Creating an Open Source LLE-based Ethernet to LonTalk adapter

Jeremy Weed

Victor Senior High School

LLE Advisors: Dave Hassett, Robert Peck, Dustin Axman

Laboratory for Laser Energetics

University of Rochester

Summer High School Research Program 2014

November 2014

1. Abstract

The OMEGA Laser System is currently controlled by a LonWorks distributed control system that includes over 1,600 Neuron 3150 chips running at 10 MHz. Communicating on more than 20 twisted pair channels, the Neurons control over 3,000 A/D channels, 2,000 DC servo motors and 4,000 digital I/O channels. All communications entering or exiting the network travel through LENA servers across an RS-232 serial bus to a Serial to LonTalk adapter and onto the LonTalk network. Creating a direct Ethernet to LonTalk adapter would allow for the retirement of the LENA computers used, provide physical connections to the LonTalk network, allow the servers to reside on virtual machines, and minimize any errors caused by high traffic volumes through the current RS-232 interface. Using a Netburner board and a Neuron card, a proof-of-concept Ethernet to LonTalk adapter was attempted. Using Neuron-to-Neuron network variables, data communication from the current firmware to an updated system was demonstrated. However, SPI data communication between the Netburner and Neuron was not achieved due to hardware limitations, so different protocols will be explored.

2. Introduction

The OMEGA Laser System currently operates over 10,000 control and acquisition points, consisting of over 3,000 A/D Channels, 2,000 DC servo motors and 4,000 digital I/O channels. A LonWorks distributed control system operates these points, and allows them to communicate across the network. The system consists of more than 1,600 Neuron 3150 chips, running at 10 MHz and communicating on over 20 twisted pair channels. To enter the LonWorks control network, data must travel through LENA servers and then across an RS-232 serial bus to a Serial to LonTalk adapter, which then propagates the data across the LonWorks network, as shown in figure 1.

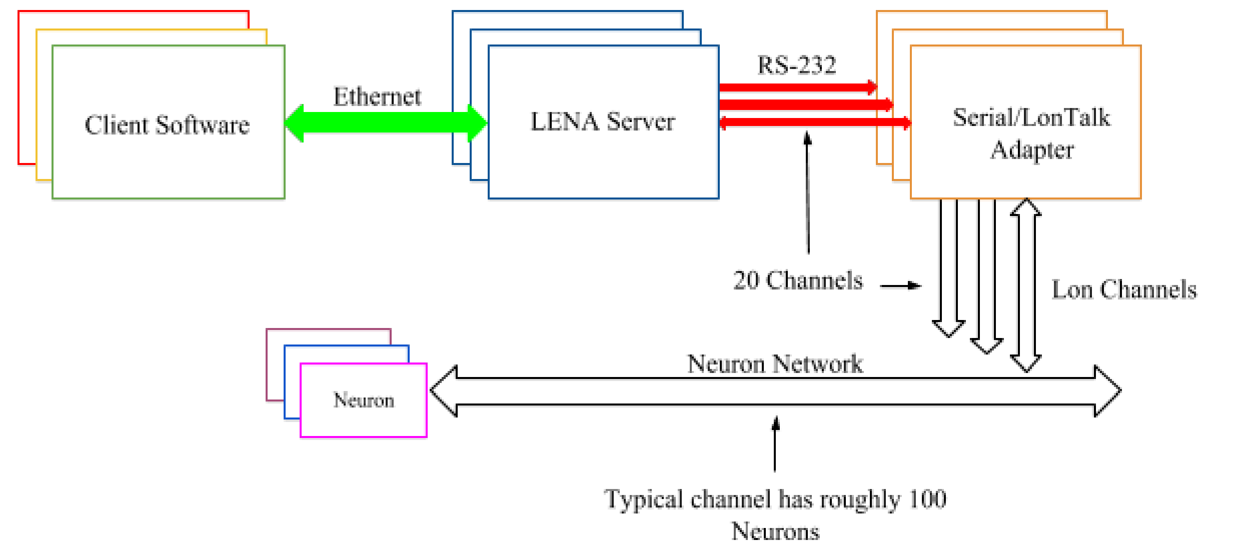

Fig 1. Diagram of current LonTalk network system.

Each Neuron on the network, commonly referred to as a node, communicates to other nodes using the LonTalk communication protocol. The LonTalk protocol was designed for building automation and industrial control, and it allows the various nodes on the network to communicate in a decentralized fashion. Neuron chips can be programmed in Neuron C, an event-driven ANSI C based language that allows automated variable communication and synchronization using the LonTalk protocol. Neuron C allows for multiple modes of communication between nodes, anywhere from explicitly created and propagated messages to autonomously propagated network variables.

