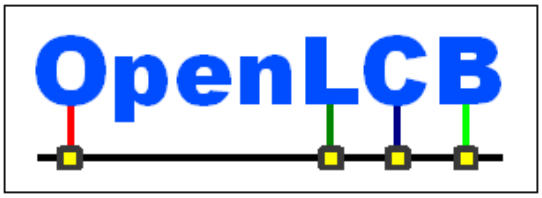

OpenLCB Technical Note	
OpenLCB-TCP Transfer	
Nov 29, 2013	Preliminary

1 Introduction

This explanatory note contains informative discussion and background for the corresponding "OpenLCB-TCP Segment Transfer Standard". This explanation is not normative in any way.

This section describes carrying OpenLCB™ across a transport mechanism that is reliable (all bytes are guaranteed to arrive in their original order) and can carry binary bytes.

Examples of this include:

- TCP/IP point-to-point links
- High-bandwidth wireless links
- Program-to-program links within computers

Reliable binary links can carry the full OpenLCB protocol. Messages are carried in full common format.

The reliable binary wire protocol is meant to attach multiple segments of the entire OpenLCB instance. Using this protocol, messages can be moved from one gateway to another, where one or more additional OpenLCB nodes can be reached through the gateway(s).

It is meant for applications from attaching a single remote console to interconnecting large modular layouts totaling 500 or more segments and 10's of thousands of devices. As such, it must have no addressing limitations beyond those built into the OpenLCB common message format and node serial number format.

This wire protocol makes no assumptions about the available bandwidth. In particular, it must work, with perhaps limited effectiveness, when the link bandwidth is small compared to the capacity on the other side of the gateways.

No assumption is made that this wire protocol is being used over point-to-point connections. Broadcast and multicast connections between nodes via this protocol are also permitted.

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

2.2 References and Context

2.3 Message Format

The various reserved bits are allocated to preserve options for future expansion. New distinct types, that don't overlap with the existing types, can use the “check for zero” bits. New modifier codes are more likely, so more bits are reserved where existing implementations just ignore their content.

Flags field:

The LSB 0x0001 is an expansion flag used to (eventually) add link control messages. These might be needed for e.g. turning gateway-filtering on and off, loading/resetting remote diagnostic information, etc.

The 0x0002 bit indicates “chaining”: Another prefix is next, before the body of the message.

The 0x0020 and 0x0010 bits provide a way of extending a single OpenLCB message across multiple TCP transfers. (Note that the terminology here is confusing, and should probably be clarified). This might be used to e.g. simplify buffering and framing in CAN-Ethernet gateways. A single OpenLCB message that extends across multiple CAN frames is best accumulated into a single TCP transmission. If that's not possible due to buffering or other issues, it can be sent as multiple TCP transmissions using the 0x0020 and 0x0010 bits. The 0x0020 bit when 0 means “start or only part”; the 0x0010 bit when 0 means “end or only part”. The 0-based coding was selected to improve human-readability of messages, as most messages are expected to be “only”.

The length code is put early to make it easier to parse messages of future, unknown format. But this doesn't make it easier to skip new prefixes of unknown length. Should there be an additional prefix length field, so new prefix formats can be skipped easily?

2.4 States

We don't have a “link up” state in OpenLCB itself, just in the link implementation. Is one needed? Might enter e.g. routing. If it existed, we could define link-specific messages to check the status.

2.4.1 Multiple-part Messages

2.5 Interactions

The chaining process is meant to provide a bread-crumb path back through the network. OpenLCB networks are not meant to have very complicated routing structures, so that only a few prefixes will get added to the average message.

2.6 TCP-specific Link Control Transmissions and Interactions

Format

2.6.1 States
2.6.2 Interactions

3 General

Development of link-specific messages and interactions

3.1 Monitoring of other protocols

There's a need to be able to pass non-common OpenLCB messages over binary links. For example, a CAN-Ethernet bridge may want to pass CAN frames through to the Ethernet unchanged for monitoring purposes. This includes both the CAN frame(s) that correspond to OpenLCB common messages, and also other frames that are used for alias reservation, etc.

