CMPT 885-3: SPECIAL TOPICS: HIGH-PERFORMANCE NETWORKS

IEEE 1394 and Isochronous Traffic:

An ns2 implementation and simulation of effective bandwidth usage.

Spring 2002

FINAL PROJECT

Glendon Holst
gholst@cs.sfu.ca

Nothing is as simple as it seems at first, or as hopeless as it seems in the middle, or as finished as it seems in the end. – Anonymous

0 - Abstract:

In this project paper, I explore the media access control mechanisms of three packet network protocols, (IEEE 802.3, IEEE 802.5, and IEEE 1394), describe the IEEE 1394 protocol in depth, describe the ns2 simulator concepts, describe how to implement IEEE 1394 in ns2, and test a variety of IEEE 1394 networks for their bandwidth utilization using asynchronous and isochronous transfer modes.

Since asynchronous and isochronous transfers are each suited to particular applications, the results had little further significance; however, they do demonstrate considerable work, and an understanding of the IEEE1394 protocol and the ns2 simulator.

1 - Introduction:

IEEE 802.3 (Ethernet) technologies coordinate access to the physical medium with a protocol that listens for activity, sends packets when no activity is detected, and detects packet collision. As contention for network access increases, the effective available bandwidth (i.e., the bandwidth remaining after packet collisions) decreases because more bandwidth is consumed by packet collisions. Packet length, and cable length also affect available bandwidth, since larger packets and longer cable lengths each increase the bandwidth wasted by collisions.

IEEE 802.5 (Token Ring) technologies coordinate access to the physical medium by sharing a token packet, thus ensuring fair access to the network without contention. While there is always the overhead of the token arbitration, this overhead remains fixed; however, there are a number of failure modes that could affect bandwidth utilization during token recovery.

In other words, the arbitration overhead for Ethernet increases with demand, while the overhead for Token Ring remains constant.

The purpose of this paper and project were to study the effective bandwidth utilization of different arbitration schemes and determine how they behave under increasing load, and varying topologies. The technology chosen for the basis for study is IEEE 1394 (aka, Firewire), a serial-bus which behaves like a packet switched network and supports two bus arbitration schemes: asynchronous and isochronous.

A further purpose of this project for myself, was to learn about IEEE 1394 and about implementing new protocols in the network simulator ns2. While my original interest in the effective bandwidth behaviour of various arbitration schemes was motivated by the differences in IEEE 802.3 and 802.5, my combined interest in IEEE 1394 and in extending ns2 (which does not presently support IEEE 1394), have flavored the

direction of the project. Which of the two delivery modes we use in IEEE 1394 (asynchronous and isochronous) is determined primarily by the type of traffic we send, and less upon the effective bandwidth, as studied here.

However, comparing these two transport schemes is still informative, and this paper provides a useful overview of IEEE 1394 and implementing new LAN level protocols in ns2.

Section 2 details the IEEE 1394 serial-bus and its protocols, sufficient for our simulation purposes in ns2. IEEE 1394 is also known as FireWire (Apple) and iLink (Sony).

Section 3 provides an overview of the ns2 network simulator and details the changes and additions required to add the portions of IEEE 1394 relevant to the following study.

Section 4 describes the testing results of IEEE 1394 networks, over a variety of topologies, network load, and delivery modes. The section concludes with an analysis of the results.

Section 5 provides a summary of the entire project and paper.

2 - FireWire:

2.1 - overview:

IEEE 1394 (aka, FireWire) is a serial-bus interface and protocol that performs much like a packet switched network. The existing IEEE 1394a is a 400 Mbps technology, with the proposed IEEE 1394b extending that all the way up to 3.2 Gbps. Later revisions improve the protocol efficiency, and both IEEE 1394a and 1394b extend the original IEEE 1394-1995 protocols to support packet transmission or bus-arbitration immediately after an ACK, instead of waiting for the idle gap times to expire.

Presently this technology shows up in computers, video cameras, stereo systems, high-speed peripherals, and more. It supports peer-to-peer communications (unlike USB which requires a single controlling host), supports dynamic plugging and unplugging of devices, and auto-detects the spanning tree for its network layout.

IEEE 1394 technologies support two delivery modes:

1) guaranteed delivery (asynchronous).
2) guaranteed delay (isochronous).

Isochronous transfer provides a fixed bandwidth channel over fixed time intervals. This

mode is ideally suited to time-based multi-media applications such as video and audio.

Since the isochronous transfer will not re-transmit lost packets, it is not suitable for all tasks (such as file transfers); further, IEEE 1394 uses bus arbitration (command signals and packets) to determine who can transmit data next (unlike Ethernet, where simultaneous attempts at transmission will consume bandwidth). Since application needs are the primary constraint in choosing the delivery mode, isochronous transfer is not the universal solution to improve efficient allocation of bandwidth. However, determining the overhead imposed by both transfer modes under a variety of network loads and topologies, is still interesting.

The following sections detail the functioning of various IEEE 1394 layers as relevant to simulation the proposed simulation.

2.2 - architecture

The IEEE 1394 architecture is defined in detail from the physical medium (cables and connectors, signaling, etc.), to the PHY layer (which controls access to the bus much like a MAC), up through the link and transaction layers, and finally to the node services such as those for power and bus management, bandwidth resource allocation, cycle control, and configuration.

IEEE 1394 is built upon the ANSI/IEEE 1212 specification for the core features of microcomputer busses. This defines the underlying control and status registers (CSRs), common transaction types, and the addressing model.

The addressing model in IEEE 1394 supports up to 63 nodes on each bus, up to 1024 busses, and a 256 terabyte (TB) memory address space for each node. Each bus supports a broadcast mode to all its nodes.

Nodes are the basic unit of connection to bus, and contain one each of a PHY and LINK layer unit, along with any higher application level units which give the node its 'personality'. From the user perspective, the devices they connect to the bus are called modules, and may contain one or more nodes. For the purposes of simulation, the node is the connecting element of interest (i.e., modules contain only one node). The IEEE 1394 node corresponds well with the ns2 LanNode object which only contains one ns2 Node instance. Details of this are covered in Section 3.

