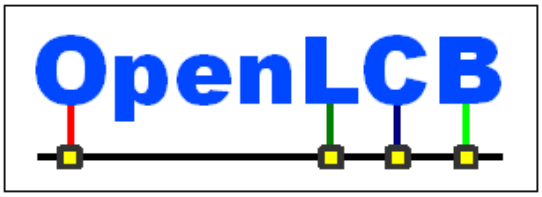

OpenLCB Technical Note	
Message Network	
July 22, 2024	Adopted

1 Introduction

This Technical Note contains information that is informative about the OpenLCB Message Network. It is meant to be read in parallel to the companion "OpenLCB Message Network Standard" document. It expands that content with the reasons for decisions and choices made, and discusses usage.

If an implementation uses any mapping for Node IDs or MTIs, these mappings must be one-to-one to the common MTIs and Node IDs. An example is the CAN implementation, which maps MTIs to shorter CAN MTIs, and Node IDs to shorter Aliases.

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

In addition to these mandatory messages, the other members of the OpenLCB Protocol Suite define optional interactions between nodes. These are meant to be structured so that they can be extended, either by adding to the existing protocols, or defining new protocols. Nodes will generally only implement a subset of these protocols, and will ignore others and any new ones defined after their manufacture.

2.2.1 References and Context

There is nothingg to add to the Standard.

2.3 Messages

2.3.1 Message Format

On some constrained segments, MTIs and/or Node IDs may be mapped to values taking less space. In all cases these mappings need to be one-to-one, and may require dynamic allocation. For an example see the CAN implementation.

2.3.1.1 Message Type Indicators

Many nodes will treat MTIs as magic 16-bit numbers, just comparing them for equality to specific values of interest. That is a perfectly fine node implementation strategy.

The all-zero MTI is reserved as it is inevitably needed for internal flags, empty buffers, etc in software implementations.

The MTI may be adapted for some transports. For example, on CAN, the MTIs are mapped to a 12 bit CAN MTI, and other aspects are encoded in the CAN frame ~~type~~format and additional bits elsewhere.

The 0x8000 MTI bit-field is reserved because it may be used in the future as part of a CAN hold-off mechanism; it has a very high priority so that nodes can send it to delay other frames from being transmitted.

The Stream or Datagram bit-field (0x1000) does not appear in the CAN-MTI, but is converted to a CAN frame-~~type~~format. It is used by Gateways to indicate need for buffering.

The Priority bit-field is the only field that explicitly determines message priority. The subsidiary Type within Priority bit-field distinguishes different MTIs, but does not contribute to their priority. However, note that every bit in the header contributes explicitly to a message's prioritization on CAN and this is a hardware feature of CAN. Therefore, the Type within Priority bit-field does have effect on priority on CAN, and the specific values were chosen with this in mind.

This section describes how the numeric values for those MTIs are allocated. The discussion in this section is not normative on OpenLCB users or node developers, but does describe the methods that are to be used for allocating new MTI values for new OpenLCB message and protocol types.

We've chosen to allocate MTI bit fields to make decoding simpler, and if possible, aligned on nibble boundaries to make it easy to read as hexadecimal numbers. We've also used a mix of bit-fields and individual flag bits to increase compatibility if and when additional MTI values are defined later.

There are two basic approaches to identify classes of message types, such as “addressed” vs “global” messages.

1. Use a dedicated bit field to distinguish the types, e.g. 1 indicates addressed and 0 means global.
2. Encode in the type number, e.g. “Type A (addressed) is 1”, “Type B (addressed) is 2”, “Type C (global) is 3”, “Type D (addressed) is 4”.

The encoded form uses less bits, particularly if there are many classes to distinguish, which would require many dedicated bits. But future expansion is easier with dedicated bit fields, because nodes can do some limited decoding of MTIs even though the node was created before the new MTI values were defined. For example, a gateway can determine whether a message is global or addressed to a particular node, even if the specific MTI was defined after the node was built.

This MTI organization allows nodes to do simple decoding of messages with MTIs that they don't recognize, perhaps because they were defined after the node was created. For example, gateways can use this to control routing of messages that they don't understand, perhaps because they were defined

after the gateway was developed, and a simple node can filter for simple-protocol messages and messages specifically addressed to it.

The 0x0000 and 0x0001 MTIs are explicitly reserved. 0x0001 may be used in the future as part of a CAN hold-off mechanism; it's a very high priority which nodes can send to delay other frames. The all-zero MTI is inevitably needed for internal flags, empty buffers, etc in software implementations.

2.3.1.1.1 MTI Bit-Fields

Top Nibble

The contents of the top nibble is still under development. Note that there are only a few values currently defined as useful: 0 and 1 for messages that will route to any CAN network; and 2 for messages that are not propagated to CAN networks. In the future, we may want to change this nibble from bit coding to value coding but those specific values will be preserved. Values 3-15 or the top 2 bits are not currently assigned and are available for the future.

Bit 13 Special

This bit defines whether a message on a segment should be propagated off that segment to others, e.g. a TCP/IP backbone link between CAN gateways. A 0 means that the message should go through gateways; a 1 indicates that it should not.

Bit 12 Stream or Datagram

This bit is set to indicate that this MTI is the data part of the stream or datagram protocols (defined elsewhere). For efficiency, these are handled in a special way on CAN and perhaps other wire protocols, and so this bit makes it easy to identify these MTIs.