The current LonTalk control system on the OMEGA Laser has hardware limitations that can lead to data bottlenecks. To pass from the LonTalk system to external clients, the data must pass through one of twenty RS-232 serial connections. The RS-232 serial bus on the system can currently transmit data at about 38,400 bits per second, much slower than the over 100 million bits per second transfer rate possible on current Ethernet systems. Data from the system and commands going to the control cards can become "backed up" at the RS-232 serial bus, causing data to be either lost or delayed.

3. Setup

The original proposed solution to the data bottlenecks in the LonTalk system involved the use of a Netburner web server to create the messages that would then be propagated across the LonTalk network, shown in figure 2. This would eliminate the existing bottleneck in the system, but would require all data going across the network to be translated by Netburner to a form that the Neuron cards could understand, and would require that the Netburner translate all data coming out of the network. There is currently no software written for a Netburner to interpret LonTalk communication, and with the multitude of ways the Neuron cards communicate over the LonTalk network, extensive work would be necessary to create a working prototype.

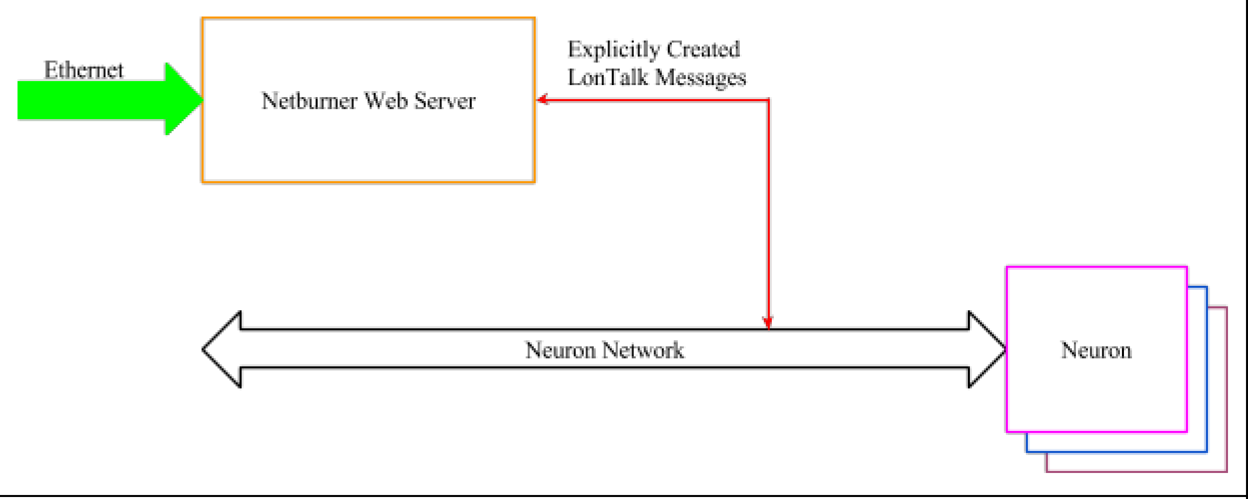

Fig 2. Diagram of Original Proposed Solution

As a result of these challenges, other solutions were explored. The use of a Neuron card to translate messages to and from the LonTalk network was proposed. The Neuron C language, used to program the Neuron chips, includes implicit handling of all communication across the LonTalk network and would allow for simple and easy translation between the Ethernet and LonTalk networks. Neuron cards cannot interpret data from the Ethernet network, so it was proposed that a Netburner web server be used to collect and send data over Ethernet. This setup requires the data going between the two networks be transferred between the Netburner and Neuron card, so a data communication protocol had to be chosen to facilitate the transmission. SPI, or serial peripheral interface, was chosen as the protocol for transmission between the Neuron and Netburner. SPI was chosen because both devices support the protocol, and SPI data transfer is relatively easy to implement in a short amount of time. The final testing setup can seen in figure 3.