The IEEE 1394 bus provides many features which serve to make it attractive to end users. These include:

Dynamic Plugin: Devices may be swapped in and out of the bus at any time.

Plug and Play: Devices can identify themselves to enable automatic driver loading.
Powered Bus: Devices may either consume or produce power, and devices may go into low powered modes.

These capabilities require support from the IEEE 1394 architecture. Dynamic plugin requires that the bus be self-configuring:

• Discovering bus topology (tree identification),
• Setting and preserving (where possible) node address and identities (Physical Identification and Self-Identification),
• Determining neighboring node speed capabilities,
• Assigning management responsibilities for Cycle Timers, Isochronous Resource allocation, and Bus Manager (i.e., the root node), and
• Providing continuity to in progress transactions where possible. Continuity also requires protocols to deal with fair resource acquisition after a bus-reset. Sleeping nodes require other nodes to act as proxies for them while they are in low powered states.

These capabilities are beyond the scope of our simulation interests. For the purposes of simulation, a fixed and given tree topology is assumed.

Since IEEE 1394 is a serial bus, data is sent as packets from node to node. For the asynchronous mode there are three types of transactions: read, write, and lock. Each of these transactions is composed from packet pairs (a request and a ack response packet) called a sub-action. Transactions are managed by the Transaction layer, which invoke the LINK layer to perform the sub-actions. For the isochronous mode, there is only a broadcast message which is handled at the LINK layer. The PHY layer functions primarily as a MAC, controlling access to the bus. When there is a packet to send, the PHY layer must arbitrate for access to the bus. When access is granted, the PHY layer sends the packet over the physical medium. The arbitration protocols ensure fair access to the bandwidth, using 125 μs cycles for isochronous transfers, and a fairness interval for asynchronous transfers. These are described in greater detail below.

2.2 - arbitration

Before sending asynchronous sub-packets or isochronous packets the PHY layer must arbitrate for access to the bus. Bus traffic toggles between an arbitration phase and a transfer phase. If a node is granted access during the arbitration phase then it can initiate a sub-action, or send an isochronous packet. For asynchronous transfers, a node is granted access only once during each fairness interval. For isochronous transfers, a node only has one access opportunity during each 125 μs cycle.

The fairness interval ends when the bus is idle for the arbitration reset gap (a gap longer than either the isochronous or sub-action gap). This approach stands in contrast to TokenRing and FDDI technologies which rely on the presence of token to ensure fairness. IEEE 1394 relies on the absence of asynchronous requests to relay fairness information. While IEEE 1394 does not need to worry about regenerating a lost token, it must handle dynamic topology changes (i.e., nodes added or removed from the bus), which could be its equivalent to the token regeneration process.

A 125 μs cycle is used to synchronize isochronous transmission. The Cycle Master, usually the root node, is responsible for broadcasting a Cycle_Start packet every 125 μs. Since the root node is also the access arbitrator, it can win any arbitration round, and send the Cycle_Start packet in a timely fashion. Delay jitter can occur when other nodes win asynchronous arbitration, and their sub-actions overlap the cycle start time. When the root sends the Cycle_Start packet it includes delay offset time so that other nodes can remain synchronized to the actual start time. Such synchronization becomes important for the protocol optimizations, such as acknowledge accelerated arbitration and fly-by arbitration introduced in 1394a and later. These optimizations allow nodes to receive implicit arbitration access in special cases. Such implicit arbitration reduces the arbitration overhead, yet remains fair (the rules of the fairness interval are still observed). Implicit arbitration can happen when a node was the target of the last sub-action (acknowledge accelerated arbitration), or when a node performs as a repeater and wishes to send a packet towards the root (fly-by arbitration). Such accelerated arbitration could prevent the root node from sending the Cycle_Start packet on time. To alleviate this, nodes which may use accelerated arbitration must keep track of the cycle time, and not use accelerated arbitration when the cycle is about to end. The synchronization information in the Cycle_Start packets ensures low delay jitter.

When a node wants access to the bus it must wait for a bus idle gap corresponding to the type of access it wants: sub-action gap for asynchronous transfers, and the isochronous gap for isochronous transfers. The isochronous gap is shorter than the sub-action gap, and this ensures that all isochronous transfers occur at the beginning of the cycle, and the remaining time of the cycle will be available for asynchronous packets.

When this gap is seen, the PHY layer sends an arbitration request packet to its parent. Each parent node which receives a request chooses the first request that knows about. i.e., the first request to come from one of its children or itself. This node then sends the request up to its parent. Contention for the physical medium is prevented because requests only travel up the topology tree, and because the signals are transmitted by holding a separate communication line high or low. The root node decides on the winning request, just as all the other parent nodes did, but upon deciding it sends a Grant_Access packet down the network tree, and this determines who won. When

nodes receive this Grant_Access the arbitration phase is over. The node that wins can now send a packet to its desired target node. Since the winning node has sole access to the bus, there are no contention issues, and the packet can propagate over the entire network tree, both up and down.

The nodes closer to the root have positional priority over other nodes; however, because of differences in timing caused by packet propagation, positional priority need not entail actual priority. In spite of this, all nodes have equal access to the the bus, since a node can only access the bus once during each fairness interval, and the interval doesn't end until all nodes have had a chance to send an asynchronous sub-action. i.e., nodes can only initiate one sub-action per fairness interval. The exception is the root node, which can send out Cycle_Start packets whenever needed.

Only the basic arbitration process is simulated in ns2.

2.2 - propagation

During the arbitration phase, nodes only send the request signals to their parent, and the access granting packets to their children.

During the data transmission phase, packets are sent out over all ports, except for the port where the packet arrived. The packets are sent out as they arrive, and in this way the total propagation time for a packet is close to the delay time for the longest link path in the bus, and is nearly independent of the number of nodes. The qualification of independence is because each PHY layer will introduce some very small delay, which is still much less than packet transmission time, into the propagation.

In this way, nodes act as repeaters and data packets are propagated over the entire bus (needed to provide information about bus idle information).

2.2 - transactions

Asynchronous transactions are composed from sub-actions, which is a pair of packets. The first packet forms the request (either a read, write, or lock), and the response packet forms the acknowledgment. The ACK packet must follow the request packet before the sub-action gap expires. Some transactions, especially reads, may occur over two sub-actions. Such split-transaction are handled by the transaction layer so that they appear as a single transaction.

