possible and remove themselves from contention.

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.

Reserving a node ID alias value is separate from actually using it to take a node from Inhibited to Permitted state. A gateway device, for example, might want want to stockpile a few reserved alias values so that they can be rapidly associated with a specific node that wants to communicate. The node ID that's broken into four parts for the CID values needs to be unique, but it does not have to be the one that's eventually attached to the alias via a Alias Map Definition frame later on.

For example, a minimal sequence is four CID frames and a RID:

```
0x17123FED
0x16456FED
0x15789FED
0x14ABCFED
0x10700FED
```

The red nibbles show that these are four CID and one RID frames. The blue nibbles are the source ID alias being tested and allocated. The yellow nibbles are the underlying Node ID of the node, in this case 12 34 56 78 9A BC.

2.6.2.2 Transition to Permitted State

Note that there is no requirement that the node's alias value be consistent from one power-up reset to the next.

The Alias Mapping Enquiry provides a mechanism for determining which alias is associated with each Node ID. This can be used when a new node attaches and needs the information. Operating nodes can also track this information by watching CAN segment traffic for control frames, without having to use the Alias Mapping Enquiry protocol.

2.6.2.3 Node ID Alias validation

Node ID alias validation 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 general, simple or regular nodes won't need to do node alias validation. Those nodes don't need to keep track of the Node ID to alias mapping, because they simply respond to frames they receive that contain an alias. They just work with the alias itself. They also don't need to track other nodes' state, etc.

In order to acquire the map between CANids and node IDs, a gateway needs to be able to send a CAN frame requiring "everybody reply with your node ID". Much like Ethernet gateway mapping, this needs

to return the node ID of just the specific CAN-attached hardware, not all the node IDs that can be e.g. reached through one or more gateways, so it needs to be a segment-specific frame defined only at the CAN level. It must not be a OpenLCB common message, though higher level protocols may want to provide a similar capability that spans the entire OpenLCB installation.

2.6.2.4 Transition to Inhibited State

The Alias Map Reset frame indicates that the alias value in the source address field may no longer be associated with the same NID, and mapping tables should be reset.

The full node ID in the Alias Map Reset frame allows nodes who only are aware of an alias for their conversation partner to convert that alias to a full node ID for later use.

This is a way to release or reuse an alias, while informing everybody that the node is no longer reachable.

The alias is only released if this node allows some other node to allocate it via CID/RID frames. The current node can still hang on to it if desired, for example to reuse it with another Node ID, by replying to CID/RID frames from another node attempting to reserve it.

Gateways have to obtain unique aliases for remote nodes they are proxying onto the segment. Gateways maintain the mapping between remote node's node ID and the local alias. If they need to break that mapping, e.g. because they need to repurpose a local alias due to resource limitations in the mapping tables, they must send a "Alias Map Reset" frame to force nodes to drop their alias information. "Alias Map Reset" is a CAN-specific frame limited to the segment, but all gateways on the segment must act on it. This is then followed by an "Alias Map Definition" when the alias is allocated to a new node ID.

Regular nodes won't need to do this. They stay active until they go instantly offline due to e.g. power-off.

When a node goes to inhibited state or releases an alias by sending an Alias Map Reset frame, that's telling other nodes that it can no longer be reached by using that alias. The last paragraph in this section of the Standard gives the node receiving the Alias Map Reset 100msec to finish processing and stop using the alias. If it still needs to send messages to the node that sent the Alias Map Reset, the receiving node can attempt to reacquire a new alias via an Alias Map Enquire frame or higher-level message(s). If the alias has changed, that will work, but if the sending node has transitioned to Inhibited state there will be no reply to the attempt to obtain the new alias.

2.6.2.5 Node ID Alias Collision Handling

During normal operation in the Permitted state on a single CAN segment, a node should never encounter a Node ID alias collision.

They can happen when network topology changes, for example when connecting two operating CAN segments. In that case, nodes on the two segments have assigned aliases independently, and can have used the same value. The underlying Node IDs do not collide, because they are assigned globally uniquely, but that's not sufficient to ensure that the shorter aliases are independent.

