Ten Problems with Using Echelon's LonWorks Technology for Embedded Control Applications

November 27, 1995

Philip Koopman
Bhargav Upender
Alex Dean

UNITED
TECHNOLOGIES
RESEARCH
CENTER

Executive Summary

When one of our customers needed an off-the-shelf embedded communications network, we at first recommended Echelon's LonWorks technology, featuring the LonTalk protocol. LonWorks has the advantage of being a relatively inexpensive, off-the-shelf solution specifically designed for embedded applications.

However, once we did some modeling and analysis, we became convinced that LonWorks has several problems that make it unsuitable for this and many other embedded applications. For example, the LonTalk protocol would frequently fail to deliver messages in our system and, with the current system design, would result in several on-site service calls daily to each network in order to restore safe system operation.

The problems we found with LonWorks product can be summarized as:

TECHNICAL LIMITATIONS:

- Inefficient interface to host computer.
- Inefficient use of available bus bandwidth.
- Insufficient support for broadcast message acknowledgments.
- Significant message losses due to random slot collisions.
- Priority slots seemingly unusable for ordinary communications.
- Design quirks and unpleasant surprises.

SUPPORT LIMITATIONS:

- Incorrect technical documentation & support.
- Inadequate system-level performance analysis.

BUSINESS-DRIVEN LIMITATIONS:

- Impediments to product evolution.
- Uncertainty as to Echelon's viability as a company.

It is possible that LonWorks may be suitable for many applications. However, we advise caution if any of the following conditions exist:

- Frequent messages that consume more than a small fraction of available network bandwidth.
- Real-time deadlines unable to tolerate repeated lost messages or arbitrarily long delays in delivery.
- Message transmission synchronization caused by control loops, noise bursts, reactions to synchronized external components, or responses to multicast messages.
- Expensive service calls required for even a small percentage of operational failures.

Because of the problems with LonWorks, we advise extensive analysis and simulation in addition to laboratory testing before adopting it for use in real time control systems.

Contents

1. Introduction

When one of our customers needed an embedded communications system, we at first recommended Echelon's LonWorks technology, featuring the LonTalk protocol. This was based on the conclusions from our paper[5] that indicated the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) approach used by LonTalk was the best fit to our requirements.

The application involved about a dozen nodes on a network 2000 feet long. Because of the highly noisy environment, LonWorks technology (which could use transformer coupling) looked much more attractive than Controller Area Network (CAN), which would need more expensive opto-isolation.

Previously, we had completed two different LonTalk laboratory demonstrations on other applications. Echelon encourages customers to do prototypes and, when they work, use them as the basis for building products. However, because we have had experience with real-time networking problems before, we decided to do some analysis before we made a final commitment.

Our analysis turned up some problems with LonWorks and with Echelon as a company that made them unsuitable for our application. Ultimately, we decided not to choose LonWorks. Although we still think that CSMA/CA is a good approach, Echelon's implementation and support leave much to be desired.

The issues we have identified are broadly applicable to a wide variety of embedded control applications. Therefore, we have written this summary of the problems we encountered when trying to understand and apply LonWorks in the hope that others will be able to steer clear of the pitfalls we have identified.

The report is organized into a set of ten major issues:

TECHNICAL LIMITATIONS:

- Inefficient interface to host CPU.
- Inefficient use of available bus bandwidth.
- Insufficient support for broadcast message acknowledgements.
- Significant message losses due to random slot collisions.
- Priority slots seemingly unusable for ordinary communications.
- LonTalk design quirks and unpleasant surprises.

SUPPORT LIMITATIONS:

- Incorrect technical documentation & support.
- Inadequate system-level performance analysis.

BUSINESS-DRIVEN LIMITATIONS:

- Impediments to product evolution.
- Uncertainty as to Echelon's viability as a company.

It is important to note that we ran out of resources before we were able to completely understand LonWorks and Echelon's business model. We did interact frequently with Echelon during the course of our investigation, so we believe we had the best available information to work with. Nonetheless, it is possible that we missed some information that makes everything turn out just fine.

2. Inefficient Interface To Host CPU

In our application, we expected to use Neurons (microcontrollers that implement the LonTalk protocol) as communication chips connected to larger, more powerful processors in some nodes, and as stand-alone processors in many other nodes. While Neurons can perform many simple control functions by themselves, it is important to be able to use high-powered processors for advanced control algorithms, comprehensive diagnostics, and failure management.

The interface between Echelon's Neuron and any potential host computer turned out to be, in our opinion, unnecessarily complex. Although Echelon outlines many options, all of them add cost and complexity to the communication system.

In particular, the technical concerns are:

- Separate hardware circuitry is required for the interface: semaphores for dual ported memory interface or address decode logic for bus interface.
- Separate hardware circuitry is required for interrupt driven data exchange.
- The parallel interface between the Neuron and the Host is based on a token passing scheme. We noticed "wait forever" loops in the token exchange software that raise concern about wasting processing cycles or even causing the processor to become caught in an infinite loop, causing the application software to fail.
- Echelon's interface software (MIPS) has licensing fees that were judged too expensive for our application.

These issues with interfacing to the host CPU reduce the attractiveness of current Neurons (chips implementing the LonTalk protocol) because of cost and complexity concerns.

