@ECHELON

Network Management in Rail Transit Applications Using LonWorks® Control Networks

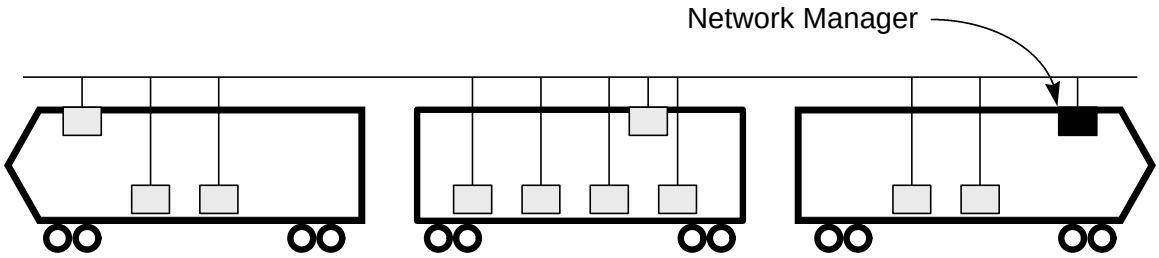
Alex Chervet
Echelon Corp.
June 1998

Automatic Configuration Upon Coupling and Recoupling/Changes

Trains fall into a network management scenario known as *continuous installation*. There is no set time when it is safe to assume that the topology of the train is fixed. Train cars may be added, removed or swapped at any time.

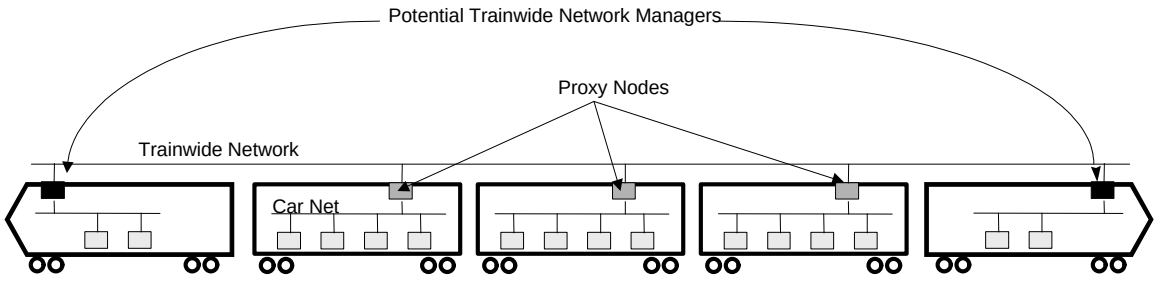
Managing nodes on a train can be accomplished in two ways:

1. *A single network manager manages every node in every car on the train*. The advantage of this scheme is that some message types may travel more quickly**.** However, this setup makes it more difficult to implement redundancy, does little to isolate local inter-car traffic, precludes data encapsulation, and costs significant time when a trainwide reconfiguration must be performed.



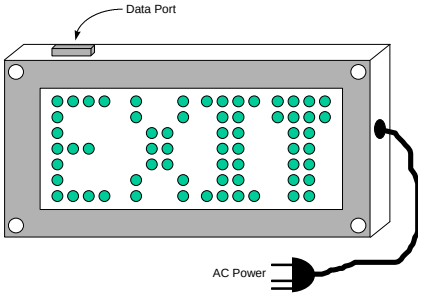
2. *Multiple network managers arranged hierarchically manage subsets of the network.* In this scheme, each car has a local network manager to manage the nodes within the car, and a trainwide network manager manages the dynamic nature of the train.

   We view this hierarchical approach as the superior solution. It offers greater configuration speed, provides complete redundancy, automatically isolates inter-car traffic, and allows the construction of software objects that very closely resemble the train's physical objects. Furthermore, data encapsulation is automatically implemented.