In the case of a collision, one or both nodes will release the current alias. If it's the alias that was used to go to Permitted state, e.g. the current alias being used by this node, the node must also drop back to Inhibited state. The node(s) then can, but is not required to, return to Permitted state with a new alias. In the process the node(s) will emit “Alias Map Reset” and “Alias Map Definition” frame that will allow any gateway(s) to update their maps.

The preceding paragraph and the language of the Standard is complicated because it has to cover the case of a gateway or similar node who has reserved a number of node ID aliases, but is not currently using them as a node. In other words, the reservation CID/RID process has completed to reserve the alias, but no mapping of the alias to a specific node ID has been published.

Simple leaf nodes don't keep internal mappings between full node IDs and aliases, so this transition will not cause any problems for them unless they are currently in communication with another node, e.g. in the middle of a sequence of frames for a datagram or stream. In that case, the node may have to invoke higher-level error handling to cause the sequence to be retried.

For example, a detachable throttle will detect that it's reattaching, reserve an alias value, and then enter permitted state with that value. Since the reservation happened while connected to the current CAN segment, there is no possibility of a duplicate.

Example: Nodes A and B are in Permitted mode with different node IDs, but the same alias values. Some gateway node C requests a list of all nodes on the CAN segment:

C: 0x10702ccc (AME frame)

A: 0x10701nnn 01 02 03 04 05 06 (AMD frame with Node ID 0x0102,0304,0506)

(For simplicity, B has not transmitted yet) B discovers that it's alias has been duplicated.

B: 0x10703nnn (AMR frame)

B then goes back into operation by reserving a new alias value and transitioning to Permitted state:

B: 0x17iiimmm (1st CID)

B: 0x16iiimmm (2nd CID)

B: 0x15iiimmm (3rd CID)

B: 0x14iiimmm (4th CID)

B: 0x10700mmm (RID)

B: 0x10701mmm ii ii ii ii ii ii (AMD with B's node ID attached)

2.6.2.6 Duplicate Node ID Handling

This should never happen, of course, but it's possible that two nodes will have the same Node ID assigned. This will result in frame loss and operational difficulties. OpenLCB-CAN does not guarantee prompt detection of this condition, at any level of the protocol stack, but it's extremely likely that it will be detected. At the Frame Transfer level, two nodes on the same CAN segment sharing a common

Node ID will be detected when a node receives a Alias Map Definition frame that contains its own Node ID.

The “in addition to any other actions” refers to e.g. acting on identical alias values, should that also be the case.

Sending the PCER message is level jumping, in that such messages are defined at a much higher level in the protocol stack. We include it here because it allows a much cleaner set of error handling operations using the event producer/consumer techniques associated with OpenLCB. This decision can be revisited.

After sending the (optional) PCER message, the node has to go offline so that it doesn't interact anymore until the user resets it to restart operation. This is necessary to prevent the PCER itself triggering additional messages back, resulting in a never-ending cycle.

Example: Nodes A and B are in Permitted mode with the same node IDs, but different aliases. Some gateway node C requests a list of all nodes on the CAN segment:

C: 0x10702ccc (AME frame)

A: 0x10701aaa 01 02 03 04 05 06 (AMD frame with Node ID 0x010203040506)

(For simplicity, B has not transmitted yet) B discovers that its node ID has been duplicated.

B: 0x195B4bbb 01 01 00 00 00 00 02 01 (Producer-Consumer Event Report (PCER) message with event ID 0x0101,0000,0000,0201)

If the two nodes have also assigned the same alias value, that interaction is also invoked here.

2.6.3 Node ID Alias Generation

We have a preferred implementation for node ID alias generation which is described in Section 6 “Preferred Alias Allocation Method” below.

The Standard is written to specify the minimum needed to ensure proper operation. This allows developers to create other implementations if needed.

“of the same type” is meant to cover “from the same manufacturer”. The idea is that manufacturer will likely number their boards sequentially, and it's possible that an OpenLCB installation will have multiple boards from the same manufacturer. This rule just says that the low bits of the sequential number appear in the alias, so that boards from a single manufacturer are not likely to collide.

Two other criteria that were considered:

- The sequence of alias values generated by a node shall depend on the node's node ID in such a way that the sequence differs from the sequence generated by every other node.
- Given sufficient iterations, the sequence of alias values generated by a node shall include every valid alias value.