3. Inefficient Use of Available Bus Bandwidth

Echelon's LonTalk protocol makes relatively inefficient use of available bus bandwidth. Echelon's published[2] theoretical message rates are limited by high overhead at low bus speeds and by slow chip processing times at high bus speeds.

For example, Echelon claims that LonTalk chips can at best hope to provide:

- 320 sustained packets per second for 78 Kbps; 400 peak (12 byte packets).
- 560 sustained packets per second for 1.25 Mbps; 700 peak (12 byte packets).

While this may seem like a sufficient number of messages per second, there are some additional constraints to take into account. First, operating near rated capacity can cause latency problems with messages that get unlucky and repeatedly defer to other messages for the bus. Second, the conditions Echelon assumes to exist when claiming this sustained throughput are quite optimistic because they involve uniformly distributed message arrivals that generate far fewer collisions than what we expect in our system.

In contrast, the CAN protocol[1] can deliver approximately:

- 600 packets per second at 78 Kbps
- 7750 messages per second at 1.00 Mbps

(these numbers are based on 64 bits data, 47 bits messaging structure, and a 16% bit stuffing overhead for a total of 129 bits per message). Furthermore, these much higher message rates are delivered under all conditions (ignoring messages lost to data errors, which are also ignored for the LonTalk numbers above).

So, LonTalk provides only 53% of the number of sustained messages per second as CAN in the best case of low data rates. In a control-based application, limits on message bandwidth and an inability to scale to high data rates can be a significant limitation. From another point of view, a requirement to use higher data rates can increase equipment costs and technical risk.

Figure 1 shows a comparison of specified (*i.e.*, best case) LonTalk vs. CAN performance.

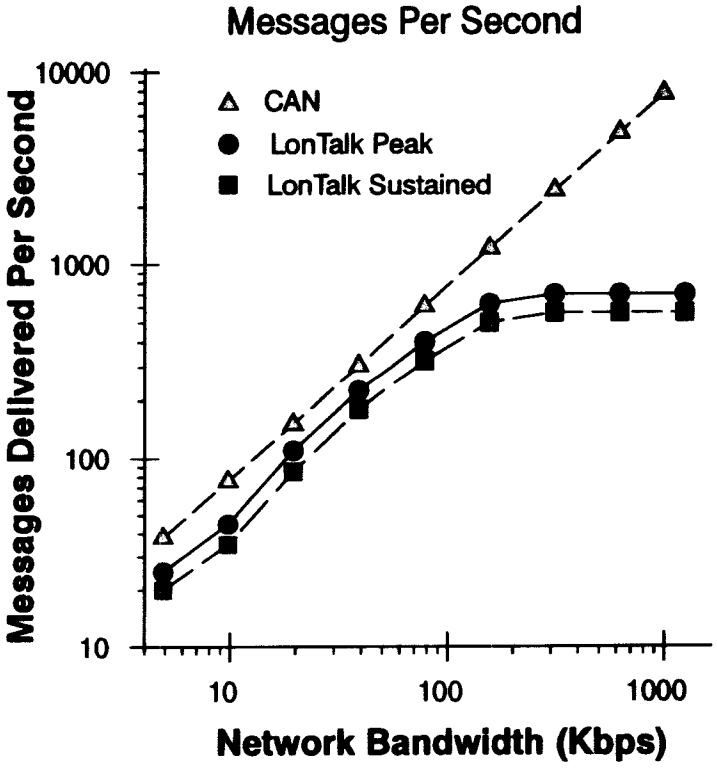

Figure 1. CAN delivers more messages per second than LonTalk. The log-log scale visually understates CAN's advantage.

An important point in this data is that LonTalk provides approximately half the sustained performance of CAN at low data rates, and has a much lower performance plateau at higher data rates. Additionally, it should be noted that the LonTalk performance numbers given are for almost ideal conditions, and can suffer greatly in the presence of message collisions, depending on operating conditions.

> *NOTE: The above analysis does not take into account the fact that the Neuron is providing a more comprehensive set of network services than an off-the-shelf CAN chip. It could be argued that a CAN solution with a low-end 8-bit microprocessor would also have a performance plateau because of the time spent by the CPU performing higher level protocol services (if they are used by the application software). However, with CAN the system designer always has the option of buying a faster CPU from among the many available designs to raise the performance plateau — this is not possible with limited LonTalk implementations currently available.*

It is possible that LonTalk performance at higher bit rates will improve in the future with the introduction of faster chips. That is because at higher bit rates the transmission capability is limited by the turnaround time of the chip in receiving messages (CAN chips do this in hardware rather than software, but offer simpler functionality). However, integrating future chips capable of faster message rates into installed systems having the current, slower chips could cause problems.

This discussion should not be misinterpreted to indicate that CAN is necessarily a better solution. CAN has its own problems, such as fixed global prioritization and inability to use noise-resistant transformer coupling hardware. We have published a paper discussing tradeoffs involved in selecting protocols [5].

In short, LonTalk makes inefficient use of available network bandwidth compared to other protocols, such as CAN.

4.1. Retransmitted messages

Another way to provide robust multicast transmission is to broadcast the message more often than necessary, and hope that at least one message copy gets through. The problem is that the effectiveness of retransmission depends on the message loss rate. The message loss rate must be low enough that at least one of the multiple transmissions of each message gets to all receivers with very high probability.