For example, if a node sends a write request, the request packet contains the entire data to write, and the receiving node can immediately respond with the acknowledge packet. Such a transfer can complete in one sub-action. A read request would be a small packet containing the address of the memory block to return. If the target node

can return the data before the ack-gap expires, then it does so and the transaction is also completed in one sub-action. In this case, the ACK packet contains the contents of the memory block as its data. If the data is not ready, a special ACK packet with a transaction ID is returned that indicates the transaction will continue in a following sub-action. The target node must then acquire the requested data (from some higher layer), arbitrate for control of the bus, and send the data as part of the request packet (it is actually a response, but it is the first packet of the sub-action). The target node of this sub-action (which was the previous requester) replies with an ACK packet to indicate the split transaction is complete.

If an ACK packet is not received, the link or transactions layers will retry. Since packets cannot be lost in the simulation by error or bus contention, we will ignore this retry behaviour.

The lock request packet is used to create complex, yet atomic, transactions formed from several sub-actions or transactions.

Isochronous transactions involve broadcast packets, which are not acknowledged. These broadcast packets are addressed by channel number (0-63) and can be twice as large as asynchronous packets, at the same bus speed. Initiating isochronous transfers involves a complex setup process which occurs by nodes communicating via asynchronous packets. This setup establishes the channel numbers, and allocates bandwidth resources.

The first phase of the setup involve transactions with the Isochronous Resource Manager (usually the root node) to allocate channels and bandwidth. The resource manager ensures that at least 20% of the bus bandwidth remains available for asynchronous transfers. These transaction are wrapped by a lock transaction to prevent race conditions. The second phase is to notify all target nodes, via asynchronous transactions, of which channel(s) to listen to. Once this is done, the node is ready to participate in isochronous arbitration that occurs during each cycle. A node may only win isochronous access once each cycle, however, it can send out a packet for each channel that it owns during that access. Because the isochronous gap is shorter than the sub-action gap time, all isochronous transfers will occur at the beginning of the cycle (i.e., because isochronous nodes respond faster to the idle bus, they will always win arbitration ahead of asynchronous nodes). When a node no longer needs to send isochronous traffic, it must tear-down the channels that it allocated, returning the channels and their bandwidth to the resource manager, and notifying the target nodes to no longer listen in on the channels.

For purposes of simulation, this setup will be simplified.

2.2 - packets

IEEE 1394 packets have a 8 to 20 byte header, a payload, and quad-word (four byte) data CRC as the trailer.

For asynchronous packets the payload can be $2^{(9+\log_2(LS/100))}$ bytes, where LS is the link speed in Mbps. For the slowest speed, 100Mbps, the payload size is 512 bytes, and for the top speed of 400Mbps for 1384a, the payload size is 2048 bytes.

For isochronous packets the payload can be $2^{(10+\log_2(LS/100))}$ bytes, where LS is the link speed in Mbps. For the slowest speed, 100Mbps, the payload size is 1024 bytes, and for the top speed of 400Mbps for 1384a, the payload size is 4096 bytes.

The payload is padded to align on a quad word boundary (four byte boundary).

Isochronous packets have a single 8 byte header with the following fields:

data_length : 2 bytes
tag (data format), channel (0..63), tcode (transaction code), sy (sync code) : 2 bytes
header_CRC : 4 bytes

Asynchronous packets have more complex headers, whose content depends upon the type of transaction they represent. The 20 byte asynchronous header commonly contains the following fields:

destination_ID : 2 bytes
tl (transaction label), rt (retry), tcode (transaction code), pri (priority) : 2 bytes
source_ID : 2 bytes
destination_offset : 6 bytes
packet_type specific data : 4 bytes
header_CRC : 4 bytes

The node ID fields (destination_ID, and source_ID) have 10 bits for the bus ID, and 6 bits to designate the node on that bus. The the 48 bit destination_offset field is a memory address of the target node.

3 - Implementation:

3.1 - overview

ns2 is a network simulation system based on C++ classes and the OTcl scripting language. Presently, it is geared towards IP over ethernet, wireless, or satellite; however, it is a sufficiently generalized discrete event simulator, and can be extended to support other protocols.

In the most abstract sense, a discrete event simulator is composed from: a graph, timers, and events. Vertices set timers which deliver events to neighboring vertices. When a vertex receives an event it updates its own state, then sets a timer for another (possibly the same) event, based upon this state. In this way events are propagated through vertices on the graph, updating the vertex states correctly, since the events arrive in their proper order (i.e., for an event at time T, all events for time < T were already delivered, and not events for time > T are yet delivered).

ns2 is not quite as general since the components (and thus the underlying graph) is specific to the predefined network component types. Fortunately, there is a fairly good correspondence between these component types and the structure of most network or protocol architectures. Subclassing these component types allows customization of their behaviour.

3.1 - packets

In ns2, events are transmitted primarily via instances of the Packet class, which inherits from the Event class.

One counter-intuitive aspect of Packet instances, is that they encapsulate every type of packet registered with the system. When adding a new packet to the system, we don't sub-class the Packet class, rather we declare structures which hold the fields of interest in our packet, and then register this struct with the Packet class. It is not possible to register packets dynamically, instead we update the `packet.h` header file directly.

3.1 - nodes

The Node class behaves like a vertex in point-to-point networks. Queues act as edges between Node instances, except they can simulate packet delay by using timers to delay delivery of the event. When a packet reaches a node, it must either be forwarded to other nodes, or it must be delivered to an agent attached to this node. Classifier objects are responsible for determining where the packet is delivered. The Address Classifier takes the incoming packet, and if it is for this node, passes it to a Port Classifier, otherwise the Address Classifier sends it out along one or more outgoing links. The Port Classifier delivers the packet to the appropriate agent.

3.1 - agents

The Agent class represents sources and destinations for packets. Agent objects are well suited to simulate middleware layers, though they could be used for any layer if properly sub-classed. Agent instances create the fully initialized and addressed

packet in response to send messages from Application layer objects that sit upon the agent instances. Agents instances send the packet to Node instance that contains them, and the Node sends the packet to the Address Classifier.

