Real-Time Ethernet

Joe Kerkes

FEBRUARY 26, 2001

Real-Time Ethernet

Popular wisdom says Ethernet is non-deterministic. This article shows that's not true and better illuminates the issues involved.

Several of my past embedded design projects have required network connectivity of one sort or another. A recurring theme among these projects has been the need for communications between an embedded system and the outside world over a standard network interface. These projects typically included a real-time operating system (RTOS) and the usual line-up of network all-stars: Ethernet, TCP/IP, and a sockets API. Ordinarily, the processing of Ethernet data in these systems was either relegated to a background task, in the case of non-real-time data, or the data was cyclic in nature and as long as it was refreshed sometime during a broad period-say in the range of 15 to 30 milliseconds-all was well. In other words, accuracy of less than several milliseconds was not usually expected.

Recently, however, I was presented with the task of designing a system that required a low-latency, deterministic network as the focal point of the project. This precipitated an investigation to determine if the network tools de rigueur were suitable for this task. Even though many excellent sources regarding network theory, standardized protocols, and structural elements were consulted in the course of this research, performance specifications were elusive due to the myriad variables in networked systems, such as: network hardware, network software, computer hardware, and operating system. (See references at the end.) From this research, I gained practical insight into the performance characteristics of TCP/IP and Ethernet, the specifics of which can be found only by empirical study, using the actual components of a given system.

The objective of the project was to reduce production and design costs by replacing a flight simulator's large centralized I/O system, that contained over 1,000 points of I/O, with a network of embedded I/O processors. Additionally, manufacturing efficiency improvements were expected due to reduced wiring complexity. The new distributed I/O system would have many remote nodes (located near the panels and instrumentation that they served) networked together to transfer data between the simulation processor and various aircraft and simulator equipment.

Initially, Ethernet was thought to be the best choice of network media; it could certainly satisfy the requirements of low-cost hardware, low production costs, and 50 to 70 I/O nodes per simulator. After all, CPU boards with integrated Ethernet are readily available from multiple vendors, resulting in reduced costs. figure 1

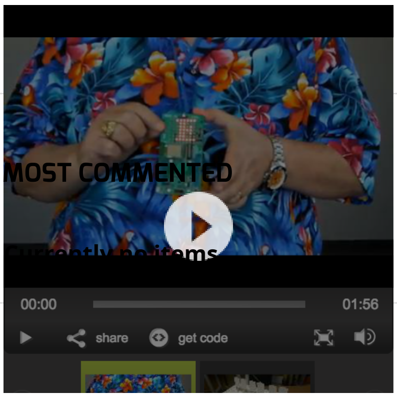
MOST READ

10.29.2013
Imagine a trillion sensor world (/electronics-news/4423942/Imagine-a-trillion-sensor-world)

05.31.2002
Serial Protocols Compared (/design/connectivity/4023975/Serial-Protocols-Compared)

10.29.2013
Cisco reorganizes for the IoT challenge (/electronics-news/4423947/Cisco-reorganizes-for-the-IoT-challenge)

EMBEDDED (/VIDEOS) TV (/VIDEOS) (/videos) video library (/videos)



(http://m.eet.com/media/1074315/0101ia1fig1.gif) depicts the simulator computer system layout originally envisioned using Ethernet as the network medium in the distributed I/O.

However, since Ethernet is non-deterministic, evaluation of other media was undertaken. Various types of network buses were evaluated, including reflective memory, field buses (common in factory automation), RS-485, and others. We concluded that none of them could meet our requirements cost-effectively. Ethernet seemed the most promising, with the single drawback (albeit a possible showstopper) of non-deterministic data transfers. Additional research was required to resolve whether or not Ethernet could be used in a system that required low latency and deterministic behavior. Ethernet, TCP, UDP, and IP

Many times I've heard colleagues mindlessly recite the platitude: "Ethernet is non-deterministic." But what exactly is non-deterministic about it? Is it without bounds or is it just a matter of defining applicable tolerances? To answer these questions, let's take a closer look at some of the fundamental concepts of Ethernet and TCP/IP.

Most application programs interact with the Ethernet through a TCP/IP stack, which provides a layered approach to implementing the various sub-protocols that comprise TCP/IP. In the networking vernacular, TCP/IP refers collectively to a suite of protocols that are built on the Internet Protocol (IP). TCP is one of the many protocols in the suite.[1] The TCP/IP stack sits on top of, and is tightly coupled to, the Ethernet driver. In other words, the TCP/IP stack orchestrates the flow of data (packets) between application software and the network driver by queuing output data, buffering input data, and providing an inter-network routing mechanism. Our test application software, to be discussed shortly, accesses the TCP/IP stack by means of a sockets API.

