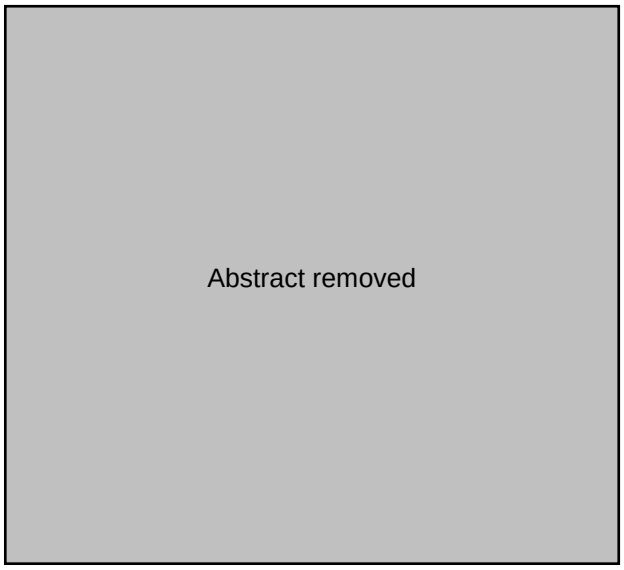
>>> To appear in IEEE LCN 2004 <<<

Design and Performance Evaluation of a New Spatial Reuse FireWire Protocol

Vijay Chandramohan and Ken Christensen
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620
vjcmn@yahoo.com, christen@csee.usf.edu

Abstract



1. Introduction

There is a growing need for visual monitoring and surveillance systems in large facilities such as airports and stadiums. New systems are envisioned with thousands of highly intelligent, interconnected cameras with rich image processing capabilities [9]. To improve safety and security, 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 including the use of intelligent cameras [6, 9]. As these applications grow in complexity and demand, the underlying networks that support video surveillance need to evolve as well.

Existing dedicated-medium switched Ethernet/ATM video surveillance systems require a communication cable per node (i.e., connected to a switch). This dedicated cabling is 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 image processing in each camera, power cannot be delivered for very long by a battery. Power distribution can be combined with wired communication [3]. 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 streams need to be investigated. We propose a new communications protocol suitable for large-scale video surveillance systems that removes the bottleneck of dedicated cabling for each camera. We address medium access control level (bus arbitration) and physical layer issues. Power distribution issues are beyond the scope of this work. We propose the Spatial reuse FireWire Protocol (SFP), which improves the effective throughput of IEEE 1394b FireWire by concurrent packet transmissions (spatial reuse) and the QoS for packet video by a real-time priority based bus access mechanism.

The remainder of this paper is organized as follows. Section 2 reviews the IEEE 1394b FireWire protocol with a focus on bus arbitration. Section 3 describes the design of the new Spatial Reuse FireWire Protocol (SFP). Section 4 evaluates the performance of SFP and IEEE 1394b FireWire for packet-based video transmission. Section 5 summarizes and describes future research.

2. Review of IEEE 1394b FireWire

IEEE 1394 FireWire is the only existing technology that supports a shared-medium, daisy-chained topology and has built-in power distribution. Each FireWire node is a part of the repeat path. A FireWire cable consists of three pairs of wires, two for data transmission and one for power. FireWire employs shielded twisted pair (STP) cabling. IEEE 1394b, which is the latest standard, can use multimode fiber (MMF) for added bandwidth and distance. A 100 meter span between nodes with up to 63 nodes at 1.6-Gbps data rate can be achieved.

This work was funded by a Spring 2001 Florida Space Research and Education Grant (reference PO A07277).

FireWire data transactions are packet based and can be classified as asynchronous or isochronous. Asynchronous transactions are guaranteed in delivery and require an acknowledgement from the receiver. Isochronous transactions are guaranteed in time with a specific bandwidth reserved for them on the serial bus. Bandwidth is allocated in portions of 125 microsecond intervals, called cycles. There is another transaction service called asynchronous streaming that is guaranteed neither in time nor in delivery. Reference [1] gives a detailed description of the FireWire architecture.