The probability of successfully delivering at least one of *M* repeated transmissions is:

$$P_{success} = 1 - (P_{msg\_failure})^M$$

where *M* is the number of transmissions, and independence of message failures is assumed.

Let us assume that failure to deliver a message after a specified number of transmissions leads to an undesirable system failure (not necessarily a catastrophic failure, but a failure that adversely affects performance nonetheless). For our application, we expect a noisy environment that has, say, 10^{-6} errors per bit. Given a message traffic load of 200 messages per second, 112 bits per message, and assuming no more than one bit error per message, mean time between failure (MTBF) would be:

$$P_{msg\_failure} = bits\_per\_message * error\ rate$$
$$P_{msg\_failure} = 112 * 10^{-6} = 1.12 * 10^{-4}$$

$$P_{success} = 1 - (0.000112)^M$$

From this, mean time between system failures (MTBF) can be computed as:

$$MTBF = \frac{1}{msgs\_per\_second * (1 - P_{success})} = \frac{1}{200 * 0.000112^M}$$

For different numbers of transmission attempts *M*, this gives:

M	*MTBF*
1	*44.6 seconds*
2	*4.61 days*
3	*112.8 years*
4	*1,000,000 years*

Barring other complications, *M*=3 transmissions of each message gives marginal performance for a large fleet of long-lived systems. Although 113 years sounds like a long time, if there are 100,000 installed units, then some system will have a fault due to communication failure about every 10 hours. So, *M*=4 is preferred, and would be needed in our system because we feel that 10^{-6} errors per bit may well be an underestimate of the number of transmission errors we expect to see.

The bad news is that *M*=4 means sending 800 messages per second, which exceeds the available LonTalk bandwidth available at any network bit rate.

4.2. Periodic messages

Rather than insist that all messages actually be delivered, it is often possible to tolerate lost messages. This can be done in cases where messages are sent periodically, and the physical dynamics of the system are such that control loops can be broken for a short interval without causing a significant enough effect to produce system failure. Our current system is, in fact, a periodic control-based application.

> *NOTE: this approach doesn't work for "event-driven systems", in which messages are sent solely in response to a discrete event. Those systems need reliable message acknowledgements to ensure message delivery. Someday, our application may change to an event-driven system; if so, the solution proposed here won't work.*

In the case of our application, it was decreed that losing two of any three consecutive messages of the same type was acceptable. In terms of MTBF calculations, this is the same as attempting M=3 multiple transmissions in hopes that one of the three gets through. But, because the transmissions are spread over three control loop cycles, no increase in message traffic over the baseline rate of 200 messages per second is required.

While the MTBF from this approach due to message data errors is marginal (113 years), it might be an acceptable price to pay for using a standard off-the-shelf protocol if there were no other problems.

5. Significant Messages Losses Due To Random Slot Collisions

LonTalk has a potentially severe problem with message losses due to random slot access collisions. Our analysis shows that in a 16-transmitter system with unacknowledged messages, up to 63% of messages can be lost due to collisions.

The problem comes about because in unacknowledged traffic, LonTalk uses a fixed number of 16 random slots to schedule messages for transmission. When the next message is to be transmitted on a busy network, each transmitter with traffic to send randomly picks one of 16 time slots after the currently transmitting message. If the first non-empty slot happens to have been selected by only a single transmitter, that message will be sent without collision. If, by chance, the first non-empty slot has been selected by two or more transmitters, a collision will occur and the messages will not get through.

The probability of a random slot collision occurring depends on the number of active transmitters (the number of transmitters with traffic to send at that instant, not necessarily the number of total transmitters). The probability of collision can be computed as follows:

Let us assume that there are S slots, numbered from 0 to S-1, and N transmitters. The probability T_i that slot i is accessed by a single transmitter, and that all other transmitters select a slot greater than i is:

$$T_i = N\left(\frac{1}{S}\right)\left(\frac{S-i-1}{S}\right)^{N-1} \qquad ; \qquad 0 \le i < S$$

The chance of successful transmission is:

$$P_{success} = \sum_{i=0}^{S-1} T_i = \frac{N}{S}\sum_{i=0}^{S-1}\left(\frac{S-i-1}{S}\right)^{N-1}$$

As an example, with S=16 slots and N=16 active transmitters, the chance of a successful transmission is:

$$P_{success} = \frac{16}{16}\sum_{i=0}^{15}\left(\frac{15-i}{16}\right)^{15} = 0.577$$

The probability of an unsuccessful transmission in this case is:

$P_{collision} = 1 - P_{success} = 0.423$; *for N=16 and S=16*

Furthermore, more than one message will be lost in each collision. Rather than derive an analytic solution, we performed a simulation of 1,000,000 bus transactions for values of N from 1 to 16. The results confirm that 42.3% of bus cycles result in a collision for 16 slots and 16 transmitters. The simulation also predicts that at N=16, approximately 63% of all messages will be lost due to collisions with on average 2.36 messages lost per collision.

Figure 3 shows the probability of message loss at various levels of bus loading. The number of active transmitters refers to the number of *instantaneously* active transmitters; that is transmitters with a message to send at that moment. The plots are from simulation data, which have been spot-checked to correspond to analytic results.