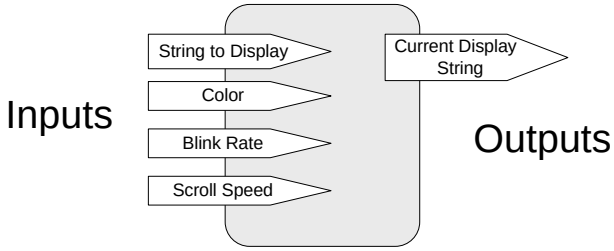


APPLYING THE CONCEPTS OF OBJECT-ORIENTED PROGRAMMING (OOP) TO NODE DEFINITION

Most "nodes" within a train car represent a single device: a door, a thermostat, a display panel, etc. Devices have hardware inputs and outputs. For example**,** a display panel has a voltage source and data-feed input hardware, and a flip-dot" or LED matrix output hardware. The physical inputs and outputs are clearly visible:



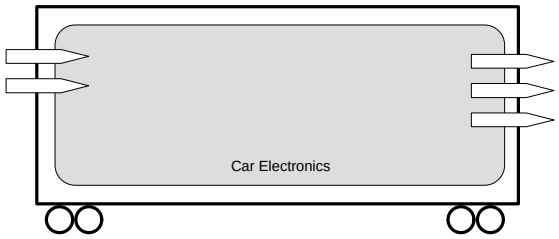
In order to ensure that the sign displays the proper message in the correct color at the proper scroll speed, the user must send the correct information to the sign's data port. Just as the sign exhibits *physical* hardware input/output connections, the software that controls the sign exhibits *logical* input/output connections. Those logical connections are encapsulated within the sign. The user simply needs to understand which inputs and outputs the sign can process. So, from a software perspective, the sign might look like this**:**



The above diagram represents the device's *interface*. By standardizing a device's interface, we provide a working software specification comprehensible to any vendor.

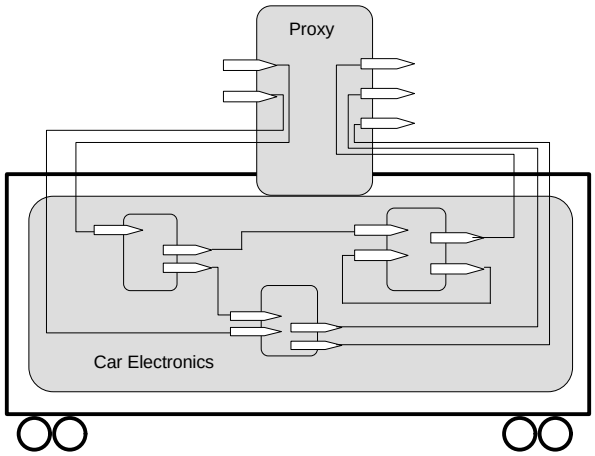
Since LonWorks provides standard data typing for such interfaces, it is possible to mix and match devices from multiple vendors without the need for special tools or gateways. Because of the standard software interface, components become interoperable. This greatly simplifies long-term maintenance and service personnel training. Transit properties and car suppliers can choose "best-of-breed" components from any company that adheres to the interoperable specification.

This encapsulation technique can be applied to an entire train car through the use of a "proxy" node**,** which establishes the external interface to the car. All communications between cars, or between a car and the locomotive, occurs through the proxy node.



While it is possible that a train car contains exactly one node (the "Car Electronics" in the diagram above), it is more likely that a train car will be composed of several nodes. Any information that these several nodes wish to share *outside* of the car must pass through the proxy node. Regardless of how the car is built internally, the proxy node maintains a consistent standard interoperable interface. This allows a transit property to precisely specify the logical interface to a train car.