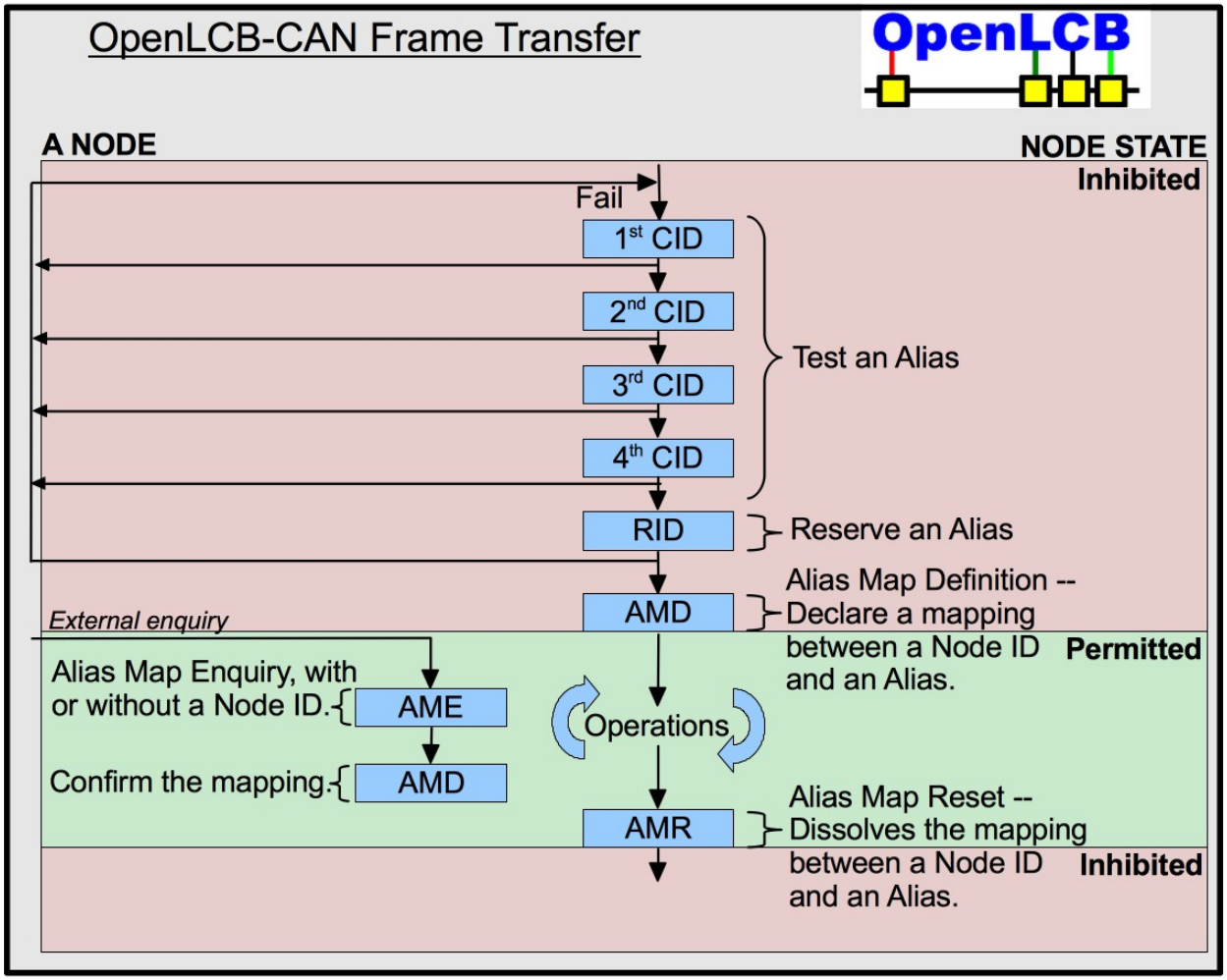
Although useful criteria, they were omitted as too hard to comprehensively test.

What's really necessary, but hard to describe in standard language, is that two separate nodes need to have different sequences of generated numbers. If node A will try 3, 4, 5, 6, … then it's important that

node B try some other sequence. If B also generates values in the sequence 3, 4, 5, 6, … then the two nodes can get locked to each other trying the same value, both rejecting it, and then trying the same new value.