3.2 Gateway-Gateway Messages

The TCP/IP gateway to gateway links effectively partition the overall OpenLCB into smaller "segments". OpenLCB nodes in any of these must be able to communicate with each other via any OpenLCB common message, including over intermediate wire-protocols that differ.

Gateway-to-gateway messages (will be) defined for control operations to make this communication transparent. These messages are used to carry information between gateways on a link, and do not propagate beyond that.

Control of gateway nodes is done via direct messages to those nodes.

Note that these are not available on e.g. CAN links.

Possible uses:

- Redirecting point-to-point links: “I'm going offline now, you should connect to him”
- Negotiating buffer sizes and traffic levels; flow control.

Control messages to individual nodes, not gateway-gateway communication, will be used for:

- Turning filtering on and off in gateways so that all messages can be forwarded
- Turning gateway message logging on and off
- Retrieving status and history information

3.3 Rendezvous & Self-Configuration Method

(Some or all of this needs to move to the Standard)

In the following, “node” means an OpenLCB node with a native TCP connection. This node might be e.g. a gateway to an entire CAN segment, but that doesn't matter to this discussion.

Two types of connections could be provided: “point-to-point” and “hub-and-spoke”.

In “point-to-point”, two nodes are connected by a direct TCP connection. This works fine for e.g. two nodes, but using only point-to-point connections doesn't scale well. If there are 10 nodes present, each

has to maintain 9 TCP sessions, which is rapidly becoming impractical. On the other hand, it's the simplest possible TCP code.

In the “hub-and-spoke” connection, one node acts as a “hub” and all others connect to it as “spokes”. Each spoke node sends traffic to the hub, and the hub then forwards it to the N-2 other nodes.

3.3.1 Protocol

When a node comes up, it enquires for a “_openlcb-hub._tcp.local.” service. If found, connect to it.

If not found, and if the node can be a hub (it's not required that all nodes be able to do that), it configures itself to act as a hub and advertises the “_openlcb-hub._tcp.local.” service. Since all that takes time, it then checks again for another node advertising to be a hub. If found, it drops back to connecting to that and removes its service if the connection succeeds.

If all that fails, start advertising as “_openlcb-node._tcp.local.”. Then, check for any other node(s) advertising as “_openlcb-node._tcp.local.” and connect to as many as possible of the ones that are found.

4 Relationship with Native-Ethernet Connections

Imagine you have five OpenLCB CAN segments, connected by small boxes to an Ethernet: You can use an Ethernet-specific protocol (e.g. ala Ethertalk, etc), or you can use IP connections (probably TCP/IP).

The TCP/IP solution seems to require more overhead, but it works about the same on a single Ethernet segment and across multiple segments, wide area links, etc.

The Ethernet-protocol solution seems somewhat simpler (though getting a protocol certified is a lot of work), but can't get traffic off the local Ethernet segment. It spans hubs, but not switches, and would require another protocol for OpenLCB-specific devices to connect to the larger Internet. So although an Ethernet-specific protocol seems simpler to create, in practice you'd still need to create Ethernet to Internet routers, resulting in an overall increase in engineering effort.

Where Ethernet-specific protocol might be useful is in reducing the configuration load for single-segment implementations. It would be nice if people could plug e.g. five OpenLCB CAN-Ethernet adapters into a single Ethernet and have them find and configure themselves to interconnect. This doesn't require new OpenLCB-specific protocols; it's much more likely that a robust solution could be created using e.g. Bonjour (a form of ZeroConf) and similar standard protocols. Some recommendations here would improve interoperability of implementing devices.

4.1 CAN-ASCII Formatted Protocol

We have also used a protocol where GridConnect-format CAN frames are directly sent back and forth to an Arduino node, which places them on the CAN bus. This is essentially an Ethernet-CAN replacement for the popular USB-CAN adapters. These advertise themselves as “_openlcb-can._tcp.” protocol, by convention on port 12021.

Table of Contents

