Insights into the IEEE 1394 Serial Bus Protocol obtained with a Novel Traffic Analyzer

Dan Steinberg, Yitzhak Birk
Technion – Israel Institute of Technology

Abstract

*The IEEE 1394 Serial Bus protocol (*FireWire®*) provides for a high-speed, plug and play, peer-to-peer interconnect supporting both asynchronous and isochronous traffic. In this paper, we investigate the unique arbitration mechanisms employed by nodes in a 1394 topology in order to gain a deeper insight into the efficiency, fairness, and robustness of the protocol. To this end, the Statistics Collector & Analyzer (SCA) has been designed, developed, and utilized to record, display, and analyze performance measurements from an active 1394 bus in real-time. Experimental data collected with the SCA exposes situations where unfairness exists, and examines the performance of higher-level protocols running over 1394 in various configurations.*

1 Introduction

The IEEE 1394 Serial Bus protocol (*FireWire*®) provides for a high-speed, plug and play, peer-to-peer interconnect supporting both asynchronous and isochronous traffic. The unique arbitration and synchronization mechanisms employed by nodes in a 1394 topology undoubtedly have a significant effect on the performance of both native and higher level protocols.

In this work, we investigate the actual performance of various 1394 topologies in different configurations in order to gain a deeper insight into the efficiency, fairness, and robustness of the protocol. To this end, the Statistics Collector & Analyzer (SCA) has been designed and developed to record, display, and analyze performance measurements from an active 1394 bus in real-time. The SCA, which is designed to run upon the CATC™ FireInpsector™ hardware platform, is a powerful tool that enables us to conduct in-depth experimental research of the 1394 protocol. Specifically, we investigate the ability of the protocol to interleave asynchronous and isochronous traffic simultaneously. We also take a close look at the arbitration mechanism for insuring fairness, and identify and demonstrate cases where "unfairness" exists. In addition, we examine the effect that gap count optimization has on native and higher level protocols, and expose some of the subtleties of the 1394 specification, including possible effects of a gap count mismatch in a poorly managed topology.

2 IEEE 1394 Overview

The IEEE 1394-1995 and 1394A specifications detail a high performance serial bus with many advanced features. IEEE 1394 targets the convergence of consumer electronics and high-speed computer peripherals. The following are some of the key attributes that are supported: Up to 63 devices connected in a single topology, throughput rates of 100, 200, and 400 Mb/s, automatic configuration with plug and play, guaranteed latency and bandwidth for isochronous applications, and peer to peer connectivity with no host intervention needed.

2.1.1 Topology Configuration

The physical topology of a 1394 network consists of up to 63 devices connected by serial cables with no loops allowed. The protocol implements a scheme for automatic configuration of the logical topology. The configuration process for a 1394 topology consists of several stages: bus initialization, tree identification, and self-identification. Bus initialization or bus reset will primarily be the result of power on initialization, device connection or removal, or software initiation. Following a bus reset the tree identification process is initiated by all nodes on the bus, which results in the determination of the root node, branch nodes, and leaf nodes in a non-deterministic manner. Loops are not allowed and will result in a non-functioning bus until the loop connection is removed. Upon completion of the tree identification phase, the root node initiates the deterministic process of self-identification that assigns a unique node ID to each device in the topology and determines the isochronous resource manager or bus manager. This node ID is used as a source or destination identifier for all ensuing packets transferred on the bus. Every time the physical topology changes or software initiates a bus reset, the selection of the root node and some or all of the node IDs may change.

2.1.2 Normal Arbitration

Arguably the most interesting aspect of the FireWire protocol is its unique arbitration mechanism. Once the topology configuration is complete, normal arbitration begins. In general, time on the bus is divided into 125 microsecond cycles. Nominally, the root node broadcasts a cycle start packet at the beginning of each cycle to indicate to all nodes that a new cycle has begun.

Nodes that have previously reserved bandwidth from the isochronous resource manager may arbitrate and transmit isochronous data on one of 63 possible channels. According to the specification, up to 80% of the total bandwidth for each cycle may be allocated for isochronous traffic. Once all of the isochronous traffic is complete, the rest of the cycle may be used for asynchronous transmission.

Nodes that wish to transmit asynchronous packets arbitrate for the bus by signaling requests up towards the root and signaling deny to other neighboring nodes further from the root. These requests work their way up to the root node that grants the bus to the first request that reaches it, and signals deny to nodes located on other branches of the tree. The node that wins arbitration then transmits the packet. Although the packet is physically broadcast to all nodes on the bus, it is only processed by the node specified by the destination node ID. The receiving node will immediately respond to the incoming packet with an acknowledge code, and then the arbitration process may begin again until the cycle completes.