As the number of active transmitters increases, the number of messages lost to collisions also increases. In our application, we expect to see approximately 9 active transmitters as an average case. This high number of active transmitters is caused by what amounts to a partial master/slave control system, in which a command is multicast from the master, and is then responded to with data values by multiple slaves simultaneously. While the application could be changed to minimize this behavior, it would mean abandoning the efficiency of multicast messages as well as significantly changing our existing software base.

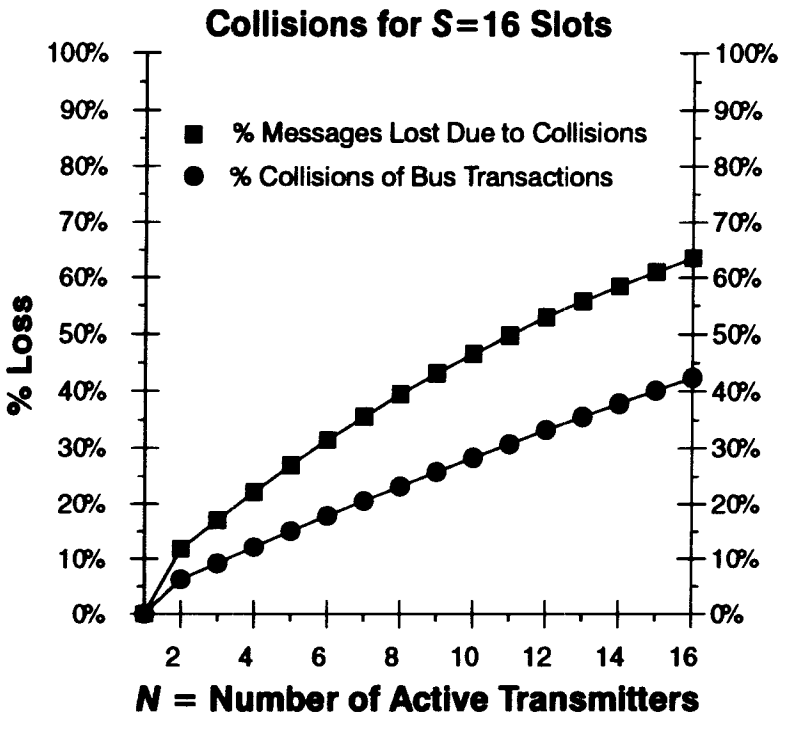

Figure 3. Severe message losses occur when many transmitters are active simultaneously.

There is, of course, a mechanism in LonTalk to adapt the number of slots *S* to the anticipated network traffic. However, this mechanism only works for acknowledged message traffic, not for large numbers of simultaneous message transmissions. And, as discussed previously, we cannot use message acknowledgements because of the limited network bandwidth available. Also, the LonTalk adaptive mechanism merely reduces the probability of collisions — it does not eliminate it.

For example, let us say that somehow we could get LonTalk to predict 9 transmitters would be active (*N*=9). LonTalk would then multiply the number of slots used by 9, giving *S*=16*9=144 slots. Simulations show that this would result in the loss of approximately 3.1% of all messages due to collisions.

Given the failure model of the previous section, losing three messages in a row would happen with probability:

$$P_{fail} = (0.031)^3 = 0.000030 \text{ failures per message}$$

Given 200 messages per second, this results in a mean time between failure (MTBF) of approximately 2 minutes 47 seconds. So, even with Echelon's adaptive prediction algorithm (which we can't use), each system will experience more than 188,000 failures per year.

In our application, there are only 16 slots for 9 simultaneous transmitters. With the resulting 43% message loss rate, each network will experience

$$(0.43)^3 * 200 * 60 * 60 * 24 * 365.25 = 500 \text{ million failures annually.}$$

Let us further assume that only "major" system errors require a service call. Even if 99 of 100 system errors are "minor" and can be screened by automated diagnosis software, the remaining 1% of problems will generate requests for 5 **million service calls** per system per year.

Even with greatly relaxed constraints, this is still a problem. For example, consider permitting 10 messages in a row to be lost before a system failure occurs, and assume that only 0.1% of such failures are "major". The result is still:

$$(0.43)^{10} * 0.001 * 200 * 60 * 60 * 24 = 3.73 \text{ service calls per system per day.}$$

The result of losing large numbers of messages in collisions is that LonTalk seems unsuitable for applications that load the bus with many active transmitters. It is important to note that this can happen even with a relatively lightly loaded (overall) bus — it suffices that the transmitters are approximately synchronized at those times that they happen to transmit. This is NOT a problem for those applications that have well-spaced message transmissions. Unfortunately, synchronized messages transmissions can be fairly common among embedded control applications having control loops, large noise bursts, multi-cast status requests, or synchronized mechanical components.

6. Priority Slots Seemingly Unusable for Ordinary Communications

Given that random slots are unusable due to excessive message collision rates for our application, we sought to use priority slots. That attempt was unsuccessful.

We expected that a CSMA/CA protocol such as LonTalk would provide efficiently implemented priority slots to guarantee collision-free operation. This can be accomplished by assigning each transmitter its own priority slot, which is adequate for a 16 transmitter design.

The problem is that LonTalk's priority slot implementation doesn't completely eliminate collisions.