Middle Byte

The next eight bits form a specific message type number. It has this substructure:

Bits 10-11 Static priority groups

The most-significant two bits, 10-11, are used to form static priority groups, with 0b00 having the highest priority, and 0b11 having the lowest. Priority processing is permitted but not required. This field is included to allow protocol designers to ensure that CAN frame reordering (which, although not always present, is a normal part of CAN that must be considered) won't result in problems for communications.

The priority mechanism may or may not work outside the priority field~~.~~ . In some cases, such as CAN, the entire MTI field will enter into the priority calculation. MTIs should be chosen to work in either case. For an example of this, see the Event Transfer Protocol Standard and Technical Note.

Bits 5-9 Type within Priority

These 5 bits are used to indicate the specific message type within a priority group and taking into account the values of all the other bits. This is the unique part that is selected during the process for designing a new OpenLCB protocol to ensure a unique MTI.

Bit 4 Simple Protocol

The Simple Protocol bit is used to indicate messages meant for "simple" or minimal nodes. A 0 in this bit means that these simple nodes can ignore this message. A 1 in this bit means that the simple nodes must process the message. See the section below for more information. This bit is reserved to 0 for all addressed message types, because a message specifically delivered to a node should be processed.

Bottom Nibble

The bottom nibble of the MTI is interpreted as flags that define the structure and format of the message type.

Bit 3 Address Present

The Address Present flag indicates whether the message carries a destination address when set to 1, or does not carry a destination address when set to 0. These are also referred to as "addressed" or "global" messages respectively. Global messages shall be delivered to all nodes.[1] An addressed message shall be delivered to the node with its destination node ID. It may, but need not, be delivered to other nodes.

Bit 2 Event Present

This bit indicates whether this message carries a P/C Event ID field when set to 1, not when set to 0. If a Event ID is present, then it is at a specific location in the message content, i.e, right after the destination address, if present, or right after the MTI if no destination ID is present.

Bits 0-1 Modifier Bits

These two least-significant bits can be used as modifiers to the specific MTI, or (later) used to create additional MTI codes. At present, unless used as MTI modifiers, these should be sent and checked as all 0 bits, i.e. 0b00. Some MTIs have additional status bits defined as part of this field. For example, there are two status bits associated with "Consumer Identified" which must be kept in the header since there is no room in the CAN data field. These are considered to be MTI modifier bits.

2.3.1.1.2 MTI Considerations on Constrained Transports

MTIs may be adapted to suit on transports that have lesser capacity. CAN is a good example, see the appropriate section of this document for adaptation to CAN.

2.3.1.2 Message Content

Since global (non-addressed) messages are distributed across the whole bus, it is prudent when designing protocols to keep these messages as small as possible. Addressed messages could be longer, but Datagrams and Streams already fulfill the need for longer addressed messages.

Note that there is no overall limit on message length. Every specific protocol's specific message has its maximum length defined, thus a node manufacturer can determine what is the maximum length of messages that the node has to be able to process. Nodes may reject or clip messages that are longer than allowed by the specs they were built for. Gateways can use the fragmentation feature of the individual transport layers to keep their buffer sizes limited when needed.

[1]The "simple node protocol" is an exception to this, which needs to be worked into this Standard.

On constrained transports this still holds, although the individual items may be mapped to less space consuming forms. For example, see section 2.7.CAN Adaptations.

2.3.2 States

The state diagram is simple, nodes start in the Uninitialized state. After initialization, the node sends a Initialization Complete message and then transition to Initialized state, after which all other messages can be sent.

The Uninitialized state is only occupied when the node is first starting up. This makes for a really simple state machine: just send Initialization Complete message first thing on coming up.

At present, there's no way to deliberately return to the Uninitialized state, as no need for this has been identified.

2.3.3 Definition of Specific Messages

2.3.3.1 Initialization Complete

This is one of the messages that informs Gateways whether its sender node is Simple or not. It has two forms, one of which has a modifier-bit set to indicate that the node is simple. For more information, see 2.4 Simple Node Protocol.

2.3.3.2 Verify Node ID

The Node Verify ID message and its reply, the Verified Node ID message, are useful for identifying all nodes, or for verifying whether a node is reachable.

There are multiple forms (MTIs) of the Verify Node ID message.

The directed (addressed) form may include an optional full Node ID in its data section. The addressed node must always reply, whether or not a Node ID is carried in the data, and whether or not there is a match when the optional Node ID is present.

The global (unaddressed) format may include an optional Node ID. If it is present, only nodes with a matching Node ID should reply, if absent, all nodes should reply.

To allow mappings on constrained systems, some of the message forms may appear to have redundant information. This is the case for Verify Node ID messages, where the destination Node ID is repeated twice. This makes more sense when when one looks at the CAN implementation. Here, the source and destination Node Ids have been reduced to their Alias equivalents, but the second copy of the destination Node ID remains full.

Full: (MTI, sourceID, destinationID, numDataBytes, destinationID)
Eg: ***(0x488, 0x123456789ABC, 0xFEDCBA987654, 6, 0xFEDCBA987654)***
Note this has two copies of the full Node ID of the destination.

CAN: [CAN-MTI, srcAlias](length) dstAlias destinationID
Eg: ***[0xXX, 0x123](8) 0xFED, 0xFEDCBA987654***
Note this includes both the Destination-Alias and the full Destination Node ID.

2.3.3.3 Verified Node ID

The node ID in the data is redundant on wire protocols that carry the full source ID, but can be very valuable for wire protocols that abbreviate ("alias") the source ID within the messages, e.g. CAN, since it maps the Alias to the Node ID.

This message could have been an addressed message, since the source information on the Verify Node ID request will be enough to get the Verified Node ID reply back to the originator. However, it was left as a global which will be carried everywhere and its routing can't fail.