The arbitration mechanism in 1394 defines a fairness interval for asynchronous traffic that is meant to insure that all nodes get an equal opportunity to access the bus and avoid situations of starvation. Arbitration between successive packets within a fairness interval are separated by a minimum amount of idle time called a subaction gap. Each node is allowed to transmit exactly one asynchronous packet per fairness interval. The order of the packets transmitted will tend to favor nodes that are closer to the root. Once a node has transmitted its asynchronous packet, it must wait until all other nodes that wish to transmit complete transmission and an arbitration reset gap is detected. This gap, consisting of a considerably larger amount of idle time than a subaction gap, indicating that the current fairness interval has ended, and a new one has begun. In theory, this mechanism should insure equal access to the channel, but we will see that in certain situations "unfairness" exists.

2.1.3 Gap Count Optimization

The gap count is a parameter that indicates the maximum propagation delay in the topology and determines the corresponding lengths of both subaction and arbitration reset gaps. This parameter is maintained individually by each node, but ideally should be equivalent for all nodes on the bus. The gap count defaults to its maximum value 63. The protocol provides a mechanism to optimize the gap count at all nodes that will reduce the size of the gaps and potentially improve overall performance. The gap count may be optimized based upon the maximum number of hops in the topology. The 1394A specification describes an enhancement called PHY pinging that utilizes actual timing measurements between PHYs to determine the optimal gap count. We will investigate the effect that gap count optimization has on higher level protocols such as SBP-2 and Ethernet over 1394. Also, we will investigate what can happen when the gap count parameter is not identical at all nodes.

3 SCA Architecture

In this section, we provide an overview of the architecture of the SCA and its capabilities. The SCA (shown in figure 1) runs on a CATC FireInpsector hardware platform, and is composed of a statistics engine, embedded SCA firmware, and the SCA application. The FireInpsector platform has a 1394A compliant PHY (PHYsical layer silicon), a high density FPGA, and an embedded microcontroller. The FPGA is loaded with the statistics engine (HDL code) that implements the real time event decoding and counting logic. The microcontroller is loaded with the SCA firmware that configures and periodically samples the counters inside the statistics engine. The sample data is then transmitted up to the SCA application running on a host PC over the USB channel. The SCA application is used to configure, record, playback, and analyze 1394 performance data in real-time. In order to create many of the desired test elements, it is necessary to have a controlled traffic generator that is both powerful and flexible. To this end, we have developed the Traffic application that also utilizes the FireInpsector platform.

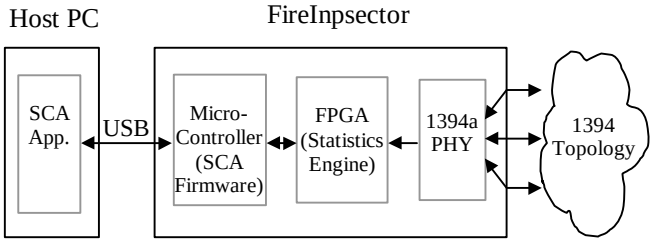

Figure 1: SCA System Architecture

3.1.1 Statistics Engine

Perhaps the most challenging portion of the overall SCA design is the real-time decoding and analysis of the 1394 traffic within the statistics engine. The statistics engine decodes the incoming 1394 traffic data from the PHY, which is an active participant in the 1394 topology. The SCA does not initiate the transmission of packet data, but the PHY does take part in the normal tree identification, self-identification, and arbitration processes. Ideally, an analyzer should be completely passive, but we feel that this minimal gain does not warrant the greatly increased complexity of building custom hardware to do so.

The inputs to the statistics engine are the control and data lines of the PHY-Link interface as specified in the 1394A specification. The PHY converts the high-speed serial data into parallel data consisting of 2, 4, or 8 streams running at 49.152 MHz. These streams are sent to the PHY decode logic which converts them into status

words, data quadlets, and acknowledge octets. Hardware counters keep track of basic events such as the total number of quadlets, and the total number of acknowledges. Event detect logic performs a more in-depth analysis of the data and status received. The incoming quadlets are sorted into packets consisting of header quadlets and data quadlets. Counters record the number and type of packets. Status words are decoded into bus resets, subaction gaps, and arbitration reset gaps which increment respective counters. In addition to regular traffic, error conditions such as bad header and data CRCs, bad acknowledge parity, PHY interrupts, reserved transaction codes, and others are recorded. Finally, user configurable counters can track 1394 traffic for a specific source-destination node pair. Periodically, all of the counters are sampled by the micro-controller with an atomic read and clear operation. This mechanism is implemented in the statistics engine to compensate for the relatively slow execution time of the micro-controller as compared to the speed of 1394 traffic.

