OpenLCB Working Note	
Stream Transport	
Apr 30, 2021	Preliminary

1 Introduction

A Working Note is an intermediate step in the documentation process. It gathers together the content from various informal development documents, discussions, etc into a single place. One or more Working Notes form the basic for the next step, which is one or more Standard/TechNote pairs.

Status of this Working Note: It's had info from various threads and HTML docs dumped into it, but it hasn't yet been organized at all (as of Oct 18, 2013).

1.1 Served Use Cases

1.2 Unserved Use Cases

Streaming is used to move large amounts of data (kilobytes and up) across a OpenLCB implementation.

This documents describes the streaming protocol(s) in OpenLCB.

1.2.1 Environment of Proposal

1.2.1.1 Requirements

Must work within OpenLCB architecture of a flat address space and unique identifiers

Receiving end must be able to throttle rate of arrival.

Multiple transfers at the same time over a single OpenLCB segment, so that transfers between disjoint nodes don't need to be coordinated.

Needs to be effective over CAN, with acceptable bit efficiency.

Streams are not required to provide real-time transfer, e.g. to move live sound.

1.2.1.2 Preferences

It should be possible to have multiple streams active at the same time to or from a single node. Nodes can choose not to support this.

Different buffers sizes can be used. High capability nodes can use large buffers for efficiency, low capability nodes can still function with small buffers.

Streams should have a way of identifying their content & purpose within themselves, in addition to through other protocols.

1.2.1.3 Design Points

Data integrity is handled by transport level, and need not be ensured in the streaming protocol.

A stream has a beginning and an end, and some number of bytes transferred in between. All other structures are implemented by higher-level protocols outside the streaming protocol or data fields within the streamed data itself. For example, the streaming protocol doesn't specify the format for sound files sent over streams.

A stream has no intrinsic block, or record structure.

Streaming is not needed for short transfers, so a certain amount of overhead is acceptable. (This implies a Datagram protocol to cover the gap between Events and Streams in size, see separate document)

We assume that routes don't change dynamically in ways that effect e.g. buffer allocation.

2 Specified SectionsAnnotations to the Standard

This is the usual section organization for a Technical Note, to accumulate the Standard and Technical Note content in its eventual order.

2.1 Introduction

Note that this section of the Standard is informative, not normative.

2.2 Intended Use

Note that this section of the Standard is informative, not normative.

2.3 Reference and Context

Lorem ipsum dolor sit amet, consectetur adipiscing elit.[1] Fusce ornare mattis justo vitae imperdiet. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

2.4 Message Formats

2.4.1 Stream Initiate Request

It carries a "Max Buffer Size" value (2 bytes), a “Type Included” boolean flag (1 bit in a byte; see below for meaning) and a "Source Stream ID" (1 byte) in the data section. The combination of source Node ID, destination Node ID and Source Stream ID must uniquely identify this stream transmission.

Source Stream ID, by itself, is not required to be unique.

Data content:

Byte	Byte	Byte	Byte	Byte	Byte
Max Buffer Size		Flags:	Additional	Source Stream	(Reserved)

[1] See the “Common Information” OpenLCB Technical Note for detailed conventions on bit and byte numbering. Briefly, the least significant bit of a field is numbered with zero in OpenLCB descriptions, but note that other technologies may use other conventions.

	LSB: Type Included 0: No type specification 1: First six bytes are unique stream content UID	Flags:	ID	

55

2.4.2 Stream Initiate Reply

Byte	Byte	Byte	Byte	Byte	Byte
Max Buffer Size		Flags LSB: Type Included 0: No type specification 1: First six bytes are unique stream content UID Bit 1: 1: Stream Rejected, out of order or other "should not happen" error 0: (nothing) Bit 2: Bit 3: Bit 4: Bit 5: Bit 6: Error Type (if Reject) 1: Permanent Error 0: Temporary Error MSB: Accept/Reject 0: Reject 1: Accept	Additional Flags If Reject LSB: 1: Information Logged Bit 1: Bit 2: Bit 3: Bit 4: if Reject, Permanent Error: Bit 5: 1: Invalid Stream Request Bit 6: 1: Source not permitted Bit 7: 1: Streams not accepted if Temporary	Source Stream ID	Destination Stream ID

		Error: Bit 5: 1: Buffer full Bit 6: 1: Internal error, out of order		

2.4.3 Stream Data Send

2.4.4 Stream Data Proceed

Byte	Byte	Byte	Byte
Source Stream ID	Destination Stream ID	Flags	Additional Flags

2.4.5 Stream Data Complete

Byte	Byte	Byte	Byte
Source Stream ID	Destination Stream ID	Flags	Additional Flags

2.5 States

A stream either exists or doesn't; those are states. There are also some un-named intermediate states during setup and take down.

2.6 Interactions

-Optionally, the destination can request that the source start a transfer. This uses some mechanism not discussed here, e.g. Datagram messages.