In studying this issue, the only measurements available to us are the effects of adding priority messages to a system that already has random slot traffic. In the LonTalk Response Time Measurements Engineering Bulletin [4], Figure 9E shows non-zero collision rates that generally increase with network traffic. These collision rates appear to be almost identical to non-priority collision rates (*e.g.*, Figure 9D), but we couldn't draw useful conclusions from the data.

The main mechanism for collision in systems that use priority slots exclusively is given on page 20 of the Engineering Bulletin [4]:

> Nodes remain synchronized only for the duration of the priority slots, plus sixteen randomizing slots, after a packet ends. If one of the nodes does not transmit during this period, the network becomes idle, and the concept of a dedicated priority slot no longer has meaning. In such cases, when a priority packet is delivered to the output queue of the Neuron Chip, the packet is sent out using the *non-priority* MAC algorithm. If another (priority or non-priority) node decides to send a message at the same time, the result is a collision.

Thus, if the bus has gone idle, messages transmitted in priority slots may still collide because there is no preceding message for synchronization. In applications with bursts of traffic (such as our application) the result is that the bus will be idle before a burst, and that some of the messages in the traffic burst may be lost to collisions, even when using priority slots. The number of potential lost messages is difficult to quantify, as it varies considerably based on details of system timing. However, even a very small number of collisions could lead to an unacceptably high message loss rate.

Even if priority slots did provide collision-free operation, there might still be a problem. Echelon seems to have assumed that priority slots are only for occasional use. However, we need to use them for all messages to avoid excessive collision rates. While we did not have resources to fully determine the implications of this mismatch, there seemed to be some priority slot design quirks (discussed in the next section) that could cause problems. In the end, we decided not to pursue using priority slots.

It is entirely possible that there is some clever way to make LonTalk work for our application using priority slots. For example, Echelon personnel suggested placing up to 4 additional "dummy" transmitters on the network to keep the bus from going idle (although we decided that this would be too expensive to build). We didn't find an approach to using priority slots that looked promising.

7. LonTalk Design Quirks and Unpleasant Surprises.

This section brings forth subjective technical issues rather than analytic results. Two Ph.D. and two M.S. computer engineers were continually surprised by the design decisions made by Echelon in building what ought to have been a relatively straightforward product. By "quirks", we mean design decisions or implementation features that are counter-intuitive, get in the way of straightforward use of the product, or are at odds with the published documentation.

An even greater issue was that these quirks showed up as we were trying to work around problems of message losses due to collisions. After a while, it seemed that no matter what we might try to do, some quirk or other surprise would arise that blocked our path. Ultimately, this lead to our loss of confidence in being able to get the system to work, and a sense of increased risk in terms of some unanticipated problem arising down the road.

Below is a listing of the things we encountered that we classify as quirks and unpleasant surprises. It should be noted that this is only a list of the issues we encountered; we did not have resources to fully explore all aspects of the product:

7.1. Back-To-Back Priority Message

LonTalk prohibits a transmitter from sending two priority messages back-to-back (LonTalk Protocol Specification [3], page 24). Instead, the second message is sent using a random slot. While the reason given is to avoid having a processor monopolize the bus, this is at odds with the expectation that a priority message will, in fact, be sent as a priority message. In particular, it can greatly increase the risk of loss of a priority message because it could be sent in a random slot and suffer a collision.

Another problem with this feature is that does not result in consistent system behavior. Two priority messages may or may not actually go back-to-back depending on how the messages happen to catch the probabilistic delays through the Neuron chip, whether another higher priority message contends for the bus between transmissions, and the Neuron chip speed versus bus bandwidth. In our application, the details of message traffic will vary depending on options installed in each system. It seems risky to have a situation in which on some installations a message could frequently suffer collisions due to being treated as a back-to-back priority message, but work fine on other installations.

While this behavior is painted as a feature, we feel it is instead a field service headache waiting to happen.

7.2. Transmissions Wait When The Bus Is Idle

Algorithm 4.12 on Page 26 of the LonTalk Protocol Specification [3] indicates (in procedure `M_Data_Request`) that when the Neuron is transmitting onto an idle bus, it waits for priority and random slots just as if it were transmitting after another message.

This behavior seems to be corroborated by page 20 of the LonTalk Response Time Measurements Bulletin, which states that:

> Nodes remain synchronized only for the duration of the priority slots, plus sixteen randomizing slots, after a packet ends. If one of the nodes does not transmit during this period, the network becomes idle, and the concept of a dedicated priority slot no longer has meaning. In such cases, when a priority packet is delivered to the output queue of the Neuron Chip, the packet is sent out using the *non-priority* MAC algorithm.

This is not the expected CSMA/CA behavior of having transmitters immediately start transmission on an idle bus, although it is was no doubt easier to build in silicon. It seems to prohibit approaches to making priority slots work by carefully sequencing accesses to an idle bus.

7.3. Collision Avoidance Doesn't Work All the Time

Page 20 of the LonTalk Response Time Measurements Bulletin [4], states that:

> Nodes remain synchronized only for the duration of the priority slots, plus sixteen randomizing slots, after a packet ends.

Thus, if the backlog of messages indicates that more than 16 random slots should be in use, a node having new traffic can jump onto the bus, increasing the probability of collisions. While the *expected* number of idle slots after a message is less than 16, there will be cases where the number of idle slots exceeds 16 because of the random selection of slots.

