Design and Performance Evaluation of a New Spatial Reuse FireWire Protocol

by

Vijay Chandramohan

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Kenneth J. Christensen, Ph.D.
Miguel Labrador, Ph.D.
Nagarajan Ranganathan, Ph.D.

Date of Approval:
September 19, 2003

Keywords: Video Surveillance, Bus Arbitration, IEEE 1394b
Medium Access Control (MAC), Sensor Networks

ACKNOWLEDGEMENTS

I would like to thank Dr. Christensen profusely for his valuable support, encouragement, and guidance throughout my graduate studies and during the course of this research. I would like to thank Dr. Ranga and Dr. Labrador for being on my committee. I extend my thanks to my family and friends who have been a constant source of encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Design and Performance Evaluation of a New Spatial Reuse

FireWire Protocol

Vijay Chandramohan

ABSTRACT

New generations of video surveillance systems are expected to possess a large-scale network of intelligent video cameras with built-in image processing capabilities. These systems need to be tethered for reasons of bandwidth and power requirements. To support economical installation of video cameras and to manage the huge volume of information flow in these networks, there is a need for new shared-medium daisy-chained physical and medium access control (bus arbitration) layer communication protocols.

This thesis describes the design principles of Spatial reuse FireWire Protocol (SFP), a novel request/grant bus arbitration protocol, architected for an acyclic daisy-chained network topology. SFP is a new extension of the IEEE 1394b FireWire architecture. SFP preserves the simple repeat path functionality of FireWire while offering two significant advantages: 1) SFP supports concurrent data transmissions over disjoint segments of the network (spatial reuse of bandwidth), which increases the effective throughput and 2) SFP provides support for priority traffic, which is necessary to handle real-time

applications (like packet video), and mission critical applications (like event notifications between cameras) that have strict delay and jitter constraints.

The delay and throughput performance of FireWire and SFP were evaluated using discrete-event queuing simulation models built with the CSIM-18 simulation library. Simulation results show that for a homogeneous traffic pattern SFP improves upon the throughput of IEEE 1394b by a factor of 2. For a traffic pattern typical of video surveillance applications, throughput increases by a factor of 7. Simulation results demonstrate that IEEE 1394b asynchronous stream based packet transactions offer better delay performance than isochronous transactions for variable bit rate video like MPEG-2 and MPEG-4. SFP extends this observation by supporting priority traffic. QoS for packet video is provided in SFP by mapping individual asynchronous stream packets to the three priority classes.

CHAPTER 1

INTRODUCTION

There is a growing need for visual monitoring and surveillance systems in large facilities like airports and stadiums, for safety and security. New systems are envisioned with thousands of highly intelligent, interconnected cameras with rich image processing capabilities. Cameras can be used to monitor crowd behavior and access to restricted areas. With facial recognition, monitoring extends to identification of profiled individuals. Many novel applications for video surveillance are envisioned [8], [21], [24]. As these applications grow in complexity and demand, the underlying networks that support video surveillance need to evolve as well.

1.1 Motivation

Existing dedicated medium switched Ethernet/ATM based video surveillance systems require a communication cable per node (connected to a switch). This dedicated cabling will soon become the cost and performance bottleneck to further deployment of large-scale (e.g., thousands of cameras in one installation) video surveillance systems. For high-resolution video and localized processing (image analysis) in each camera, power cannot be delivered for very long by a battery. Power distribution can be combined with

communication [15]. For economical installation of large-scale video surveillance systems, there is a need for new shared-medium, daisy-chained network technologies with built-in power distribution. Very significantly, new bus arbitration protocols capable of supporting multiple, high bit-rate video traffic must be investigated.

1.2 Thesis contributions

This thesis investigates new communication protocols suitable for video surveillance systems, in particular at the medium access control level (bus arbitration) and physical layer. The main contributions of this work are:

- Review of suitable technologies for low-cost networking for video surveillance.
- Performance evaluation of FireWire as a candidate technology for video surveillance.
- Design and performance evaluation of Spatial reuse FireWire Protocol (SFP), a new bus arbitration protocol, which improves the effective throughput of FireWire by spatial reuse of bandwidth, and QoS support for packet video by a real-time priority based bus access mechanism.

1.3 Thesis organization

The remainder of this work is organized as follows.

- Chapter 2 describes the evolution of networks for video surveillance. This chapter studies the performance issues in FireWire protocols, especially the bus arbitration mechanisms.
- Chapter 3 describes the design of the Spatial reuse FireWire Protocol (SFP).
- Chapter 4 evaluates the queuing delay and the throughput performance of SFP and of existing FireWire protocols for packet-based video transmissions.
- Chapter 5 presents a summary of results and discusses directions for future research.

CHAPTER 2

BACKGROUND

Sensor networks, also called Embedded Networks (EmNets) [7], are envisioned to tie together embedded systems. EmNets can enable distributed processing and entirely new computational models. In many applications, sensor networks are wireless [2], [4], [6]. For applications such as habitat monitoring in a jungle [2] or tracking of items in a warehouse [6], a wired network is not possible. However, for sensor applications with fixed locations and high bandwidth and power demands a wired sensor network (WSN) is needed [3]. One such application of national importance is video surveillance. Open problems in video-based sensor networks include; collaboration of the large volume of sensor information, developing efficient image processing and video compression algorithms, and most importantly designing scalable, low-cost network technologies capable of providing the required QoS for high bit-rate video traffic [8], [21].

2.1 Evolution of networks for video surveillance

With image processing included as part of an intelligent camera, many new applications for video surveillance can be envisioned [8], [21], [24]. Common to all of these applications is the need to communicate video, still images, or event notifications

between peer cameras and/or to one or more monitoring or sensor fusion points. New applications force an evolution in the underlying networks that support video surveillance systems. Four generations of networks for video surveillance systems are identified:

- Existing – based on analog cameras and dedicated coax cabling.
- Emerging – based on digital cameras and Ethernet or ATM with dedicated, unshielded twisted-pair cabling.
- Near future – based on intelligent digital cameras with processing capability and shared cabling in acyclic topologies.
- Future – based on intelligent digital cameras and arbitrary topologies where shared-medium acyclic clusters and store-and-forward nodes capable of routing will co-exist.

The near future and future generations are predictions. Figures 1, 2, 3 and 4 show the four generations of video surveillance systems in order.

For the existing generation, the coax cables for all cameras are brought into a common control room in which the analog video signals are monitored by a human and/or recorded. In the control room human operators observe a large wall of monitors, each with rotating views from multiple cameras. The operators may have the ability to lock a monitor to a specific camera and physically control the orientation and/or zoom-in of a camera, in order to manually follow a suspicious target. This existing generation is limited by the cost of the coaxial cabling (which can exceed camera costs) and the number of video streams that can be monitored by a human. Image processing of the video streams would be difficult given that only centralized processing is possible.

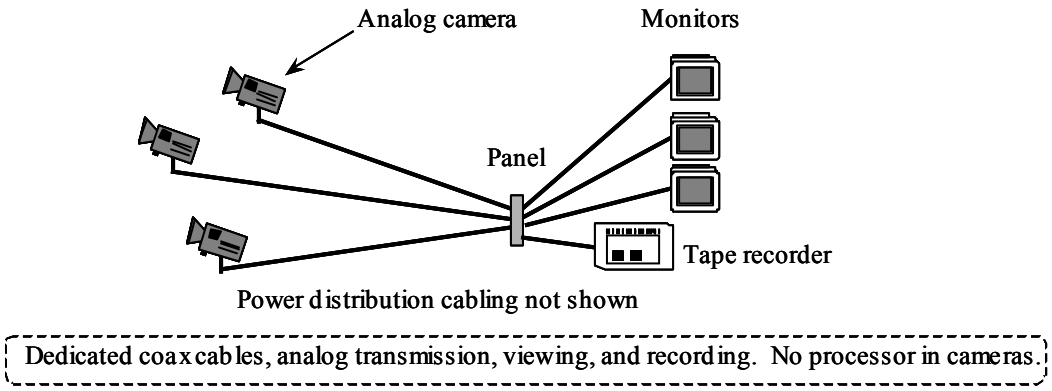

Figure 1. Existing video surveillance systems

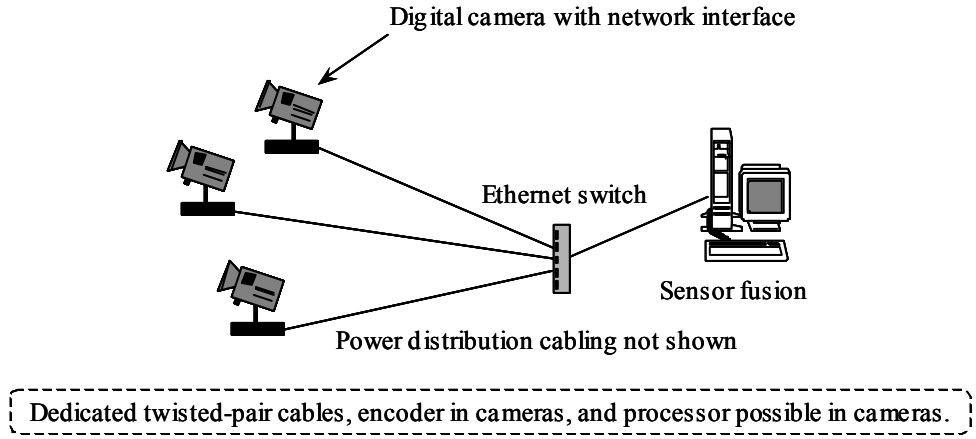

Figure 2. Emerging video surveillance systems

The emerging generation uses low-cost digital cameras and incorporates an Ethernet [19] (or an ATM network [22]) connection in the camera unit. Video is transmitted from the camera to a central point as MPEG-2 using IP. By using Ethernet, lower cost unshielded twisted-pair cabling (e.g., UTP-5 for 100BaseT) can be used. With the video stream already in digital and packet format, transmission over an existing IP network and/or recording on a PC hard disk are easily accomplished. With a continued decrease in the cost of cameras, but no similar decrease in the cost of copper or labor to install cabling, a bottleneck will soon be hit.

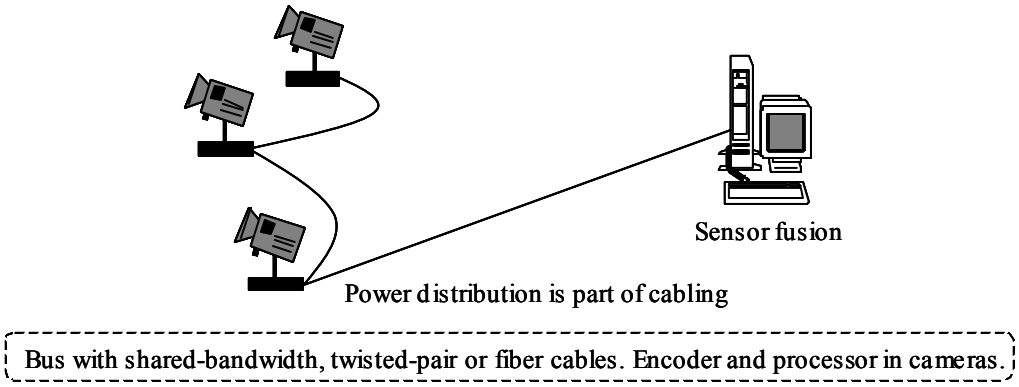

Figure 3. Near-future video surveillance systems

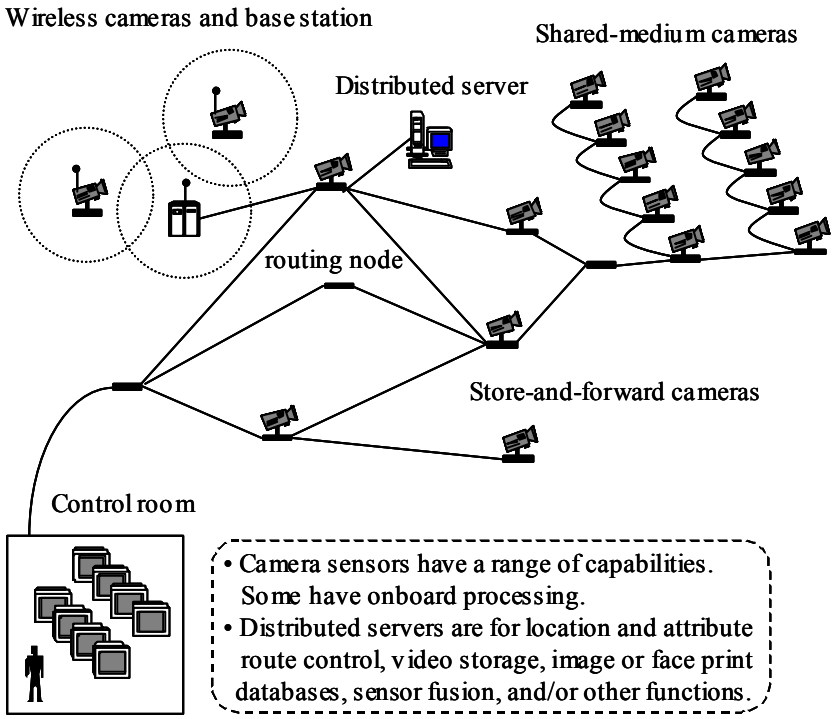

Figure 4. Future video surveillance systems

It is envisioned that the future generations of video surveillance systems will require shared-medium and direct-wired networks to reduce cabling costs and allow for ad hoc installation of cameras. In the near future, acyclic networks with intermediate clustering points are predicted. Tree branches can be extended with a new camera, or a camera

inserted into a branch. In the longer-term future, shared-medium acyclic clusters will co-exist with new store-and-forward nodes. These store and forward nodes will include routing and caching capabilities. With such nodes, arbitrary network topologies become possible with redundant links for reliability and added bandwidth. Very significantly, these future WSNs can enable new models of distributed image processing. These systems may have distributed information (routing) servers to facilitate novel location and attribute based routing between video sensor nodes. Routing issues in WSNs with an arbitrary topology are studied in [3].

Image processing localized in, or even distributed between, cameras is needed to build highly autonomous video surveillance systems that will require human intervention only on the detection of critical events. Existing, low-cost cameras are already capable of motion-detection and tracking [5]. For localized image processing in each node, battery power is not sufficient. Thus, wiring is needed for power distribution. Existing video surveillance systems have two cabling systems, one for power and another for communications. Power distribution can be combined with communications. IEEE 802.3af standardizes power distribution on an Ethernet link to allow for nodes without separate power wiring [15]. FireWire includes power distribution as part of the standard cable bundle. Wired sensor networks will combine power distribution with communications [3].

2.2 Overview of FireWire

FireWire was originally developed by Apple (in 1987) as an extended serial bus technology intended to replace expensive parallel peripheral buses such as PCI (Peripheral Component Interconnect) and ISA (Industry Standard Architecture bus). Over the time, FireWire has evolved into a versatile method for interconnecting wide variety of high-bandwidth consumer electronic devices, peripherals and computers. FireWire is the only existing technology that supports a shared-medium daisy-chained topology and has built-in power distribution. It is believed that a shared-medium is necessary to support ad hoc installation of nodes and reduce cabling costs compared to dedicated medium technologies (such as switched Ethernet or ATM). Each FireWire node is a part of the repeat path. Nodes may have one or more ports to support branching and hence tree topologies. A FireWire cable consists of three pairs of wires, two for data transmission and one for power conductors as shown in Figure 5. All FireWire standards employ shielded twisted pair (STP) cabling. IEEE 1394b also uses plastic optic fiber (POF) and multimode fiber (MMF) for added bandwidth and distance. A FireWire cable cross-section is approximately 5 millimeters in diameter.

The number of nodes on a FireWire serial bus is limited to 63. Upto 1024 serial buses can be bridged together in a single network. The standard for FireWire bridges was still under development at the time of this writing [16]. There exist three versions of the FireWire standard, IEEE 1394-1995 [14], IEEE 1394a [13], and IEEE 1394b [12]. Each standard has successively increased the bandwidth and the reach of the serial bus network. Each

standard has also improved upon the bus arbitration mechanism. Table 1 gives a performance summary of the three FireWire standards. Performance of IEEE 1394 has been studied analytically in [20]. References [11] and [17] describe methods of transmitting IP packets over FireWire. In [9], IP over FireWire was compared to IP over Gigabit Ethernet, and it was found that throughput was very similar. Reference [27] provides a detailed capacity utilization analysis of IEEE 1394 FireWire. The rest of this chapter presents the Firewire architecture and performance issues. Bus arbitration mechanisms are described in detail.

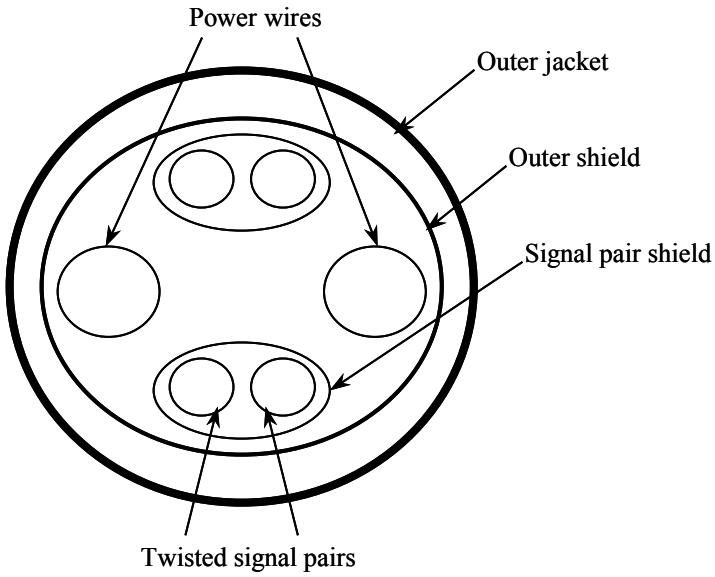

Figure 5. FireWire cable cross-section, taken from [1]

2.3 Basic operation of FireWire

Prior to the normal operation of FireWire a bus configuration phase must take place. Bus configuration is responsible for the "plug-and-play" feature of the network and occurs

whenever a node is added to or removed from the bus. This phase includes tree identification and self identification. During tree identification, nodes exchange a series of handshake signals to establish a parent / child relationship among them, and to determine the root node. The root node claims the bus ownership, and plays an important role in bus arbitration and several bus management activities. Usually tree identification fails if there is a loop in the topology. IEEE 1394b provides a solution to this problem by selective disabling of links [12]. Bus configuration also establishes the topology of the network. During self identification each node in the serial bus is assigned a unique address called "self id", which ranges between 1 and 63. A detailed study of bus configuration in FireWire is given in [1].