Formally, if A_i is defined as a particular value in node A's sequence, B_i is defined as a particular value in B's sequence, then "if $A_i = B_i$, then A_{i+1} != B_{i+1} most of the time" is the important condition.

It's not required that two sequential aliases be different. It's possible that the sequence generator will come up with the same value occasionally, and the RID/CID mechanism is constructed to handle that. The sequence on the CAN bus will be clearer, and in some cases the allocation process will be faster, if nodes identify duplicate aliases in sequence, and just go on to the next one.



3 Gateways and segment management

Multiple CAN electrical regions can be connected into a single CAN segment via repeaters that allow the entire segment to share arbitration. The OpenLCB-CAN frame-level protocol was designed to work with these: It doesn't use RTR frames, and negotiates aliases over top of the CAN arbitration.

Bridges, which work by passing frames between separate segments, are also supported. In normal operation, OpenLCB-CAN frames can be transported through those bridges unchanged. While the bridge is active, the alias negotiation system will work properly. If a bridge is inactive, e.g. the two segments are separate, while nodes are coming up and allocating their aliases, it's possible that duplicate aliases will be allocated. Bringing up the CAN bridge hardware before bringing up the nodes prevents duplicate assignment. If duplicates are assigned, e.g. due to bringing up a bridge between active CAN segments, the nodes will discover the duplicates on their first transmission, and will rectify the situation using the procedure defined in the “Node ID Alias Collision Handling” section.

Higher-capability bridges, and gateways to non-CAN OpenLCB networks, will want to handle the assignment of local aliases directly. This prevents alias collisions between separate CAN segments, and allows very large OpenLCB networks, but requires more work from the gateway. It will allocate a pool of aliases using the conventional technique, and then assign them to off-segment nodes as they need to communicate on the segment. Many algorithms are possible for that assignment.

If/when a full NID is needed, it can be obtained by sending a "Alias Mapping Enquiry" frame 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 frame body.

4 Throughput and Temporarily Deaf Nodes

29-bit extended header and 8-byte payload results in a total transmission taking about 135 bit times, varying slightly with bit-stuffing.

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.The CAN network itself does not require any inter-frame spacing, limitations on bandwidth usage, etc; the segment can be allowed to run at 100%.

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 a node to broadcast that it's back up if the hardware detected frames lost. This is because there could have been frames of some form that modified the OpenLCB node state, which are now lost (e.g. drop alias, CID/RID in absence, or even higher level events)

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 in general. Nodes must handle full rate frames, specifically including frames not

addressed to them. 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 for a header only frame.

Eventually OpenLCB may have to discuss use of Overload Frames to throttle for e.g. NVRAM writing, but various CAN hardware implementations may also be doing this already. This mechanism provides only a little more time, due to restrictions in the CAN specification that have become limitations on the silicon implementations.

5 On the choice of a 12-bit alias length

The size of the 12-bit NodeID alias field was selected by balancing two constraints:

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

5.1 Issues

5.1.1 Addressing

The alias 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 100 nodes[6], but communication also involves nodes off the segment. For example, chaining N CAN segments via bridges will generally make up to 100*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.

Finally, one can imagine connecting a few CAN segments in a star pattern to a central OpenLCB-CAN “backbone”, which must route the full cross-segment traffic.

In some cases, we want two aliases (source and destination), plus a few more bits, in the 29 bits total available in the header. Using 12-bits for each address leaves 29-(2*12) = 5 bits for these purposes, which seemed about right, and allows a maximum of 2^12, or 4096 nodes.

5.1.2 Collisions during Allocation

The alias 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.

[6]The OpenLCB-CAN Physical Layer Standard says in its Intended Use section uses 50 nodes as an expected size. This is merely an informative statement of expectations, however, and some layouts will no doubt exceed that. We use the larger number 100 here to avoid placing any stronger limitation on what can be achieved electrically.

5.1.3 Size

Streaming provides the most stringent test of alias sizes. We'd like to have all 8 bytes of CAN frame payload available for transporting stream data, which requires putting both source and destination addresses, plus whatever control information is needed, into the header.

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.

6 Preferred Alias Allocation Method

This section documents the preferred algorithm for generating unique Node ID aliases.

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 a 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.