This is one of the messages that informs Gateways whether its sender node is Simple or not. It has two forms, one of which has a modifier-bit set to indicate that the node is simple. For more information, see 2.4 Simple Node Protocol.

2.3.3.4 Optional Interaction Rejected

(Nothing to add to the Standard)

2.3.3.5 Terminate Due to Error

The use of the optional data value is to be defined by the node implementer in the node documentation, or perhaps in the documentation for the specific protocol being exchanged when the error happened.

2.3.3.6 Protocol Support Inquiry

(Nothing to add to the Standard)

2.3.3.7 Protocol Support Reply

These will be extended as necessary as new protocols are added.

OpenLCB documentation is big-endian. This means that when sending 0x80 00, the set bit is in the first byte sent.

2.3.4 Interactions

2.3.4.1 Node Initialization

Sending the Initialization Complete message is required to insure that higher-level tools are notified that they may start to work with the node. For example, this allows Gateways to build their tables.

There is no guarantee that any other node is listening for the Initialization Complete message, and no reply is possible.

Nodes must not emit any other OpenLCB message before the "Initialization Complete" message.

The initialization complete message indicates that a node has just started to operate, but it can also mean that the node failed and restarted.

2.3.4.2 Node ID Detection

This can be used as check that a specific node is still reachable. When wire protocols compress the originating and/or destination Node ID, this can be used to obtain the full Node ID.

The standard Verify Node ID Number interaction can be used to get the full 48-bit Node ID from a node for translation. At power up each node must obtain an Alias that is locally unique. Gateways will also have to obtain unique Aliases for remote nodes they are proxy-ing on to the segment.

Note that nothing here says that the node can't send a Verified Node ID message at other times, e.g. as a heart-beat indicator.

The following are some examples of using these messages to perform certain tasks. The message format used in these examples is rather informal: (MTI, Source Node ID, Length of data, Data).

2.3.4.2.1 Example: Finding all nodes

To find all nodes, send the global form with no node ID – each node will send a Verified Node ID message.

Node 0x112233445566 sends:
(0x0490, 0x112233445566, 0) – global without Node ID;
and receives multiple responses;
(0x0170, 0x333344445555, 6, 0x333344445555) – unaddressed Verified Node message from first Node;
(0x0170, 0x817263544536, 6, 0x817263544536) – unaddressed Verified Node message from second Node;
...

2.3.4.2.2 Example: Confirming that a specific node can still be reached

If you have the necessary information (i.e., the Node Alias) use the addressed form, as it's less load on the entire system. Otherwise use the global form with the Node ID in the data.

Node 0x112233445566 sends:
(0x0490, 0x112233445566, 6, 0x0817263544536) *– global with Node ID;*
and receives:
(0x0170, 0x817263544536, 6, 0x817263544536) *– unaddressed Verified Node message;*
The originating node confirms Node 0x817263544536 is reachable.

2.3.4.3 Protocol Support Inquiry and Response

OpenLCB defines various optional protocols. If another node attempts to use a protocol that the target node doesn't implement, there are well-defined rules for how the target node will either signal an error or ignore the request.

For some uses, it's more convenient to be able to tell whether a node implements a protocol before attempting to use it. The Protocol Support Inquiry and Protocol Support Reply messages define a method for doing that.

To determine which protocols a node implements, a Protocol Support Inquiry message is sent to the specific node. It will reply with a Protocol Support Reply message, where each specific bit position has been reserved for each defined protocol. If the bit is zero or not present, the protocol is not supported and requests to use it will result in an error. If present and 1, the protocol is supported.

As indicated above, it is not necessary to use these messages to check whether an addressed protocol is supported by a node before attempting to use the protocol. If it is not supported, the standard error handling mechanism will indicate that. However, these messages provide a way to check whether the protocol is supported without errors, and avoiding errors provides a cleaner system. Further, this method can check support for protocols that use global (non-addressed messages), and since nodes are not permitted to return errors for global messages, it is more difficult to detect that a node is ignoring a protocol attempt.

We don't explicitly reference all the protocol definitions that are associated with specific bits. Implementors can find them from the protocol names, and we don't want to clog up this section of the Standard with a set of references that we'll have to update continually.

Low-end nodes may want to implement this protocol so that higher-function nodes can easily learn their limitations. Generally, the node designer can just provide a simple fixed value for the reply.

The requirement for messages of "one or more bytes" is to allow for future expansion. It's not necessary to send extra zero bytes past any bits that need to be asserted.

OpenLCB is big-endian, so the protocol bits have been assigned from the MSB of the 1st byte.

In general, nodes using this interaction don't need to know the meaning of bits defined after the node was created. A node is looking to see whether a particular protocol is present or not so that it can select what actions to take. A newly-defined protocol isn't among the things that the node will try to reason about. This allows us to eventually extend the length of the reply without causing trouble for existing nodes.

We define the bits here, rather than in the individual protocol definitions, to reduce the risk of duplicate assignments. Duplicate assignments would be obvious here, but not so much when spread across separate documents.

This interaction is optimal for requesting information from a single node, but not for requesting which nodes on the entire network can provide a service. Event-based methods, whether a capability is announced via an event or the Identify Producer and Identify Consumer mechanism is used to find which nodes can source the event that identifies the capability, may be more effective for that use case. The information returned is intended to be considered static: A node may request it and never have to request it again, because it won't change. For development purposes, a node might be reprogrammed, so it might be useful if configuration tools have a way to force rescan of this data.