There is a risk here that an engineer, having in mind typical CSMA/CA behavior, would contrive to stagger issuance of messages from multiple nodes by a little bit in order to avoid collisions on idle buses. Instead of avoiding collisions, they could actually be increased because of the priority or randomized delays between starting a Neuron transmitting and the actual transmission.

7.4. Slot Adaptation Only Works With Acknowledged Transmissions

LonTalk has a mechanism for dynamically adjusting the number of random slots in order to reduce collisions. Unfortunately, this only works for acknowledged messages. Since acknowledgements are impracticable for our application, the entire scheme is useless to us.

It would have been more useful if Neurons were capable of being programmed with a number of random slots based on our analysis of expected traffic patterns, rather than having an adaptive mechanism that is beyond our control. We would have expected the design to come with both options.

7.5. Erroneous and Unclear Documentation

There were numerous substantive as well as minor errors in LonWorks documentation. While they are discussed in more detail in a the next section, the process of reading the documentation, forming an expectation of behavior, then finding that the actual system behaved differently added to the perception of Neurons as being quirky.

8. Incorrect Technical Documentation & Support

We have found numerous documentation mistakes that hampered our efforts to understand the suitability of LonTalk for our application. Eventually, we felt that we had to question each and every technical detail, because so many of them had turned out to be misleading or simply incorrect.

While it is normal to have occasional typographical errors, the problems with Echelon's technical material go much beyond that. Several times, Echelon support personnel were not able to supply correct information when asked specific technical questions (in all fairness, usually we got correct and timely information, but in this case the exceptions caused us significant problems). In some cases, Echelon personnel refused to acknowledge the correct answers even after we supplied mathematical formulas backing up our assertions. This was especially true with issues dealing with probability theory (and may reflect an underlying cause for the lack of high-quality system analysis provided by Echelon).

In several cases, Echelon support staff were unable to realize that, based on the problem we told them we were solving, there were additional issues that we should be aware of. This caused us to expend significant energy going down blind alleys in search for workarounds to LonTalk limitations.

Based on the fact that we were given direct access to Echelon's engineers, we believe that these problems are indicative of Echelon's support abilities and are not isolated incidents. Because the communications we have received from Echelon are all marked "confidential", we cannot quote the specifics of Echelon's response to the points in the following subsections.

The following is a non-exhaustive list of problems with technical documentation that we have encountered. We ran out of resources long before we ran out of problems to resolve. It should be noted that most of the problems we encountered originated with statements in the LonTalk Protocol Specification[3] (because that is the document we studied most). Our copy is "version 3.0" and is not stamped draft. Therefore, we had an expectation that it would be reasonably complete and correct.

The below errors are in an approximately decreasing order of severity.

8.1. Incorrect Multi-Link Delivery Probability

On page 29 of the Specification[3], the probability of end-to-end communication failure is given as:

$$\sum P_e$$

Where P_e is the probability of error on each link in the path.

The maximum communication distance D is given as:

$$D < \frac{1}{P_e}$$

Verbally, Echelon staff explained that if the probability of message loss on a single link was 10%, then the maximum distance was 10 such links. The fallacious reasoning is that if 10% of the messages are lost at each link, no messages will

make it over all 10 links (10 times 10% is 100%). A different defense was given later, but still did not make sense to us.

These formulae are obviously incorrect in the face of elementary probability theory. The correct formula is:

$$P_{failure} = 1 - \prod (1 - P_e)$$

or, assuming that all P_e values are the same, then the probability of failure for transiting N links is:

$$P_{failure} = 1 - (1 - P_e)^N$$

Thus, if P_e is 10% for each link, the probability of failure is 10% for one link, 19% for two links, 27% for three links, and so on to 65% for 10 links. Contrary to Echelon documentation, there is no notion of an absolute maximum distance, but rather a decreasing probability of success as the number of links increases.

These particular mistakes are not simply a typographical or mathematical error. Rather, they are indicative of a conceptual error in understanding multi-link system operation. We assert this because page 33 of the Specification[3] states that "the probability of delivery is inversely proportional to the number of channels traversed", and it is clear that a linear proportional inverse relationship is not the case. In a private communication on 6/21/95, Echelon held to that formula. Curiously, in the Appendix on page 108 a correct equation is given for multi-hop delivery.

8.2. Incorrect Definition of Collision Rate

On Page 21 of the Specification[3], Collision Rate is defined to be:

Collision Rate = Error Pkt Cycles / Error Free Pkt Cycles

It should instead be:

Collision Rate = Error Pkt Cycles / (Error Pkt Cycles + Error Free Pkt Cycles)

Echelon acknowledge this as a previously unknown typographical error. Although this revised formula is correct, it is not intuitive because it counts bus collision cycles instead of messages lost due to collisions.

8.3. Incorrect Timing Equations

On pages 27-28 of the Specification[3], the timing equations have several problems. They have typographical errors, they use a poor choice of notation, and they fail to match the values in the operating manuals.