Table 1. Summary of FireWire standards

	IEEE 1394	IEEE 1394a	IEEE 1394b
Internode distance	4.5 meters (max)	4.5 meters (max)	100 meters (max)
Maximum hops	16	63	63
Physical medium	STP	STP	STP, POF, MMF
Cable bandwidth	100, 200, 400 Mbps	100, 200, 400 Mbps	Up to 1.6 Gbps
Loop prevention	No	No	Yes
Arbitration	Large idle gaps	Small idle gaps	No idle gaps

All FireWire data transactions are packet based and can be broadly classified as asynchronous or isochronous. Asynchronous transactions are guaranteed in delivery and require an acknowledgement from the receiver. They are unicast in nature. Isochronous

transactions are guaranteed in time with a specific bandwidth reserved for them on the serial bus. Up to 80% of the bus bandwidth can be allocated for isochronous transactions. Bandwidth is allocated in portions of 125 microsecond intervals, called cycles. Isochronous transactions are multicast in nature, addressed to one or more nodes based on a channel number. FireWire supports another transaction service called asynchronous streaming, which is guaranteed neither in time nor in delivery. Asynchronous streaming can be unicast or multicast. FireWire data packets are variable in size. Maximum data payload size depends upon the type of transaction and the bandwidth of the FireWire serial bus. Figures 6 and 7 show different packet formats in FireWire. Asynchronous stream and isochronous packets follow the same format.

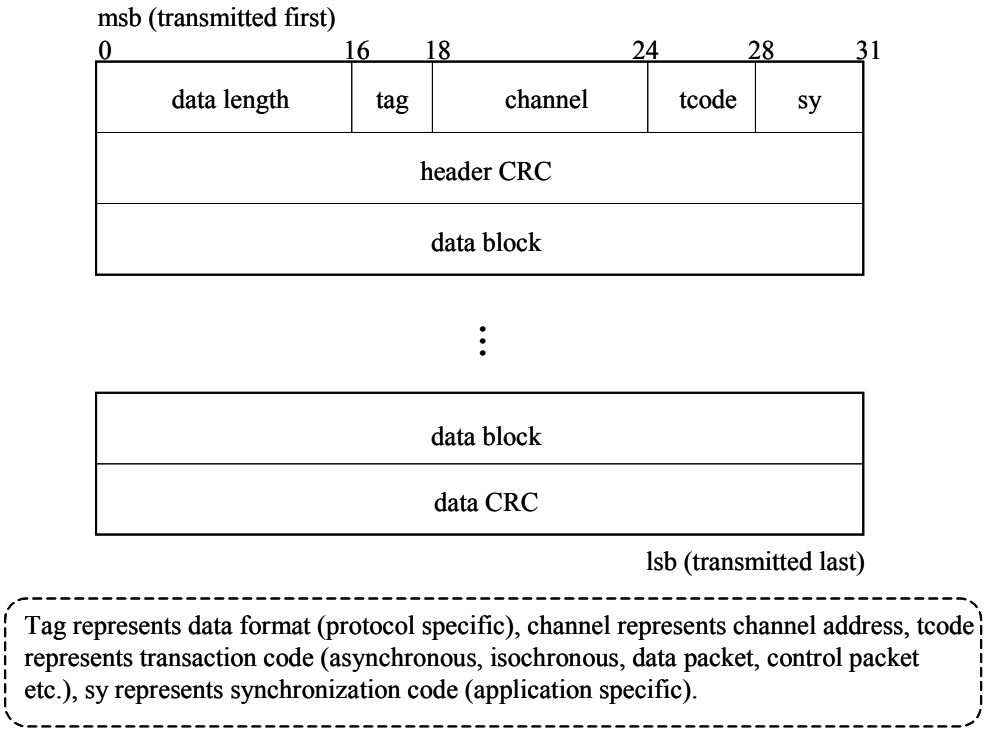

Figure 6. FireWire asynchronous stream and isochronous packet format, taken from [1]

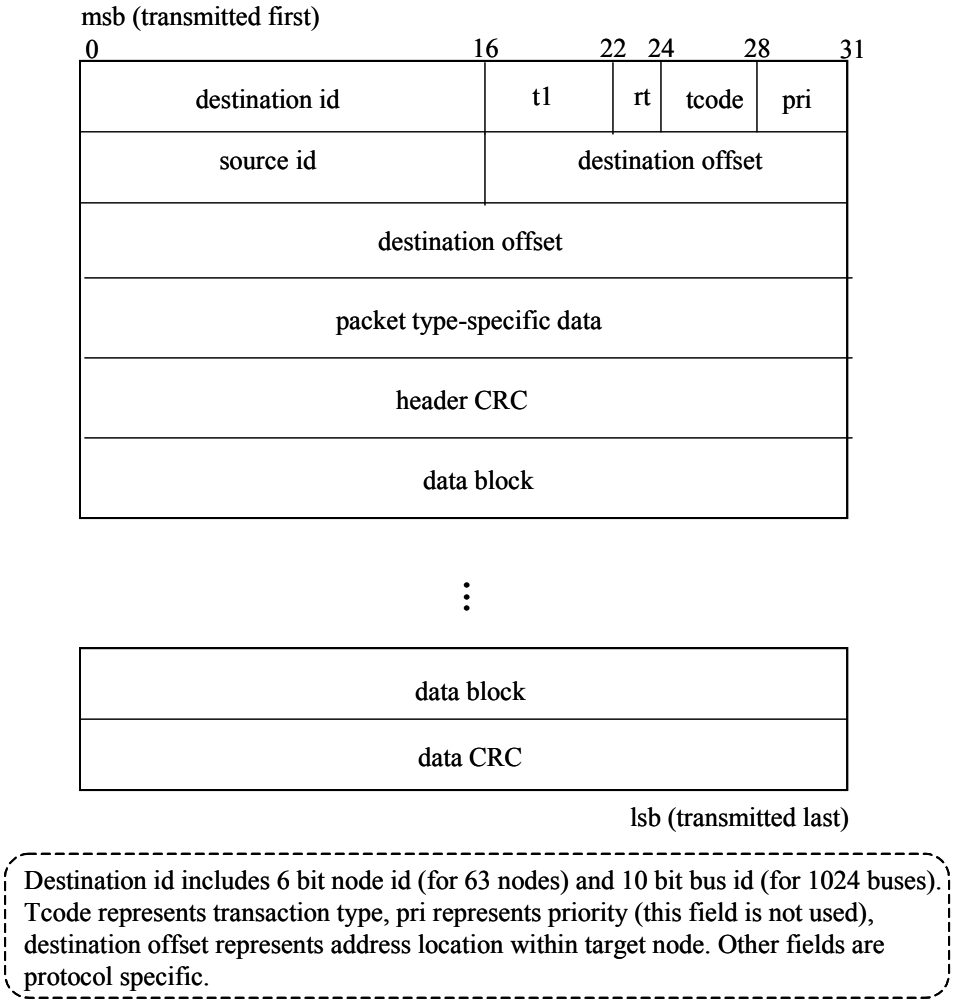

Figure 7. FireWire asynchronous packet format, taken from [1]

FireWire architecture is built upon a four-layer protocol stack as shown in Figure 8. The physical layer implements bus arbitration, defines electrical signaling for data transmission and mechanical interface for cables and connectors. The link layer provides address and channel number decoding and CRC generation and verification for transmitted and received packets. The transaction layer provides request-response services for asynchronous transactions. Isochronous transactions operate independent of

this layer. The bus management layer provides support for several bus management activities and bus configuration. FireWire requires three primary bus management nodes for normal operation. They are cycle master, Isochronous Resource Manager (IRM), and bus manager. The cycle master generates and broadcasts cycle start packet every 125 microseconds. A cycle start packet denotes the beginning of the periodic 125-microsecond interval. The root node plays the role of cycle master. The IRM manages serial bus isochronous bandwidth and also allocates multicast channel numbers. The bus manager manages cable power distribution and publishes the topology map and the speed map of the serial bus. A speed map is necessary since FireWire can support nodes/cables of different bandwidth capacity in a single network. Usually, all nodes are capable of performing these bus management activities. However, the operational bus management nodes are elected during the bus configuration phase.

2.4 Bus arbitration in FireWire

FireWire employs a request / grant arbitration mechanism to control access to the shared-medium network. A simple arbitration scheme works as follows:

- Nodes that wish to transmit a packet request the bus owner for permission.
- The bus owner selects a best request based upon certain criteria and issues a grant to the corresponding node.
- Only the granted node transmits its packet, the other nodes continue to request until they receive a grant from the bus owner.

In IEEE 1394-1995 [14] and IEEE 1394a [13] the bus owner is always the root node. In IEEE 1394b [12] all arbitrating nodes perform the role of bus owner in a round-robin basis. A detailed description of FireWire bus arbitration is given in the following sections.

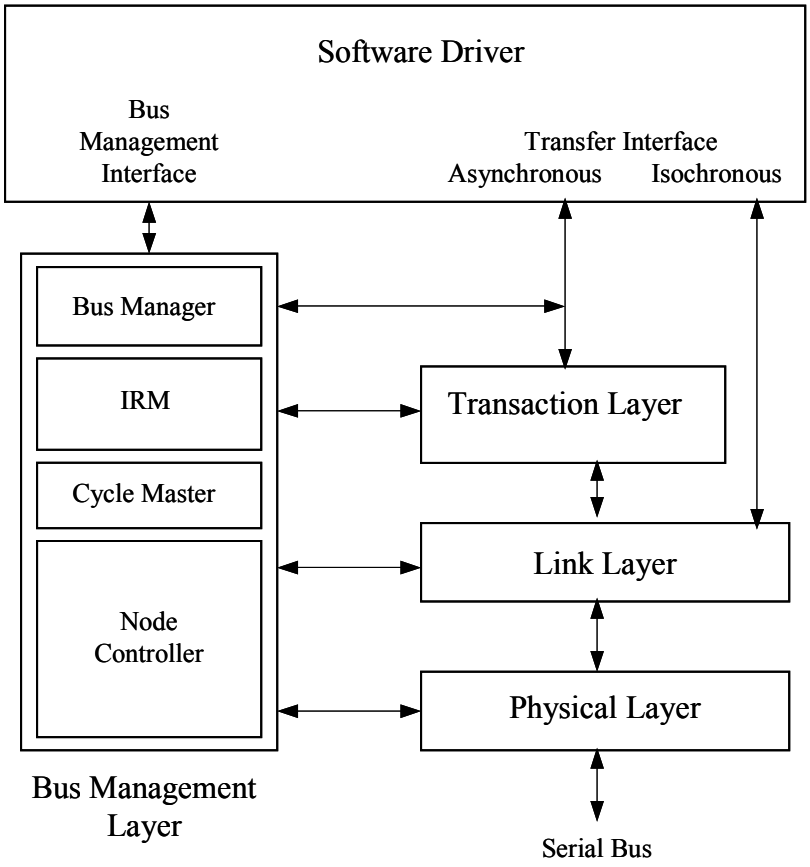

Figure 8. FireWire protocol stack, taken from [1]

2.4.1 Bus arbitration in IEEE 1394 and IEEE 1394a FireWire

The basic arbitration mechanisms employed in IEEE 1394 and IEEE 1394a are the same, with a few arbitration enhancements proposed in the latter [13]. Figure 9 shows a FireWire data transmission interface (physical layer) with two twisted pairs TPA and

TPB that are crosswired within the cable (between nodes). Data transmission in IEEE 1394a FireWire is done via data / strobe signaling. Binary data is transferred across one twisted pair and the strobe signal across the other [12]. The strobe signal changes if the data stays the same. This makes data transfer operation essentially half-duplex. Bus arbitration is performed using arbitration signals. Arbitration signals are not clocked data but rather are steady state signals across the twisted pairs [1]. Arbitration signaling can be bi-directional. Two connected nodes are permitted to drive their lines at the same time. In Figure 9, twisted pairs TPA and TPB have arrows at both ends, which indicates that both the lines must be driven simultaneously for data transmission or arbitration signaling.

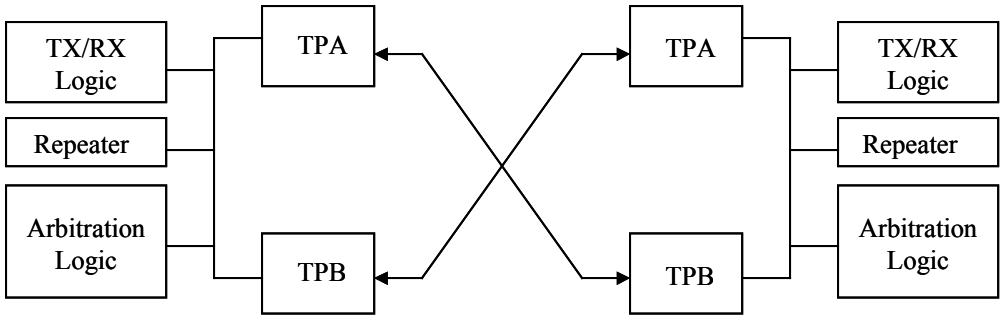

Figure 9. IEEE 1394 and IEEE 1394a data transmission interface

When a node wishes to perform a data transaction it must arbitrate for the bus. Bus arbitration can be isochronous or asynchronous depending upon the type of transaction. Arbitrations are based upon the periodic 125-microsecond cycle, the start of which is indicated by a broadcast cycle start packet. Isochronous arbitrations can begin immediately after nodes detect the cycle start packet. Only those nodes that have reserved a specific bandwidth on the bus can perform isochronous arbitration. An arbitrating node

signals a request towards its parent. Each parent in turn repeats the request upwards towards its parent until the request reaches the root node. When both parent and child arbitrate for the bus, the parent overrides the child's request. The root issues a grant signal for the received request, which in turn is repeated downwards until it reaches the requesting node. When the root receives multiple requests (i.e. on several ports) the request at the lowest numbered port is granted. The winning node transmits its isochronous data. The next isochronous arbitration can begin only after all nodes detect a specific amount of idle bus time, the isochronous gap, to make sure that the previous data transmission has completed. Every node can perform only one successful isochronous arbitration in a cycle. The end of isochronous arbitrations is marked by a larger bus idle time. This idle gap time is called the subaction gap. At the detection of a subaction gap nodes can begin their asynchronous arbitrations.

Asynchronous arbitrations also employ the same request / grant signaling as used in isochronous arbitrations. After completion of an asynchronous data transmission, the next arbitration can begin only after all nodes detect a subaction gap. This ensures that asynchronous nodes receive their acknowledgements before a new arbitration begins. An acknowledgement packet does not require arbitration and can be sent immediately on the receipt of an asynchronous packet. Asynchronous transactions are divided into fairness intervals. Every node can successfully arbitrate maximum of once during an asynchronous fairness interval. When a node completes a data transaction it must give up any further arbitration in the current fairness interval. This ensures equal and fair sharing of the FireWire bandwidth between asynchronous nodes. When all arbitrating

asynchronous nodes complete their data transmission, the bus goes idle for a long arbitration reset gap. This gap indicates the end of a fairness interval and the beginning of a new one. The arbitration reset gap is larger than the subaction gap. Figure 10 shows a typical arbitration sequence in IEEE 1394 and IEEE 1394a FireWire. IEEE 1394a proposes acknowledge accelerated arbitration and fly-by arbitration that reduce the idle bus period to some extent [13]. Every FireWire node transmits data packets (both isochronous and asynchronous) on all active ports. Intermediate nodes repeat the packet on all ports except on the one on which it was received. Data packets are stripped by the end nodes in the network (no destination stripping is permitted).

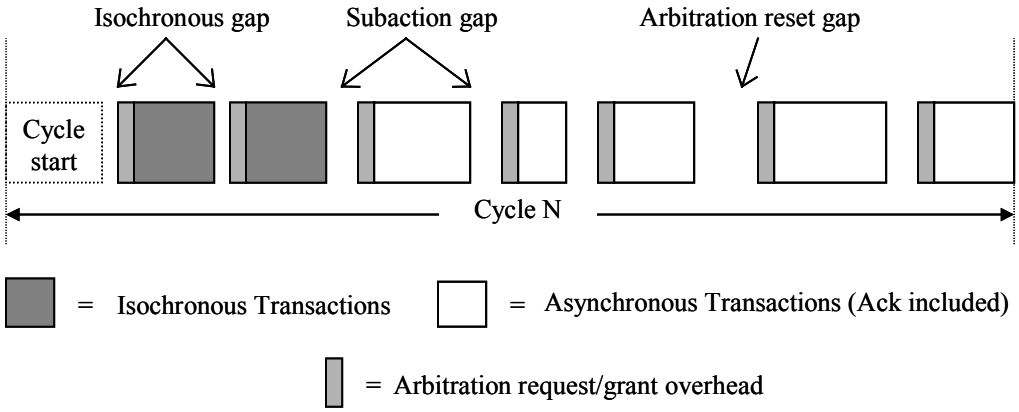

Figure 10. Arbitration sequence in IEEE 1394 and IEEE 1394a

2.4.2 Bus arbitration in IEEE 1394b FireWire

Earlier versions of FireWire alternate between arbitration and data transmission that were separated by distinct idle bus times. Idle bus occupancy vastly reduced the performance of FireWire. IEEE 1394b employs a new beta mode signaling that helps arbitration requesting to be overlapped with data transmission [12]. Arbitration overlapping

completely eliminates the idle bus occupancy seen in the previous standards. Beta mode signaling is a version of 8b/10b signaling protocol that is used in Gigabit Ethernet and Fibre Channel specifications. Beta mode signaling does not require both signal pairs for unidirectional data transfer. The signal pairs TPA and TPB can transmit data separately and continuously in opposite directions as shown in Figure 11. TPA and TPB have arrows at opposite ends, which indicates that only one of the lines need to be driven for data transmission or arbitration signaling. This full-duplex nature of the IEEE 1394b bus enables overlapping of arbitration with data transmission. In IEEE 1394b arbitration signals are not steady line states across the twisted pairs but rather are 10-bit symbols called tokens.