With TCP/IP you have the choice of two protocols for data transfer, TCP or UDP. The most important difference between the two is that TCP provides a reliable connection, which ensures that the intended receiver receives each packet that is sent. UDP, on the other hand, is a send-it-and-forget-it protocol; if receipt verification is desired, the application must provide its own means of doing so. One might be quick to embrace the more reliable of the two, but tread carefully little grasshopper, for reliability comes at a price.

Every packet sent via TCP protocol triggers the receiving stack to send back an acknowledgment packet. The performance price can be pretty steep for small quantities of data. For example, if one byte of data was to be transferred, a 64-byte packet (minimum size for Ethernet) would be sent over the network and a 64-byte acknowledge packet would be sent in response, essentially doubling the total network traffic. Using UDP in this same scenario would result in only the original packet being sent. Therefore, the use of TCP in this project would add unnecessary overhead to our closed network, where the added reliability was not needed.

An Ethernet hardware device, known as a Network Interface (NI), is used to connect to the network medium. The NI employs a two-stage bus arbitration scheme, Carrier Sense Multiple Access with Collision Detection (CSMA/CD), whereby devices output to the bus when they sense that it is not in use. This is the CSMA part of the bus arbitration protocol. In this scheme, however, it's possible for two or more devices to transmit on the bus simultaneously; this is where the CD (collision detection) part of the name comes in. When simultaneous accesses to the bus occur, the devices sense the "collision" and retry outputting to the bus after random delays until they transmit successfully. Therein lies the most rudimentary non-deterministic aspect of Ethernet; if a bus arbitration scheme was implemented at a higher level, however, the CSMA/CD scheme could possibly be circumvented.

To overcome the unpredictable arbitration scheme of Ethernet, we employed a software protocol that allows network devices to access the bus only when commanded to do so. This protocol does not inhibit CSMA/CD; it merely places restrictions on the higher level (application) software, which controls access to the Ethernet medium. For this scheme to be effective, some rules had to be established:

- Only our proprietary nodes could be attached to the network
- One device will be designated the controller; all other devices will be considered remote nodes
- No remote node may access the network bus until the controller sends a request to it
- Remote nodes will have a predetermined amount of time to respond to a controller request

A test program was created for the controller node, in which data is sent via the network to a remote node. The program waits for response data from the remote node. Any response data is compared to the original transmitted data to check for errors. If a data mismatch occurs or if the remote node fails to respond after the predetermined time, an error message is generated. The controller software also records worst, best, and average times for round-trip packets. The remote node software simply waits for

RELATED CONTENT

02.26.2001 | DESIGN

Real-Time Ethernet (/design/connectivity/4023291/Real-Time-Ethernet)

01.03.2013 | CONTENT GROUP

Hardware-in-the-loop simulation (/collections/4404128/Hardware-in-the-loop-simulation)

06.01.2013 | NEWS

Real-time Java workshop deadlines extended (/electronics-news/4416544/Real-time-Java-workshop-deadlines-extended)

11.23.2009 | PRODUCT

Aonix adds VxWorks/Intel support to ObjectAda (/electronics-products/electronic-product-reviews/embedded-tools/4114068/Aonix-adds-VxWorks-Intel-support-to-ObjectAda)

09.10.2010 | PRODUCT

RTS launches IEEE 1588 Precision Time Protocol for Windows (/electronics-products/electronic-product-reviews/connectivity/4207571/RTS-launches-IEEE-1588-Precision-Time-Protocol-for-Windows)

incoming network data from the controller; once data is received, it will be echoed back to the controller. Round-trip data transaction timing, produced by these cooperating programs, will be observed to evaluate system performance.

Putting it to the test

A prototype network consisting of two nodes-the controller and a remote node, connected by 10BaseT Ethernet-was constructed to facilitate the gathering of empirical network throughput and timing measurements. Phar Lap's TNT real-time kernel was chosen as the run-time environment on all nodes. In addition to the kernel, Phar Lap Ethernet device drivers and TCP/IP stack were also used. Ampro 486-class 100MHz PC/104 boards were chosen for each node. Even though our aim was to use a CPU with an integrated Ethernet 10BaseT port for our product, it was not yet available, so for test purposes an Ampro PC/104 network interface card with an SMC 91C94 Ethernet interface chip was used. An oscilloscope connected to the parallel port of each node facilitated timing analysis; certain bits were set and cleared around significant events in the application software to serve as timing indicators. Using an oscilloscope was probably overkill for this project, but it provides a "sanity check" and it looked cool, especially around a bunch of software engineers. figure 2 (http://m.eet.com/media/1074316/0101ia1fig2.gif) illustrates the test system.