2.3.5 Error Handling

2.3.5.1 Reject Addressed Optional Interaction

This is a response to a message directly addressed to this node, as such the originating node will be expecting some action, or a reply. The intent is that a node would always reply with OIR if it received a self-addressed MTI that it was not going to process via some other interaction. That way, the sending

node would be told that the receiving node wasn't able to properly handle a request, and could invoke some error handling. The goal here was to provide that error handling (without need for timeouts) for future interactions that might involve MTIs that pre-existing nodes didn't implement.

Example: A node receives an addressed message with an unrecognized MTI:
(0x04F9, 0xAAAAAAAAAAAA, 0x0123456789AB, 0)
The receiving node replies with:
(0x0068, 0x0123456789AB, 0xAAAAAAAAAAAA, 4, 0x10 00 04 F9)
The 0068 indicates Reject Optional-Interaction, note that it has its addressed-bit set. The ***0x04 F9*** is the original MTI. The ***0x10 00*** is the error code, which in this case indicates this is a permanent error. The MTI is included so that the originating node can determine which type of interaction failed.

2.3.5.2 Reject Unaddressed Optional Interaction

We do not want every node to be replying if it is not targeted (addressed), as this would result in a large amount of unnecessary traffic.

2.3.5.3 Reject Addressed Standard Interaction Due to Error

Note that the Standard doesn't say whether the Node in consideration, or another Node, started the interaction, it could have been either.

This is a very coarse mechanism that is meant to handle rare events that should not routinely occur. The “most recent MTI received” value is not always sufficient to determine which interaction is being referred to. Higher level protocols should define more focused and reliable mechanisms if they are likely to encounter errors.

Nodes must handle messages of this type that arrive without MTIs and/or error code information.

The optional fields may be included, as necessary. Although the use of these fields is to be defined, buffer space must be reserved for messages, so the maximum number of bytes is specified.

2.3.5.4 Duplicate Node ID Discovery

OpenLCB nodes are required to have unique node IDs. While the Frame Transfer protocol will detect duplicate node IDs on a single CAN OpenLCB segment, it is not intended to detect duplicate node IDs across multiple segments. This section is meant to detect duplicates across the entire connected OpenLCB.

The “whatever indication technology is available” in the Standard means: use LEDs if the node is a board with LEDs; put something on the screen if the node is a program in a PC; etc. The Standard does not care how you do it, just that you do. If possible, it should emit a Producer-Consumer Event (PCER) message with the “Duplicate Source ID detected” global event, which carries the duplicate node ID in the Source Node ID field. After sending the “Duplicate Source ID detected” global event, the node should not transmit any further “Duplicate Source ID detected” messages until reset because this message will be received at the other duplicate-ID node(s), resulting in additional “Duplicate Source ID detected” global events and causing a possible message loop. Optionally, it could allow to send again after e.g. 5 seconds.

Yes, sending a PCER message is level jumping, but it's the best way to do it within the existing structure, since the vast majority of nodes implement the Event protocol. See the Event documents for more detail.

To further improve the reliability of this detection, nodes may, but aren't required to, emit a Verified Node ID message every 30 to 90 seconds. As an implementation detail, it's recommended that CAN-attached nodes use their Node ID Alias to pick that interval so that messages don't bunch up.

2.3.5.5 Error Codes

These error codes are meant as a basis for a more complete set specified by other protocols, and are gathered in one section so that they are consistent, and can be applied to other protocols in a consistent manner. Eventually, it might be necessary to separate them if they start to diverge in meaning.

Note that, at the very least, a node will return 0x0000, which is not ideal, but acceptable.

The Permanent Error Code 0x1040 is designed to indicate that a particular function is not implemented. This includes when the specific data content is not recognized or not supported. For example, while the Datagram Protocol might be supported, the specific Data Content ID is not recognized, or is not supported. See Datagram Transport documents.

The Temporary Error Code 0x2080 indicates that the message was garbled such that it cannot be interpreted, and should be resent. On some transports this might reflect an incorrect CRC or check-sum. On CAN this is detected and automatically taken care of by the CAN hardware.

2.3.6 Routing

While the routing and transport of addressed messages is most simply done by forwarding them to all segments, this is both inefficient and could result in swamping slower segments of the network. This traffic can be reduced by Gateways building tables of the routes to nodes, and only forwarding Addressed messages according to those tables.

Similarly, Event traffic can be reduced since OpenLCB is designed to allow the use of *Interest-based Routing/Filtering*. Events are by definition unaddressed and it is simplest to forward them everywhere. However, it is clear that a specific Event needs to be forwarded to those nodes which are interested in it, and by implication to only those segments that contain such nodes. A node's interest in an Event can be determined by using and monitoring Identify Consumer and Identify Producer messages, and their replies. In this way, Gateways may construct routing-tables of node interest, and use these to determine whether to forward a specific Event to a particular segment. This mechanism has the potential to significantly reduce bus traffic on some segments, which may be particularly im~~n~~portant on slower bus segments.

The discussion of event routing is an example of protocol-specific routing of global messages, and the “Simple Protocol Subset” is another example. The message protocol is designed to allow more of these to be added later.

Global messages must be presented to and acted upon by the node sending them. For example, a message that requires global replies from all nodes with a certain condition must result in a reply message from the original sending node if that condition is true for that node. This enables passive condition monitoring by other nodes on the network.

A Gateway will need to have sufficient buffering to handle:

- translation from one transport-type to another;
- efficient buffering of messages, including multi-part messages from constrained transports, such as CAN;
- routing tables
- Alias caching, if necessary.