2.1 Bus arbitration in IEEE 1394b FireWire

FireWire employs a request/grant arbitration mechanism to control access to the shared bus. Nodes that wish to transmit a packet request permission from the bus owner supervisor selector (BOSS) node. The BOSS node selects a best request based upon specified 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 BOSS node. All arbitrating nodes perform the role of BOSS node in a round-robin fashion. The last node to transmit a packet that does not require an acknowledgement acts as the next BOSS node. Arbitration requests and grants are 10-bit symbols called tokens. The full-duplex nature of the IEEE 1394b bus enables overlapping of arbitration with data transmission. The FireWire data transmission interface (physical layer) has two twisted pairs TPA and TPB that are crosswired within the cable (between nodes). The signal pairs TPA and TPB can transmit data separately and continuously in opposite directions.

When a node wishes to perform a data transaction it sends out an arbitration request token towards the BOSS. Arbitration request tokens are sent out on any active port that is not transmitting (repeating) a data packet. Arbitrations are divided into isochronous and asynchronous intervals. Both isochronous and asynchronous intervals alternate between "even" and "odd" arbitration phases. Any node that has transmitted an asynchronous/isochronous packet in the current phase can arbitrate only for the next/opposite phase. Arbitration request tokens are classified as isochronous or asynchronous and are also prioritized based on the phase of arbitration. Each node transmits request tokens based upon the transaction type and the current phase (even or odd). 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. 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. The priority mechanisms in IEEE 1394b FireWire simply provide a means for alternating between isochronous and asynchronous arbitrations and ensuring bandwidth fairness. A detailed description of IEEE 1394b arbitration is given in [2] and a performance analysis in [5].

2.2 Performance limitations in FireWire

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 1 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. These transactions can only be scheduled to occur one after the other. 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.

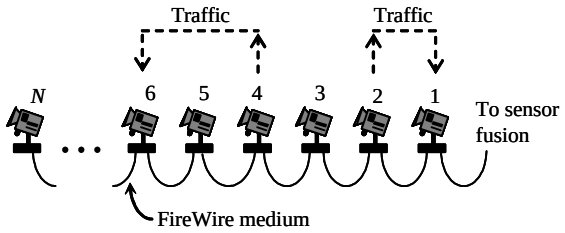


Figure 1. Daisy-chained FireWire network

FireWire provides QoS guarantees for real-time traffic by isochronous bandwidth reservation. The isochronous scheme, however, lacks the flexibility to react to the rate variations as seen in variable bit rate video (VBR) such as MPEG-2 and MPEG-4 video. Reserving bandwidth corresponding to the peak bit-rate will result in a waste of resources. A real-time priority-based packet scheduling mechanism can be more suitable for VBR video and can result in a more efficient use of resources [8].

3. The Spatial reuse FireWire Protocol

The new Spatial reuse FireWire Protocol (SFP) supports overlapped request-grant cycles and uses existing IEEE 1394b FireWire protocols as much as possible. This section describes SFP in detail.

3.1 Key definitions and concepts

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 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. Two requests are compatible if their corresponding data transactions occupy distinct segments of the network. 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.

Arbitration granting: The *bus owner* broadcasts a grant packet with information about the granted nodes. Nodes that explicitly see a grant for them can transmit their data packet concurrently. The grant packet also identifies the destination nodes that are 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. We assume only unicast packets in this work.

3.2 The data transmission interface

Figure 2 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, but operate as two independent half-duplex lines. Standard FireWire cabling can be used. TPB is called the request line and is dedicated to carrying arbitration requests. TPA, or the data line, carries data and grant packets. TPA and TPB are driven by separate half-duplex transmitter/receiver logic and can independently and concurrently carry data.

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 data 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. Each node has knowledge of the 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.