Listings 1 and 2 contain the logic sequence of the controller and remote node programs respectively; the controller code was called at a 60Hz rate while the remote node ran in an endless loop. During the controller's initialization phase a packet was also sent to the remote node. This forced the ARP protocol to be affected at initialization so as not to interfere with our real-time measurements.

Listing 1: Controller program logic

```
Record start time
Set parallel port bit 1
Send pre-constructed buffer to remote node
Clear parallel port bit 1
Set parallel port bit 2
Wait for response packet or timeout of 14ms
Clear parallel port bit 2
Record stop time
If response packet was received
If data mismatch
Log error
Else if timeout
Log error
Calculate best, worst, and average times
```

Listing 2: Remote node program logic

```
Loop forever
    Wait for received packet
    Set parallel port bit 1
    Retrieve data
    Clear parallel port bit 1
    Set parallel port bit 2
    Send data back to controller
    Clear parallel bit 2
```

Analysis of the resulting waveforms and software accumulators revealed that 2ms was the best round-trip time that could be achieved for a 64-byte payload to be sent to a remote node and a response packet received by the controller. The worst-case time was 6ms.

A required cycle time of 16.7ms (the reciprocal of 60Hz) and a worst-case time of 6ms to complete a round-trip transaction for a single node, only two remote nodes could be supported reliably by this system. Clearly, the paltry amount of throughput offered by this scheme would require 25 or more of these independent networks for a single flight simulator. A quantum leap in performance is needed to make this system a viable alternative to our traditional rack-and-stack I/O.

Hold the TCP/IP

Two problems were readily apparent. First, even the best case time of 2ms seemed like a lot of latency for a 10Mbps signaling rate. Second, a huge amount of jitter results. We'll address the jitter later. As far as latency goes, we thought a closer look at what actually traverses the wire would give us some insight into its timing characteristics.

In the test case, we sent 64 bytes of data to the TCP/IP stack. With the added overhead of the protocol headers, shown in figure 3 (http://m.eet.com/media/1074317/0101ia1fig3.gif), our 64 bytes were transformed into a whopping 126 bytes by the time the packet hit the bus. Let's calculate the minimum time for 126 bytes to make a round trip over 10Mbps Ethernet. 126 bytes equates to 1,008 bits, multiply that by 100ns (the reciprocal of 10MHz) to get 100.8s, multiply that by two (because the data is echoed back) and we end up with a round-trip time of 201.6s. Minor factors such as propagation delay were omitted, and in reality we cannot expect to reach this theoretical limit, but by our measurements, the best case time was about 10 times greater than this. Not very efficient!

Early tests performed on the kernel, with varying loads, revealed that the kernel performance was very solid as far as timing and scheduling were concerned. Therefore, we felt that the largest share of latency and jitter was probably caused by the TCP/IP stack or the device driver. Hence, I decided to rewrite the software without using TCP/IP. Since the Phar Lap network driver interface was well documented and utilizing the existing drivers would still provide some hardware independence, the plan was to revise the network access module of the application code to interface directly to the device driver rather than the socket library.

This modification was not a trivial task. Buffer management and packet construction code had to be created. This change makes the application code less portable because it becomes operating system-specific. However, the trade-off would be worth it if the performance increase results in a cost-effective solution.

After the software modifications were completed, the same programs (Listings 1 and 2) were run again. The best-case round-trip time was 0.45ms and the worst-case time was 0.6ms. The modification proved to be a prodigious step in the development of the distributed I/O project; it improved worst-case performance by an order of magnitude. Part of this performance gain is due to the reduced packet overhead, which came from losing the UDP and IP headers, but most of it is the result of not having to pass data through a stack that has more functions than a Swiss Army knife.