8.3.1. Incorrect Beta 1 Formula

The first problem is that the formulae for beta 1 are incorrect, in that they multiply CT by all the terms. The formulae given in the Specification are:

*beta1 = CT * (583 + beta 2 + PAD) { direct mode }*
*beta1 = CT * (577 + beta 2 + PAD) { special purpose mode }*

The correct formulae are:

*beta1 = CT * 583 + beta 2 + CT * PAD { direct mode }*
*beta1 = CT * 577 + beta 2 + CT * PAD { special purpose mode }*

8.3.2. Poor Choice of Notation

The formulas for beta 2, beta 1, PAD, and preamble all make use of "v, a tuning parameter". It turns out that v is actually a set of values, all different. It would have been better to somehow indicate that the v values are different for each equation. This is a point of clarity rather than an outright error. Nonetheless, it caused considerable confusion in interpreting the Specification.

8.3.3. Inconsistent *v* Values

The values for v published in the development system manuals, when placed into the equations, are not close to the values for beta 1 and beta 2 that were verbally given to us by Echelon. We lacked resources to investigate this problem further.

It should be noted that most of the problems discussed above occurred in a relatively small section of the Specification. We did not study most of the rest of the specification with any detail, and therefore cannot know how many errors would be found elsewhere.

9. Inadequate System-Level Performance Analysis

Echelon sells LonWorks by promoting the notion that it is an off-the-shelf technology that works, and detailed understanding by the application specialists is unnecessary. We, on the other hand, specialize in finding and correcting problems caused by the lack of an adequate up-front analysis. In order to avoid repeating the mistakes of the past, we wanted analysis information from Echelon. Unfortunately, Echelon couldn't supply everything we needed. The lack of analysis information seems to stem from two sources:

- Echelon's use of experimental data rather than simulations or detailed analysis (the analysis Echelon gives is for steady-state and average values, not the worst-case data needed for a real-time control system)..
- Echelon's preference for talking about what they can measure rather than what's important to any particular customer.

9.1. Experimental Approach

Excepting a few analytic equations, most of Echelon's performance information is derived experimentally rather than via simulation. Performance information has been available from only two sources: the LonTalk Response Time Measurements Bulletin[4], and the Echelon Protocol Specification[3].

The LonTalk Response Time Measurements Bulletin used an experimental apparatus that measured performance with uniformly distributed messages. In many systems that is an unrealistic assumption. Furthermore, Echelon personnel seemed unaware that this assumption made a critical difference in performance results. After we made this point with Echelon (and, we hear, they ran experiments to verify the findings), they deleted the bulletin from their offerings (we do not know for certain if the interaction with us was the reason for the deletion). We are aware of no replacement.

Today, the only source of performance information is the equations in Section 12 of the Protocol Specification[3]. Unfortunately, these equations assume perfect backlog prediction, which is unrealistic for networks having mostly unacknowledged messages. The equations either assume zero collisions or use an optimistic estimate, which is similarly a questionable assumption. No equation is given to probabilistically determine expected worst case delay.

So, plainly put, we think that Echelon provides wholly inadequate information for a user to understand the real time behavior of their networks.

When confronted with users who need to understand real time behavior of the network, we have observed Echelon's response to be that users should build a system in the lab, and then test that it works. The problem with this approach is that it will not find infrequent system failures caused by protocol problems. For example, we have an application for which the network should exhibit fewer than approximately 0.001 failures per year from protocol problems. In order to perform a laboratory test of that behavior, several thousand system-years of testing would likely be required. But, at the time we talked to Echelon, their longest data collection run was a few days long.

Yes, if a system works in the lab for a few days, then it is likely that fielded systems will also mostly work. However, if it is important that the systems in the field work reliably, rather than simply work most of the time, brief laboratory testing of stochastic communication protocols just isn't enough. On the other hand, a simulation system would allow readily setting up worst-case conditions as well as collecting data without investing in large-scale prototype hardware.

9.2. Echelon Talks about What They Can Measure

Probably because of their experimental outlook, what information Echelon has provided is in terms of what their development support tools can measure, rather than the parameters of most interest when doing performance analysis. The best example of this is the definition of collision rate.

Echelon defines collision rate as the frequency of message collisions. They then suggest that a 4% collision rate is to be expected. Setting aside the potential inaccuracy of that prediction, the statement is misleading. An engineer who has not studied the mathematical equations in detail would likely believe that this means 4% of the messages are lost. Instead, more than 8% of the messages are lost, because Echelon's definition for "collision rate" is the number of collisions that are observed by the network monitor, not the number of messages lost.

It took us a while to understand that Echelon's definition was differing from ours (especially in light of the equation error in the Protocol Specification).

Echelon's definition is not only potentially misleading, but does not pass the following common sense test. Let us suppose that in an extreme case, 50 nodes each sent a message, and all the messages were lost due to a collision. Then, a single node sends one message that goes through. The Echelon definition would be that 1 in 2 bus transactions completed successfully, for a collision rate of 50%. A much more useful measure that we would prefer is that 1 of 51 messages was transmitted successfully, for a collision rate of approximately 98%.

It turns out that the network monitor can not know for sure how many messages have been lost, because of the possibility of more than 2 messages colliding at once. So, it may be that Echelon adopted their definition of collision rate to correspond to their available development system capabilities. It would be useful for Echelon to report that the actual number of messages lost will vary considerably depending on the system under consideration.

In fact, with some analysis and simulation, Echelon could probably have provided rules-of-thumb that relate bus collisions to lost messages for various operating conditions. Having a network simulation program to study this and other issues would be useful, but Echelon has consistently resisted that approach even though we have requested that they develop a simulation capability over a period of several years.