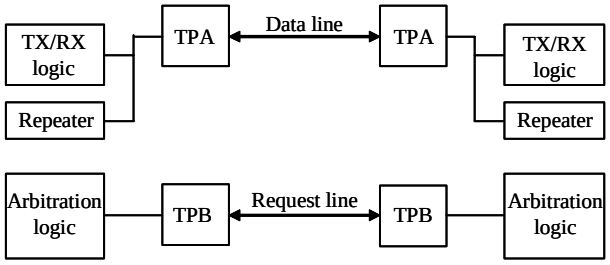


Figure 2. SFP data transmission interface

3.3 Arbitration requesting

Arbitration requests in SFP 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* is 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, which 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 (can be *High*, *Medium*, or *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.

It is assumed that all nodes are synchronized to a common clock. This synchronization takes place during the network configuration. It is expected that each node runs an *arbitration cycle master* whose time cycle continuously alternates between *odd* and *even* request intervals. Since the nodes are synchronized, the cycle changes occur in all nodes at the same time. Every node caches any new request it receives and also retransmits it to the neighbors in the appropriate request interval (i.e., odd addressed nodes in an *odd* interval and even addressed nodes in an *even* 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. In SFP, arbitration requesting is never blocked by data traffic and occurs continuously and independently of data transmissions.

3.4 Packet priority and fairness

SFP provides support for three priority classes, *Low*, *Medium*, and *High*. Each node implements three priority queues (transmit buffers) corresponding to the three classes of priority. 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 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 and overrides the old (lower priority) cache entry.

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. An 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.

3.5 Operation of the bus owner

The *bus owner* makes the arbitration decision (i.e., selects a group of nodes for bus access). SFP nodes take turns in playing the role of *bus owner* and there is always one active *bus owner*. After making 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 (the address of the next bus owner) is included in the grant packet that it broadcasts. In the absence of new requests, the current *bus owner* retains its control.

Figure 3 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 "15" is called the right end. A connection (dashed line) between any two nodes, A and B, indicates that a data transaction (or simply "transaction") needs to be established between them (e.g., A wishes to transmit a packet to B, or vice versa). Figure 3 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 3, 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 the information shown in Figure 3.

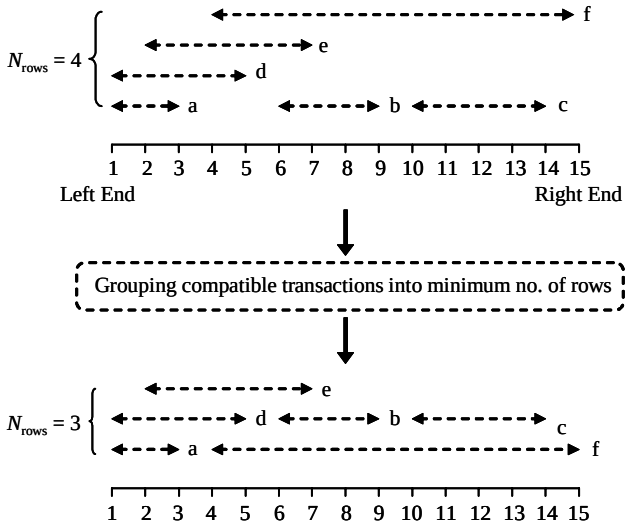


Figure 3. Grouping of compatible requests

The *bus owner* arbitration decision algorithm groups compatible transactions into a minimal number of rows (or sets). 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.

3.6 Design of the 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). Each array element is associated with a one-bit flag. 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, packet phase, and packet priority fields of the received requests. The source address field of a request serves as its unique "signature". Figure 4 shows a request cache.

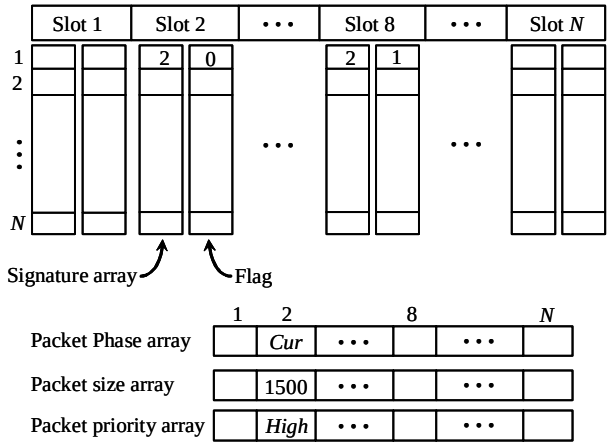


Figure 4. SFP request cache

The following example (see Figure 4) 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 size – 1500 bytes
- 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. 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.