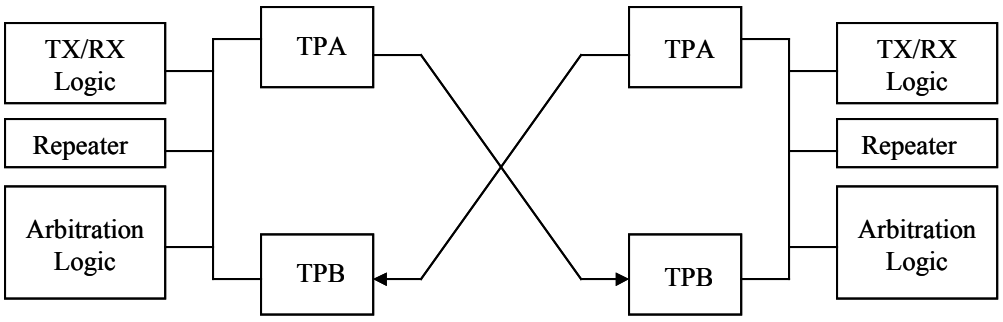

Figure 11. IEEE 1394b data transmission interface

In IEEE 1394b the bus owner is not a fixed root node. All arbitrating nodes perform this role in a round-robin fashion. The last node to transmit a packet that does not require an acknowledgement acts as the next bus owner. The node claiming bus ownership is called the BOSS (Bus Owner Supervisor Selector). A node that transmits an isochronous packet, an acknowledgement packet, or an asynchronous stream packet becomes the BOSS and is

responsible for making the next arbitration decision. When a node wishes to perform a data transaction it sends out an arbitration request token towards the BOSS. Arbitration tokens are sent out on any active port that is not transmitting (repeating) a data packet. Arbitration tokens propagate in the opposite direction from a data packet. As in IEEE 1394a, IEEE 1394b arbitrations are divided into isochronous and asynchronous intervals. Both isochronous and asynchronous intervals alternate between "even" and "odd" arbitration phases. The concept of an arbitration phase is similar to the fairness interval scheme seen in IEEE 1394a. Any node that has transmitted an asynchronous / isochronous packet in the current phase can arbitrate only for the next / opposite phase. Each asynchronous phase is a fairness interval. In IEEE 1394a, two fairness intervals were separated by an idle bus period, called an arbitration reset gap. However, in IEEE 1394b the BOSS explicitly advances fairness intervals by sending out an "arbitration reset token" that specifies the beginning and the phase of a new fairness interval. When the BOSS sees no pending asynchronous requests for the current phase, it advances the phase by sending an ASYNC_EVEN / ODD token corresponding to the new phase.

Isochronous arbitrations begin when nodes see a cycle start token. When there are no pending isochronous arbitrations the BOSS begins an asynchronous arbitration interval by sending out an ASYNC_EVEN / ODD token. Each node transmits request tokens based upon the current phase and its transaction type. Arbitration request tokens are classified as isochronous or asynchronous and are also prioritized. Intermediate nodes always forward the highest priority request token to the next node. The BOSS issues a grant token towards the highest priority request that it receives. When the BOSS receives

two or more requests of the same priority then the request at the lowest port number is granted. Each grant token identifies the current phase and transaction type of the granted request. Every intermediate node can keep the grant for itself or forward it to other nodes based upon the priority of its own request and other requests. A detailed description of IEEE 1394b arbitration is given in [12]. Figure 12 shows a typical arbitration sequence in IEEE 1394b FireWire. It can be seen that successive isochronous and asynchronous packet transactions are separated only by a small arbitration grant overhead. The arbitration grant overhead is the time taken by a grant token to reach the source node and the amount of overhead depends upon the propagation and the repeat path delays.

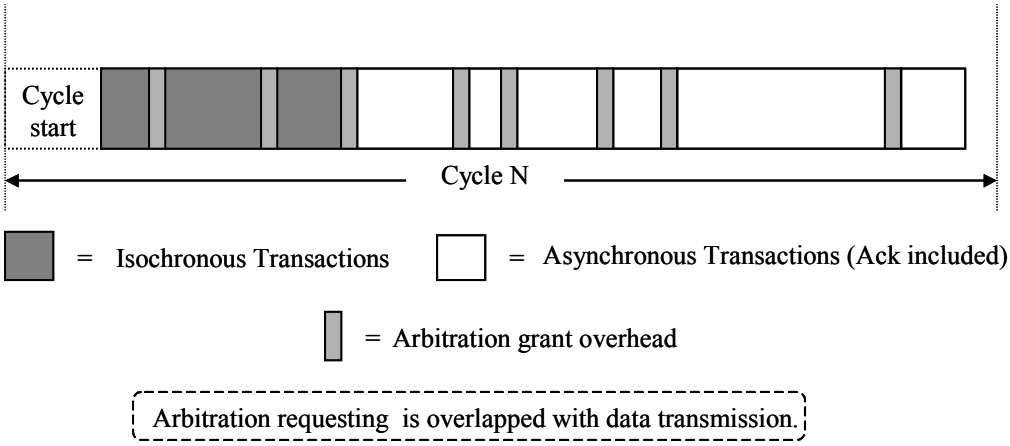

Figure 12. Arbitration sequence in IEEE 1394b

2.5 Performance limitations in FireWire

Though IEEE 1394b offers a higher throughput by completely eliminating the idle bus occupancy seen in earlier versions it still has certain performance limitations, which are discussed in this section.

2.5.1 Lack of spatial reuse

IEEE 1394b envisions the entire network as a single logical serial bus. Every node transmits (repeats) incoming packets on all out-going ports and destination stripping of data packets is not possible. FireWire does not permit concurrent packet transmissions (spatial reuse) over distinct segments of the network. For example, Figure 13 shows an *N* node FireWire video network with nodes linked in a daisy-chained fashion. In this example, node 2 is sending traffic to node 1 and node 4 to node 6. Though these transmissions occupy non-overlapped (distinct) segments of the network, FireWire does not permit them to occur simultaneously. FireWire bus arbitration schedules these transactions to occur one after one. This limits the throughput of FireWire to single link capacity. To increase the effective throughput of FireWire and to improve its scalability beyond the 63-node limit, it is necessary to incorporate spatial reuse in FireWire. The idea of supporting spatial reuse in large (wide area) daisy-chained networks is not new. An emerging technology that supports this concept is Cisco's SRP [26]. The scope of SRP is a metropolitan area ring topology network with a limited size of 32 to 64 nodes. SRP nodes store and forward incoming packets and have layer 3 routing capabilities. Congestion and fairness control is accomplished by a distributed control mechanism where control packets are continuously propagated between adjacent nodes in opposite direction from the data packets. Each data transaction involves a processing overhead (packet scheduling) and a store-and-forward overhead at every node. One goal of this thesis is to incorporate spatial reuse feature in IEEE 1394b while preserving the simple

repeat path functionality (physical layer) and the request/grant bus arbitration model of FireWire.

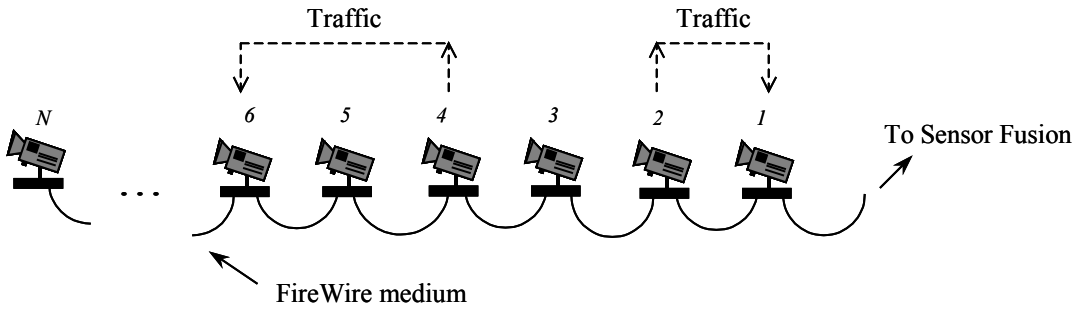

Figure 13. Lack of spatial reuse in FireWire

2.5.2 Lack of support for priority traffic

FireWire provides QoS guarantees for real-time traffic (like packet video) by isochronous bandwidth reservation. Isochronous nodes reserve a fixed amount of bandwidth on a per cycle basis. This service is not suitable for high-resolution variable bit-rate (VBR) encoded video like MPEG-2 and MPEG-4. Figures 14 and 15 show the rate snapshot of an MPEG-2 and an MPEG-4 video, respectively. The MPEG-2 video rate is 25 frames per second with a mean data rate of about 5 Mbps. The MPEG-4 video rate is 25 frames per second with a mean data rate of about 0.766 Mbps. The isochronous bandwidth reservation scheme lacks the flexibility to react to the rate variations as seen in MPEG-2 and MPEG-4 video traffic. Reserving a bandwidth corresponding to the peak bit-rate will result in a waste of resources. A real-time priority based packet scheduling mechanism will be more suitable for widely used VBR video and will provide efficient use of computing resources [25]. The priority mechanisms in IEEE 1394b FireWire (i.e., the

request token priorities) provide a means for alternating between isochronous and asynchronous arbitrations and ensuring bandwidth fairness. It is necessary to incorporate a priority service in FireWire so that QoS for packet video and mission critical applications can be provided by mapping traffic in different priority classes.

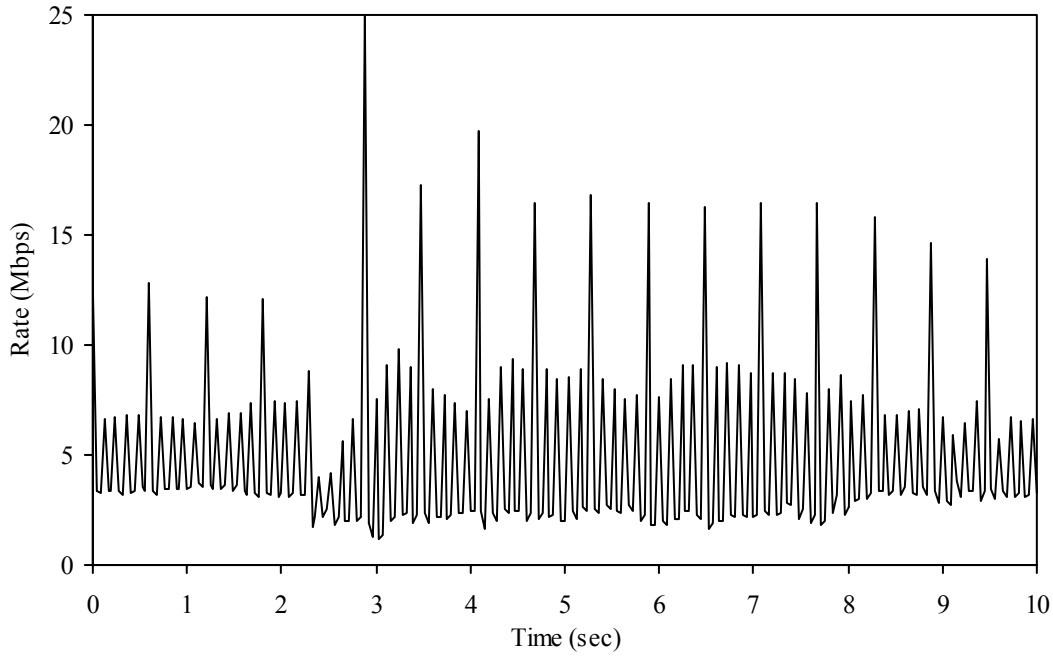

Figure 14. Rate plot for MPEG-2 video

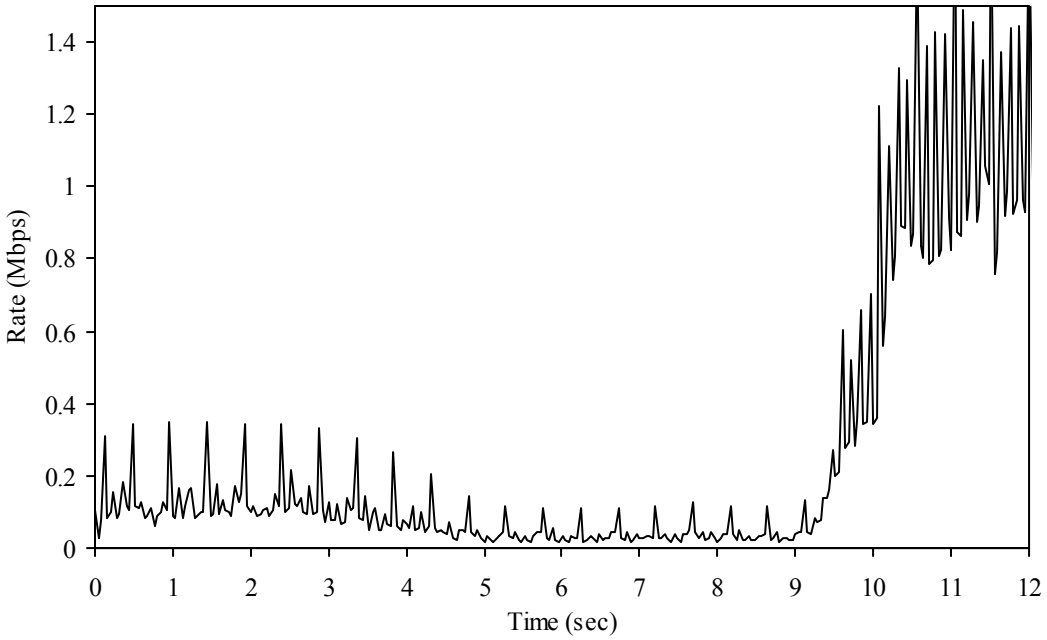

Figure 15. Rate plot for MPEG-4 video

CHAPTER 3

SPATIAL REUSE FIREWIRE PROTOCOL (SFP)

Spatial reuse FireWire Protocol (SFP) is a request/grant bus arbitration protocol architected for an acyclic daisy-chained network (bus) topology. SFP preserves the simple repeat path architecture of IEEE 1394b FireWire while providing two significant improvements. 1) SFP increases the aggregate throughput of the network by spatial reuse of bandwidth by simultaneous data transport in multiple, distinct segments of the network. 2) SFP provides support for priority traffic, which forms the basis for real-time scheduling towards improved QoS support for packet video. This chapter describes the design principles of SFP.

3.1 Overview of bus arbitration in SFP

The core of SFP is the bus arbitration mechanism used for controlled access to the shared-medium network. A simple SFP arbitration scheme works as follows:

- *Arbitration requesting*: Nodes that wish to perform a data transaction broadcast a request packet (or "request") that is cached by every node in the network. Request packets are informative, they contain details about the source and destination

nodes involved in a transaction and other data packet properties (such as packet size, priority, etc.)

- *Bus owner arbitration decision*: The current *bus owner* (i.e. the arbitration decision making node) examines the multiple requests in its cache and "selects" a group of "compatible" requests. The request selection procedure is described in section 3.4. Two requests are compatible if their corresponding data transactions occupy non-overlapped segments of the network. For example, in Figure 16, the transactions 'A' and 'B' are compatible. The source nodes corresponding to the selected compatible requests are "granted" (permitted) bus access. The knowledge of multiple requests and the informative nature of requests enable the *bus owner* to make an "intelligent" arbitration decision. The arbitration decision encompasses several "selection" constraints, such as maximizing the throughput of the network, providing support for high priority traffic and ensuring fairness among like priority nodes.
- *Arbitration granting*: The *bus owner* broadcasts a grant packet with information about the "granted" nodes. Nodes that explicitly see a grant for them (in the grant packet) can transmit their data packet concurrently. The grant packet also identifies the destination nodes, which are supposed to strip the next data packet that they receive. Destination stripping enables spatial reuse by limiting bandwidth consumption to the used segments of the network. This work assumes only unicast packets. One of the granted nodes explicitly identified as the next *bus owner* (in the grant packet) takes-up its role at the end of data transmission.

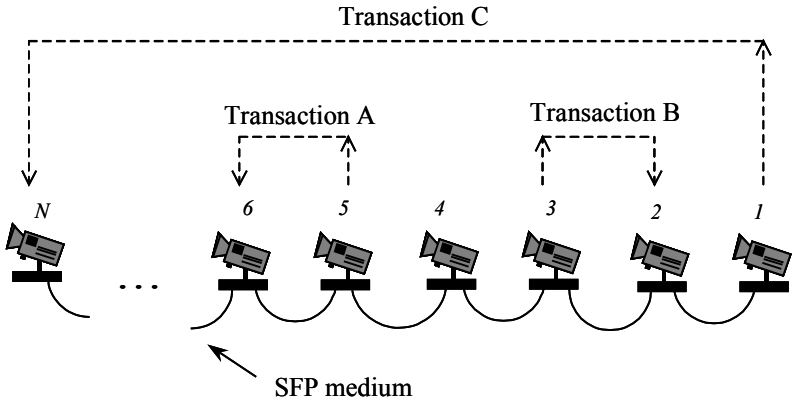

Figure 16. SFP network

3.2 SFP data transmission interface

Figure 17 shows a high-level connection interface between two SFP nodes. The communication link has two twisted signal pairs, TPA and TPB. TPA and TPB are not crosswired within the cable, but operate as two independent half-duplex lines (i.e. there is a TPA-TPA link and a TPB-TPB link between adjacent nodes). Standard FireWire cabling can be used in SFP. TPB is called the request line and is dedicated to carrying arbitration requests. TPA or data line exclusively carries data traffic (and also grant packets) between nodes. TPA and TPB are driven by separate half-duplex transmitter/receiver logic. It is expected that TPA and TPB can independently and concurrently carry traffic between nodes. A signaling method similar to beta mode signaling in IEEE 1394b is assumed. SFP will combine power distribution with communication, but power distribution issues are beyond the scope of this work.