In the new configuration, the system could support over 20 remote I/O nodes per network with the minimum sized payload at the highest required rate. Figure 4 (http://m.eet.com/media/1074318/0101ia1fig4.gif) is a screen capture from the oscilloscope connected to the parallel ports of the prototype system. Here is a description of the traces:

- Top-Controller node signal set high when a prepared packet is sent to the device driver. Returns low when control is returned from the driver. The rising edge of this signal is where packet round-trip timing begins
- Second-Controller node signal set high when receive packet is expected. Returns low when packet is received. The low-going edge of this signal indicates the end of a packet's round-trip journey
- Third-Remote node signal set high when notified by driver that a packet is available. Returns low when the application program has retrieved the packet
- Bottom-Remote node signal set high prior to transmit packet preparation. Returns low when control returns from the call to the driver to send the packet. This time includes copying the data from received packet to transmit packet

Based on empirical values taken from the prototype system, figure 5 (http://m.eet.com/media/1074319/0101ia1fig5.gif) illustrates the effect of average packet size on round-trip time. Notice the time never goes much below 500s; this is due to the minimum packet size requirement of Ethernet.

Building a system

The controller node will parse a configuration file and store configuration information for each remote node. This information will be used to dynamically configure the remote nodes via the network as they are brought on-line. To ensure system determinism, no remote node in the network may transmit on the Ethernet bus until it first receives a transmission from the controller, at which time it will reciprocate by sending a single packet appropriate for the type of packet that it received from the controller. That is, if the controller sends a data packet, the designated remote node must return a data packet. Likewise, if a configuration packet is sent by the controller, the only acceptable response is a configuration response packet. The remote node must perform all of its data processing and I/O scanning between packet receptions from the controller.

To achieve maximum CPU utilization, the controller always processes the previous node-received packet, and then the next node-transmit packet, while waiting for the current node-response packet to arrive. For example, if the controller has just sent a packet to remote node 3, the controller will then process the data (if any) previously received from node 2. Next, the packet destined for node 4 (or whichever node is next) will be constructed. The controller waits for a packet from node 3 or a timeout. The cycle then repeats by sending the previously constructed packet to node 4 then processing the data received from node 3, and so on.

Packets in the distributed I/O network conform to the Ethernet frame shown at the top of figure 3 (http://m.eet.com/media/1074317/0101ia1fig3.gif). The frame-type field is used to identify the the function of the frame to the receiving node. Table 1 shows the descriptions of the types used. The eight-bit field following the type field is the node ID. It always carries the ID of the remote node regardless of the direction of the packet. Following that is an 8-bit error code field, which is followed by user data for the remainder of the packet.

Table 1: Type field descriptions

Type	Description
080A	MAC address query
080B	MAC address query response
080C	Configuration packet
080D	Configuration response packet
808	Data packet

Each node has an 8-bit port, on the proprietary I/O board, which is jumpered to identify that node to the system. For each remote node that has an entry in the configuration file, the controller will broadcast a request for the node's Ethernet MAC address using the (home brew) MAC Address Query (MAQ) protocol. In a properly configured system only one remote node should respond to a given MAQ. The remote node, with the node ID that matches that of the MAQ packet, responds by sending a MAQ Response packet. The remote node also stores the MAQ packet's source address, which is used to fill the destination address field for all further transmissions. The MAQ Response packet contains the responding node's MAC address in the source field. The controller then stores this address in a table to reference this remote node in the future. (Hey! That sounds a lot like ARP.)

After a node's MAC address is determined, configuration packets are sent to it. The remote node responds to each configuration packet with a configuration response packet, to let the controller know the packet was received. Configuration information received by the remote node is stored and used for real-time data conversion and scaling. Each distributed I/O network will have a configurable rate, known as the frame rate, at which the controller cycles. Frame rates are in the range of 1Hz to 100Hz. Sub-rates of one-half, one-fourth, one-eighth, and one-sixteenth of the controller frame rate can be assigned on a node basis. These sub-rates allow for a denser system (more nodes on the network) without violating timing requirements of higher speed nodes.

Figure 6 (http://m.eet.com/media/1074320/0101ia1fig6.gif) is a screen shot of a network analyzer that has captured the distributed I/O in action. Another remote node has been added to the network. Using Table 1 as a reference, you can follow the sequence of events for remote MAC address acquisitions, configuration cycles, and normal data transfers. Notice the time field, which has a resolution of 1ms. Multiple transactions take place in a single millisecond. Observing the time field also reveals the 60Hz frame rate of the controller. Conclusion

I have no intention of swearing off TCP/IP; to the contrary, this project has given me a greater understanding of that venerable protocol. Hopefully, the future will bring new opportunities to put that understanding into practice. When it becomes economically feasible to use diminutive CPUs with integrated 100Mbit (or 1Gbit) Ethernet ports, I'm certain we will once again entertain the possibility of using TCP/IP for our distributed I/O.