10. Impediments to Product Evolution

One of the main reasons to use an outside vendor to supply a piece of technology is to get continual product evolution without investing large amounts of engineering resources. In the case of the performance limitations of LonTalk and other technical problems we have discussed, it seems important for Echelon to continue aggressive development and improvement efforts.

However, there are signs that Echelon as a business may not be able to invest significant efforts in improving the basic technology of the Neuron:

- Echelon makes most of its money on support and development tools, not on the Neuron chips themselves. Making a change to improve the performance of the Neuron may entail many expensive engineering changes to the development systems, including increasing speed (if the Neuron is made to go faster), or updating software functions (if the Neuron is changed in some way). Because Echelon has adopted a support/maintenance model for their development tools (users pay a yearly support fee), they could have troubles simply telling their customers to go buy a brand new next-generation tool, and instead be forced into providing major upgrades at no additional charge. At the same time, it would be no surprise if their revenue stream from maintenance/support contracts was too small to do a massive revision of all their products.
- LonTalk is not designed as a well-encapsulated system, despite the fact that it uses the 7-layer OSI model. By that we mean that implementation details (such as the many quirks we discovered) show through to the user who is concerned about verifying correct system operation. Rather than establish a performance-based specification (*e.g.*, all messages are delivered within such-and-such a latency with probability so-and-so), Echelon has provided an implementation-based specification (*e.g.*, the Neuron chip behaves in this particular way in this particular situation). This is likely to reduce flexibility for improving implementations in the future. For example, some implementors may rely on the fact that priority messages are not sent back-to-back, so it may not be possible to change that decision even if it is later found out to be sub-optimal.

If Echelon is not able to provide sustained product evolution over time, then one of the important reasons to use a standard protocol is unfulfilled.

11. Uncertainty as to Echelon's Viability as a Company

This is not a detailed analysis of Echelon's business, which is beyond the scope of this report. Instead, we focus on a single point: **How does Echelon plan to give value to its investors?** If Echelon doesn't remain in business over the long term, then that negates a fundamental reason to use an off-the-shelf protocol — expectation of vendor support.

At last report, Echelon had spent over $80 Million in investments (it is probably more by now). Echelon seems to have spent a significant portion of that money on fairly successful marketing to make customers believe that they will inevitably win in the marketplace.

From what we understand, royalties on Neurons (the integrated circuits that implement the LonTalk protocol) will be quite modest. Let's assume 1% on sales of $3 per chip in large volume production. That means that Echelon needs to sell 2.7 billion chips to break even with its initial investments. Currently Echelon has sold less than one thousandth of that amount (a million chips as of October 1995 per a news release on their World Wide Web site). To put this into perspective, 2.7 billion chips is about a full year of worldwide consumption for all 8-bit microcontrollers in the marketplace. Furthermore, while Echelon is selling chips, the cash already invested will be losing millions in time value. Also, Echelon will still be spending money in the years that will take for such sales to materialize.

A newer business model seems to be that Echelon will sell development tools and support services. Judging by our experience of Echelon's lack of ability to provide accurate technical information it is difficult to see how they can recoup $80 million selling support. Third party vendors are supplying development tools at lower prices. Furthermore, now that the protocol is "open", these vendors seem to be able to provide comparable tools without the burden of an $80 million "mortgage" to make payments on.

We are in the position of having to find a partner to supply technology for 30- to 50-year product lifetimes. While current parts will probably be available from Motorola and Toshiba as long as there is market demand, it is unclear how long Echelon will remain viable without a solid mechanism to repay its current and future investors. Clearly within a few years current implementations will become obsolete, and we will be counting on a solid partner to be in business to implement next-generation, compatible technologies. Echelon's position seems more precarious than we'd like.

12. Conclusion

We did the best we could to apply Echelon's LonTalk protocol work to our embedded real-time control problem. But, within the limits of our resources, we were unable to make LonTalk work. Furthermore, we have analytic results backed by simulations that suggest it is not possible to make LonTalk work for our application.

It *is* possible that LonTalk may be suitable for many applications. However, we advise caution if any of the following conditions exist:

- Frequent messages that consume more than a small fraction of available network bandwidth.
- Real-time deadlines unable to tolerate repeated lost messages or arbitrarily long delays in delivery.
- Message transmission synchronization caused by control loops, noise bursts, reactions to synchronized external components, or responses to multicast messages.
- Expensive service calls required for even a small percentage of operational failures.

Because of the problems with LonTalk, we advise extensive analysis and simulation in addition to laboratory testing before adopting it for use in real time control systems.

13. References

[1] *CAN Specification*, Version 2.0, Robert Bosch GmbH, 1991.

[2] *LonTalk Protocol*, LonWorks Engineering Bulletin, April 1993. Echelon Publication 005-0017-01.

[3] *LonTalk Protocol Specification Version 3.0*, LonWorks Engineering Bulletin, 1994. Echelon Publication 078-0125-01/19950.

[4] *LonTalk Response Time Measurements*, LonWorks Engineering Bulletin, March 1992. Echelon Publication 005-0010-01.

[5] Upender, B. & Koopman, P., "Communication protocols for embedded systems," Embedded Systems Programming, 7(11) 46-58, November 1994.