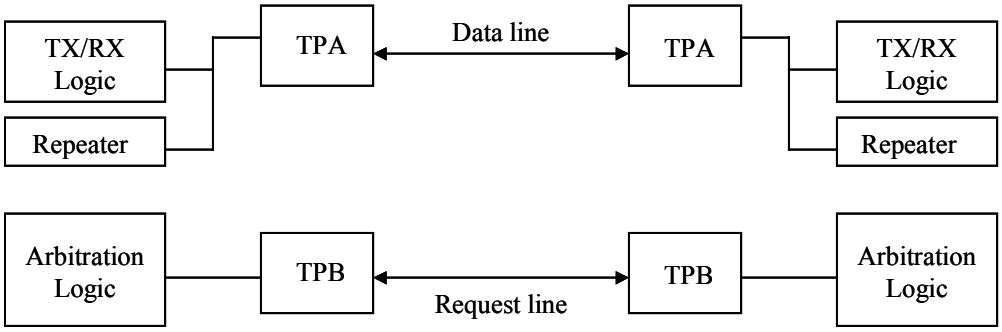

Figure 17. SFP data transmission interface

The TPA interface can operate in two functionalities, *repeat mode* and *blocking mode*. When a node operates in *repeat mode*, it repeats an incoming packet towards its neighbor. When operating in *blocking mode*, nodes strip the next incoming packet. *Blocking mode* enables destination stripping of a data packet without the requirement of destination address lookup (a delay overhead) at every node. Normally, nodes always operate in *repeat mode*. *Blocking mode* operation is permitted only when nodes see their address explicitly identified in the "destination address list" of a grant packet. *Blocking mode* nodes switch to *repeat mode* immediately on stripping the next incoming data packet. Grant packets are always repeated while a data packet can be repeated or stripped. Each node has knowledge of the simple network topology and data packets are always routed towards the destination. A node can source data in one port and concurrently receive (strip) a packet from another port. A network configuration phase as seen in FireWire is assumed in SFP. Network configuration plays a key role in node addressing, topology discovery, and establishment of a *root* node. The SFP *root* node plays an important role in various bus management activities and fault tolerance (like assuming the role of *bus*

owner in the failure of one). This work does not study network configuration and fault tolerance issues in detail. It is assumed that there is no loss of arbitration request and grant packets. Network recovery methods used in IEEE 1394b can be easily extended to SFP.

3.3 Arbitration requesting

Arbitration requests in SFP are not 10-bit tokens as used in IEEE 1394b, but rather are distinct packets of information. For every data packet a node wishes to transmit, it must broadcast a request packet claiming access to the shared data line. Each request packet contains the following fields of information:

- Source id: Address of the node from which the data packet originates. Nodes are addressed 1 to *N*, *N* being the number of nodes in the network
- Destination id: Address of the node to which a data packet is destined.
- Packet phase: Phase of arbitration. Can be *Current* or *Next*. The arbitration phase ensures fairness among like priority nodes.
- Packet size: Size (in bytes) of the data packet for which the request is made.
- Priority: Priority of the data packet for which the request is made. SFP supports three priority classes *High*, *Medium*, and *Low*.

Arbitration request packets are transmitted on the request line (TPB). Since TPB operates in a half-duplex mode there is a need for controlled access to it to prevent packet collisions. This is accomplished by the *synchronous request transfer* mechanism.

Synchronous request transfer: It is assumed that all nodes in an SFP network are synchronized to a common clock. This synchronization takes place during the network configuration phase before the normal network operations begin. It is expected that each node run an *arbitration cycle master* whose time cycle continuously alternates between *even* and *odd* request intervals. Since the nodes are synchronized, the cycle changes occur in all nodes at the same time. At the start of an *even* request interval, even numbered (addressed) nodes can transmit newly received request packets (if any) to their right and left neighbors. At the start of an *odd* request interval, odd numbered nodes can transmit newly received request packets to their neighbors. Every node caches the request it receives and also retransmits it to the neighbors in the appropriate request interval. Nodes do not retransmit an incoming request that is already present in their cache. It can be observed that at any point in time a node may have a maximum of three new request packets to transmit (its own request packet and the packets from its left and right neighbors). So, the duration of a request interval (*even* and *odd*) must be long enough to accommodate three request packet transmissions. Request interval length also depends upon the worst-case hop delay in the network. The duration of a request interval, T_{req}, is,

$$T_{req} = 3\left\lceil \frac{L_{req}}{R} \right\rceil + D_{\max} T_{prop} . \quad (1)$$

In (1), L_{req} is the size of a request packet in bits, R is the bandwidth of the SFP link in bits per second, D_{max} is the maximum internode distance in meters, and T_{prop} is the propagation delay of electrical signals (5 nanoseconds per meter). In SFP, arbitration requesting is never blocked by data traffic and occurs continuously and independent of data transmissions. Nodes can transmit a request packet and a data packet concurrently on

their respective lines. Figure 18 shows arbitration requesting in SFP. Nodes 2, 4, and 6 transmit their request packets at cycle n (*even* request interval). Nodes 1,3, and 5 transmit their request packets at cycle n+1 (*odd* request interval). A more sophisticated approach could be employed for request transfer between nodes. However, the basic idea is to support unblocked arbitration that is overlapped with data transmission. The continuous nature of arbitration combined with the request caching enables the *bus owner* to have a global knowledge of all arbitrating nodes. This knowledge is necessary to make an "intelligent" arbitration decision. The arbitration decision procedure is described in section 3.4.

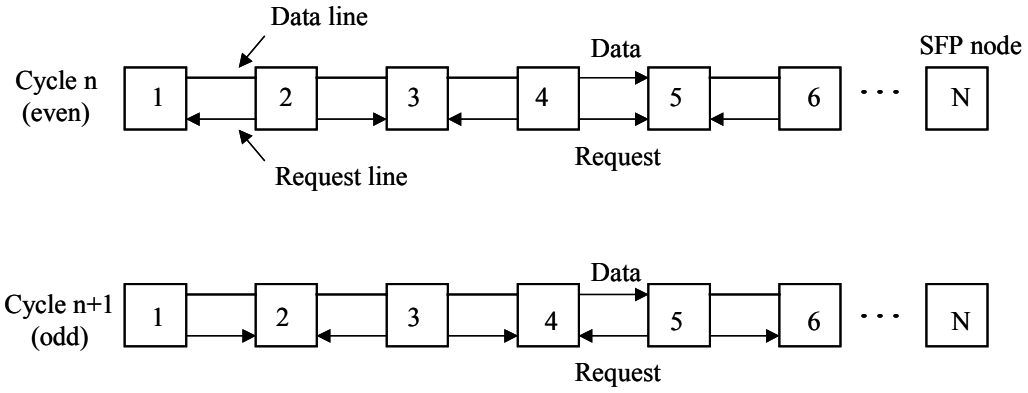

Figure 18. Arbitration requesting in SFP

3.3.1 Support for priority traffic

In SFP, every data packet is prioritized and the *bus owner* ensures expedited bus access to high priority packets. SFP provides support for three priority classes, *High*, *Medium*, and *Low*. Each node implements three priority queues (transmit buffers) corresponding to the three classes of priority. Nodes enqueue packets to be transmitted in appropriate buffers

based on priority. This work assumes infinite capacity for all three transmit buffers and hence no buffer overflows. However, in a real environment buffers may be limited in size and new transmit packets may be dropped due to buffer overflow.

Arbitration requesting can be done for only one buffered data packet at a time (i.e. for the head-of-line packet in the highest non-empty priority queue). After a node arbitrates for the bus it cannot send another request packet (for an additional data packet) until the previous packet transmission is triggered (started). However arbitration for a *Low* / *Medium* priority packet may be preempted if a higher priority data packet is enqueued. If arbitration is preempted, a new request packet corresponding to the higher priority data packet is sent out. A new request overrides the old (lower priority) cache entry. Figure 19 illustrates the arbitration scheduler algorithm executed in every SFP node. The statement WAIT ("event") specifies that the arbitration scheduler holds (or performs no action) until the appropriate event is detected. The first IF block (lines 2-5) describes the *High* priority arbitration. It can be seen that after a *High* priority arbitration is done, the next arbitration is put on hold until a grant is received and the *High* priority packet transmit is triggered. The second (lines 6-10) and the third (lines 11-15) ELSE IF blocks describe the *Medium* and *Low* priority arbitrations, respectively. It can be seen that *Medium*/*Low* priority arbitrations result in a packet transmit (if a grant is received) or a new arbitration (if a higher priority packet is enqueued).

```
ALGORITHM Arbitration Scheduler
1. While (TRUE) do
2.     If (High priority transmit buffer has packets) then
3.         Send request packet with priority field = High
4.         WAIT (until a grant for the request is received)
5.         Trigger packet transmit
6.     Else if (Medium priority transmit buffer has packets) then
7.         Send request packet with priority field = Medium
8.         WAIT (until a grant for the request is received or a High priority packet is enqueued)
9.         If (grant for the request is received) then
10.            Trigger packet transmit
11.    Else if (Low priority transmit buffer has packets) then
12.        Send request packet with priority field = Low
13.        WAIT (until a grant for the request is received or a High/Medium priority packet is
           enqueued)
14.        If (grant for the request is received) then
15.            Trigger packet transmit
```

Figure 19. Arbitration scheduler algorithm

3.3.2 Fair sharing of bandwidth

Arbitration requesting in SFP alternates between *Current* and *Next* arbitration phases. The arbitration phase ensures fairness among nodes of the same priority class. Every node that has transmitted a packet (of any priority) in the *Current* phase can arbitrate only for the *Next* phase. Arbitration phase is independent of packet priority. Each node implements an *Arbitration_status* flag. If this flag is set to TRUE, *Current* phase requesting is done and if set to FALSE, *Next* phase requesting is done. To start all nodes have *Arbitration_status* flag set to TRUE. As soon as a node transmits a data packet it sets *Arbitration_status* flag to FALSE. This flag is again set to TRUE when the *bus*

owner indicates arbitration reset (i.e. changes the phase of arbitration). The change of phase information is included in the grant packet that the *bus owner* broadcasts. The *bus owner* performs an arbitration reset when it sees no requests for the *Current* phase. When the *bus owner* performs an arbitration reset, the old requests that are already present in the cache automatically get updated to the *Current* phase. Nodes are not required to send a new request packet to update the change of arbitration phase. Among requests of the same priority class, *bus owner* provides higher precedence (in bus access) to *Current* requests than to *Next* requests.

3.4 Bus owner

The *bus owner* is responsible for making the arbitration decision. The *Bus owner* is not a fixed node. SFP nodes take turns in playing the role of *bus owner* and there is always an active (only one) *bus owner*. After taking the arbitration decision, the present *bus owner* explicitly relays control to a node that will be its successor in the network. The transfer of control information (address of the next bus owner) is included in the grant packet that it broadcasts. In the absence of new requests, the *bus owner* retains its control. In case of unexpected network conditions (like loss of a grant packet carrying the transfer of control message) the *root* node assumes the role of *bus owner* after detecting a specific amount of network idle time. Immediately after the network configuration the *root* node is initially assigned the role of the *bus owner*.

3.4.1 Grouping compatible transactions

The bus owner arbitration decision is the core of the arbitration process in SFP. In simple terms, arbitration decision is nothing but "selecting" a group of arbitrating nodes that can be granted simultaneous access to the bus. The bus owner has a global knowledge about arbitrating nodes, and the properties (size, priority, etc.) of packet transactions that arbitration is done for. For better understanding of the bus owner arbitration decision process, it is necessary to study a few design issues.

Figure 20 shows an indexed line that illustrates an SFP topology where each index represents a node. The end of the topology that has node "1" is called the left end, and the other end that has node "*N*" is called the right end. A connection (dashed line) between any two nodes, A and B, indicates that a data packet transaction (or simply "transaction") needs to be established between them (i.e. A wishes to transmit a packet to B, or vice versa). Figure 20 shows several connections, indicating many possible transactions, placed over several rows. Any two transactions that overlap (incompatible) must be placed in different rows (i.e. one above and one below). Non-overlapping (compatible) transactions can be placed in the same row. Compatible data transactions can occur concurrently (i.e. the paths between the corresponding source and destination nodes do not overlap and packet collisions will not occur). In Figure 20, transactions 'a' and 'd' are incompatible and transactions 'b' and 'c' are compatible. Since each request packet defines the source and the destination addresses of nodes involved in a transaction, the *bus owner* is able to envision information as shown in Figure 20.

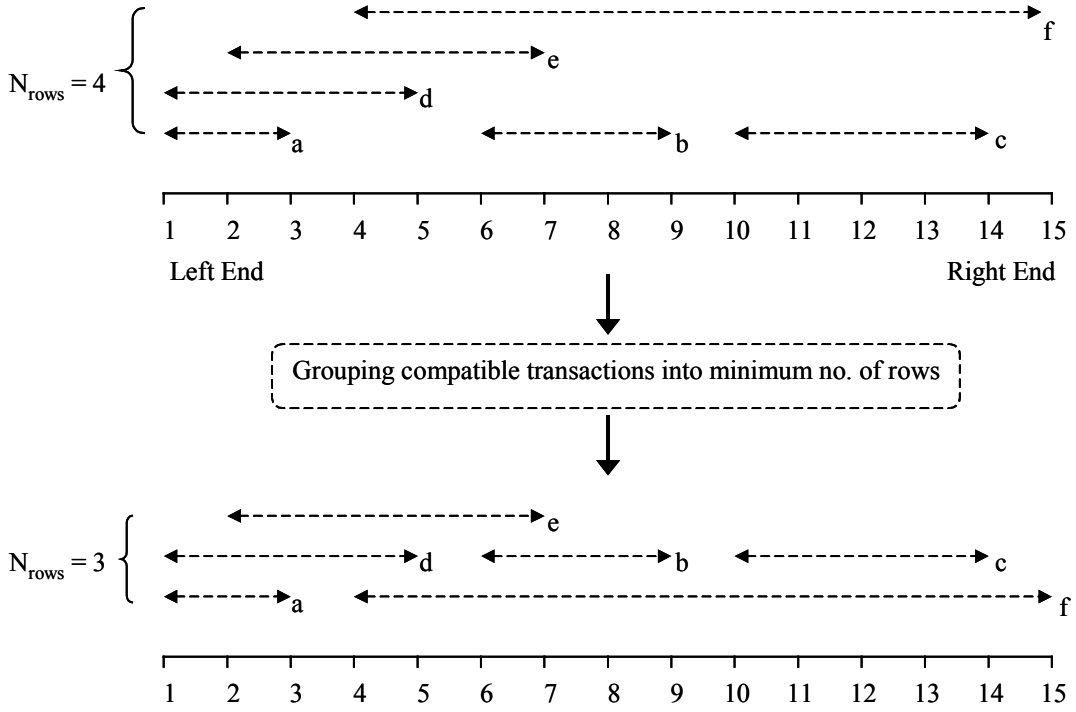

Figure 20. Grouping compatible requests

It is assumed that all data transactions take a fixed duration of transmit time, T_{data} (i.e. all packets are of a same fixed size). Since compatible transactions occur concurrently, the time taken for all transactions in the same row to complete is T_{data}. If N_{rows} denotes the number of rows it takes to accommodate a total of N_{trans} transactions, then the total time duration (total transmit time) taken for completing all data transactions, T_{total} is equal to $N_{rows} \times T_{data}$. Throughput is defined as the number of data transactions completed in unit time and is equivalent to N_{trans} / T_{total} (total number of transactions divided by the total transmit time). It can be observed that a minimum value of T_{total} can be obtained by accommodating all transactions in the minimum possible N_{rows} as shown in Figure 20.

This will maximize the throughput of the SFP network. This observation is true only if all data transactions take the same time duration (i.e. all data packets are of a same fixed size). Finding an optimal minimum value of T_{total} for a group of transactions with different time durations (i.e. different packet sizes) is difficult. Moreover, in an SFP network new requests continue to arrive in a random fashion. At any point in time it is not possible to predict future requests. So this problem is essentially an online-scheduling problem. Applying a greedy strategy at any given time to minimize T_{total} will not necessarily guarantee a total minimum transmit time (and maximize the throughput). However, the bus owner arbitration decision algorithm employs a similar strategy for grouping compatible transactions into minimal number of sets (or rows). The design of the request cache enables this grouping to be done in linear time (in a single memory sweep). After grouping compatible transactions into sets, one of the sets is selected such that packet priority and fairness properties are respected. All source nodes corresponding to the transactions in the selected set are issued a grant. Before presenting the bus owner arbitration decision algorithm, the design of the request cache is studied.