2.3.7 Delays and Timeouts

The protocols are all designed to have error-responses (error reply to datagram or Optional Interaction Rejected) instead of silently failing for directed messages. Timeout logic is only necessary to handle transport failures and failures of nodes to execute the protocols, including for example when they power off in the middle of some transaction.

When implementing the protocol in a node, one has to allow for a delay before receiving a reply and take this into consideration when arranging state machines and buffering. In addition, interactions from multiple protocols can overlap, and this also needs to be considered.

Any specific issues with timeout recovery can be discussed in the message-specific parts of the Technical Notes for specific protocols.

2.4 Simple Node Protocol Subset

This note describes the Simple Node Protocol. It is not normative on node implementors.

It is designed to reduce traffic on those segments that contain only self-declared Simple-nodes.

Simple nodes are defined as the leaf nodes that do basic layout control (input/output) operations. These nodes need to be able to[2]:

- Indicate their presence, by sending Initialization Complete Simple messages and Verified Node Simple messages;
- Send and receive event reports, via PCER messages and associated reports;
- Be configured, through related messages, datagrams, or streams;
- and perhaps other things in the future.

The messages mentioned above form the Simple Node Protocol subset (Simple-set) of the full OpenLCB protocols. Note that this is an asymmetric subset: simple nodes can send some types of messages but receive other types. In particular, they do not receive non-directed messages that do not have their Simple-bit set. These include the Consumer and Producer Identified messages, and this will reduce traffic, especially at system start-up.

On the other hand, gateways, configuration tools and other network-aware nodes such as sniffers and monitors are not simple nodes. These nodes need access to the full OpenLCB protocol so they can:

- Learn about the appearance of other nodes (by receiving Initialization Complete messages)

[2]Simple nodes also have to do whatever is needed to function with their specific wire protocol, e.g. send CID/RID frames on CAN.

- Learn about other producers and consumers (by receiving status messages)

and similar activities. To do this, they must be able to send and receive every type of message.

For ease of filtering, a specific bit in the MTI identifies the global messages which are needed by simple nodes. This bit allows OpenLCB to define new MTIs in the future and still include them in the Simple Node Protocol subset without having to modify existing nodes.

Gateways that are serving network segments that contain only simple nodes (e.g. single CAN segments) may suppress unaddressed (global) messages that do not contain MTIs without the Simple-bit set.

2.4.1 Protocol Description

This section describes the reasoning behind message choices in the current MTI definitions.

2.4.1.1 Messages Transmitted

Messages sent from Simple nodes are treated the same as any node.

2.4.1.2 Messages Received

Further to the Standard, Simple nodes need to receive each of these messages:

- Verify Node ID – because they need to reply to it;
- Verified Node ID – because it may be the reply to their own request, and it might be used to e.g. locate a node for delayed sending of status;
- Protocol Support Inquiry – because they need to reply to it;
- Identify Consumers, Identify Producers, Identify Events – because others will ask this of them, and they may need to reply;
- Learn Event – so they can be programmed;
- P/C Event Report – so they can take actions.

2.4.1.3 Messages Not Received

Further to the Standard, these are brief descriptions of why the following message types are not necessary for simple nodes:

- Initialization Complete: These are used to indicate that a node is newly available to the network. Simple nodes only care about their specific tasks, and by definition are not interested in the overall structure and availability of the network.
- Consumer Identified, Consumer Identify Range: These are of interest to gateways and configuration tools, but an individual producer does not need to know which (if any) nodes are consuming its produced events.
- Producer Identified, Producer Identify Range: These are of interest to gateways and configuration tools, but an individual consumer does not need to know which (if any) nodes are producing its consumed events.

2.4.1.4 Messages Directed at Gateways

In order to filter global non-Simple messages, Gateways must determine whether the target segment contains only Simple nodes. They can do this by monitoring Initialization Complete and Verified Node ID messages to see if they are of the Simple Set Sufficient variety, thus indicating that the sending node is Simple. If the segment is determined to contain only Simple nodes, then the Gateway can start filtering non-Simple global messages. Addressed messages and messages in the Simple-set would continue to be sent onto the Simple-only segment.

2.5 Gateway Processing

Gateways are responsible for:

- Routing messages from a source segment to one or more destination segments;
- Translating protocols from one transport to another;
- Optionally performing interest-based filtering of traffic.
- Adequate buffering to ensure sufficient bandwidth.

Gateways are responsible for ttransferring traffic from one transport to the another. Between two capable transports, this is can be as simple as a copy operation. For example, transfers between Ethernet segments is straight forward. If filtering is instituted, then the Gateway needs to record and maintain tables of events, so that only traffic which is of interest to the second transport is actually passed on. Translating to and from a less capable transport, such as CAN, also involves the recording and maintenance of shortened forms of Node Ids, and with CAN this means keeping a translation table between the general 48-bit node IDs and their 12-bit CAN-Aliases.

In order to do these things, as well as transfer the three work-horse message types of Events, Datagrams, and Streams, a Gateway requires meta-messages to obtain and transmit meta-data about the system and its segments. Messages such as the Initialization Complete, Verified Node ID, Identify Consumer and Identify Producer messages are used to build the Gateways' routing and filtering tables. For segments with only Simple nodes, many of these messages do not need to be sent to that segment, because none of the nodes are interested in them by definition.

In addition, some messages are specific to a segment and are never routed, but are confined to that segment of the network. Examples of these are the CAN specific messages to obtain unique node Aliases.

It will be useful to optionally route addressed messages to specific or all segments for snooper/monitoring support. This will necessitate messages directed to the Gateways themselves.