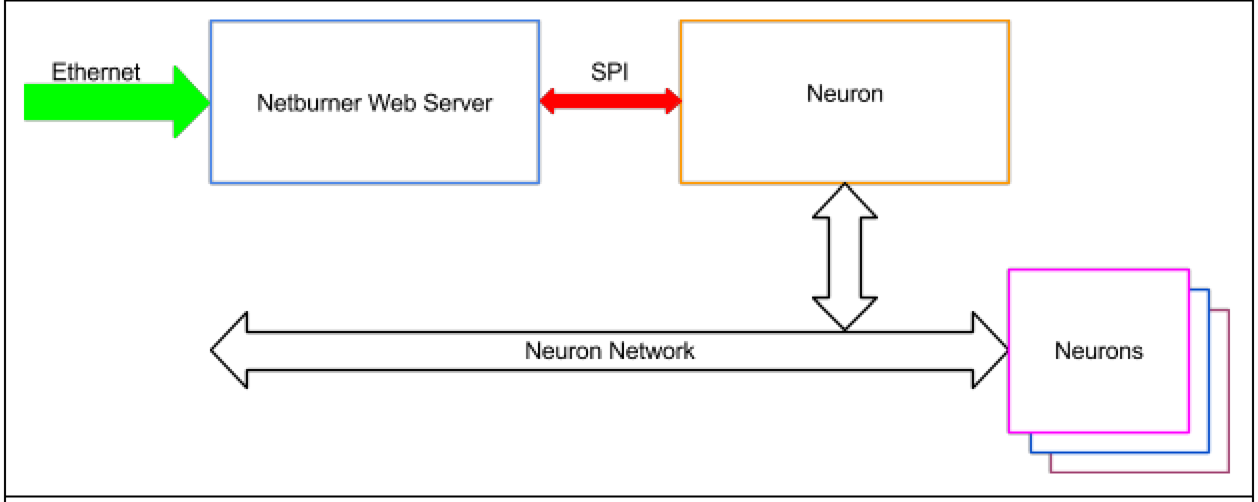

Fig 3. Diagram of Final Testing Setup

4. Data Communication

There are multiple ways to transmit information over the LonTalk network. Two different methods were explored, explicit messages and network variables. Both forms of communication require a form of binding between the recipient and the sender. Explicit messages can be bound in code, but it is recommended that network variables be bound with an external tool, such as the LonMaker Integration Tool.[1] For this reason, explicit messages were explored first for LonTalk data communication. Although easier to implement, explicit messages are not autonomously propagated like network variables and are more resource intensive.

Explicit messages were sent between two different Neuron cards in a test setup. Before sending the message, the destination must be bound. First, the type of message must be defined (i.e., is it domain-wide or directed for one specific node) and then the destination for the message is defined. For subnet-node type addressing, which was used in this demonstration, the domain, subnet, and node location of the message recipient must be defined. An example of a subnet-node addressed message can be found in figure 4. Further exploration of explicit messages in Neuron C showed that messages could request responses, and respond directly to messages received without defining a recipient explicitly. Neuron C also provides the ability to check whether a message has been received. Both of these abilities were successfully demonstrated.

```
msg_out.tag = test_message;
msg_out.data[0] = 0;
msg_out.dest_addr.snode.type = SUBNET_NODE;
msg_out.dest_addr.snode.domain = 0;
msg_out.dest_addr.snode.subnet = 1;
msg_out.dest_addr.snode.node = 3;
```

Fig 4. Example of Subnet-Node Type Addressing in Neuron. This example shows a message called `test_message` *that is destined for a Neuron Card located at domain 0, subnet 1, and node 3, containing the value 0. The structure definitions and typedefs can be found in the Neuron C header files ADDRDEFS.H and MSG_ADDR.H.*

After exploring explicit messages, Neuron C network variables were investigated. While faster and more powerful than explicit messaging, network variables require more setup than messages. It is recommended that network variables be bound with a separate tool, so as to leave the specific addresses and locations of other network variables ambiguous to the software on a Neuron chip. This tool was not available, so an operation called "self-binding" had to be performed on the variables. Shown in figure 5, binding a network variable requires two things: an address and a selector. Addresses work similarly to explicit messages, with multiple types defining where and to how many network variables the variable is synced with. Unlike explicit messages, multiple network variables can exist on one Neuron at a time. The selector allows network variables to specify the specific variable they are bound to, allowing multiple variables to coexist on one Neuron.