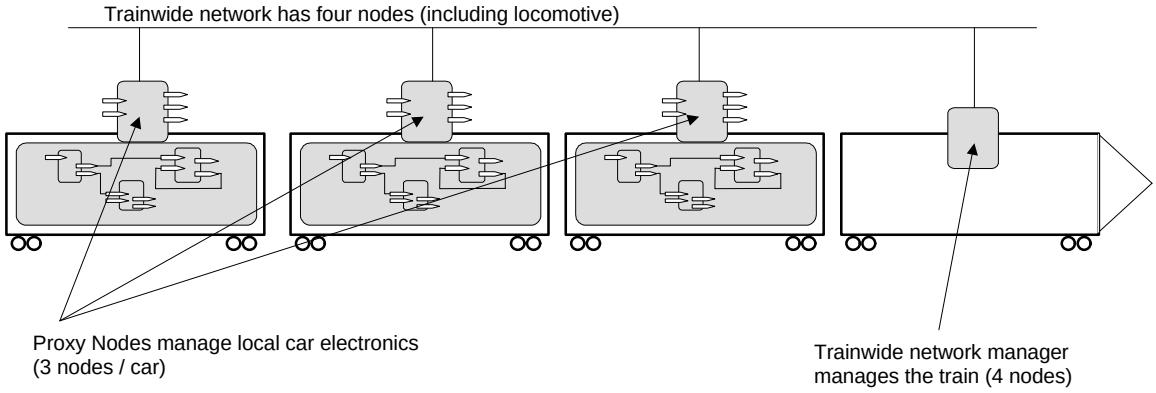
The result is that cars created by multiple vendors can be linked together transparently to form a train.



From a network management perspective, this simplifies the view of a train car down to a single device. Instead of the network manager in the locomotive having to manage every device on the train, it only has to manage one device per car. The network management problem is effectively partitioned into devices within a car, and cars within a train.

Devices within the car may be managed by a local network manager or by the proxy node itself. This is purely an implementation question. For this discussion, let us assume that the proxy node manages the "in-car" devices (nodes).

The "proxy node" provides another less obvious advantage. It has the capability to aggregate traffic destined for the locomotive's train operator status panel. This reduces bandwidth utilization on the train backbone.



MANAGING NODES

In any network, there must be some core of logic that specifies which node is connected to which other node. These connections may take the form of peer-to-peer connections, master/slave connections, or a combination of both. LonWorks provides facilities to automatically connect and configure such networks.

Let us assume the following system design characteristics:

1. Train management will occur from locomotives only. Any locomotive can potentially become the train network manager; however**,** a given train will never have more than one network manager, even if it has several locomotives. The locomotive **with** the motorman's key inserted is the network manager. *(Note: Although the following scenarios are described in the context of locomotives and **cars,** it is recognized that propulsion equipment may reside anywhere throughout the train, and that the operating position may be located in a passenger car. The scenarios described in the following pages are equally valid if any car can become a network manager, and if it is identified as such with the motorman s key.)*

2. A train operator may not move the train without first inserting his key. It is assumed that the motorman is always at the front of the train.

3. Each train car is equipped with a proxy node as described above. The proxy node serves four principal functions:

    a) It encapsulates all devices in the car, thus presenting a standard interface to the locomotive

    b) It acts as a car network manager, establishing and maintaining the logical interconnections of nodes within each car