2.6 Expansion

OpenLCB is designed to allow expansion. This includes addition of new messages and new protocols, but also includes the extending of messages with additional data. Nodes made before any of these additions will, of necessity, simply ignore these additional messages, protocols or data.

Expansion past 16 bit MTIs for both global and addressed messages is possible. While it is easy to add another byte, it is harder to get optimal use from it. To preserve the CAN priority structure, a CAN-MTI of low and high priority level have been reserved. Expanded messages will use one or both of

these CAN-MTIs and can treat the start of the data (after the destination address et al, if present) as another byte of CAN-MTI. Nodes created before these expanded MTIs will just ignore them.

2.7 CAN Adaptations

2.7.1 Introduction

CAN is a very robust transport with good availability, excellent noise resistance, load characteristics, and good hardware support for error detection and correction. Its main limitations are its small frame (data packet) size, and its bus speed and length. 125 kbps was chosen as it allows CAN segments to be a useful length while maintaining a useful bit-rate. The CAN frame is also divided between a header and data-part. The header is used to implement automatic prioritization and arbitration on frames but this does impose the requirement that headers be unique. OpenLCB ensures this by including the source Node's Alias in every header.

The following table shows the general scheme used for the CAN mapping:

Message Type	CAN Mapping	Comment
Most messages	Single frame	Longer messages can use multiple frames, using the Only, First, Middle, and Last paradigm, similarly to Datagrams, carried in the first data-byte.
Datagram	First-, Middle-,…, Last-frames	Small Datagrams may be carried by a single Only-frames.
Stream	Stream-Data-frames	A common Stream-message will be mapped to one or more Stream-Data-frames.

To maximize payload space, several mappings were necessary: the Common MTI is reduced to a 12 bit CAN MTI, with additional bits being included when necessary; Node IDs are mapped to segment unique 12 bit Aliases; and special frame formats were developed to transfer Datagrams and Streams efficiently as multiple frames.

2.7.2 Intended Use

CAN is very useful for the small layout. As a layout outgrows a single segment, additional segments can be added with the use of a Gateway. Eventually, a faster transport can be used to link a number of CAN segments together – this assigns CAN the role of the 'last-mile' in telephone parlance.

2.7.2.1 References and Context

(Nothing to add to the standard)

2.7.3 Messages

2.7.3.1 Message Format

2.7.3.1.1 CAN Prefix Field

The CAN Prefix Field is used to indicate CAN specific messages, which are used to perform Link layer functions involving obtaining and managing CAN Aliases. See CAN Transfer documents for more details.

2.7.3.1.2 CAN MTI Mapping

Having different ~~types of~~ frame formats allowed the formats to be tailored to their purpose, and increase message density and efficiency. ~~Type~~Format 1 maintains the CAN MTI in the header to allow the possibility of hardware filtering on it, and includes the destination Alias in the data part, only if it is necessary. ~~Types~~Formats 2-5 and 7 place the destination Alias into the header, preserving more room in the data part for actual data transport.

Most MTIs will map to frame ~~type~~format 1. These carry their MTI, shortened to a 12 bit CAN-MTI, in the header, in the form 0x19mmmsss, where mmm are the CAN-MTI bits, and sss are the source Node's Alias. Global messages do not have a destination Alias, but addressed messages carry the destination Alias in the 12 low-order bits of the data-segment, as 0xfddd, where ddd is the Alias, see 8.3.1.2.

Frame ~~type~~formats 2, 3, 4, 5 (Datagram frames) and 7 (Stream Data) are special cases chosen for efficient processing of large amounts of data on CAN. They have their MTIs compressed to 3 bits, and are carried in the five high order bits of the header in the form 0b11mmm , where mmm is the frame-~~type~~format (ie 0x1A, 0x1B, 0x1C, 0x1D and 0x1F, respectively). These forms allow the source and destination Aliases to be located in the header, leaving the entire data-segment for other data transfer. See Datagram Transport and Stream Transport documents.

A 12 bit CAN-MTI short form is used until future expansion makes more necessary. The possibility of longer CAN-MTI values has been reserved, see below.

2.7.3.1.2.1 Future Expansion of CAN MTI

If MTIs are expanded to be larger than 16 bits for global and addressed messages, then it is relatively easy to take up another byte of the data-part, but it is harder to get optimal use out of it. To preserve the CAN priority structure, an MTI within each of the priority levels has been reserved. Messages with one of these MTIs will treat the start of the data-part of the message (after the destination address et al, if present) as another byte of MTI. Nodes created before the expanded MTIs need to be defined to just ignore these. Two CAN frame-~~type~~formats have been reserved for possible future expansion. Specifically, ~~type~~format-0 has the highest priority which may be useful in the future, say for a null- or pause-message to slow the segment down. Similarly, ~~type~~format-6 is reserved and has a lower priority.

[2.7.3.1.3] Global and Addressed Messages, CAN Frame ~~Type~~Format 1

The ability to form multi-part messages was included because often 6-8 bytes of data is not sufficient for many purposes. However, designers of protocols should use this feature prudently. Longer addressed messages are already served by Datagrams and Streams.

2.7.3.1.2.2 [2.7.3.1.3.1] Longer Messages / Fragmentation

If the implementers want to send longer messages, for example a OpenLCB Simple Node Info Protocol (SNIP) reply, then they have several choices:

- Send separately as multiple messages, in some agreed way;
- Send as a Datagram;
- Send as a Stream; or
- Send as an extended message. Intermediate Gateways only need to combine and send as many frames as they have buffering for. It is likely that they would use the same buffer pool as for Datagrams, and may need to send multiple message fragments to complete the transfer. The receiving node will know the protocol, and will have means to receive the message fragments.