3.7 Bus owner arbitration decision algorithm

The first task for the *bus owner* is to group the requests into a minimum number of sets. It can be observed 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. Requests are sorted based on their *left* and *right* addresses. A request cache is used to keep requests in a sorted order. A stack data structure that stores the signature of requests is used. 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 a stack data structure eliminates the need for scanning already created sets. The use of a stack to group requests is adopted from [10], which presents a linear time left edge algorithm for channel routing in VLSI circuits. The requests are grouped in a single sweep of the request cache.

The bus owner arbitration decision algorithm is presented in Figure 5. In lines 1 to 12, the requests in the cache are partitioned into a minimal number of sets of compatible requests. The request cache is scanned from slot 1 to slot *N*. 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. 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. The variable *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 to 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.

```
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.             Increment index
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 max requests and
           at least one priority j request
16.      Issue grant to all reqs in the selected set
17.      Exit from this algorithm
```

Figure 5. Bus owner arbitration decision algorithm

3.8 Arbitration granting

After making the arbitration decision, the *bus owner* broadcasts a grant packet. Every node makes a local copy of the grant packet and repeats it to the neighbor. The grant packet includes several fields of information:

- 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 the 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 field.
- Arbitration reset status: When the *bus owner* sees no *Current* phase requests it performs an arbitration reset by setting this field value to TRUE. Otherwise 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*. From the list of granted nodes the current *bus owner* selects the node that will be the last to complete data transmission. This node is the next *bus owner*. The node that sees its address in this field, must take control of the bus owner operation at the end of its data transmission.

4. Performance evaluation of SFP

Using simulation, the queuing delay and the throughput performance of SFP and IEEE 1394b were evaluated. Discrete-event queuing simulation models of the two protocols were built using the CSIM18 function library [7]. All models include propagation (5 nanoseconds per meter) and repeat path (144 nanoseconds per node) delays. A response delay of 244 nanoseconds for the bus owner is also included. These delay values are from the IEEE 1394b standard [2].

4.1 Traffic models for simulation experiments

Two traffic models were used to evaluate performance. The first traffic model was twenty 5-Mbps MPEG-2 video sources based on frame traces [4]. The MPEG-2 frame traces were converted into packet sizes with 48 bytes of overhead (representing LAN and upper layer headers) per packet. When the number of simulated nodes is greater than 20, copies of the frame traces are randomly assigned between nodes. For the 20 MPEG-2 sources, the mean packet length was 1460 bytes. The second traffic model was Poisson arrivals of 1460-byte fixed-length packets.

4.2 The simulated configuration

A bus topology of *N* nodes, as shown in Figure 1, was modeled. Each node was an independent traffic source. Each node was assumed to have an infinite capacity buffer for packets being sent on the link. The distance between node pairs was equal and fixed at 10 meters. All internode links had equal bandwidth capacity, which was varied between the experiments. Source-destination traffic distributions between the nodes were based on four models as described below. Each model was characterized by a distinct value of a spatial reuse factor; *S*. The value of *S* is the average number of concurrent packet transmissions that can occur in the network.

- *Spatial_min*: All packets (of all nodes) are destined to the head end, which acts as the sensor fusion node (control unit). Since no concurrent packet transmissions are possible, $S = 1$.
- *Spatial_average*: For every packet, a source node uniformly selects a destination node. The value of *S* for this model is equivalent to the total number of nodes divided by the average distance between two nodes, which is $S = N/(N/2) = 2$.

- *Spatial_video*: For 90% of the time nodes send packets to their right or left neighbors. For 10% of the time packets are destined to the head end. In a video surveillance system most of the traffic will occur between peer cameras (e.g., 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. This models the expected traffic distribution in a typical video surveillance system. S for this model is given as,

$$S = \sum_{i=1}^{N} i \cdot P_{head} \cdot P_{adj}^{i-1} \quad (1)$$

  where P_{head} is the probability that packets are destined to the head end (0.1) and P_{adj} the probability that packets are for adjacent nodes (0.9).
- *Spatial_max*: All nodes send packets to their right neighbors. For this mode, $S = N$.