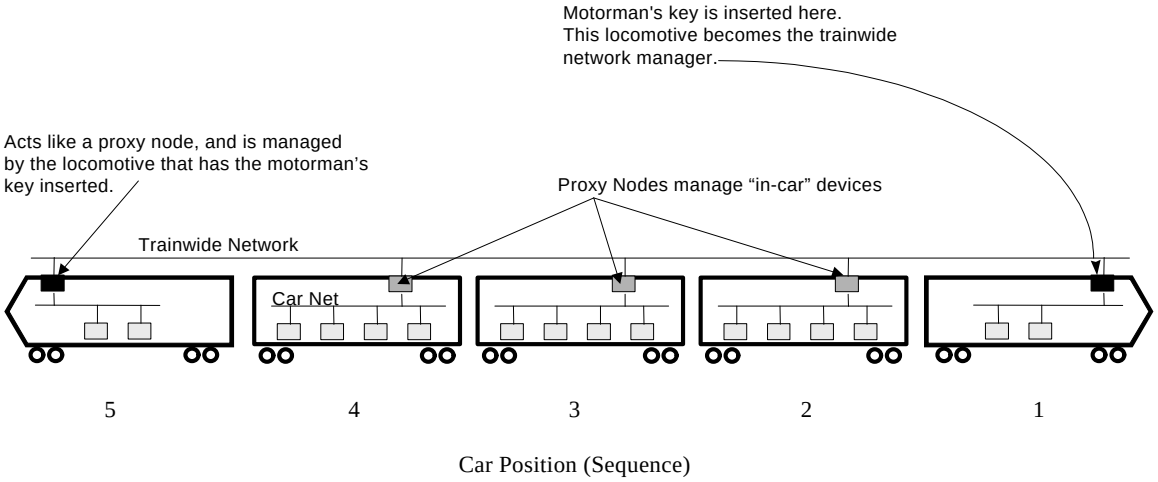
    c) It serves to aggregate traffic that must pass from within the car to the locomotive's train operating status panel

    d) It implements the train sequencing logic and detects car orientation

4. When a train is sequenced, the locomotive at the front of the train (where the motorman's key is inserted) is considered car position 1. The next car is position 2, etc.

5. If there are several locomotives, and therefore several potential trainwide network managers, the locomotive with the motorman's key inserted will assume the duties of managing the train. Other locomotives are available as "backup" managers in the event of a failure.

6. There is a maximum allowable limit to the number of cars in a train. The train operator's status panel is designed to handle the maximum number of allowable cars, and dynamically adjusts its display screens depending upon the number of cars actually in the train.



Initial Power-Up

When the train operator's key is inserted into the locomotive, it activates the trainwide network manager in that locomotive. The network manager checks to see if it is still attached to the same train it was attached to the last time a key was inserted. This is a simple matter, since all network managers keep a database of their current network in nonvolatile memory. Car sequencing follows:

1) Power is applied to a train.

2) The train network manager initiates a signal that is read by all subsequent cars to determine relative position and car orientation. (Remember that we have defined the locomotive as the first car in the train.)

3) Each car's proxy node reads the locomotive's signal and determines its relative position and orientation within the train relative to the locomotive.

4) Each proxy node applies a connection matrix to the internal car devices. Note that all proxy nodes can apply their matrices simultaneously. This reduces the total time required to configure a train. Also, note that in most cases, the proxy node does not need to configure devices within its car**,** since devices retain their connection information across power cycles. It is only when a new device has been added or removed that the proxy node must actually apply its matrix.

5) At power-up, a locomotive has no way of knowing which cars it has been attached to, so the locomotive's trainwide network manager must discover the cars in the train. This process is supported by the network management messages built into the Neuron Chip firmware. Each discovered car's proxy node reports car location within the train relative to the locomotive. The locomotive's trainwide network manager assigns a logical address to each car it discovers and keeps a table to correlate car location (sequence) and logical address.

NOTE: Only the locomotive with the motorman's key inserted initiates a discovery process. Locomotives without a motorman's key act like proxy nodes and simply report position. This is necessary to support trains with multiple locomotives.

Optimizations

Just as devices (nodes) maintain connection information, network managers maintain a database of all devices under management. This database is persistent across power cycles. Database persistence allows for optimizations under several circumstances:

No Change in Train Configuration Since Last Power-Up

The first step in the trainwide network manager's discovery process is to check that all cars that were under management on the previous power cycle still exist and report the same sequence number. This indicates that no cars have been moved from their previous position.

The next step is to discover any additional cars. If cars are discovered**,** it indicates that the train has "grown" since the last power cycle. If no additional cars are found**,** it indicates that this locomotive is connected to exactly the same train as the last power cycle. In fact**,** this is the usual case for transit trains.

Since nothing has changed, the trainwide network manager does not have to reconfigure any devices (cars).