3.1 - lans

Missing from Node object above is the ability to simulate contention for the physical medium. The LanNode object connects Node object together via a Queue, Link Layer, Mac Layer, Physical Wire, and Channel objects. This linkage allows finer grained simulation of LANs, especially the channel contention and the separate layers.

These are the constructs that we will use to simulate IEEE 1394.

The ieee1394Mac class simulates the PHY layer, and part of the LINK layer, of IEEE 1394. It appears in the ns2 OTcl layer as Mac/ieee1394. This class handles:

Arbitration: Participates in arbitration process as specified by IEEE 1394, and only sends data packets after gaining access.
Granting access: Done if this Mac belongs to the root node.
Sending Cycle_Start: Done if this Mac belongs to the root node.
Packet Routing: sends data packets out on all but incoming port. Sends packet to link layer if it is for this node.
Fairness interval: Recognizes the start of another fairness interval if the bus is idle for long enough.
Sub-action gaps: Recognizes the end of a sub-action and initiates arbitration if the Link Layer (ieee1394LL) already sent a packet.
Isochronous gaps: Recognizes the end of an isochronous transfer and initiates arbitration if the Link Layer (ieee1394LL) has already sent an isochronous packet.
Sub-Actions: When asked to send an asynchronous packet, waits for the ACK response. This naturally implements retries. Note: normally this would be associated with the LINK layer, but it is more natural to implement as part of the Mac layer, since in ns2 the connection between Link and Mac is narrow and limited (designed for packets only), while in IEEE1394, there is a direct connection between the two (the PHY layer provides services which are invoked directly by the LINK layer).

The ieee1394LL class simulates portions of the LINK layer of IEEE 1394. It appears in the ns2 OTcl layer as LL/ieee1394. This class handles:

Packet Routing: sends data packets out to the ieee1394Mac instance via a Queue. Sends packet to node if the packet is from the Mac layer.

The ieee1394Agent class simulates the transaction layer of IEEE 1394. There is not much to do for this class except create packets and send them to the Queue. The ieee1394AgentAsync agent sends either asynchronous packets and the ieee1394AgentIsoch agent sends isochronous packets. If the ieee1394AgentIsoch is used to send isochronous packets, the user must ensure that the maximum channels and bandwidth are not exceeded (i.e., the user is the Isochronous Resource Manager). Packets are delivered to the corresponding agent just as asynchronous packets are.

A Node, Queue/DropTail, LL/ieee1394, and Mac/ieee1394 connected together is called a ieee1394 LanNode. These ieee1394 LanNodes are connected together via a Phy and Channel instances. The configuration is shown in Figure 1 and explained in Section 3.2.2.

An example of creating a network of ieee1394 LanNode instances in OTcl follows:

```
create-ieee1394-lan $num_nodes $start_time $end_time $interval $pkt_size \
                    $delay $trace $prefix
```

`$num_nodes` is the number of nodes in the network, `$start_time` is the start time for the simulation, `$end_time` is the end time for the simulation, `$interval` is the time period used to record bandwidth used, `$pkt_size` is actually the data chunk size to send to the Agent/ieee1394/* classes, `$delay` is the delay time for the Channel, `$trace` sets Mac level tracing, and `$prefix` is used to determine the Agent makeup of the network (aync, isoch, or both).

3.2 - added classes

A number of files were modified or added to the ns2 system to simulate a IEEE 1394 network.

• `Makefile.in` - add `ieee-1394/*.o` files, to rebuild Makefile, use: `./configure --x-includes="/Library/Tools/X11R6/include" --x-libraries="/Library/Tools/X11R6/lib"`

• `tcl/lib/ns-packet.tcl` - added ieee1394 packet label.

• `tcl/lib/ns-default.tcl` - at end of file, added ieee1394 section for default values for ieee1394 agent and mac.

• `ieee-1394/testing/firewire-lan.tcl` - contains the functions which construct ieee1394 LanNodes, construct the network topology, and control the simulation. Invoke as: `~/Development/NS/ns-allinone-2.1b8a/ns-2.1b8a/ns firewire-lan.tcl`

from ieee-1394/testing/.

• packet-1394.* - These source (*.cc) and header (*.h) files declare and implement hdr_ieee1394 and ieee1394Packet. hdr_ieee1394 contains the header fields needed by a IEEE 1394 packet. ieee1394Packet can create new Packet instances initialized for IEEE 1394 packets.

• agent-1394.* - These source (*.cc) and header (*.h) files declare and implement the agent classes ieee1394Agent, ieee1394AgentAsync, and ieeeAgentIsoch. ieee1394Agent is the base class that accepts messages from an Application, packetizes them and sends them downwards toward the LL/ieee1394 (via a Queue). This base agent class will sent several packets if necessary (if the message size is larger than the maximum packet size). The ieee1394AgentAsync and ieee1394AgentIsoch subclasses just create the appropriate type of packet (asynchronous or isochronous).

• link-1394.* - These source (*.cc) and header (*.h) files declare and implement the ieee1394LL class, which is a simple link layer that passes packets up and down the ieee1394 LanNode. It should adjust the packet size to add and remove the header, but the Mac layer does this presently. From the perspective of the simulation, this makes no difference.

• mac-1394.* - These source (*.cc) and header (*.h) files declare and implement the core ieee1394Mac class, along with two transceiver classes based on ieee1394MacHandler, four timing classes also based on ieee1394MacHandler, and ten finite automata state classes based upon ieee1394MacState. The ieee1394Mac class is a composite class that uses the transceivers, timers, and state objects to send and receive packets, time the sending and receiving times or packets, time the bus idle gaps, and specify the IEEE 1394 protocol. More information on the state objects can be found in Figure 2, and Section 3.2.2. More information on the transceiver and timer objects can be found in the Appendix, Section 6.8.

3.2.1 - capturing 1394 in ns2

The implementation captures essential elements of IEEE 1394 behaviour, but ignores some aspects of the protocol. The following section explores how the protocl is captured in ns2.