4.3 Description of simulation experiments

Five experiments were defined to evaluate the performance of IEEE 1394b and SFP. 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. Unless otherwise specified the response variable for experiments is mean queuing delay. Control variables are offered packet load on the link and the number of nodes.

Isochronous experiment (IEEE 1394b): Isochronous bandwidth reservation is evaluated against asynchronous packet streams. The number of nodes is increased from 2 to 19. Link bandwidth is fixed at 100 Mbps. MPEG-2 sources are used.

Load experiment (SFP vs. IEEE 1394b): The offered packet load on the link is increased from 10% to as high as 4500%. The number of nodes is fixed at 60 and the link bandwidth at 400 Mbps. Nodes are Poisson sources.

Node count experiment (SFP): The number of nodes is increased from 4 to 1000. The link bandwidth is fixed at 100 Mbps. Each node is a 5 Mbps Poisson traffic source. All the spatial reuse models are evaluated.

Priority experiment (SFP): The offered load on the link 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. Nodes are MPEG-2 sources. The *Spatial_average* model was used.

Packet priority ratio (PPR) experiment (SFP): 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. Poisson sources are used. The *Spatial_average* model used was used.

4.4 Results from the simulation experiments

Figure 6 shows the isochronous experiment results. IEEE 1394b asynchronous stream transactions offer a better delay performance than isochronous transactions for packet-based 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.

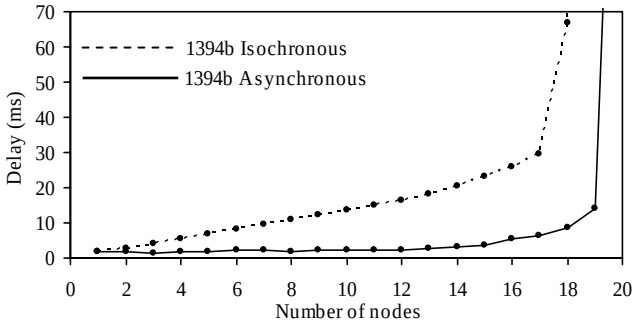


Figure 6. Isochronous results

Figure 7 shows the load experiment results. For the *Spatial_min* traffic pattern (a restricted case that permits no spatial reuse), SFP and IEEE 1394b offer similar performance. Both reach a throughput maximum at 98% load. IEEE 1394b offers identical performance for all four traffic models, as it does not support spatial reuse. Maximum throughput is implied when queuing delay increases at a large rate and is much higher than the tolerance of human response time (100 milliseconds). SFP reaches a maximum throughput at 165%, 650%, and 4250% loads for *Spatial_average*, *Spatial_video*, and *Spatial_max*, respectively. 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*, respectively. A similar improvement factor is seen in the node capacity of SFP at saturated network conditions.

Figure 8 shows the node count experiment results. For a saturated network, SFP is able to support 19, 34, 145 and more than 1000 nodes for *Spatial_min*, *Spatial_average*, *Spatial_video* and *Spatial_max*, traffic distributions, respectively. From the priority experiment results in Figure 9, it is clear that SFP priority arbitration distinctly separates the three priority classes in delay performance. *High* priority packets experience 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. From the PPR