3.4.2 Request cache

Each SFP node implements a request cache. A request cache is structured as a two-dimensional source address pool (i.e. there are N slots each holding an N element array of source addresses). N represents the maximum number of nodes the SFP network may support. Each array element is associated with a one-bit flag. In addition, a request cache has three independent N element arrays called packet size array, packet phase array, and

packet priority array. Each array respectively stores the values of the packet size field, packet phase field and packet priority field of the received requests. Each received request is identified by its source address field. The source address of a request serves as its unique "signature". Figure 21 shows a request cache. The following example illustrates how the cache is updated when a request is received. Assume that a request with the following field values is received:

- Source address – 2,
- Destination address – 8,
- Packet Phase – *Current*,
- Packet length – 1500 bytes, and
- Packet priority – *High*

For every request, its signature is updated in the slots indexed by its source and destination addresses. Since the source address of this request is 2 its signature is 2. In this example, the signature entry should be made in the slots "2' and "8" corresponding to the source and the destination addresses. Source and destination addresses are classified as either *left address* or *right address* based on their closeness to the left end or right end of the topology. In other words the smaller of source or destination addresses is *left address* and the other is *right address*. In this example the source address is *left address* and the destination address is *right address*. Signature "2" is entered in the array (in the next non-empty position) of slot "2" and its associated one-bit flag is set to 0 (since source address is *left address*). Signature "2" is entered in the array of slot "8" (corresponding to the destination address) and its associated one-bit flag is set to 1 (since destination address is right address). The other arrays are updated as:

- Packet phase array [signature] = *Current*
- Packet size array [signature] = 1500
- Packet priority array [signature] = *High*

There is also a request counter that stores the number of requests present in the cache. The request cache can be implemented using content addressable memory (CAM). A CAM-based design for a request cache helps SFP nodes in easy update and removal of cache entries and enables the *bus owner* to make a fast arbitration decision.

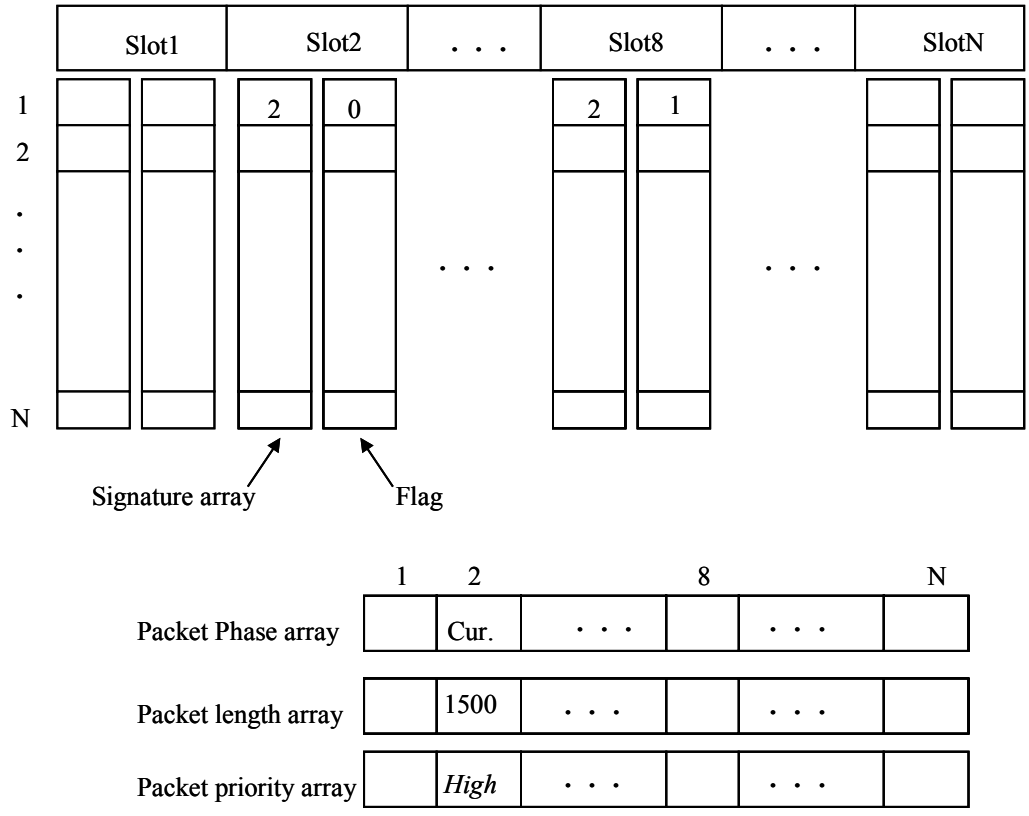

Figure 21. Request cache

3.4.3 Bus owner arbitration decision algorithm

The bus owner arbitration decision algorithm has two tasks. The first task is to group the requests into minimum number of sets as shown in Figure 20. The second task is to select a set such that packet priority and fairness properties are respected. Grouping of requests can be done by two methods. The first method involves sorting of requests based on their *left address*. This method is shown in Figure 22. Initially there is just one empty set (designated as Set_1). Each request of the sorted list is assigned to the lowest possible set if it is compatible with the other request in it. If it is not possible to assign the request to any set, then a new set is created and the request is assigned to it.

ALGORITHM Grouping of Requests – Method 1

1. For (each request of the sorted list) do
2. For (each set Set_i, i from 1 to current number of sets) do
3. If (the request is compatible with all requests in Set_i) then
4. Assign the request to Set_i
5. Else
6. Create a new set Set_{i+1} and assign the request to it

Figure 22. Grouping of requests – method 1

If there are *n* requests, then sorting takes *O(nlogn)* time and grouping of requests takes $O(n^2)$ time. So the time complexity of this method is $O(n^2)$. Grouping of requests takes $O(n^2)$ time because every request is compared to every set and in the worst case (all transactions to the head end) the number of sets can be proportional to n^2.

The second method is based on the observation that two requests can be placed in the same set if the *left address* of one request is greater than or equal to the *right address* of the other. The second method requires that requests be sorted based on their *left* and *right* addresses and scanned in that order. A request cache can ensure that requests are kept in a sorted order (based on their *left* and *right* addresses). So the time for sorting is saved. A new stack data structure that stores the signature of requests is created. The requests are scanned in order (i.e the *left address* of a request is encountered before the *right address*). Whenever the *right address* of a request is encountered its signature is pushed onto the stack. Whenever the *left address* of a request is encountered the stack is checked. If the stack is non-empty then the request is assigned to the same set as the request in the stack top. If the stack is empty then the request is placed in a new set. The use of stack data structure eliminates the need for scanning every already created set. The use of a stack to group requests in this way is adopted from the work [28]. Reference [28] presents a linear time left edge algorithm for channel routing in VLSI circuits. This second method does not involve sorting of requests and requests are grouped in a single sweep of the request cache. The time complexity of the second method is *O(n)*. The SFP bus owner arbitration decision algorithm follows the second method for grouping requests.

The bus owner arbitration decision algorithm is presented in Figure 23. Lines 1-12 of the algorithm deal with partitioning the requests in the cache into a minimal number of sets of compatible requests. This is similar to the grouping of compatible transactions in minimum possible rows as seen in Figure 20. The request cache is scanned from slot 1 to slot *N* so that requests are scanned in a sorted order (based on their left and right

addresses). For each slot, whenever a request signature with the associated flag set to 1 (indicating *right address*) is encountered it is pushed onto a stack. For each slot, when a request signature with the associated flag set to 0 (indicating *left address*) is encountered, it is placed in the same set as the request found in the top of the stack or it is placed in a new set if the stack is empty. Grouping requests in this way ensures that they are compacted into a minimal number of sets. In the algorithm, R_i denotes the signature of any request *i*, where *i* ranges from 1 to the number of requests in a slot. *Index* represents the identification of a set, and is initialized to zero. S_{index} denotes a set with identification *index*, and *j* is a loop counter. In lines 13 – 17 a set of requests is selected for grant to bus access. A set that has the maximum number of requests is selected such that it has one or more of the highest priority level requests present at that time. The three priority classes in SFP and the two phases of arbitration combine to provide six levels of priority as illustrated in Table 2. After selecting a set, the *bus owner* issues a grant to all the requests (i.e. the corresponding source nodes) in the selected set. The granted source node that is expected to complete its packet transmission in the end will be assigned the role of the next *bus owner*. The *bus owner* broadcasts a grant packet with information about the granted nodes and the next *bus owner*. Arbitration granting is explained in the next section.

ALGORITHM Bus Owner Arbitration Decision

```
1. For (each slot of the request cache) do
2.     For (i=1 to number of requests for this slot) do
3.         If (Ri has associated flag set to 1) then
4.             Push Ri on the stack
5.     For (i=1 to number of requests for this slot) do
6.         If (Ri has associated flag set to 0) then
7.             If (stack is empty) then
8.                 index = index + 1
9.                 Assign Ri to the set Sindex
10.            Else
11.                Pop R from stack
12.                Assign Ri to the same set as R
13. For (j = 1 to number of priority levels) do
14.     If (There are any priority j requests) then
15.         Select a set containing the maximum requests and at least one priority j request
16.         Issue a grant to all requests in the selected set
17.         Exit from this algorithm
```

Figure 23. Bus owner arbitration decision algorithm

Table 2. Priority levels in SFP

Priority class	Arbitration phase	Priority level
High	*Current*	1 (highest)
	Next	2
Medium	*Current*	3
	Next	4
Low	*Current*	5
	Next	6 (lowest)

3.5 Arbitration granting

After making the arbitration decision, the bus owner broadcasts a grant packet. The grant packet is transmitted in the data line. Since the bus owner is the last node to complete its data transmission (among a group of transmitting nodes) it is ensured that the grant packet will not collide with other data packets. Every node makes a local copy of the grant packet and repeats it to the neighbor. The grant packet includes several fields of information whose significance is described below:

- Granted address list: This list contains the address of all source nodes (corresponding to the requests in the selected set) granted by the bus owner. The listed nodes can transmit their data packet immediately on receiving the grant. Based on the granted address list, nodes clear cache entries corresponding to the granted requests.
- Destination address list: This list contains address of all destination nodes whose corresponding source nodes are granted bus access. This knowledge comes from the request packets. Nodes operate in *blocking mode* when their address is included in this list (it is implied that the next data packet is destined to the listed node).
- Reset status: This field indicates the status of arbitration reset. When the *bus owner* sees no *Current* phase requests it performs an arbitration reset by setting this field value to TRUE. If *bus owner* sees one or more *Current* phase requests then this field is set to FALSE. When this field is set to TRUE, nodes update their

Arbitration_reset flag to TRUE and hence can start arbitrating for the *Current* phase again.

- Bus owner: This field identifies the address of next *bus owner*. One of the granted nodes that identifies its address in this field, must take control of the bus owner operation at the end of its data transmission. For each granted source node the present bus owner computes a drain time, T_{drain},

$$T_{drain} = \frac{L_{pkt}}{R} + N_{hops} T_{repeat} + D_n T_{prop} . \quad (2)$$

In (2), L_{pkt} denotes the size (in bits) of the data packet arbitration is done for. N_{hops} is the number of intermediate nodes between the present *bus owner* and the granted node, D_n is the rough distance estimate between the present bus owner and the granted node, and T_{repeat} is the repeat path delay per node. The granted node, which has the maximum value of drain time, is the next bus owner and its address is included in the bus owner field of the grant packet. Figure 24 illustrates a typical arbitration sequence in SFP.

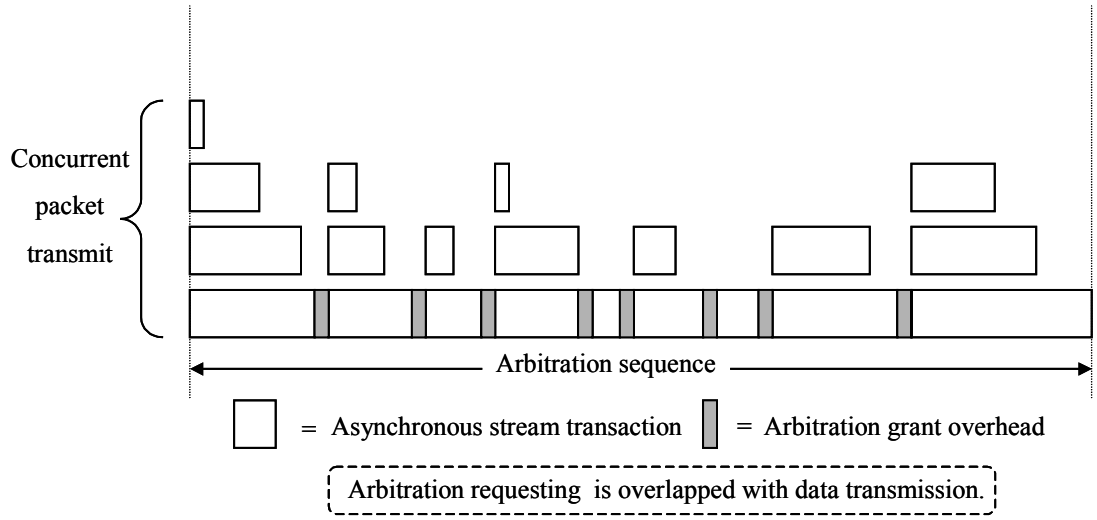

Figure 24. Arbitration sequence in SFP

3.6 Traffic classes in SFP

All data transmissions in SFP are packet based and SFP supports variable sized packets. SFP supports two types of data transactions described as follows:

- Asynchronous transactions: These are unicast transactions that provide reliable data delivery. Each asynchronous packet requires an acknowledgement from the receiver. In FireWire, an acknowledgement packet does not require arbitration and can be transmitted by any node immediately on the receipt of an asynchronous packet. However, acknowledgement packets may or may not require arbitration in SFP. The performance study of tradeoffs between the two methods is left for future study. Asynchronous packets may be assigned to different priority classes and fairness among nodes may be ensured by the SFP fairness mechanism.

- Asynchronous streaming: This work focuses on asynchronous streaming, which is suitable for carrying video and other real-time traffic. These transactions may be unicast or multicast. The scope of this work is limited to unicast asynchronous streaming. Asynchronous streaming is an unreliable service and packets do not require an acknowledgement from the receiver. SFP does not provide support for isochronous service as seen in FireWire. QoS support for time critical applications, such as voice and video, is provided by mapping individual asynchronous stream packets to the different priority classes supported by SFP.

3.7 Summary

The following summarizes the SFP design principles:

- SFP proposes a new physical layer data transmission interface that uses the existing FireWire cable. A communication link uses two twisted pairs (TPA, or data line and TPB, or request line) that operate as two independent half duplex lines. Synchronous request transfer mechanism permits unblocked, overlapped arbitration with data transmission.
- Request packets are informative; containing source address, destination address, packet phase, packet size, and packet priority fields. Caching of request packets enables the *bus owner* too see multiple requests at the same time and make an intelligent arbitration decision. The request cache has an efficient design permitting easy look-up of requests and quick response time (processing time) for the *bus owner*.

- SFP preserves the simple repeat path functionality of FireWire and still achieves destination stripping of packets. A data packet does not require destination address look-up (involving a delay overhead) at each node. A grant packet explicitly informs destination nodes to operate in *blocking mode*.
- SFP supports three priority classes and arbitration ensures fair sharing of bandwidth among like priority nodes. Isochronous service is not supported. Data transactions are asynchronous or asynchronous streaming.

CHAPTER 4

PERFORMANCE EVALUATION OF SFP

Using simulation, the queuing delay and the throughput performance of SFP, IEEE 1394b and IEEE 1394a are evaluated. Discrete-event queuing simulation models of the three protocols were built using the CSIM18 function library. All models include T_{prop} (5 nanoseconds per meter) and T_{repeat} (144 nanoseconds) delays. A response delay (i.e. time to make an arbitration decision and broadcast a grant) of 244 nanoseconds for the bus owner is included. These delay values are based on the IEEE 1394b FireWire standard. Packetized video transmission is done using asynchronous stream packets.

4.1 Traffic models for simulation experiments

Two traffic models are used to evaluate performance. The first traffic model is based on MPEG-2 frame length traces from the 1996 Olympic games [18]. Each trace was for 40 minutes of a sporting event and a total of 20 traces were available. The MPEG-2 frame traces were converted into packet sizes with 48 bytes of overhead (representing LAN, IP, UDP, and RTP headers) per packet. Fragmentation of MPEG-2 frames into Ethernet packets was assumed to occur in zero time. The MPEG-2 video rate is 25 frames per second with a mean data rate of about 5 Mbps. For the simulation evaluation, frame

traces from 20 different Olympic events were used. When the number of simulated nodes is greater than 20, copies of the available frame traces are randomly assigned between nodes. For the 20 MPEG-2 sources, the mean packet length was 1459.7 bytes and the total offered packet load was 101.5 Mbps. The frames from multiple sources were not synchronized. The second traffic model was Poisson arrivals of fixed length packets. The packet length used was the mean packet length of the MPEG-2 video sources unless otherwise specified. This traffic model was synthetically generated with no limit on the number of nodes. MPEG-4 frame length traces are used for a single experiment. Each MPEG-4 trace was for 60 minutes of a movie sequence and a total of 20 traces were available [10]. The mean data rate of all MPEG-4 sources is 0.67 Mbps and video rate is 25 frames per second. The same packetization method as applied to the MPEG-2 traces is used for the MPEG-4 traces.

4.2 The simulated configuration

The three simulation models (SFP, IEEE 1394b, and IEEE 1394a) were designed to model a daisy-chained network configuration as shown in Figure 16. Each node is an independent traffic source. Each IEEE 1394b and IEEE 1394a node is assumed to have an infinite capacity buffer for packets being sent on the link. Each SFP node is assumed to have three infinite capacity buffers (corresponding to the three priority classes). The distance between any of pair nodes is equal and fixed at 10 meters. All internode links have equal bandwidth capacity, which is varied between the experiments. Source-destination traffic distributions between the nodes are based on four models as described

below. Each model is characterized by a distinct value of spatial reuse factor; *S*. The value of *S* represents the average number of concurrent packet transmissions that can occur in the network. The performance of SFP is expected to vary for the different traffic distribution models. However, IEEE 1394b and IEEE 1394a will offer similar performance for all the four models because no spatial reuse is permitted in them.