• packet size and travel time: The max packet size for asynchronous and isochronous packets is specify in ns-defaults.tcl for 100Mbs connections. Normally the max size depends on link speed, but this is not captured. If the link speed were to change (set in the bw variable in the create-ieee1394-lan function in firewire-lan.tcl) then

the defaults would also have to change. This can be done programmatically after recompiling. The bandwidth is specified to the Phy/WiredPhy instances, and the delay (0.1 μs for our tests) is specified to the Channel instances.

• different packet types (acks, isoch, async, etc.): The async and isoch data packets are generated by `ieee1394AgentAsync` and `ieee1394AgentIsoch` respectively. The other packets are generated by the Mac level classes. All packet creation is done via the `ieee1394Packet` class, then modified by the sender.

• bus arbitration: Bus arbitration is part of the IEEE 1394 protocol that is implement by the finite automata contained in `ieee1394Mac`, and represented by the `ieee1394MacState` subclasses. Because of the topological layout created by `create-ieee1394-lan` in `firewire-lan.tcl`, each `ieee1394MacHandlerRecv` instance that connects the `ieee1394Mac` to the Phy/WiredPhy instance knows wether it represent an connection to a child or a parent. In this way we can send request packets only to the parent. All other packets are broadcast to the downtarget of each `ieee1394MacHandlerRecv` object, except for the one where the message arrived at. IEEE 1394 uses signals over a separate control line to implement requests, while we simulate them via very small packets over the data line. This can cause a collision to appear, but we account for this by cancelling the late request packet. The end result is the same, since packet collision cannot occur in the IEEE 1394 protocol because access is arbitrated, and we do detect any real collisions (there are none detected because the protocol is properly implemented).

• bus cycles: The isochronous cycle is tracked via the `ieee1394MacHandlerCycleTimer` instance, which sends a cycle-restart event to the current state when the timer expires. This instance is only re-activated if the owning `ieee1394Mac` instance belongs to the root node. If the `ieee1394Mac` instance is a root node, then at the next opportunity to arbitrate, the root node wins the arbitration and sends a CYCLE_RESTART packet. The bus idle times are tracked by the `ieee1394MacHandlerSubactionGapTimer`, `ieee1394MacHandlerIsochGapTimer`, and the `ieee1394MacHandlerFairnessGapTimer`. The isoch gap indicate that the Mac can request to send an isochronous packet (if it hasn't sent one before during this isochronous cycle). The subaction gap indicates that the Mac can request to send an asynchronous packet (if it hasn't sent one before in this fairness interval). The fairness gap indicates that all Macs can once again request to send an asynchronous packet.

• timing values: Except for the cycle timer, which is specified as 125 μsec in the IEEE 1394 standard, the values for the idle gaps were not specified, and appear dependent upon the topology and size of the network. We have chosen values that work well for our tests.

• node addressing and message forwarding: For addressing we use the existing address structure for IP packets in ns2. Since every Mac will see every packet, (they are broadcast out to the downtarget of each `ieee1394MacHandlerRecv` object, except for the one where the message arrived at), a Mac can send a packet up to its LL if the packet is address to the Node atop this LanNode. The Node takes care of sending the packet to the appropriate agent using classifiers, just as is typically done in ns2 for IP packets.

• transmission errors: The Mac will continue attempting to send an asynchronous packet until an ACK packet is received. The sent packet will be counted as used bandwidth if it arrives at its destination (i.e., only the ACK was lost).

• topology discovery: The `create-ieee1394-lan` function ensures that the topology is a tree, so there is no need to discover it. Since we do not support topology changes there is no need to re-discover the topology..

• handling topology changes and disconnects: This capability is ignored by the implementation.

• transactions and delayed ACKs: This simulation of the IEEE 1394 protocol does not support transactions, or the sending of actual data. Since applications cannot request to read data from another node (a limitation of our model), there is no need to support transactions.

• isochronous resource management: This capability is ignored by the implementation. The user must ensure that bandwidth demands do not exceed 80%. The implementation does not use channels numbers for isochronous packets (they are addressed the same as asynchronous packets). One reason for this is so that bandwidth utilization figures are accurate (otherwise we may count the same packet several times if it was being multi-cast.

3.2.2 - topology and states

The topological layout of the previously discussed classes, and the state transition diagram for the IEEE 1394 protocol are explained below.