For example, the Protocol Support Reply message uses fragmentation bits to break the message into a number of frames, if required, as illustrated below.

2.7.3.1.2.3 [2.7.3.1.3.2] Multi-frame Messages

Note that this encoding results in messages with less than 9 bytes of data can use the Only form, and therefore the initial nibble is 0. This applies to the large majority of general messages.

Example of Multi-frame message:
The general Protocol Support Reply message:
(0x0668, 0x456456789789, 0x123456123456, 0x19, 0x1122334455667788990011223344556677889900)
becomes:
[0x19668, 0x456](8) 0x1123, 0x112233445566
[0x19668, 0x456](8) 0x3123, 0x778899001122
[0x19668, 0x456](8) 0x3123, 0x334455667788
[0x19668, 0x456](3) 0x2123, 0x99

An implementation would need to either process each frame as it arrives or have sufficient buffers to hold it for later processing. The maximum length is chosen so that buffers can be shared between Datagram and General Message processing.

Because most frames are “only”, and to simplify reading the MTI bytes in these cases, the first and last bits are defined as active-0.

[2.7.3.1.4] Datagram and Stream Messages, CAN Frame ~~Type~~Formats 2-5 and 7

These formats are designed to include the Source and Destination Node ID Aliases in the CAN-header, allowing the full use of the CAN data-part for message data.

The following Table is only for convenience of the reader and completeness, refer to the appropriate OpenLCB document for definitive information.

29-bit CAN Header								
Field	**CAN prefix**		**Frame ~~Type~~Format**		**Destination ID/ddd[3]**		**Source ID/sss**	
Size & location	2 bits 0x1800,0000		3 bits 0x0700,0000		12 bits 0x00FF,F000		12 bits 0x0000,0FFF	
Value(s)	3		**2,3,4,5,or 7**		**Destination Node Alias**		Source Node Alias	
Up to 8-Byte Data-Part								
Byte#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Value	(data)	(data)	(data)	(data)	(data)	(data)	(data)	(data)

Table 1: CAN Datagram and Stream Format

2.7.3.2 States

(Nothing to add to the standard)

2.7.3.3 Definition of Specific Messages

(Nothing to add to the standard)

2.7.3.3.1 Initialization Complete

(Nothing to add to the standard)

2.7.3.3.2 Verify Node ID

The optional Node ID idea just serves as documentation of the intent of the request. This makes sense on CAN, as it allows other nodes to determine the mapping between Node ID and it Alias.
For example of node verification:

CAN: [CAN-MTI, srcAlias](length) dstAlias destinationID

Eg: ***[0xXX, 0x123](8) 0x0FED, 0xFEDCBA987654***

Note that this includes both the Alias and the full Node ID of the destination.

2.7.3.3.3 Verified Node ID

The node ID in the data is redundant on wire protocols that carry the full source ID, but can be very valuable for wire protocols that abbreviate ("alias") the source ID within the messages, such as in this section, as it allows other nodes to determine the mapping between a Node ID and its Alias.

[3] "ddd" and "sss" refer to the Destination Node Alias and the Source Node Alias, respectively.

2.7.3.3.4 Optional-Interaction Rejected

(Nothing to add to the standard)

2.7.3.3.5 Terminate Due to Error

Example: A node receives an addressed message with an unrecognized MTI, and it sends:
[0x04F9, 0xABC](2) 0x0123
The receiving node replies with:
[0x0068, 0x123](6) 0x0ABC 0x10 00 04 F9

The 0x0068 indicates Reject Optional-Interaction, note that it has its addressed-bit set. The ***0x04 F9*** is the original MTI. The ***0x10 00*** is the error code, which in this case indicates this is a permanent error.

2.7.3.3.6 Protocol Support Inquiry

(Nothing to add to the standard)

2.7.3.3.7 Protocol Support Reply

On CAN, the data-parts of the reply frames are fixed except for destination Alias, and that can be taken directly from the request frame. It is likely that the number of protocols will eventually exceed 48, and when that happens this reply will have to be a multipart message. For a discussion, see below.

Before the CAN message start/end bits were defined, the 0x00 00 00 00 00 0F bits in the Protocol Support Reply message were designed for expansion. Nodes were to ignore any frames with one or more of those bits set. That mechanism has been deprecated and all known implementations updated to remove it. It's possible that nodes implementing that early version will misinterpret Protocol Support Reply messages once those bits represent specific protocols, but bits in the first CAN frame are valuable, so rather than reserve bits for that unlikely case, we've just required that those (few) nodes be fixed.

2.7.3.3.8 Extensibility

Note that the system is designed so that additional functionality can be added by extending the data carried in a message. On CAN this may result in the fragmentation of a single Common Message into multiple frames. Since simpler, or earlier nodes designed before the extension, may not need not process the entire data included in the message, these nodes may return a complete or error message before the final frame-part is sent. The sending node must be able to process this 'early' result. This was chosen as it simplifies the receiving node, as it does not need to maintain state until the end of the transmission, it may ignore the remaining frame-parts.

To handle extensions, a node should on receipt of:

- a first or last frame with expected content – handle as expected;
- a first or last frame with additional content – handle original content as expected;
- a first or middle frame with additional content – handle content;
- a middle frame with only additional content, and you've already replied – ignore;

- a middle or last frame with only additional content, and you've already replied – ignore.

Nodes must be able to handle the early receipt of a complete or error message.