- *Spatial_min*: All packets (of all nodes) are destined to the head end, which acts as the sensor fusion node. Since no concurrent packet transmissions are possible *S* for this model equals 1 (minimum possible).
- *Spatial_average*: For every packet, a source node uniformly selects a destination node, which can be any other node in the network. *S* for this model is equivalent to the total number of nodes divided by the average distance between two nodes (in node count), and can be given as $\frac{N}{N/2}$. So the value of *S* is 2.
- *Spatial_video*: For 90% of the time nodes send packets to their right or left neighbors (45% of time to right neighbor and 45% of time to left neighbor). For 10% of the time packets are destined to the head end. It is expected that traffic distribution in a typical video surveillance system will be similar. In a video surveillance system most of the traffic will occur between peer cameras (to track a profiled individual or notify significant events). A communication with the head end is established only for control messages and/or for recording data. *S* for this model is given as

$$S = \sum_{i=1}^{i=N} i \times P_{head} \times P_{adj}^{i-1} \tag{3}$$

In (3), P_{head} is the probability that packets are destined to the head end (0.1 here), and P_{adj} is the probability that packets are destined to adjacent nodes (0.9 here).

- *Spatial_max*: All nodes send packets to their right neighbors. The *N*th node in the network is assumed to have a dummy right neighbor. *S* for this model is *N* (maximum possible).

4.3 Description of simulation experiments

Seven experiments are defined to evaluate the performance of existing FireWire protocols and SFP. The first two preliminary experiments evaluate the performance of IEEE 1394a and IEEE 1394b. The other five experiments evaluate the performance of SFP. To achieve a target offered packet load (the control variable for experiments) for MPEG-2 sources, the link rate (*R*) is varied as the total bandwidth of sources divided by the target offered load. For Poisson sources packet arrival rate (λ) is varied to achieve a target load. When the control variable is node count, packet size, or priority ratios the load is not maintained at any fixed value. Unless otherwise specified, all packet transactions are asynchronous stream based, *Low* in priority and follow the *Spatial_min* traffic distribution. SFP request packets are assumed to be 10 bytes and SFP grant packets 100 bytes in length.

Preliminary experiment #1: This experiment evaluates the performance of IEEE 1394b and IEEE 1394a. The response variable is queuing delay (mean) and the control variable is the offered packet load on the link, which is increased from 10% to 97%. The number

of nodes is fixed at 20. This experiment is performed for Poisson and MPEG-2 sources. For Poisson sources, the link bandwidth is fixed at 100 Mbps.

Preliminary experiment #2: This experiment evaluates the performance of IEEE 1394b for isochronous and asynchronous packet streams. The response variable is queuing delay (mean) and control variable is number of nodes, which is increased from 2 to 19. For isochronous streams, the entire 125-microsecond arbitration cycle is allocated for packet transactions and bandwidth is shared equally among the isochronous nodes. There is no cycle start overhead. Link bandwidth is fixed at 100 Mbps. This experiment is performed for MPEG-2 and MPEG-4 sources.

Load experiment: This experiment evaluates the performance of SFP and IEEE 1394b for different traffic distribution models. The response variable is queuing delay (mean and 99%) and the control variable is the offered packet load (throughput) on the link, which is increased from 10% to as high as 4500%. The number of nodes is fixed at 60 and the link bandwidth at 400 Mbps. This experiment is performed for Poisson sources.

Node count experiment: This experiment evaluates the performance of SFP for different traffic distribution models. The response variable is queuing delay (mean and 99%) and the control variable is the number of nodes, which is increased from 4 to 1000. The link bandwidth is fixed at 100 Mbps. This experiment is performed for Poisson sources. The packet arrival rate (λ) is adjusted so that each node is a 5 Mbps traffic source.

Packet size experiment: This experiment evaluates the throughput performance of SFP for different packet sizes. The response variable is the maximum offered throughput on the link (in factors) and the control variable is the fixed packet size of Poisson sources, which is increased from 100 to 20,000 bytes. The number of nodes is fixed at 100 and the link bandwidth at 400 Mbps. The traffic distribution model used is *Spatial_video*. This experiment is performed for Poisson sources.

Priority experiment: This experiment evaluates the performance of SFP for different priority traffic. The response variable is queuing delay (mean and 99%) and the control variable is the offered load on the link, which is increased from 10% to 165%. The number of nodes is fixed at 60. Packets are prioritized such that 20% of the packets are *High* priority, 30% are *Medium* priority, and 50% are *Low* priority. This experiment is performed for MPEG-2 and Poisson sources. The traffic distribution model used is *Spatial_average*.

Packet priority ratio (PPR) experiment: This experiment evaluates the performance of SFP for different PPR. The response variable is the maximum offered throughput (in %) on the link and control variable is PPR. PPR is the ratio of *Low* to *Medium* to *High* priority traffic. The number of nodes is fixed at 60. This experiment is performed for Poisson sources. The traffic distribution model used is *Spatial_average*.

4.4 Results from the simulation experiments

Figures 25 and 26 show the preliminary experiment #1 results for Poisson and MPEG-2 traffic sources, respectively. In IEEE 1394b and IEEE 1394a mean queuing delays increase with the load. For Poisson sources, IEEE 1394b delay is always a magnitude less than the IEEE 1394a delay. IEEE 1394a reaches the maximum tolerable delay (delay exceeds the tolerance of human response time of 100 milliseconds) at about 92% load. At 97% load IEEE 1394b delay is around 3 milliseconds. IEEE 1394b reaches a bottleneck at about 99% load (not shown in graph). For MPEG-2 sources, the queuing delay trend is similar. However, the IEEE 1394b delay is only slightly lesser than the IEEE 1394a delay until about 90% load. IEEE 1394a delay exceeds 100 milliseconds at 92% load. At 97% load IEEE 1394b delay is 15 milliseconds.

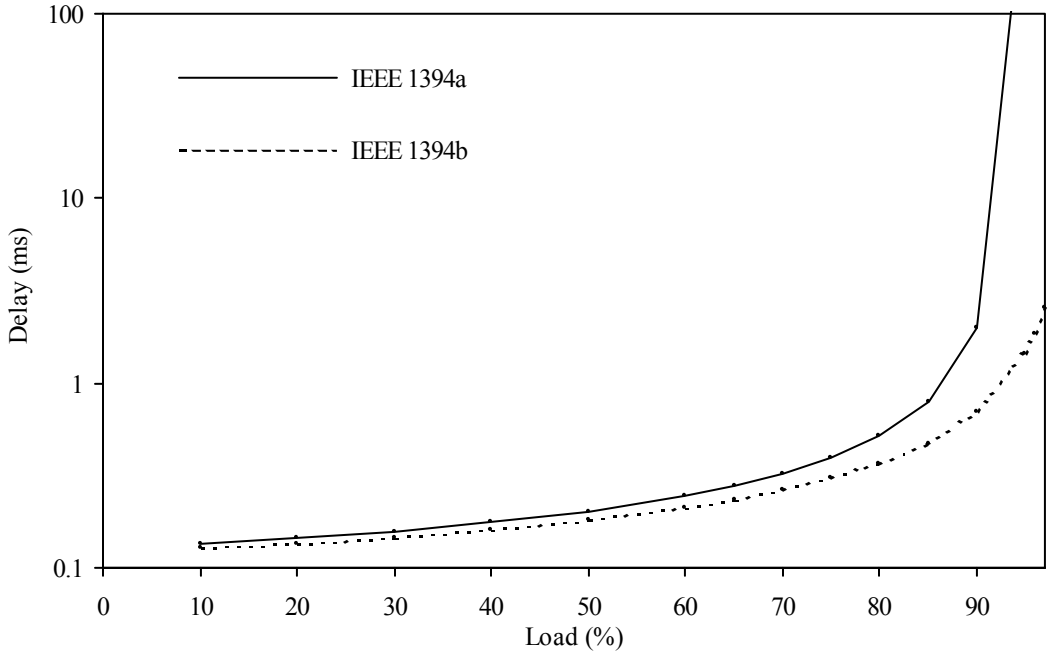

Figure 25. Preliminary experiment #1 results for Poisson source

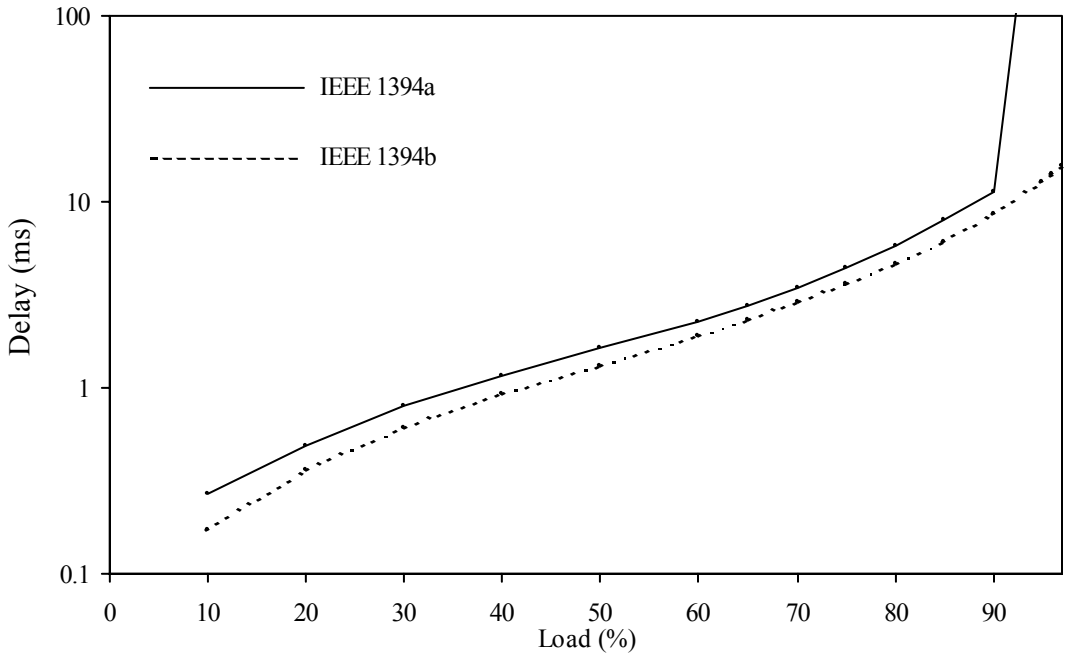

Figure 26. Preliminary experiment #1 results for MPEG-2 source

Figures 27 and 28 show the preliminary experiment#2 results for MPEG-2 and MPEG-4 traffic sources, respectively. For both traffic sources, the asynchronous delay is always a magnitude less than the isochronous delay and increases at an exponential rate. For both traffic sources, isochronous delay increases at a constant rate. For MPEG-2 traffic, asynchronous delay is 8 milliseconds at 18 nodes, which is one-eighth the corresponding isochronous delay. For MPEG-2 traffic, isochronous delay exceeds the maximum tolerable delay at 19 nodes and asynchronous delay at 20 nodes. Maximum tolerable delay is reached since the total offered bandwidth of the 20 MPEG-2 sources (100 Mbps) is equal to the link capacity (saturated network condition). For MPEG-4 traffic, both isochronous and asynchronous delays stay within 5 milliseconds, even for 20 nodes. This is because the bandwidth of MPEG-4 sources is very small (roughly 8 times less) compared to MPEG-2 sources, and the offered load at 20 nodes is just 13%.

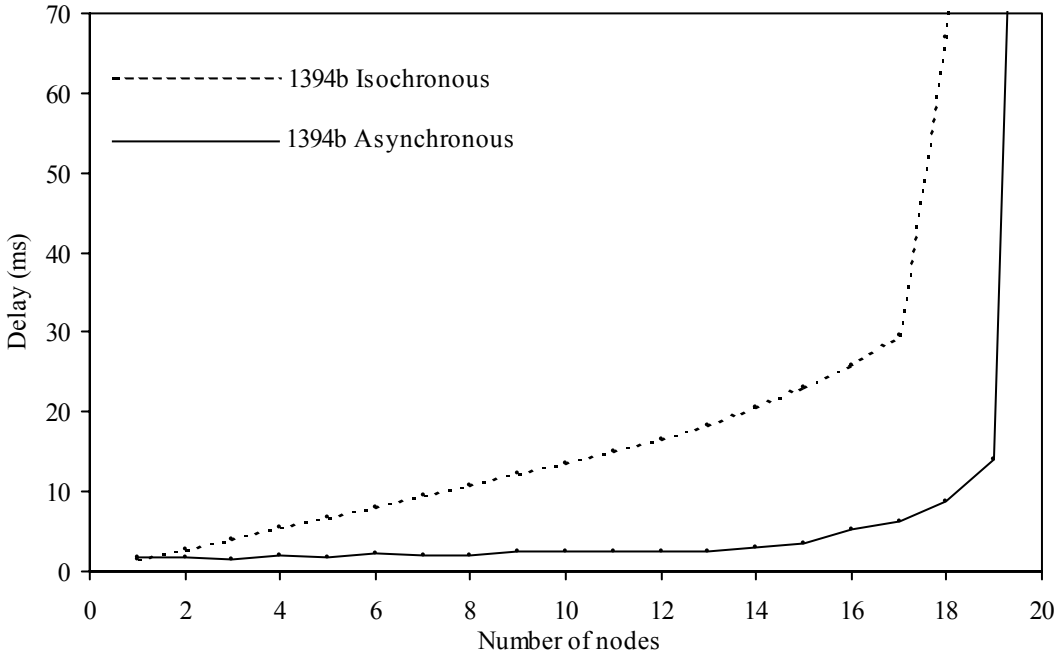

Figure 27. Preliminary experiment #2 results for MPEG-2 source

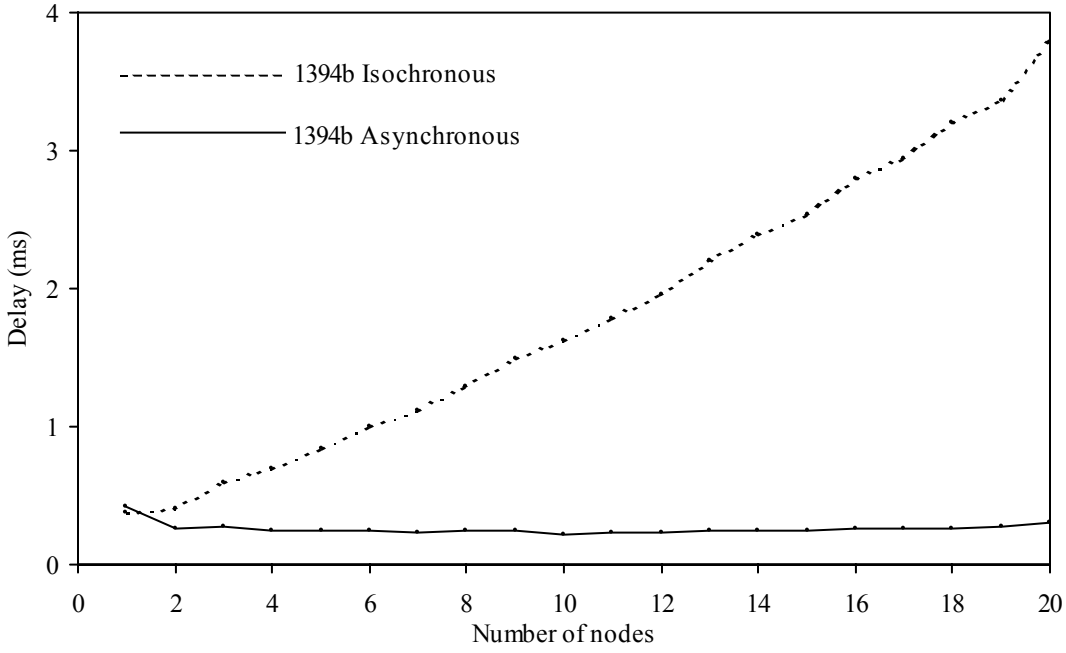

Figure 28. Preliminary experiment #2 results for MPEG-4 source

Figures 29 and 30 show the load experiment results. Queuing delay increases exponentially with the offered network load. The queuing delay trends of SFP with *spatial_min* traffic model and IEEE 1394b are very similar and the lines are not distinguishable for mean and 99% results. Both reach a throughput maximum at 98% load. Maximum throughput is implied when queuing delay increases at a large rate and is much higher than the bottleneck value (100 milliseconds). IEEE 1394b offers identical performance for all four traffic models because it does not support spatial reuse. SFP offers no spatial reuse for *Spatial_min* traffic model, and hence, its performance is similar to IEEE 1394b. SFP reaches a maximum throughput at 165%, 650% and 4250% loads for *Spatial_average*, *Spatial_video* and *Spatial_max* traffic models, respectively. The maximum possible throughput is,

$$\rho_{max} = \frac{\left\lceil \frac{L}{R} \right\rceil S}{\frac{L}{R} + D_{boss}(T_{repeat} + T_{prop})} \tag{4}$$

where L is the average packet length in bits (11677 bits here) and D_{boss} is the average hop count between consecutive bus owner nodes. The value of D_{boss} is 1, $N/2$, N, and N for *Spatial_min*, *Spatial_average*, *Spatial_video* and *Spatial_max* traffic patterns, respectively. The only delay bottleneck in SFP arbitration is the arbitration granting overhead and is equivalent to $D_{boss}(T_{repeat} + T_{prop})$. Substituting all parameters, the values of ρ_{max} obtained from the equation are 168%, 700% and 4289% for *Spatial_average*, *Spatial_video* and *Spatial_max* models, respectively. For *Spatial_average* and *Spatial_max* models, the variation between experimental and theoretical results for maximum throughput is less than 1.8%. For *Spatial_video* model the variation is nearly

7%. Variation is higher because, the *Spatial_video* model is unbalanced (i.e. the traffic load is not uniformly distributed across the network). This model requires that every node send packets to the head end 10% of the time. Nodes far from the head end will experience a higher delay because their traffic is interfered by many intermediate nodes. The delay experienced by each node grows proportionally with the number of intermediate nodes between the node and the head end. It can be seen that 99% delay for *Spatial_video* model exceeds the maximum tolerable delay at 580% load while mean delay exceeds the maximum tolerable delay at 655% load. This is due to the unbalanced nature of traffic distribution. It is very difficult to present a precise delay analysis for this traffic distribution. *Spatial-min*, *Spatial_average* and *Spatial_max* models are balanced and 99% and mean delays exceed the maximum tolerable delay at approximately the same load.