New Cars Added to the Train Since Last Power-Up

The first step in the trainwide network manager's discovery process is to check that all cars that were under management on the previous power cycle still exist and report the same sequence number. This indicates that no cars have been moved from their previous position.

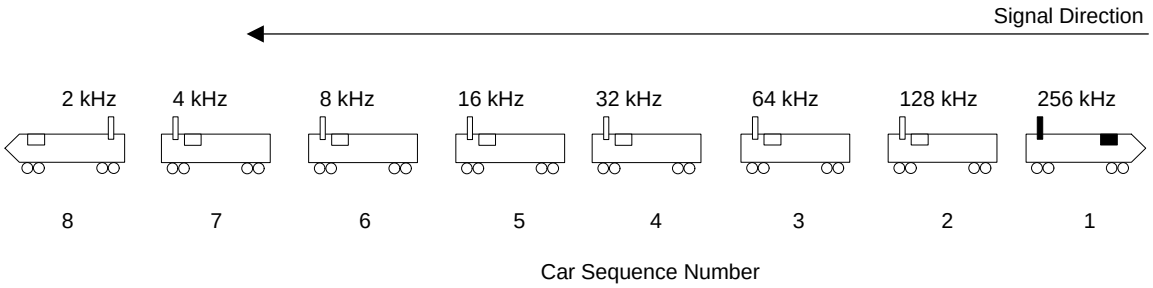
The next step is to discover any additional cars. If cars are discovered**,** it indicates that the train has "grown" since the last power cycle.

In this case**,** it is only necessary for the trainwide network manager to commission the newly discovered cars.

CAR SEQUENCING AND ORIENTATION

Several solutions exist for sequencing train cars following initial power-up. A discussion of exactly how each solution is implemented is beyond the scope of this document; however**,** for the purposes of discussion**,** let us assume the following:

1. Any locomotive is capable of generating a 256-kHz square wave signal.
2. Each proxy node can read the signal and halve it before allowing the signal to pass to the next car
3. A maximum of 8 cars is allowed in the train



This allows each proxy node to determine its location relative to the locomotive. A proxy node that reads a 32-kHz signal (before halving) knows it is the **fifth** car in the sequence. Car position is computed as

$$\text{Car Position} = \log_2 (256 / f) + 1$$

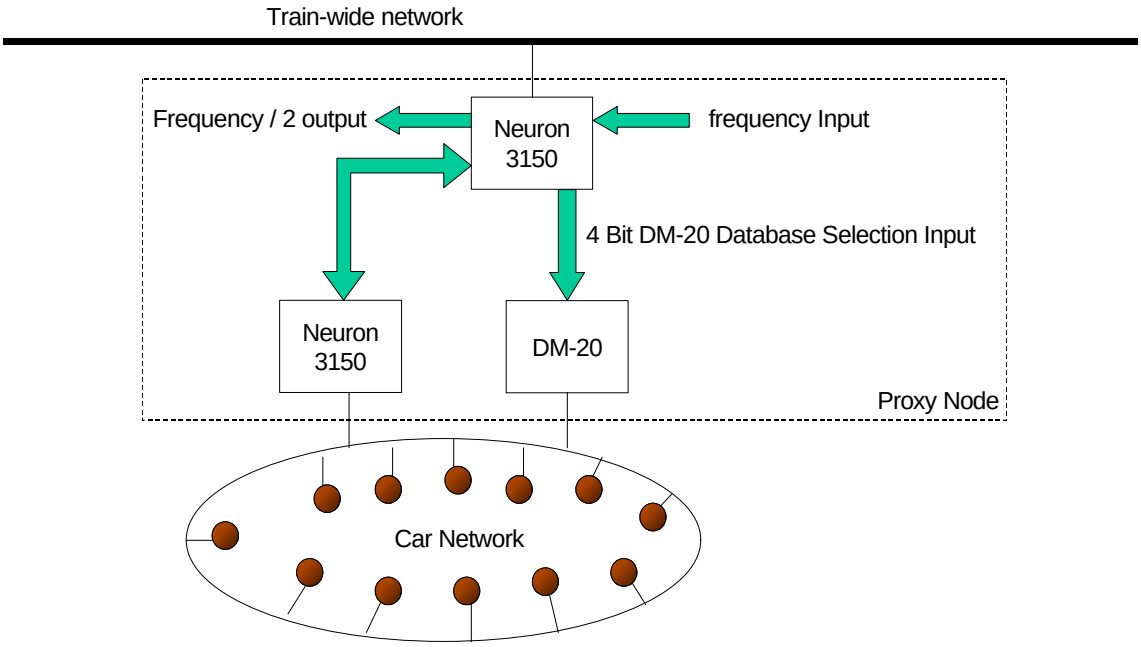
Where f = the observed frequency (before halving)