In order to make an intelligent analysis of the traffic on the 1394 bus, it is vital to have an up-to-date view of the logical topology. In this vein, upon detection of a bus reset the statistics engine saves the self-ID packets in an internal memory block. This data is extracted by the micro-controller and sent to the application, which can accurately recreate and display the current topology from the self-ID packets.

3.2 SCA Firmware

The SCA firmware has the responsibility of configuring and collecting counter data from the statistics engine and passing them up to the application over the USB channel. The application initiates the collecting of samples at a rate that may be configured within the application. Currently, the polling rate may be as fast as 50 times a second, which enables investigation and analysis of 1394 performance with fine grain resolution.

3.3 SCA Application

The SCA application primarily provides the user interface to configure and record statistical performance data from the active 1394 topology under test. The application is implemented with multiple threads to provide for the real-time capture, analysis and graphical display of the data. Currently, the application tracks over 50 different statistics per sample including: overall effective throughput and utilization of the bus, throughput rates for specific source and destination nodes, error rates, breakdown of different packet types and speeds, and monitoring of specific isochronous channels. Although data speeds over 1394 can approach 400 Mb/s, the hardware analysis and data reduction performed in the statistics engine allows the application to record data for hours or days with minimal storage requirements.

Once the performance data is recorded, the SCA application can perform an in-depth analysis of the data file tabulating totals, and detecting trends that may not have been noticeable during the recording. The application also has the ability to open previously recorded files and play them back recreating the graphical display and analysis.

3.4 Controlled Traffic Generator

The controlled traffic generator was developed in order to generate configurable and repeatable traffic stimulus as well as help in configuring the topology. The generator consists of a dialog application that interfaces to the FireInpsector executing SCA firmware. The application allows the user to configure the type, length, speed, and destination of the packet to be generated. The desired packet may be sent continuously, periodically, or in bursts. In addition, an interactive mode is supported that enables two traffic generators to communicate using a simple stop and wait protocol complete with acknowledges. Finally, the traffic generator may be used to initiate bus resets, force a specific node to become root, and change the gap count value.

4 1394 Performance Analysis

In this section, we use the SCA to collect meaningful statistics that will provide insight into various performance aspects of the 1394 protocol.

4.1 Isochronous & Asynchronous Mix

The first group of test measurements focuses on how well the protocol supports interleaving real-time and asynchronous traffic. Specifically, we verify that as the bus saturates only nodes that wish to transmit asynchronous traffic are affected, and isochronous traffic proceeds smoothly. Also we examine the question: What is the optimal amount of bandwidth to allocate to isochronous traffic in relation to its effect upon the transfer of asynchronous traffic? According to the specification, the isochronous resource manager may allocate up to 80% of each cycle for isochronous traffic, but what effect will this level of allocation have on higher-level asynchronous protocols such as IP over 1394?

In the first experiment, the setup consists of five nodes: two OHCI cards for isochronous traffic generation, two FireNet™200 cards for asynchronous traffic generation, and an SCA for performance monitoring. Substantial asynchronous traffic is generated using the "NetTraffic" test application provided by Unibrain. This application performs repeated file transfers between two host PCs using the IP over 1394 protocol. The offered asynchronous traffic remains constant at about 50 Mb/s or

25% utilization of the bus, and data points are collected as the offered isochronous load is increased. The results are plotted in figure 2. The curve for the isochronous traffic is linear indicating that it was unaffected by the background asynchronous traffic. The asynchronous throughput was also relatively unaffected by the real-time traffic until the total bus utilization increased above 85%. As the bus reaches saturation, the isochronous data continues without incident, but the asynchronous throughput decreases rapidly. It is important to note that even with more than 80% of the cycle allocated for real-time data, the asynchronous file transfers are able to complete successfully.