The source sends an "Stream Initiate Request (Addressed)" to the destination. It carries a "Max Buffer Size" value (2 bytes), a “Type Included” boolean flag (1 bit in a byte; see below for meaning) and a

"Source Stream ID" (1 byte) in the data section. The combination of source, destination, and Source Stream ID must uniquely identify this stream transmission.

If the destination node does not wish to receive the stream, it returns a "Stream Initiate Reply (Addressed)" marked “reject”, with a "Max Buffer Size" value (2 bytes) of zero, the “Type Included” boolean flag (1 bit in a flag byte), "Source Stream ID" (1 byte) and "Destination Stream ID" (1 byte). The message includes flags set to represent exactly one of several conditions:

“Permanent error” - This node does not process streams, will not accept a stream from this source, or for some other reason will not ever be able to accept this stream. The Stream Initiate Request should not be retransmitted. Optionally, the node can mark the reply with one or more of several conditions:

“Information Logged” - the node supports the logging protocol and information was logged for later retrieval.

“Invalid Stream Request” - something made the stream request improper, such as longer than the max permitted length. Proper requests might be acceptable.

“Source not permitted” - streams from this source will never be accepted

“Streams not accepted” - this node will not accept stream requests under any circumstance. A node can also reject the interaction instead of sending Stream Initiate Reply with this code.

“Buffer shortage, resend” - The node isn't able to receive the stream because of a shortage of buffers. The sending node should resend at its convenience.

“Stream rejected, out of order” - Should not happen, but an internal inconsistency was found in the CAN frames making up a stream. Sender can try again if desired.

If the destination node wishes to receive the stream, it returns a "Stream Initiate Reply (Addressed)" marked “accept”, with a "Max Buffer Size" value (2 bytes), the “Type Included” boolean flag (1 bit in a byte), "Source Stream ID" (1 byte) and "Destination Stream ID" (1 byte). The "Max Buffer Size" is less than or equal to the value in the Initiate Stream Request, and is the negotiated buffer size for this transfer. If it's zero, the request to start the stream has been rejected, and the exchange is over. The source can try again later.The Source Stream ID is the same as the value in the Initiate Stream Request, and is returned for identification and the convenience of the source. The destination doesn't do anything with it except return it. The source can use it to match up multiple operations, as a way of identifying buffers, or for any other purpose.The Destination Stream ID is used to tag the data sent to the destination. It has no meaning to the source. The destination can, but need not, use it to associate the stream data with a particular buffer or usage. Multiple simultaneous streams can use the same Destination Stream ID value.

The source starts sending bytes using Stream Data Send (Addressed) messages, each carrying up to the message size limit on the particular wire protocol and the Source and Destination Stream ID. After sending exactly Max Buffer Size bytes in one or more messages, the source pauses.

If the “Type Included” flag was true during initialization, the first 6 bytes of the data are a unique data-type indicator. These are allocated the same way that Node IDs, etc, are allocated, but from a separate name space. If that “Type Included” flag is false, some higher-level protocol must identify the data-type of the stream data. The idea is that a stream is a lot of data; there's not much use for one otherwise because of the setup overhead (code and time). So six bytes for a stream type identifier isn't a large cost

(unlike e.g. a datagram, where it would be a 10% overhead). A UID as a Stream type ID has the advantage that it's not ever going to collide, so people developing protocols don't have to coordinate it (e.g. useful for future expansion, where a protocol can be locally developed and then deployed more widely). And it still fits in the 1st CAN Frame of the stream, which simplifies what nodes have to do to figure out what type of (unsolicited) data this is. On the other hand, streams have a stream ID, so that a protocol can use some other mechanism (messages, datagrams, or even other streams) to pass the information about what a specific new stream means. That means the type info at the start of the stream isn't as necessary as in e.g. datagrams. (We really wanted to avoid situations like "The next datagram you receive carries the data for this request", because "next" is a hard concept to ensure in code that's doing several things independently)

On CAN, the Stream Data Send message does not carry the Destination Stream ID field. Messages to CAN get split into frames and forwarded without the field. Messages from CAN have to have it inserted by the gateway as part of the translation process. The gateway can do this as part of the (optional) buffer accumulation from frames into larger transfers.

Upon receiving Max Buffer Size bytes via one or more messages, the destination sends a Stream Data Proceed (Addressed) message to the source, carrying the Source Stream ID and Destination Stream ID. This tells the source that there is enough buffer space available that another Max Buffer Size bytes can be sent. The destination can also set flag bits (error codes) in this message to indicate that no more data should be sent and the transmission should be terminated early.

The destination may, but is not required to, send a Stream Data Proceed (Addressed) message to the source before receiving the full count of Max Buffer Size bytes. In that case, the source does not stop transmission after sending Max Buffer Size bytes, but continues for an additional Max Buffer Size bytes of transmission.

When the last data has been sent, the source sends a Stream Data Complete message carrying the Source Stream ID and Destination Stream ID to indicate that all data has been sent. Optionally, this can carry a four-byte length of the stream data transferred for checking. A zero value, if present, means the length is unknown. When this message is received by the destination, the transfer is complete.