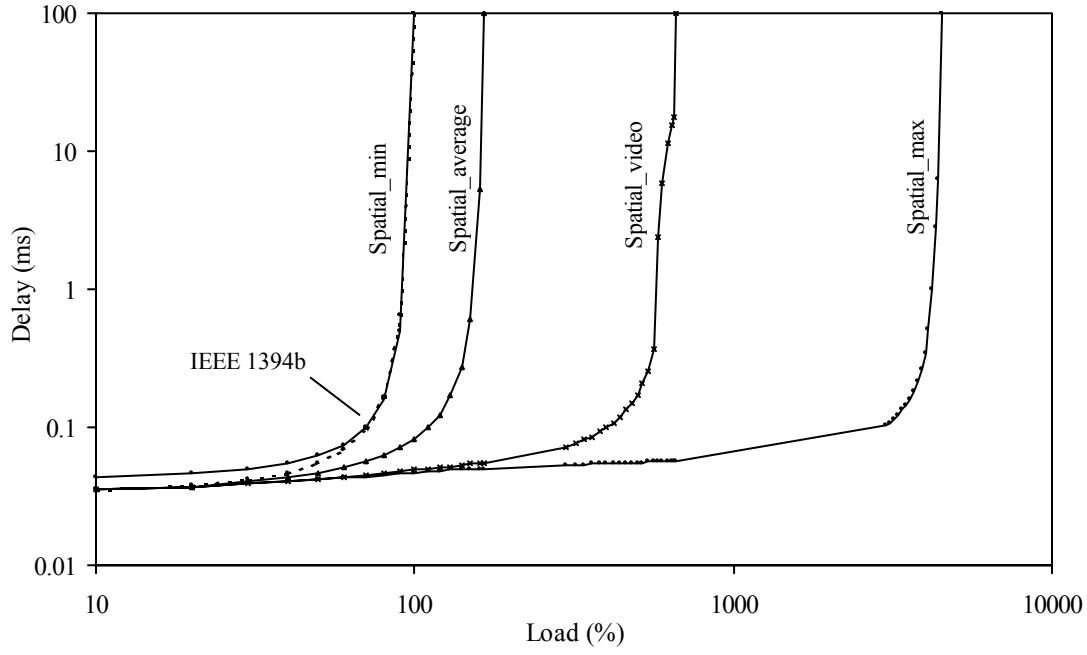

Figure 29. Load experiment results (mean delay)

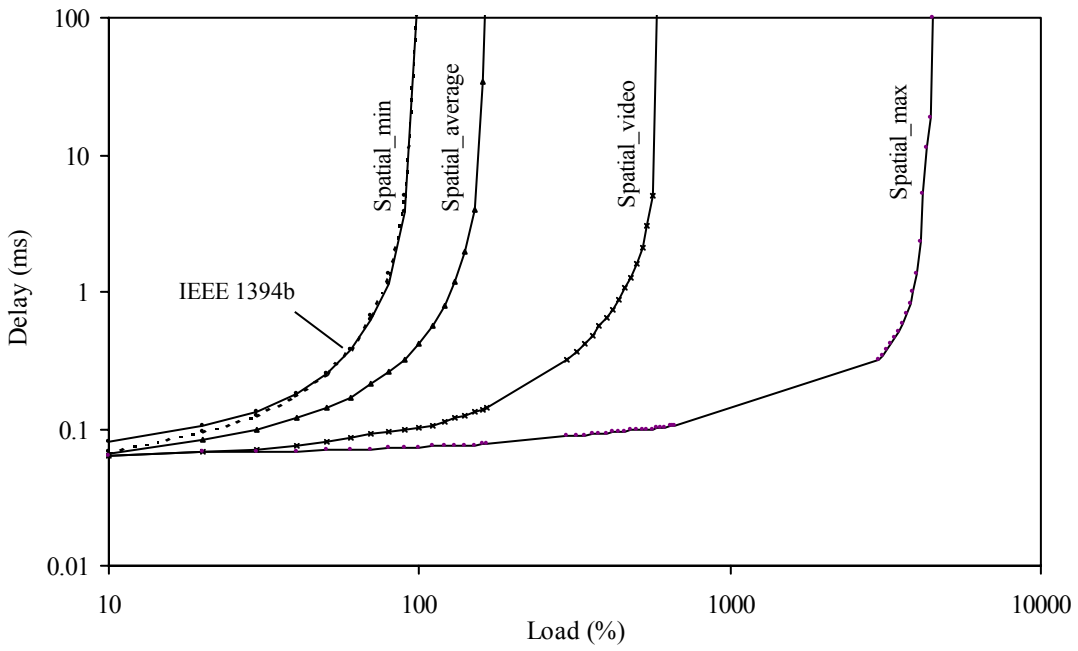

Figure 30. Load experiment results (99% delay)

Figures 31 and 32 show the node count experiment results. Queuing delay increases exponentially with the node count. It is seen that mean delay of *Spatial_min* traffic is less than 3 milliseconds, and 99% delay is less than 10 milliseconds up to 19 nodes, and at 20 nodes both exceed 100 milliseconds (reach bottleneck). For the *Spatial_average* model the bottleneck is reached at 35 nodes (for mean and 99% results). For the *Spatial_video* model the mean delay is less than 13 milliseconds for 160 nodes and exceeds 100 milliseconds at 165 nodes. For *Spatial_video*, 99% delay exceeds the maximum tolerable delay at 145 nodes itself. This is due to the unbalanced nature of *Spatial_video* model. For *Spatial_max* model mean and 99% delay are less than 3 milliseconds even for 1000 nodes. It is seen that SFP is able to support 19, 34, 145 and more than 1000 nodes for

Spatial_min, *Spatial_average*, *Spatial_video* and *Spatial_max*, traffic models respectively. The maximum node capacity is,

$$N_{max} R_{node} \leq \frac{L \times S}{\frac{L}{R} + N_{max}(T_{repeat} + T_{prop})}. \quad (5)$$

where N_{max} is the maximum number of nodes and R_{node} is the average data rate of a node in bits per second (5Mbps here). Substituting all parameters, the values of N_{max} obtained are, 19, 37, 158, and more than 1000 nodes for *Spatial_min*, *Spatial_average*, *Spatial_video* and *Spatial_max*, traffic models respectively. The theoretical and experimental results match closely for *Spatial_min* and *Spatial_average* (variation less than 5%). For *Spatial_video*, variation between theoretical and experimental maximum nodecount is 10%. This is again due to the unbalanced nature of traffic distribution in *Spatial_video*.

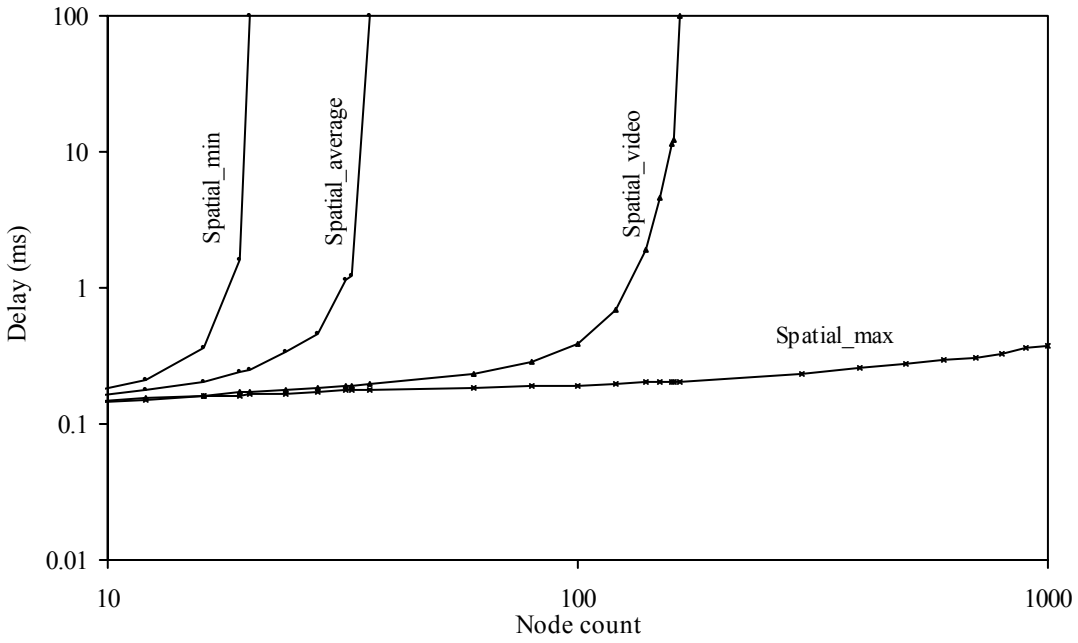

Figure 31. Node count experiment results (mean delay)

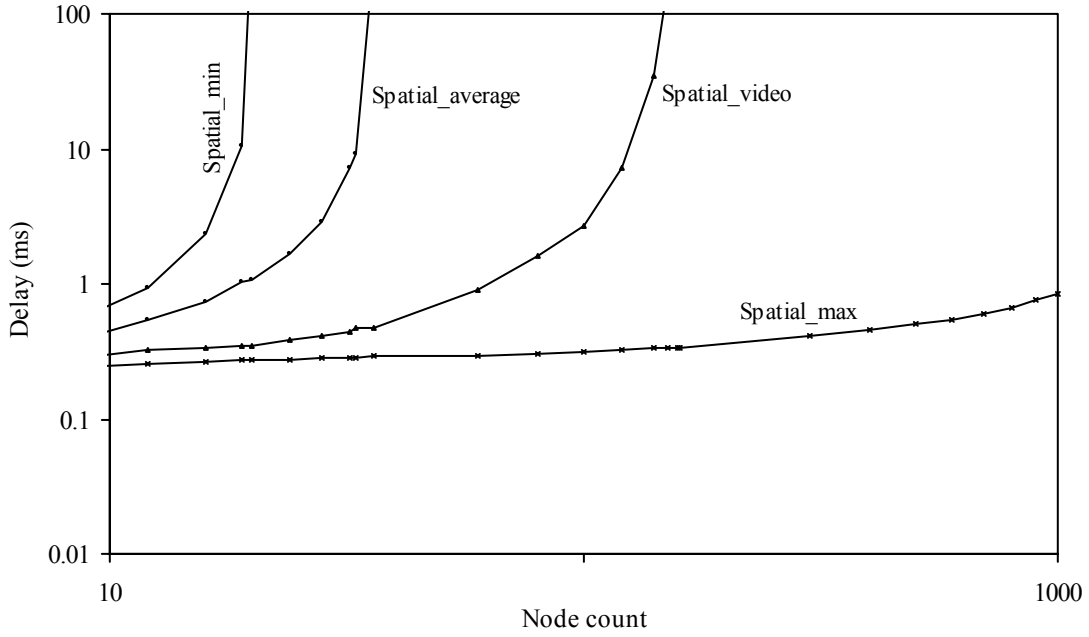

Figure 32. Node count experiment results (99% delay)

Figure 33 shows the results for packet size experiment. It is seen that throughput increases with the packet size. However the rate of increase dampens with the increase in packet size, and throughput gradually reaches a constant value for large packet sizes. For 100 byte packets, throughput is 0.94 (or 94%). This is similar to the throughput of IEEE 1394b. Throughput is 6 (or 600%) and 960% for 1500 and 20,000 byte packets, respectively. After 20,000 bytes the increase in throughput is negligible.

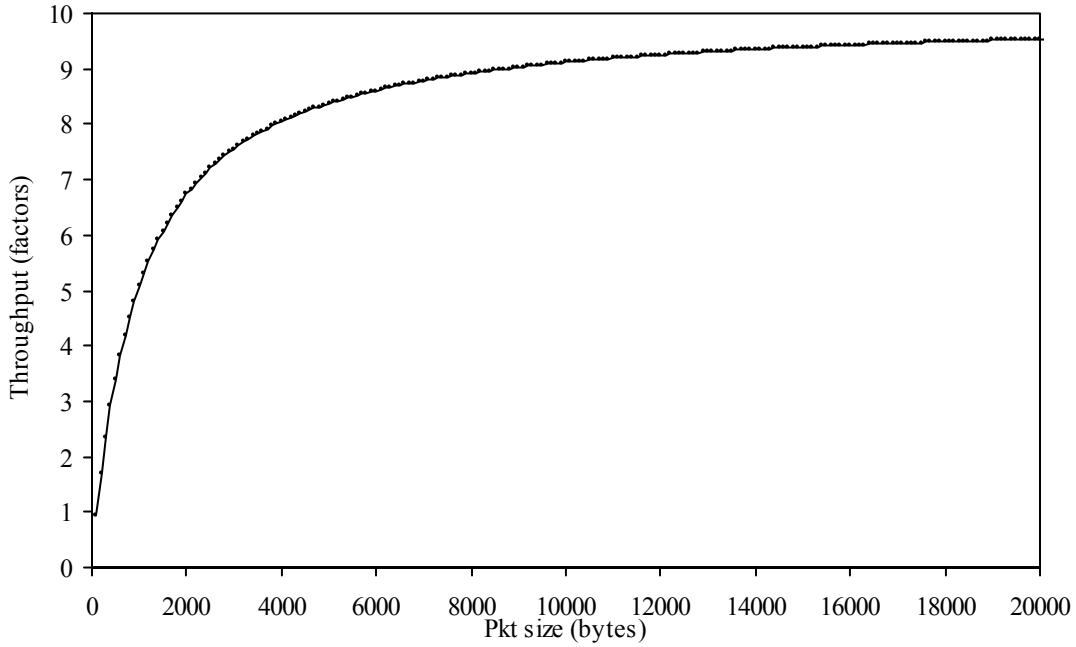

Figure 33. Packet size experiment results

Figures 34, 35, 36 and 37 show the priority experiment results. At low loads the queuing delays of *High*, *Medium*, and *Low* priority traffic are alike. Queuing delay increases with load, however the rate of increase of *Low* priority delay (mean and 99%) is many magnitudes higher than that of *High* priority delay. *Medium* priority delay falls between the *High* and *Low* priority delays. For Poisson sources, at 160% load the mean delays are 0.08, 0.3, and 240 milliseconds and 99% delays are 0.18, 1.3 and 2000 milliseconds for *High*, *Medium* and *Low* priority traffic, respectively. For MPEG-2 sources at the same load, the mean delays are 4, 9 and 190 milliseconds and 99% delays are 8, 30, and 2000 milliseconds for *High*, *Medium* and *Low* priority traffic, respectively.

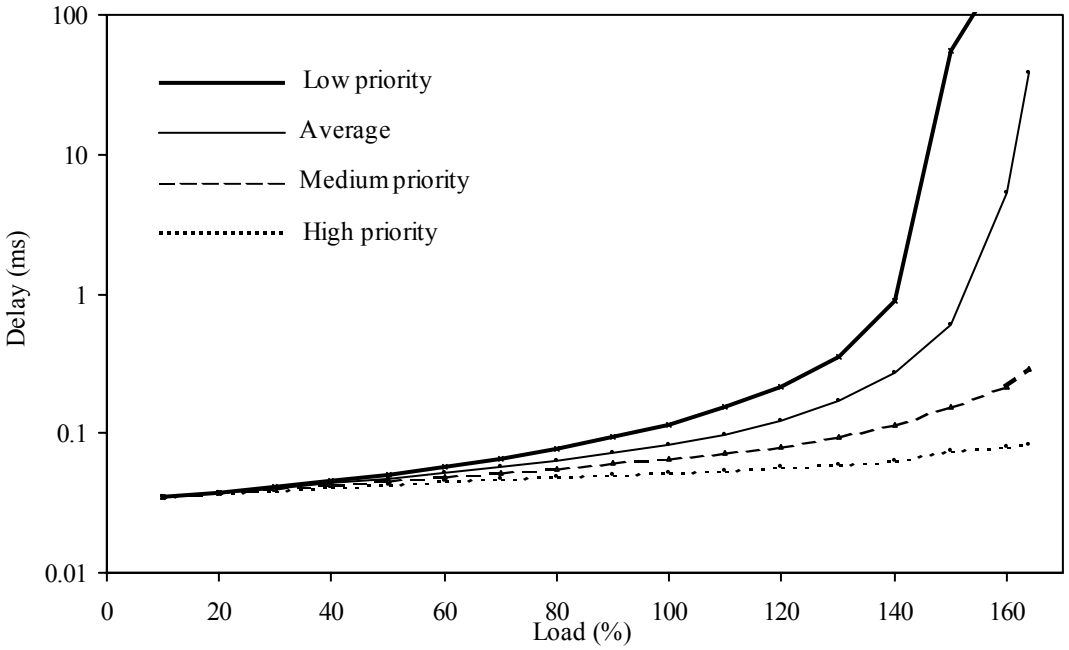

Figure 34. Priority experiment results for Poisson source (mean delay)