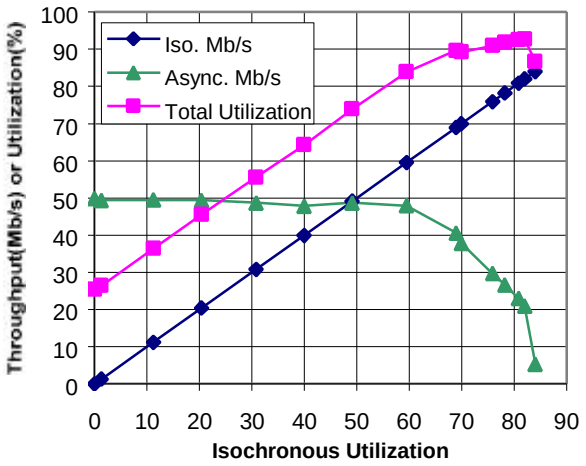

Figure 2: Effect of Isochronous Traffic on Asynchronous Traffic I

In the second experiment, we have repeated the same principle, but chosen a different offered asynchronous load in order to verify that the attainable utilization does not depend upon the working point. The offered isochronous load is the same as before, but this time we have used two controlled traffic generators to generate the asynchronous traffic. Each generator is composed of a FireInpsector unit running our "Traffic" application, which simply sends a fixed length packet repeatedly to a specified node. The packet length and speed selected correspond to roughly 168 Mb/s or 42% utilization of the bus. Again, data points are collected as the offered isochronous load is increased. As the graph in figure 3 indicates, isochronous throughput is linear and unaffected by the asynchronous traffic. The asynchronous traffic, however, begins to decrease as the bus saturates, and utilization rises above 80%.

The primary conclusion that can be made from the above tests is that asynchronous and real-time traffic can be interleaved successfully in 1394. As long as the overall bus utilization stay below 80%, both traffic types are virtually unaffected by each other. Above 80% utilization, however, the asynchronous throughput is dramatically affected and the isochronous traffic continues to flow smoothly.

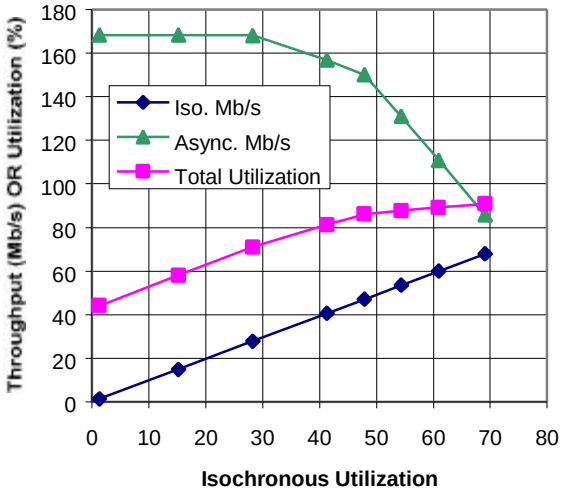

Figure 3: Effect of Isochronous Traffic on Asynchronous Traffic II

4.2 Fairness Protocol

As stated previously, the arbitration mechanism in 1394 uses the concept of a fairness interval to insure that nodes on the bus wishing to transmit asynchronous packets will each have equal opportunities to access the bus. While conducting the experiments above, we noticed a specific situation where both controlled traffic generators did not seem to get fair access to the channel, and we began to investigate further.

Upon close examination of the arbitration protocol, it is clear that there will exist cases of unfairness. The problem stems from a relatively minor detail in the specification which causes nodes further from the root to, on average, have a longer latency from the time they wish to transmit until the time they are granted ownership of the bus. At first glance this scheme seems fair, since the longer initial arbitration time should be offset by the fact that each node is guaranteed the ability to transmit one asynchronous packet during each fairness interval. In some cases, a transmitting node may be modeled as a system that has a fixed amount of processing time that begins upon completion of a packet transmission and ends when the next packet is ready to be sent to the PHY. If this is the case, then two identical nodes located at different distances from the root will have unequal success transmitting if the total time required by each node to process, arbitrate, and send a packet are different.