6.1 Preferred alias generation algorithm

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

The entropy starts in the node ID values, which cover a 48-bit space (albeit non-uniformly). This provides an initial 48-bit value that can be mapped to a 12-bit value.

To create a sequence of values, you want a computationally cheap way of sequencing through all those possible 48-bit values. (Running through the 48-bit values and compressing each to a 12-bit alias value is less likely to get multiple nodes attempting the same sequence than just working through a 12-bit sequence) A suitable pseudo-random number generator does exactly this: it runs through full 48-bit sequence, with each node starting at separate point in the sequence, so the 48 bit values are unique.

A computationally convenient way to create a 12-bit value, while preserving as much of the randomness as possible, is to break the 48-bit value into four 12-bit values, align these, and exclusive-or them. Each bit-change in the original 48-bit number then becomes a change in the final 12-bit value.

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 with just one new bit shifting in at the top. (Traditionally, shift-register PRNGs shift toward the LSB) Only one bit changes. 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.

6.1.1 Preferred 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. 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.

It is initialized with the node ID as a 48-bit key, with the most significant byte of the node ID as the most-significant byte of the PRNG value.

6.1.2 Preferred PRNG to node ID alias mapping

The 12-bit node ID alias is made from the 48-bit PRNG value by splitting the PRGN value into four 12-bit values and XORing them together. This preserves the information content of every bit in the final result.

6.1.3 Example C implementation[7]

The PRNG state is stored in two 32-bit quantities:

```
uint32_t lfsr1, lfsr2; // sequence value: lfsr1 is upper 24 bits, lfsr2 lower
```

The 6-byte unique Node ID is stored in the `nid[0:5]` array.

To load the PRNG from the Node ID:

```
lfsr1 = (nid[0] << 16) | (nid[1] << 8) | nid[2];
lfsr2 = (nid[3] << 16) | (nid[4] << 8) | nid[5];
```

To step the PRNG:

```
// First, form 2^9*val
uint32_t temp1 = ((lfsr1<<9) | ((lfsr2>>15)&0x1FF)) & 0xFFFFFF;
uint32_t temp2 = (lfsr2<<9) & 0xFFFFFF;

// add
lfsr2 = lfsr2 + temp2 + 0x7A4BA9l;
lfsr1 = lfsr1 + temp1 + 0x1B0CA3l;

// carry
lfsr1 = (lfsr1 & 0xFFFFFF) | ((lfsr2&0xFF000000) >> 24);
lfsr2 = lfsr2 & 0xFFFFFF;
```

Form a 12-bit alias from the PRNG state:

```
uint16_t LinkControl::getAlias() {
  return (lfsr1 ^ lfsr2 ^ (lfsr1>>12) ^ (lfsr2>>12) )&0xFFF;
}
```

[7]The code in this section is from the OpenLCB C sample implementation available at http://www.openlcb.org/trunk/prototypes/C/libraries/OlcbCommonCAN/ The specific code is available at http://www.openlcb.org/trunk/prototypes/C/libraries/OlcbCommonCAN/LinkControl.cpp

7 Additional References

The OpenLCB-CAN Physical Layer TN contains useful references on the characteristics of CAN.

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)

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

Appendix A: Calculation Examples

Starting 48-bit seed	Alias	Next 48-bit seed
00 00 00 00 00 00	0	1B 0C A3 7A 4B A9
1B 0C A3 7A 4B A9	11E	4F 60 3B 8B E9 52
4F 60 3B 8B E9 52	521	2A E3 F5 D8 D8 FB
2A E3 F5 D8 D8 FB	42D	0D DC 4B 05 1A A4
0D DC 4B 05 1A A4	663	E1 7E F8 B4 AE 4D

Table: Sequence of aliases and seeds starting from zero

490

Starting 48-bit seed	Alias	Next 48-bit seed	Next Alias
02 01 21 00 00 12	113	1F 4F C4 7A 6F BB	62D
02 01 12 00 00 21	113	1F 31 B5 7A 8D CA	A24
02 01 11 00 00 22	113	1F 2F B4 7A 8F CB	625
02 01 22 00 00 11	113	1F 51 C5 7A 6D BA	A2C

Table: Four seeds (of 2^{36}) that give the same alias, and their different subsequent seeds and aliases

Table of Contents