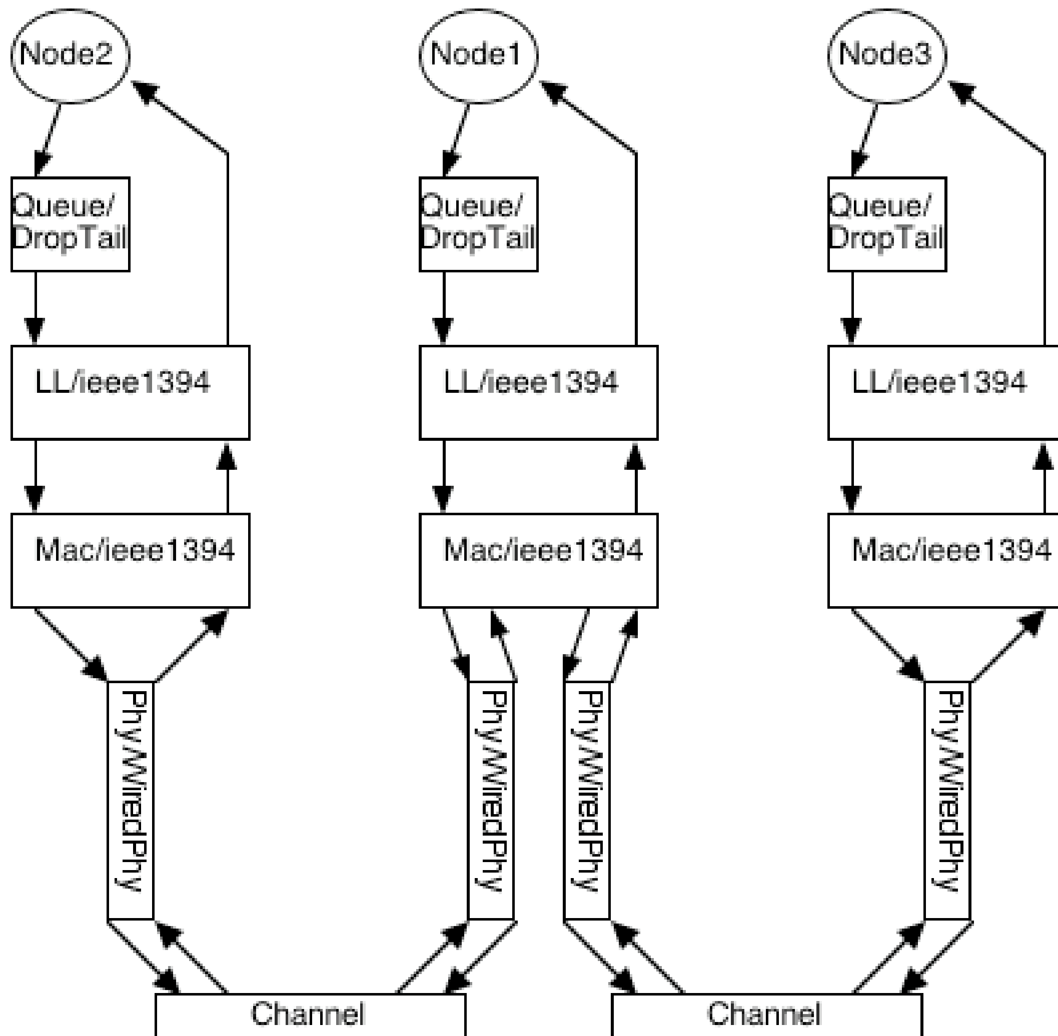

Figure 1 - IEEE1394 LanNodes connected together via Channels and Phy/WiredPhy.

Unlike ns2 LanNodes, which share a single Channel between all nodes and a single Phy connection for each Mac instance, the ieee1394 LanNodes have one Channel connecting two nodes and separate Phy/WiredPhy instances connecting each end.

The ieee1394 LanNode is created by the `create_ieee1394Node_fromNode` function in the `firewire-lan.tcl` source file. The ieee1394 LanNodes are connected together by the `connect_ieee1394Nodes` function, which adds the Phy/WiredPhy and Channel instances and connects the parent node to the child node. A ieee1394 LanNode which was never connected as a child is considered the root node.

The Mac/ieee1394 instance shown in Figure 1 is actually a composite of several other object instances. Each Phy/WiredPhy object is connected to the Mac via a `ieee1394MacHandlerRecv` receiver object which handles the timing and delivery of incoming packets. Outgoing packets are sent directly to the Phy/WiredPhy object, via this receiver (which doesn't participate further in their delivery).

The Phy/WiredPhy object determines the bandwidth of the connection and the Channel determines the delay time (set to 100Mbs and 0.1 μs respectively in `firewire-lan.tcl`). Together these determine packet transmission and arrival times.

The `ieee1394Mac` object also uses ten `ieee1394MacState` type objects to represent the finite automata that determines the Mac level behaviour.

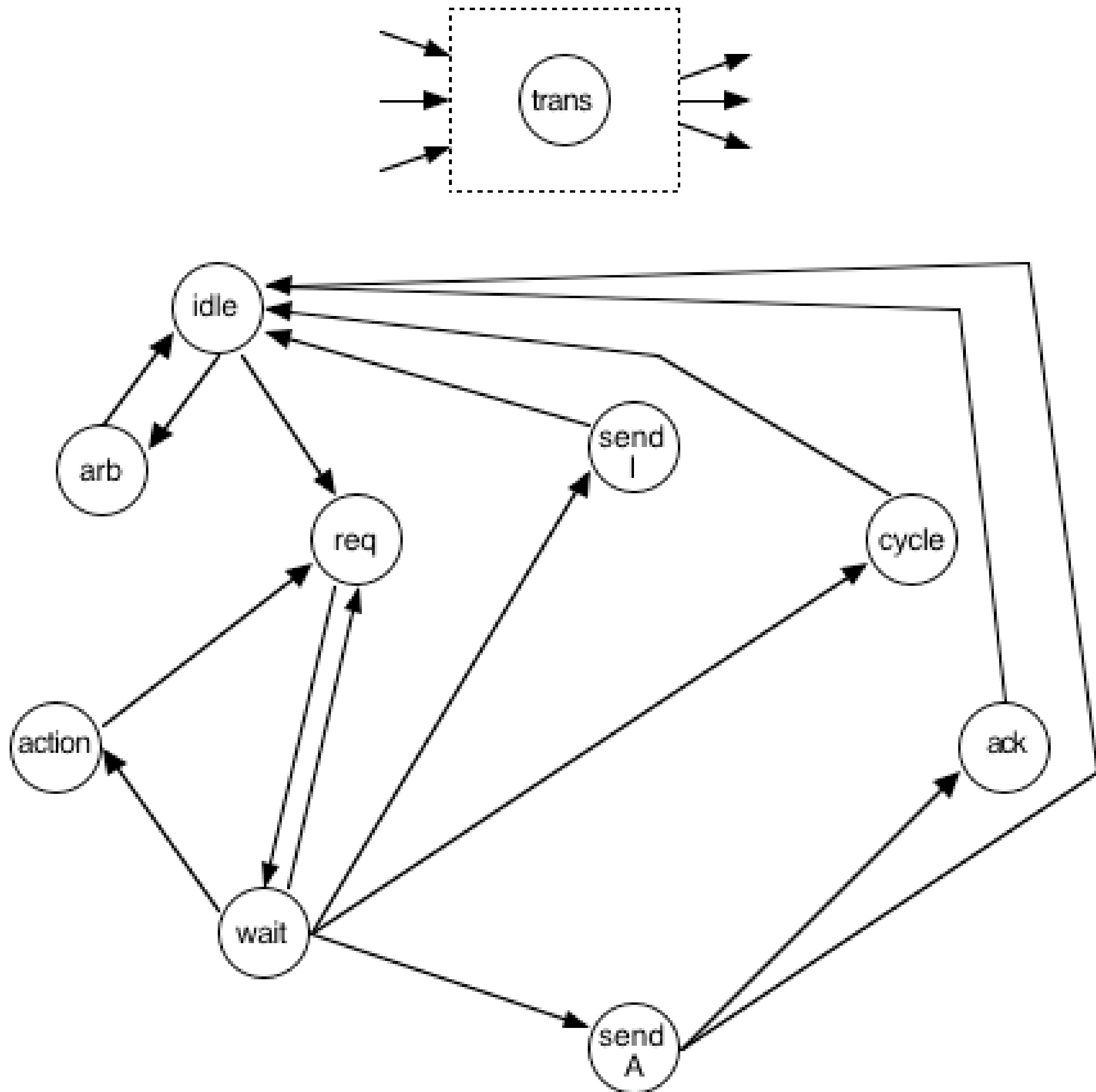

Figure 2 - Finite Automata State Space for Mac Layer

A simplified representation of the finite automata for the ieee1394Mac layer appears in Figure 2. **trans** is a special transition node used throughout the finite automata, and is shown separately for clarity, and to indicate that any node can transition to **trans**, and

trans can transition to any other node (see below for details).