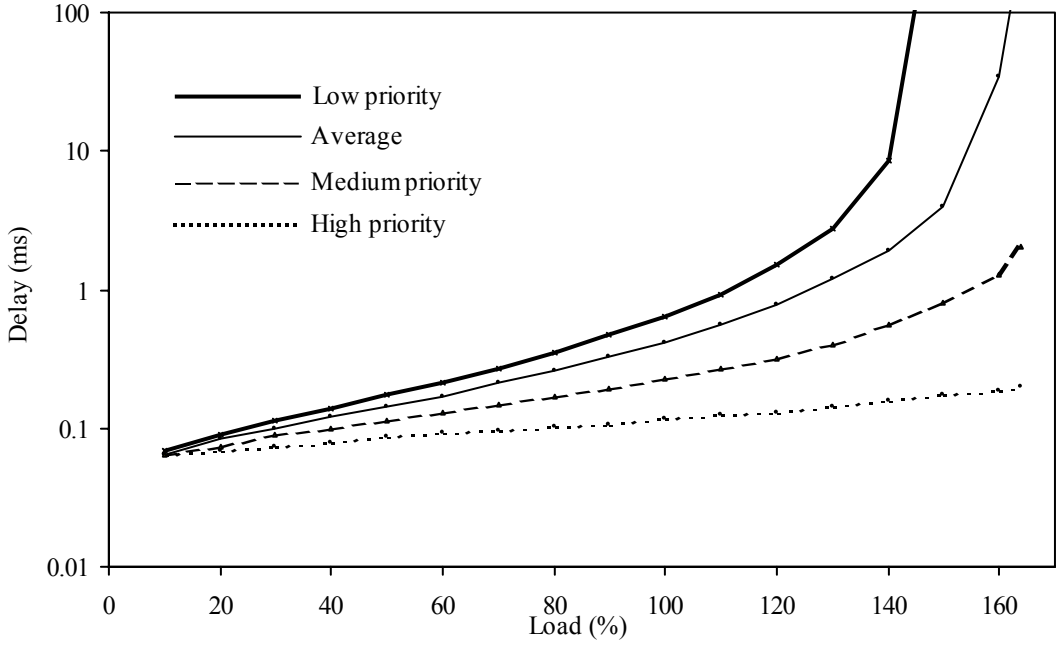

Figure 35. Priority experiment results for Poisson source (99% delay)

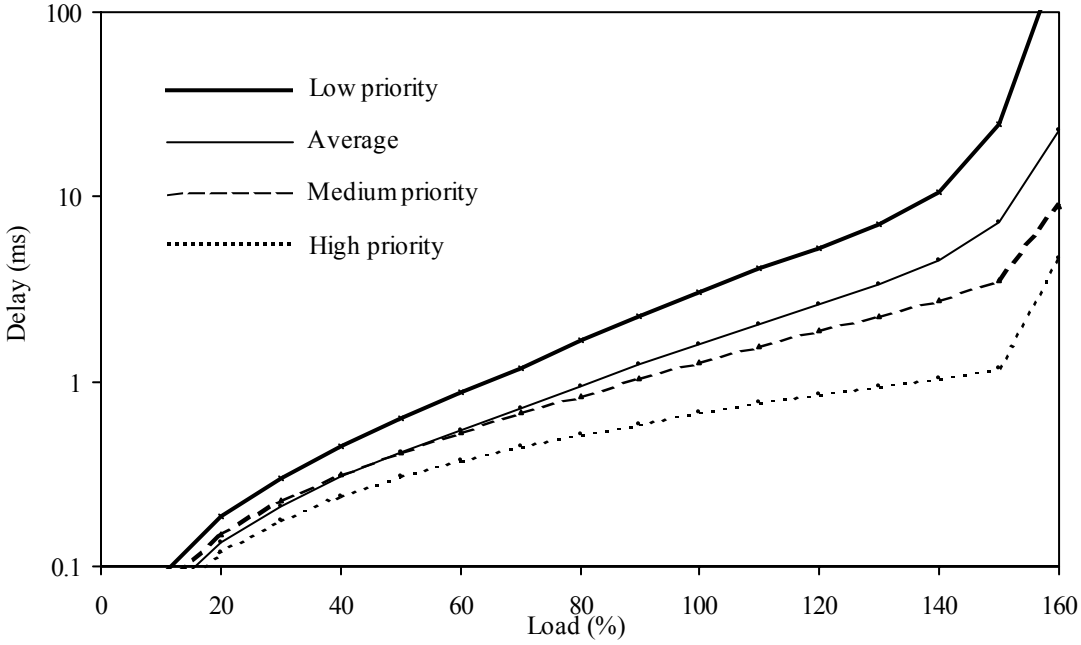

Figure 36. Priority experiment results for MPEG-2 source (mean delay)

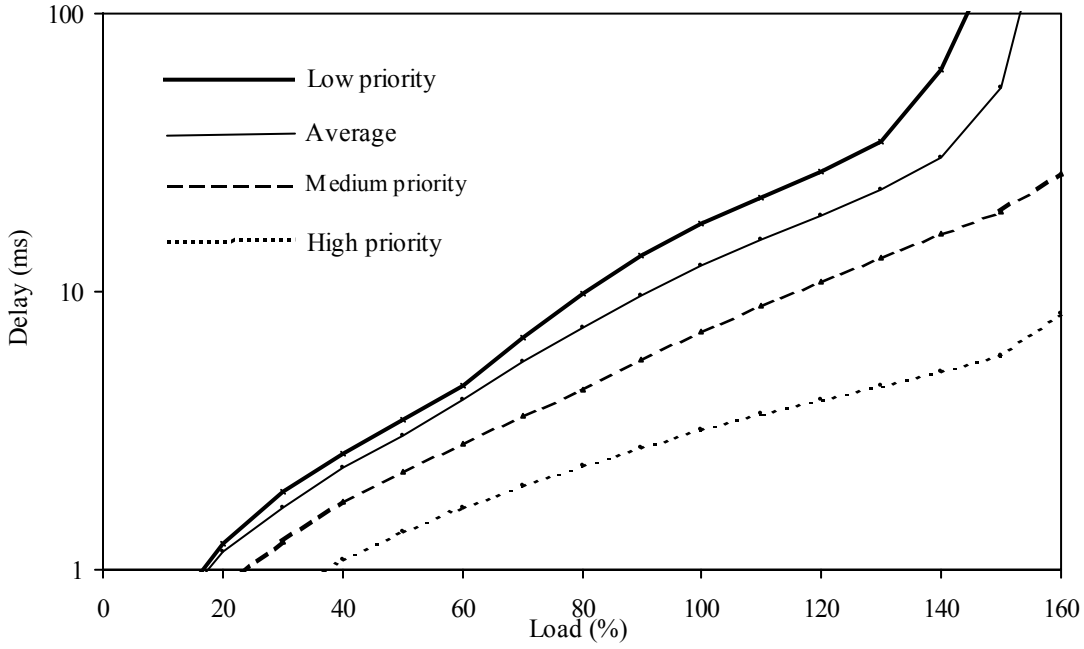

Figure 37. Priority experiment results for MPEG-2 source (99% delay)

Figure 38 shows the results for packet priority ratio (PPR) experiment. PPR is the ratio of *Low* to *Medium* to *High* priority traffic. It is seen that throughput changes with PPR (different combinations of priority traffic), but the variations are a very small factor. Maximum throughput is 162%, seen at PPR of 1.0:0.0:0.0 (*Low*:*Medium*:*High*), 0.0:1.0:0.0 and 0.0:0.0:1.0. Minimum throughput is 154.1% seen at 0.2:0.4:0.4. It is seen that, throughput variations are always lesser than 5% of the maximum value.

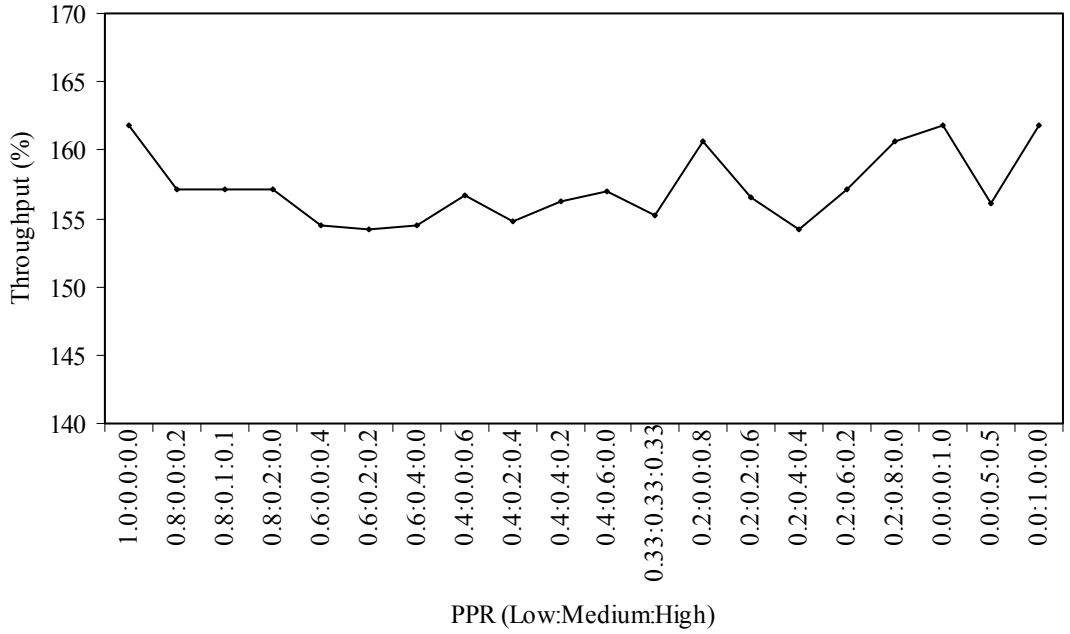

Figure 38. Packet priority ratio experiment results

4.5 Discussion of results

IEEE 1394b exhibits considerably less queuing delay than IEEE 1394a. The difference is especially visible at higher loads (over 90% load). The better performance of IEEE 1394b can be attributed to the overlapping of arbitration and data transmission, and the complete elimination of idle arbitration gaps. IEEE 1394b asynchronous stream transactions offer a better delay performance than isochronous transactions for packet-based MPEG-2 and MPEG-4 video transmissions. For a saturated (fully loaded) network the queuing delay of asynchronous stream packets is nearly 15 times less than the queuing delay of isochronous packets. This is a motivation for completely eliminating the isochronous service in SFP.

SFP improves the throughput of IEEE 1394b by a factor of 1.7, 6.8, and 43.9 for *Spatial_average*, *Spatial_video* and *Spatial_max* traffic patterns, respectively. For the *Spatial_min* traffic pattern (a restricted case that permits no spatial reuse), SFP and IEEE 1394b offer similar performance. A similar improvement is seen in the node capacity of SFP at saturated network conditions. The better performance of SFP can be attributed to the spatial reuse of bandwidth. From the packet size experiment it is clear that SFP offers better throughput for large (mean) packet sizes (greater than 1500 bytes). SFP throughput increases with packet size because the percentage of overhead (arbitration granting overhead) per packet transaction decreases with the increase in packet size. From the results, it is clear that SFP priority arbitration distinctly separates the three priority classes in delay performance. *High* priority packets offer nearly 6 times lower queuing delay than

Medium priority packets, whose delay is more than 100 times lower than *Low* priority delay. Asynchronous stream packets mapped to the different priority classes can provide a flexible service for MPEG-2 and MPEG-4 video. It is seen that for different combinations of priority traffic the throughput variations are not much and fall within 5% of the maximum value. This makes it clear that SFP does not compromise in (maximizing) throughput while providing service for priority traffic and strikes a good balance.

CHAPTER 5

CONCLUSION

This thesis presented the Spatial reuse FireWire Protocol (SFP), a novel bus arbitration protocol architected for an acyclic daisy-chained topology. Shared-medium daisy-chained network technologies are necessary to support economical installation of large-scale video surveillance systems. SFP is based upon the IEEE 1394b FireWire architecture and preserves the simple repeat path functionality of FireWire. SFP improves the effective throughput of FireWire by spatial reuse of bandwidth and QoS support for packet video by a real-time priority based bus access mechanism.

5.1 Summary of contributions

This thesis investigated new communication protocols suitable for video surveillance systems, in particular at the medium access control level (bus arbitration) and physical layer. The main contributions of this work are:

- A comprehensive study of the evolution of networks for video surveillance systems was made. FireWire was identified as a potential low-cost technology for video surveillance systems.

- IEEE 1394b FireWire was investigated as a candidate technology for video surveillance. Performance limitations in FireWire, such as lack of spatial reuse and lack of support for priority traffic, were identified. Simulation results demonstrate that FireWire asynchronous stream transactions offer a better delay performance than isochronous transactions for widely used variable bit-rate video like MPEG-2 and MPEG-4. This result motivates the elimination of isochronous service in SFP.
- Designed and evaluated the performance of Spatial reuse FireWire Protocol (SFP). SFP improves the throughput of IEEE 1394b by a factor of seven for a video surveillance traffic pattern and a factor of two for a homogeneous traffic pattern. SFP provides support for variable size packets, asynchronous stream and asynchronous transactions, three classes of priority, and ensures fairness among like priority nodes.

5.2 Future research

In SFP arbitration requesting is overlapped with data transmission. However, arbitration granting overhead (which depends upon the propagation and the repeat path delays) increases with node count. At a high node count, arbitration granting overhead can become considerable. One possible way to minimize this overhead is for the bus owner to issue grants to multiple sets of requests at the same time. Data transmission between consecutive sets must be properly synchronized and the next bus owner should be one of the nodes from the last transmitting set of nodes. Another approach is to have multiple

operational bus owner nodes at the same time, each taking care of a certain small portion of the network. For data traffic between distinct portions, the bus owners should carefully synchronize among themselves.

Future research directions include; extension of SFP to accommodate multicast traffic, extension of SFP to support a tree topology network where every node can have two or more ports for branching, improving the robustness of SFP to handle packet losses (especially arbitration request and grant packets), and performance evaluation of higher layer protocols (such as TCP/IP) over SFP.

REFERENCES

[1] D. Anderson, "FireWire System Architecture (Second Edition)," MindShare, Inc., 1999.

[2] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "Habitat Monitoring: Application Driver for Wireless Communications Technology," *Computer Communications Review, Supplement issue*, pp. 20-41, 2001.

[3] V. Chandramohan and K. Christensen, "A First Look at Wired Sensor Networks for Video Surveillance Systems," *proceedings of the High Speed Local Networks Workshop at the 27th IEEE Conference on Local Computer Networks (LCN),* pp. 728-729, November 2002.

[4] K. Delin and S. Jackson, "Sensor Web for In Situ Exploration of Gaseous Biosignatures," *Proceedings of the IEEE Aerospace Conference*, pp. 465-472, 2000.

[5] Detect and Photograph Intruders With a Portable, Motion-Sensing Camera!, SMARTHOME, Inc., 2002. URL: *http://www.smarthome.com/764801.html*.

[6] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," *Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 263-270, 1999.

[7] *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers*. Committee on Networked Systems of Embedded Computers. National Academy Press, Washington, DC, 2001.

[8] W. Feng, J. Wadpole, W. Feng, and C. Pu, "Moving Towards Massively Scalable Video-Based Sensor Networks," *Large Scale Networking Workshop*, 2001.

[9] "FireWire versus Gigabit Ethernet: Dare to Compare," 2002.
URL: *http://www.unibrain.com/products/ieee-1394/fw_vs_gbit.htm*.

[10] H. P. Fitzek and M. Reisslein, "MPEG-4 and H.263 Video Traces for Network Performance Evaluation," 2003.
URL: *http://www-tkn.ee.tu-berlin.de/research/trace/trace.html*.

[11] K. Fujisawa, “Transmission of IPv6 Packets Over IEEE 1394 Networks,” *RFC 3146*, October 2001.

[12] IEEE Std. 1394b – 2002 IEEE Standard for a High-Performance Serial Bus – Amendment 2, 2002.

[13] IEEE Std. 1394a – 2000 IEEE Standard for a High-Performance Serial Bus – Amendment 1, 2000.

[14] IEEE Std 1394-1995, Standard for a High Performance Serial Bus, 1995.

[15] IEEE P802.3af, Draft Standard for DTE Power via MDI (*http://grouper.ieee.org/groups/802/3/af/*), May 16, 2002.

[16] IEEE p1394.1 – High Performance Serial Bus Bridges Working Group, 2003. URL: *http://grouper.ieee.org/groups/1394/1*.

[17] P. Johansson, “IPv4 Over IEEE 1394,” *RFC 2734*, December 1999.

[18] A. Mukherjee and A. Adas, “An MPEG2 Traffic Library and its Properties,” (*http://www.knoltex.com/aboutKnoltex/people/amarnath/papers/mpeg2Library.htm*), 1998.

[19] Network Camera, DVR and Video Servers, Axis Communications, Inc., 2002. URL: *http://www.axis.com/products/camera_servers/index.htm*.

[20] T. Norimatsu, H. Takai, and H. Gail, “Performance Analysis of the IEEE 1394 Serial Bus,” *IEICE Transactions on Communications*, Volume E84-B, Issue 11, pp. 2979-2987, 2001.

[21] K. Obraczka, R. Manduchi, and J.J. Garcia, "Managing the Information Flow in Visual Sensor Networks,"*Fifth International Symposium on Wireless Personal Multimedia Communications*, October 2002.

[22] People-Mover Project Brings 21st Century Surveillance System to Dallas Airport, Telindus, 2002.
URL: *http://www.cellstack.com/news_info/case_dallas_airprt.pdf*.

[23] H. Qi, S. Iyengar, and K. Chakrabarty, “Distributed Sensor Networks – A Review of Recent Research,” *Journal of the Franklin Institute*, Vol. 338, pp. 655-668, 2001.

[24] T. Radford (science editor), “In-Flight Cameras to Curb Hijack Fears,” *The Guardian*, Thursday May 9, 2002.

[25] M. Sjodin, “Response-Time Analysis for ATM Networks,” *Licentiate Thesis, Department of Computer Systems*, Uppsala University, 1995.

[26] D. Tsiang and G. Suwala, “The Cisco SRP MAC Layer Protocol,” *RFC 2892*, August 2000.

[27] J. Walles, "On Capacity Utilization in IEEE-1394 FireWire," M.Sc. Thesis in Computer Science.
URL: *http:// www.d.kth.se/~d92-jwa/exjobb/dok/rapport-en.doc.*

[28] S. Zhang and W. M. Dai, “Linear Time Left Edge Algorithm,” *Proceedings of the International Conference on Chip Design Automation*, August 2000.