*** One can imagine a train sequencing system based on analog voltages or relays, equivalent to the frequency divide idea presented above. The frequency divide process is presented here simply for discussion purposes. This specific approach may or may not be correct for a specific train implementation. Each approach has advantages and disadvantages that should be considered within the design parameters of a given project.*

An innovative solution might combine data communication wires and train sequencing wires, thereby reducing the required number of train lines.

Possible Proxy Node Design Using Off-the-Shelf Components from Motorola, Toshiba, and Echelon

Several off-the-shelf components may be useful in creating a proxy node. A simple solid design would include a DM-20 component, a Neuron Chip connected to the car network, and a Neuron Chip connected to the trainwide network. Optionally, a higher order processor could also be used, although it is probably unnecessary.



In the diagram above, the Neuron Chip connected to the trainwide network is responsible for figuring out where it is within the train (based on frequency input) and reporting car position to the network management device in the locomotive. This Neuron Chip also supplies a database input to a DM-20 device. (Please refer to Echelon's website at www.echelon.com or Echelon's product catalog for complete information on the DM-20.) The input frequency is divided by 2; and the new frequency propagates down the train line to the next proxy device. (Note that additional hardware may be required.)

The Neuron Chip connected to the car network aggregates and passes information to the Neuron Chip connected to the trainwide network via parallel or serial connection. Information can also flow from the trainwide network to the car network.

The DM-20 manages all devices in the car network. This includes node discovery, commissioning, presence detection, fault monitoring, and binding network variables. The DM-20 also maintains an internal event log that can be extracted by portable test equipment when performing network diagnostics on a car.

Conclusion

It is important to separate the task of sequencing the train from the task of managing nodes on the network.

Once a train has been sequenced, the architecture described above allows the following benefits:

- Precise node definitions are possible because of the built-in type declarations provided by LonWorks.
- The LonMark Interoperability Association acts as a neutral third party to help with node definition and specification. Further, the LonMark organization maintains a database of device profiles that can be referred to when updates or changes must be made to a system.
- A trainwide network manager can verify that no cars are "missing" in the train by checking that all sequence numbers are discovered. (Depending on the implementation, it may be necessary to add an "end of train" device to mark the end of the train.)
- Proxy nodes can be "swapped" from car to car. (Each time a proxy node is powered, it reads the frequency of the square wave to determine its car position.)
- In-car devices can be swapped from car to car without the use of any special tools. (Since proxy nodes have the ability to manage a local network, "swapped" devices are automatically detected and integrated into the local car network using the proxy node's connection matrix.)
- Large amounts of information can be passed from cars to the locomotive's train operating panel because the proxy nodes can aggregate many individual in-car messages into a single large message. This greatly reduces the bandwidth utilization on the train backbone.
- Allowing the proxy nodes to meter traffic can further optimize backbone bandwidth utilization. (It may be that a man/machine interface does not need as many updates per second as a peer device.)
- Individual cars can be serviced without the benefit of a locomotive because each car is individually managed by its proxy node.
- The time required to initially configure a train is reduced because each car has a network manager; and all network managers can work in tandem.