For example, consider the following Protocol Support Reply message:
[0x19668, 0x456](8) 0x1123, 0x112233445566
[0x19668, 0x456](8) 0x3123, 0x778899001122
[0x19668, 0x456](8) 0x3123, 0x334455667788
[0x19668, 0x456](3) 0x2123, 0x99
Older nodes may only need to process the first-frame, which would include most of the common protocols. The remainder of the frames would be ignored.

2.7.3.4 Interactions

Although there are no special provisions for the CAN Transport layer, it is useful to show some examples of how these messages can be used to manage nodes. The following examples show how to do a variety of useful functions.

2.7.3.4.1 Example: Finding all nodes

To find all nodes, send the global form with no node ID – each node will send a Verified Node ID message.

Example: Node 0x112233445566, Alias 0x123 sends:
[0x19490, 0x123](0) – global without Node ID;
and receives:
[0x19170, 0x345](6) 0x333444445555 – unaddressed Verified Node message from first Node;
[0x19170, 0x876](6) 0x817263544536 – unaddressed Verified Node message from second Node;
...

2.7.3.4.2 Example: Confirming that a specific node can still be reached

If you have the necessary information (I.e.Node Alias) use the addressed form, as it's less load on the entire system. Otherwise use the global form with the Node ID in the data.

Example: Node 0x112233445566, Alias 0x123 sends:
[0x19498, 0x123](8) 0x06E6, 0x999988887777 – addressed with Node ID;
and receives:
[0x19170, 0x6E6](6) 0x999988887777 – unaddressed Verified Node message;
The originating node confirms Node *0x999988887777* is reachable, and has Alias *0x6E6*.

Example: Node 0x112233445566, Alias 0x123 sends:
[0x19498, 0x123](2) 0x6E6 – addressed without Node ID;
and receives:
[0x19170, 0x6E6](6) 0x999988887777 – unaddressed Verified Node message;
The originating node confirms Node *0x999988887777* is reachable, and has Alias *6E6*.

Example: Node 112233445566, Alias 123 sends:

[0x19490, 0x123](6) 0x999988887777 – global with Node ID;
and receives:
[0x19170, 0x6E6](6) 0x999988887777 – unaddressed Verified Node message;
The originating node confirms Node 0x*999988887777* is reachable, and has Alias 0x*6E6*.

2.7.3.4.3 Example: Finding a full node ID *from* a local alias

CAN requires Node Aliases to send messages, and Verify Node ID can be used to get the full node ID from a local alias by sending the addressed form using that alias. Only the desired node will reply, the reply can be identified by the known alias in the source ID part of the message, and the full node ID will be in the data part of the message.

Example: Node 0x112233445566, Alias 0x123 sends:
[0x19498, 0x123](2) 0x0876 – addressed without Node ID;
and receives:
[0x19170, 0x876](6) 0x817263544536 – unaddressed Verified Node message;
The originating node determines that the Node ID for the Node with Alias *0x876* is *0x817263544536*.

2.7.3.4.4 Example: Node obtaining local Alias from full Node ID

CAN requires Node Aliases to send messages, and Verify Node ID can be used to get the CAN Alias assigned to a known Node ID by sending the global form with the full node ID in the data. Only the desired node will reply, that reply can be identified by the full node ID in the data part of the message, and the desired node's aAlias will be in the source ID part of the message.

Example: Node 0x112233445566, Alias 0x123 sends:
[0x19490, 0x123](6) 0x817263544536 – global with Node ID;
and receives:
[0x19170, 0x876](6) 0x817263544536 – unaddressed Verified Node message;
The originating node determines the Alias for Node *0x817263544536* is *0x876*.

2.7.3.4.5 Example: Confirming a local Node is still reachable

To see if a local Node is still reachable, the addressed Verify Node ID message can be used with the destination Node's alias and Node ID, and only the addressed Node will answer. If the use of that Alias has changed, the returned Node ID will not match that sent.

Example: Node 0x112233445566, Alias 0x123 sends:
[0x19498, 0x123](8) 0x06E6, 0x999988887777 – addressed with Node ID;
and receives:
[0x19170, 0x6E6](6) 0x999988887777 – unaddressed Verified Node message;
The originating node confirms Node with Alias *0x6E6* is reachable, and has Node ID *0x999988887777*.

2.7.3.5 Error Handling

(There is nothing to add to the standard)

2.7.3.6 Routing and Sequencing

The requirement that frames of a message be sent consecutively is meant to reduce and simplify buffering requirements. Note that this language does not require that the CAN frames be sent with no inter-frame idle time. It's strongly recommended that the frames be sent with no inter-frame idle time, because that can reduce and simplify buffering requirements even further, but that is not required by this Standard.

OpenLCB only has 4 priority levels, so ~~Although~~the nesting of higher-priority messages ~~can be sent~~ in the middle of a lower-priority message is restricted to 4 levels per sending source Node. Nodes initiating addressed interactions are generally in control of how many buffers they need to keep around for reassembling multi-frame responses.

In the presence of many potential source Nodes, such as in the situation of a Gateway, the number of buffers is not theoretically bounded. It is expected that handling a very large network requires a Gateway that is not memory-constrained, such as a Single-Board Computer (e.g. Raspberry Pi, BeagleBone)., ~~there are only N priority levels, so a receiving Node only need to provide N buffers to reassemble CAN messages.~~

2.7.4 Message Type Indicators

On constrained transports, such as CAN, the General MTI needs to be mapped to a smaller or different configuration. This mapping needs to be one-to-one. In the case of CAN, some of the MTI bits are converted to a frame type, and some are carried as the CAN-MTI. See the section 2.7.3.1.2 above.

Table of Contents