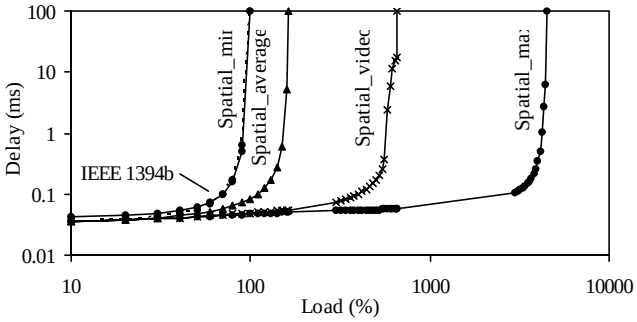


Figure 7. Load results

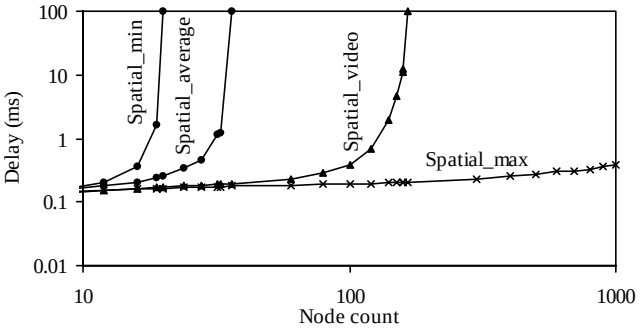


Figure 8. Node count results

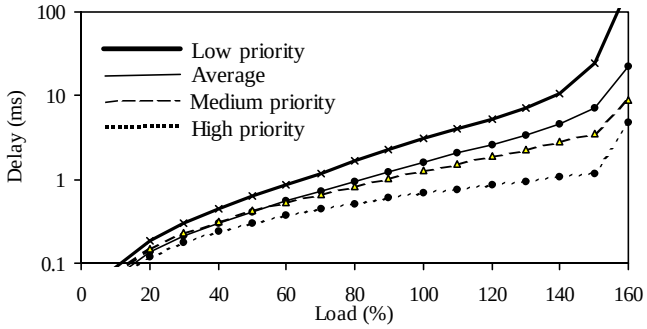


Figure 9. Priority results

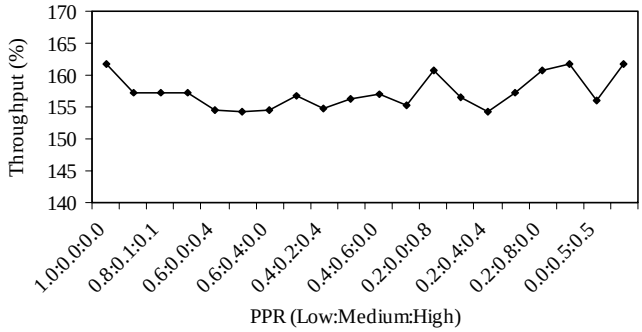


Figure 10. Packet priority ratio (PPR) results

experiment results in Figure 10 it is seen that for different combinations of priority traffic the throughput variations are small 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

5. Summary and future work

In future local area networks and wired sensor networks the cost of dedicated cabling will likely exceed the cost of the interconnected devices (such as of video cameras). Daisy-chained, shared-medium networks are needed to reduce this large cabling cost. In this paper, the Spatial reuse FireWire Protocol (SFP) was presented and evaluated. The evaluation showed that the fully-distributed SFP protocol can provide near ideal spatial reuse of a shared FireWire-like channel. The SFP protocol uses a "left edge" algorithm to efficiently group compatible transactions and enable concurrent transactions on a single, shared FireWire network. SFP also supports multiple priority levels for simultaneous video and data transport. Future work includes extending SFP to accommodate multicast traffic, improving the robustness of SFP to handle packet losses, and performance evaluation of higher layer protocols (such as TCP/IP) over SFP.

References

[1] D. Anderson, *FireWire System Architecture: IEEE 1394A*, Second Edition, Addison-Wesley Professional, 1998.

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

[3] IEEE P802.3af, Draft Standard for DTE Power via MDI, May 16, 2002.

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

[5] T. Norimatsu, H. Takagi, and H. Gail, "Performance Analysis of the IEEE 1394 Serial Bus," *Performance Evaluation*, Vol 50, pp. 1-26, 2002.

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

[7] H. Schwetman, "CSIM18 - The Simulation Engine," *Proceedings of the 1996 Winter Simulation Conference*, pp. 517 - 521, December 1996.

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

[9] WeSpot AB, Lund, Sweden, 2003. URL: *http://www.wespot.se*.

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