3 ~~Section Title~~Background Information

3.1 Stream Content Type Identification

Stream senders can, but don't have to, identify the content type (format, meaning, etc) of the data stream at the front. As part of their private protocol, they can use whatever structure they'd like for the data.

General content types can be defined using unique identifiers. The “Type Included” flag identifies that the first eight bytes of stream data are to be interpreted as stream content type information. These can be allocated via the same mechanisms as EventIDs. Well-known ones will be published by OpenLCB. These identifiers are recorded in a separate worksheet.

3.2 Buffer Size

The buffer size can be up to 216-1 bytes = 64KB-1. This is driven by the size of the initiate messages, which we want to keep in a single CAN frame. Although some transport mechanisms might be able to

profitably use larger buffers, it seems unlikely that a layout control bus will need to have a higher message efficiency or lower latency than this size will allow. Should that turn out to be necessary in the future, an alternate form for the stream initialization messages can be defined for those large-message nodes. (It's unlikely they're running over CAN)

3.3 CAN Discussion

On a CAN segment, the data limit is 8 bytes. This requires use of as little control information as possible, so that as many of those eight bytes can be used for data as possible.

OpenLCB/S9.6 CAN frames can hold the source and destination aliases in the header. A dedicated "stream data transfer" format can also be coded entirely in the header. This then allows 8 bytes of data per transfer if the Source Stream ID and Destination Stream ID need not be carried. By specifying that only one stream can be transferred on a CAN link between any two nodes at a time, no Stream ID would be needed. Although initially consider, this it thought to be too inflexible. Instead, we chose a format where the Source Stream ID is carried, reducing the payload to seven bytes per frame. This requires that the Source Stream ID be unique among source-destination node pairs, without relying on the Destination Stream ID to be part of the unique stream identification. Since a node knows that it's originating the stream, this can be ensured in advance for point-to-point connections. Gateways will need to track the Destination Stream ID to restore it to the full format. Since they are tracking the stream in any case for buffering purposes, this is thought to be not a large problem.

3.4 Implementation Notes

The "Stream Data Proceed" from the destination is clearance to send another buffer-size-worth of data. To achieve better performance, the destination can send it before receiving the entire buffer-size-worth of data, as soon as it has room to receive what's already been OK'd plus one more buffer size. For example, a destination with a 4kB buffer could reply with Max Buffer Size of 2K, followed by an immediate Stream Data Proceed, to do single overlap of the transfer.

Intermediate nodes need to be able to handle transfers, and therefore need permission to lower the Max Buffer Size on the outbound Stream Initiate Request message. The length in the returned Stream Initiate Reply can't be changed, as the destination need to know when to send clearance for another buffer's worth of data.

A CAN ↔ Ethernet bridge might receive several kilobytes of data from the Ethernet side at a time. It's then responsible for breaking that up into CAN frames and forwarding it. It can reduce the buffer size of transfers if need be to ensure there's a place to store the data while this is done.~~Quisque sollicitudin tempor bibendum. Donec consectetur condimentum sollicitudin. Sed dignissim velit id felis lacinia at eleifend nisi laoreet. Vivamus tristique porta ornare. Vivamus feugiat dolor id lectus aliquet luctus.~~

~~4 Section Title~~

~~4.1 Subsection Title~~

~~4.1.1 Sub-subsection Title~~

~~Praesent gravida pulvinar vehicula. Vivamus eu nisl eget sem rutrum suscipit id a magna. Integer id ante odio.~~

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Sed consectetur, ipsum et egestas accumsan, felis erat ornare mauris, sit amet suscipit ante orci sed massa.

5 Section Title

Quisque sollicitudin tempor bibendum. Donec consectetur condimentum sollicitudin. Sed dignissim velit id felis lacinia at eleifend nisi laoreet. Vivamus tristique porta ornare. Vivamus feugiat dolor id lectus aliquet luctus.

Quisque sollicitudin tempor bibendum. Donec consectetur condimentum sollicitudin. Sed dignissim velit id felis lacinia at eleifend nisi laoreet. Vivamus tristique porta ornare. Vivamus feugiat dolor id lectus aliquet luctus.

Quisque sollicitudin tempor bibendum. Donec consectetur condimentum sollicitudin. Sed dignissim velit id felis lacinia at eleifend nisi laoreet. Vivamus tristique porta ornare. Vivamus feugiat dolor id lectus aliquet luctus.

Quisque sollicitudin tempor bibendum. Donec consectetur condimentum sollicitudin. Sed dignissim velit id felis lacinia at eleifend nisi laoreet. Vivamus tristique porta ornare. Vivamus feugiat dolor id lectus aliquet luctus.

Quisque sollicitudin tempor bibendum. Donec consectetur condimentum sollicitudin. Sed dignissim velit id felis lacinia at eleifend nisi laoreet. Vivamus tristique porta ornare. Vivamus feugiat dolor id lectus aliquet luctus.

Table of Contents