In order to verify this, a topology is created as shown in figure 4 composed of the SCA, three controlled traffic generators, a FireInspector, and an OHCI card. Generator B and The OHCI card are used to generate background asynchronous and isochronous traffic respectively. The FireInspector functions as a passive

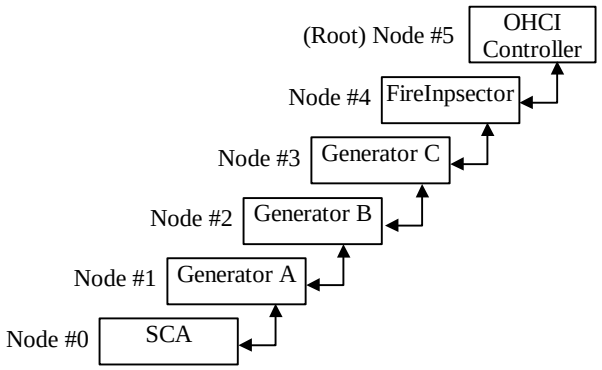

Figure 4: Topology for Fairness Tests

node that may be used to snoop individual packets. Nodes A and C are configured to generate bursts of asynchronous packets separated by a fixed amount of processing time. Furthermore, they are both running identical firmware with identical parameters. The only difference among them is that generator C is physically located closer to the root node. The SCA is configured to monitor performance of Generators A and C in particular and the overall bus in general.

In order to measure the relative unfairness, we define the following:

Q_A = Total Quadlets transmitted from node A
Q_C = Total Quadlets transmitted from node C
R = Relative unfairness in % = $| Q_C - Q_A | / Q_A * 100$

Adjusting the various parameters, a value for R as high as 19% was attained. In other words, node C was 19% more successful in asynchronous packet generation that node A even though they are identical. In order to verify that this phenomenon occurs with real applications and not just in synthetic tests, a similar setup was created replacing the controlled traffic generators with FireNet cards. Performance was measured for two topologies that differed only with respect to the location of the FireNet cards. Again the relative unfairness was calculated yielding a result slightly greater than 1%. Although this difference is smaller than the synthetic tests, it does demonstrate that even real world applications will be affected by this minor flaw in the 1394 protocol.

4.3 Gap Count Effect on Performance

The gap count parameter is an essential part of the arbitration mechanism in 1394, and here we investigate its effect on performance. In the first set of tests, we look at the effect on higher level protocols such as SBP-2 and IP over 1394. In a configuration with two FireNet cards and the SCA, we vary the gap count over the full range of possible values and note that the throughput between the FireNet cards remains unchanged. A similar test is run with an OHCI controller and an SBP-2 disk drive. Again the gap count value is varied over the full range and the asynchronous throughput from the drive to the host is unchanged. In both cases, we can see that delays between back to back packets are shorter, but significant processing delays remain the same. The conclusion here is that the bottleneck is in the processing of packets at the host and not in the 1394 channel. For this reason, gap count optimization may be unnecessary in many topologies.

In a bus that is poorly managed or unmanaged, a gap count mismatch may occur. In practice, we have observed this situation quite frequently, and decided to investigate what may happen when nodes disagree on the value of the global parameter. A topology is setup with two controlled traffic generators and the SCA. Generator B has a gap count that is configured to a value of 4, which is the minimum value recommended for a topology with two hops. It is connected to the SCA and Generator A, which both have the maximum gap count value of 63. When generator A alone is configured to generate packets at maximal speed continuously, the result is a throughput of 119 Mb/s. Generator B alone is able to generate a throughput of 188 Mb/s. This difference indicates that the gap count will indeed have a significant effect upon native 1394 protocols with little or no latency. When both generators attempt to transmit identical packets continuously the combined throughput reaches 189 Mb/s. Using the source destination match feature of the SCA, we can see that the breakdown of the traffic among the generating nodes greatly favors the node with the smaller gap count. Specifically, node A only succeeds in transmitting 36206 quadlets in the same amount of time that node B succeeds in transmitting 3095201 quadlets. The resulting ratio favors node B by more than 85:1! The explanation for this result is that node B will, in general, begin arbitrating for the channel very soon after transmitting a packet, while node A waits. Node A will time a longer gap before it recognizes that it has the right to arbitrate. When node A finally does arbitrate for packet transmission, it will likely already have lost the arbitration and wait for the next gap. In summary, a mismatch in the gap count parameter can, under certain circumstances, have a dramatic effect on the arbitration mechanisms resulting in unfairness or even starvation.

5 Conclusions

In this paper we described the architecture of a novel performance analysis tool termed SCA that was developed to investigate the IEEE 1394 serial bus protocol. The usefulness of the SCA is demonstrated by exposing strengths and weaknesses inherent in the protocol. Experimental data indicates that isochronous and asynchronous traffic mix well together until the bus saturates. Also, the fairness protocol was shown to have certain flaws that result in unequal access to the bus. Finally, examples of the types of protocols that are affected by the gap count parameter were presented as well as the possibly severe results of a gap count mismatch in a poorly managed bus.