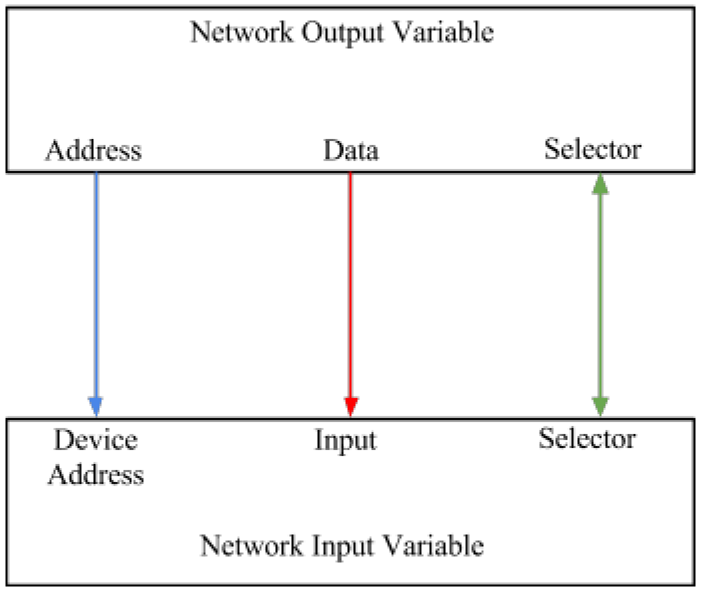

Fig 5. Example of Properly Configured Network Variable. In order for two network variables to communicate, two criteria must be met: that the address of the output variable matches that of the device with the input variable, and that their selector IDs match. If these two requirements are met, automatic data transmission can take place.

There are multiple types of network variables, and all require a slightly different binding process. At their most basic, there are two types of network variables: input and output. To bind an output variable to an input, the selector value for each variable must be the same, they must be of the same type, and the address of the output variable must match the address of the device that contains the input variable. Once these criteria are met, the value of the output variable can be changed, and the Neuron will automatically update the input variable on the other chip. Other types of network variables exist, such as propagated, which must be explicitly told to update, and polled, where the roles of the output and input variables are reversed, and the input variable must ask for the value of the output variable. On the test setup, basic, polled, and propagated network variable communication was demonstrated. Communication with a Neuron running firmware currently deployed on the OMEGA Laser system using network variables was also demonstrated, which is not possible with explicit messages. Disregarding the initial bind, network variables were both faster and easier to implement than explicit messages, and were chosen as the communication protocol for this project.

For communication from Ethernet to the Neuron card, a Netburner web server[2] was used. A Netburner was chosen because it offered an easy and simple web interface and data output. A web interface was first created and tested on the Netburner. It allowed for simple, byte-sized data transfer from the web to the Netburner. The Neuron card and the Netburner were then connected to transmit data over an SPI bus. SPI, or Serial Peripheral Interface, was designed for simple, serial data transmission between one master and many slaves. SPI uses four wires, one to signal that a data transmission is occurring, two for data transmission, and one, called the clock, to specify the rate at which data will be transferred. Unfortunately, data communication between the Neuron and Netburner over SPI was not possible. The first issue that arose was that the Neuron operated with a 5V signal as high, and the Netburner interpreted 3V as high. This issue was resolved using level shifters integrated into the Netburner board, to allow it to communicate with 5V as high. After this was fixed, it was discovered that the lowest possible clock speed for SPI data communication for the Netburner was multiple magnitudes faster than the fastest clock speed available on the Neuron chip, and during the data transfer, both boards wished to be masters. To resolve these issues, bit-banging code, or code that explicitly created and interpreted SPI signals, was written for the Netburner web server. Unfortunately, this code was not successful in solving the issue in communication between the two devices. In the future, the SPI bus could easily be replaced with another communication protocol that is compatible with both the Netburner and Neuron.

5. Conclusion

To solve the current issues with the OMEGA Laser LonTalk system, an LLE-based Ethernet to LonTalk adapter was explored. Research into various methods of LonTalk communication protocols showed that network variables show the most promise for a future adapter. Several proof-of-concept ideas were also demonstrated, including integration with the current firmware on the laser. Due to incompatible hardware, a full Ethernet to LonTalk adapter was not possible, but with a few small changes, a proof-of-concept adapter could now be easily created.

Acknowledgements

I would like to thank Dr. Craxton for giving me this opportunity to participate in the internship program. I would also like to thank Dustin Axman for providing me with guidance and debugging help with the early stages of this project, Dave Hassett for helping me to understand the LonTalk system at the facility and providing me with the resources I needed to work on my project, and Robert Peck for overseeing my project and guiding me throughout the summer. Lastly, I would like to thank all of the teachers, mentors, and friends who have inspired me over the years, and provided me with the knowledge I needed to work on this project.

6. References

1. "Neuron C Programmer's Guide." "http://www.echelon.com/assets/blte1a7d6a3b37598bf/078-0002-02H_Neuron_C_Programmers_Guide.pdf "1990. PDF file.
2. "Netburner MOD5441X Datasheet.""http://www.netburner.com/mod54415/255-1-7/file " File last modified on 25 July 2014. PDF file.