Joe Kerkes has been with BAE Systems, in the Flight Simulation and Training business unit, for 18 years. He is currently a Senior Computer Systems Engineer and has been designing embedded applications and systems for the last 11 years. Prior to that he designed test programs and interfaces for automated test equipment. He can be reached by e-mail at jkerkes1@tampabay (mailto:jkerkes1@tampabay). rr.com.

Endnotes

1. Just to give you an idea of how far some people are willing to stretch the jargon, I saw one vendor that advertised a hardware product with an embedded TCP/IP stack, but this product did not actually support the TCP protocol at all, only UDP. Go figure.
Back

Resources

Comer, Douglas E. *Computer Networks and Internets, 2nd ed.* Upper Saddle River, NY: Prentice Hall, 1999.

Stevens, Richard W. *TCP/IP Illustrated, Volume 1: The Protocols.* Reading, MA: Addison-Wesley, 1994

Figures

- Figure 1 (http://m.eet.com/media/1074315/0101ia1fig1.gif) Simulator topology with distributed I/O
- Figure 2 (http://m.eet.com/media/1074316/0101ia1fig2.gif) Test system
- Figure 3 (http://m.eet.com/media/1074317/0101ia1fig3.gif) Ethernet frame and IP datagram sizes
- Figure 4 (http://m.eet.com/media/1074318/0101ia1fig4.gif) Timing events of prototype system (without TCP/IP)
- Figure 5 (http://m.eet.com/media/1074319/0101ia1fig5.gif) Round trip time vs. packet size
- Figure 6 (http://m.eet.com/media/1074320/0101ia1fig6.gif) Network analyzer screen shot of distributed I/O network initialization sequence

Share G+1 0 Tweet Like 0

(mailto:?subject=Real-Time Ethernet&body=http://www.embedded.com/design/connectivity/4023291/Real-Time-Ethernet)

WRITE A COMMENT

Subscribe to RSS updates all articles (/rss/all) or category ▾

DEVELOPMENT CENTERS

All Articles (/development)

Configurable Systems (/development/configurable-systems)

Connectivity (/development/connectivity)

Debug & Optimization (/development/debug-and-optimization)

MCUs, Processors & SoCs (/development/mcus-processors-and-socs)

Operating Systems (/development/operating-systems)

Power Optimization (/development/power-optimization)

ESSENTIALS & EDUCATION

Products (/products/all)

News (/news/all)

Source Code Library (/education-training/source-codes)

Webinars (/education-training/webinars)

Courses (/education-training/courses)

Tech Papers (/education-training/tech-papers)

COMMUNITY

Insights (/insights)

Forums (http://forums.embedded.com)

Events (/events)

ARCHIVES

Embedded Systems Programming / Embedded Systems Design Magazine (/magazines)

Newsletters (/archive/Embedded-com-Tech-Focus-Newsletter-Archive)

Videos (/videos)

Collections (/collections/all)

ABOUT US

About Embedded (/aboutus)

Contact Us (/contactus)

Newsletters (/newsletters)

Advertising (/advertising)

Editorial Contributions (/editorial-contributions)

Site Map (/site-map)

Programming Languages & Tools (/development/programming-languages-and-tools)

Prototyping & Development (/development/prototyping-and-development)

Real-time & Performance (/development/real-time-and-performance)

Real-world Applications (/development/real-world-applications)

Safety & Security (/development/safety-and-security)

System Integration (/development/system-integration)

GLOBAL NETWORK **EE Times Asia (http://www.eetasia.com/)** **EE Times China (http://www.eet-china.com/)** **EE Times Europe (http://www.electronics-eetimes.com/)** **EE Times India (http://www.eetindia.co.in/)** **EE Times Japan (http://eetimes.jp/)** **EE Times Korea (http://www.eetkorea.com/)** **EE Times Taiwan (http://www.eettaiwan.com/)** **EDN Asia (http://www.ednasia.com/)** **EDN China (http://www.ednchina.com/)** **EDN Japan (http://ednjapan.com/)** **ESC Brazil (http://www.escbrazil.com.br/en/)**

(http://ubmcanon.com/)

Communities

EE Times (http://www.eetimes.com/) | **EDN (http://www.edn.com/)** | **EBN (http://www.ebnonline.com/)** | **DataSheets.com (http://www.datasheets.com/)** | **Embedded (http://www.embedded.com/)** | **TechOnline (http://www.techonline.com/)** | **Planet Analog (http://www.planetanalog.com/)**

Working With Us: About (http://www.aspencore.com/) | Contact Us (http://www.aspencore.com/#contact) | Media Kits (http://go.aspencore.com/mediakit)