The automata states are:

idle - corresponds to `ieee1394MacStateIdle`: This corresponds to the standard state for nodes which have no packets to send. It broadcasts all incoming packets as appropriate. If the node is the root node, then this state handles arbitration requests, granting them to the first request it receives. When granting an arbitration request, this state transitions to **arb** to wait for the grant to propagate fully (in this way the idle state will only grant to the first request it receives). If this state receives an async data packet for this node, it sends it up to the link layer, and broadcasts an ACK packet. If there is a data packet to send, this state transitions to **req**.

req - corresponds to `ieee1394MacStateWaitToRequestArb`: This state behaves much like **idle**, except that any incoming request is converted into a request for this node it can send a packet during the current fairness interval or isochronous cycle. The transition after sending on a request is to the **wait** state.

wait - corresponds to `ieee1394MacStateWaitForGrantArb`: This state waits to receive a arbitration grant packet, and broadcasts any grant packets it receives. If it receives one for this node (i.e., it won the arbitration), then it transitions to the appropriate sending state: **send A** for asynchronous data, **send I** for isochronous data, and **cycle** for the cycle restart packet. If it receives a grant for another node, it transitions to the **action** state to await the completion of the sub-action. Receiving any timing gap event, causes a transition to the **req** state.

arb - corresponds to `ieee1394MacStateWaitForArbCompletion`: This state ignores arbitration requests, but transitions back to the **idle** state after the grant packet is delivered, or after any timing gap event.

action - corresponds to `ieee1394MacStateWaitForActionCompletion`: This state waits to receive the last packet in a sub-action, before transitioning back to the **req** state again. Any timeouts also cause a transition back to the **req** state. Like the **idle** state, this state rebroadcasts all packets, send incoming packets for this node up to the link layer, and responds with an ACK packet where appropriate.

send A - corresponds to `ieee1394MacStateAsyncSendData`: This state simply sends out the pending data packet. It transitions to the **ack** state to await the response.

ack - corresponds to `ieee1394MacStateWaitForAck`: This state awaits an ACK packet (it will be for this node only) and re-broadcasts it. It transitions to the **idle** state

after receipt of the acknowledge, or after any timing gap event.

send I - corresponds to `ieee1394MacStateIsochSendData`: This state simply sends out the pending data packet, then transitions to the **idle** state.

cycle - corresponds to `ieee1394MacStateCycleRestartSend`: This state simply sends out a cycle restart packet, then transitions to the **idle** state.

trans - corresponds to `ieee1394MacStateTransitionSend`: This is a special interim state that is used to transition between two states, where a packet send occurs during the transition. The destination state is specified before transitioning to the **trans** state. When the packet send is complete, or canceled, the secondary transition to the previously specified state occurs.

4 - Results:

4.1 - expected results

I expect that the bandwidth utilization of both transfer modes will be similar since both modes are arbitrated in much the same way, and the un-utilized bandwidth of idle time is governed by the protocol timing constraints.

Asynchronous transfers can utilize all bandwidth, while isochronous transfers only have 80% of the available bandwidth (this threshold can be adjusted for testing purposes). However, asynchronous transfers have the required ACK packet and the longer idle gap times than isochronous transfers. Also, isochronous packets can be twice as long as asynchronous packets. In these tests, however we will not limit isochronous transfers to 80% of the bandwidth.

The results can be calculated analytically using the following values for the timers, bandwidth, delay, and packet sizes:

subaction gap time (SAGT): 0.00001s
isochronous gap time (IGT): 0.000005s
fairness interval gap time (FIGT): 0.00002s
cycle interval (CI): 0.000125

bandwidth (B): 100,000,000 bps
delay (D): 0.0000001 s

async packet size (APS): 532 bytes (includes header) = 4,256 bits
isoch packet size (IPS): 1044 bytes (includes header) = 8,352 bits
request packet size (RPS): 8 bits. (we are simulating a signal not a packet).

other control packets size (OPS): 64 bytes (includes header) = 512 bits.

Let N be the number of nodes, and H be the height of the topology (in links, not nodes). On average, the time to send a single async packet (SSAP) is:

$SSAP = [SAGT]_1 + [H*D + RPS/B]_2 + [H*D + OPS/B]_3 + [H*D + APS/B]_4 + [H*D + OPS/B]_5$

Where 1 is the time to wait until we can request, 2 is the time to send up a request, 3 is the time to broadcast a grant, 4 is the time to send the data packet, and 5 is the time to send the acknowledge.

The time to send N async packets (SNAP) is:

$SNAP = [N * SSAP]_1 + [FIGT]_2$

Where 1 is the time to send N packets, and 2 is the time to wait until we can send N more.

For an 7 node tree with height 2 (in links):

{0.00001 + 0.0000002 + 8/100,000,000 + 0.0000002 + 512/100,000,000 + 0.0000002 + 4,256/100,000,000 + 0.0000002 + 512/100,000,000 =

0.00001 + 0.0000002 + 0.00000008 + 0.0000002 + 0.00000512 + 0.0000002 + 0.00004256 + 0.0000002 + 0.00000512 =

0.00001028 + 0.00000532 + 0.00004276 + 0.00000532} =

SSAP = 0.00006368 s

SNAP = 0.00046576 s

So, we can send 3,548 bytes of data (no headers), every 0.00046576 s, so we can transfer 60,941,257 bits on a 100 Mbs link. As the number of nodes increases, the bandwidth utilization also increases, since there are fewer fairness interval gaps.

On average, the time to send a single isoch packet (SSIP) is:

$SSIP = [IGT]_1 + [H*D + RPS/B]_2 + [H*D + OPS/B]_3 + [H*D + IPS/B]_4$

Where 1 is the time to wait until we can request, 2 is the time to send up a request, 3 is the time to broadcast a grant, and 4 is the time to send the data packet.

The time to send N isoch packets (SNIP) is:

$SNIP = [N * SSIP]_1 + [max(CI - (N * SSIP), 0)]_2$

Where 1 is the time to send N packets, and 2 is the time to wait until we can send N more (if we go over the cycle interval time, then we can restart without delay).

For an 7 node tree with height 2 (in links):

{0.000005 + 0.0000002 + 8/100,000,000 + 0.0000002 + 512/100,000,000 + 0.0000002 + 8,352/100,000,000 =

0.000005 + 0.0000002 + 0.00000008 + 0.0000002 + 0.00000512 + 0.0000002 + 0.00008352 =

0.00000528 + 0.00000532 + 0.00008552} =

SSIP = 0.00009612 s

SNIP = 0.00067284 s

So, we can send 7,168 bytes of data (no headers), every 0.00067284 s, so we can transfer 85,226,799 bits on a 100 Mbs link.

4.1 - topologies

All topologies are binary trees (i.e., each node has at most two children). Only one non leaf node has one child.

The following test are distinguished by the number of nodes in the tree.

4.2 - asynchronous

The following list shows the max bandwidth utilization for 2, 3, 5, 7, and 12 nodes with a packet size of 512.

2 nodes ==> 60.071936 Mbps
3 nodes ==> 60.854272 Mbps
5 nodes ==> 65.536000 Mbps

7 nodes ==> 65.536000 Mbps
12 nodes ==> 65.540096 Mbps

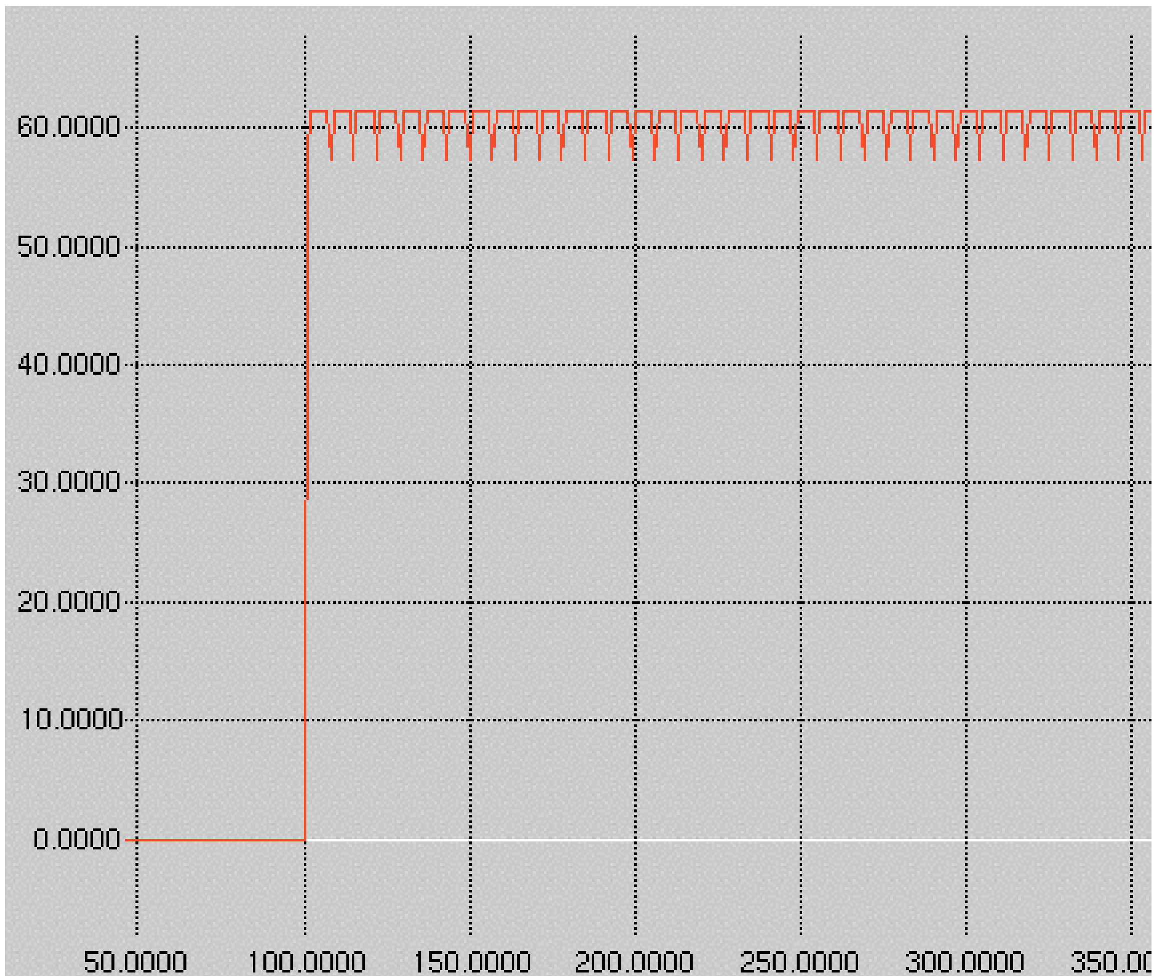

Figure 3 - Graph of bandwidth utilization from 0.05 secs to 0.350 secs, for 3 asynchronous nodes, with a 0.001 sec sampling interval.

4.3 - isochronous

The following list shows the max bandwidth utilization for 2, 3, 5, 7, and 12 nodes with a packet size of 1024.

2 nodes ==> 65.536000 Mbps
3 nodes ==> 65.536000 Mbps
5 nodes ==> 65.536000 Mbps
7 nodes ==> 87.384064 Mbps
12 nodes ==> 78.643200 Mbps

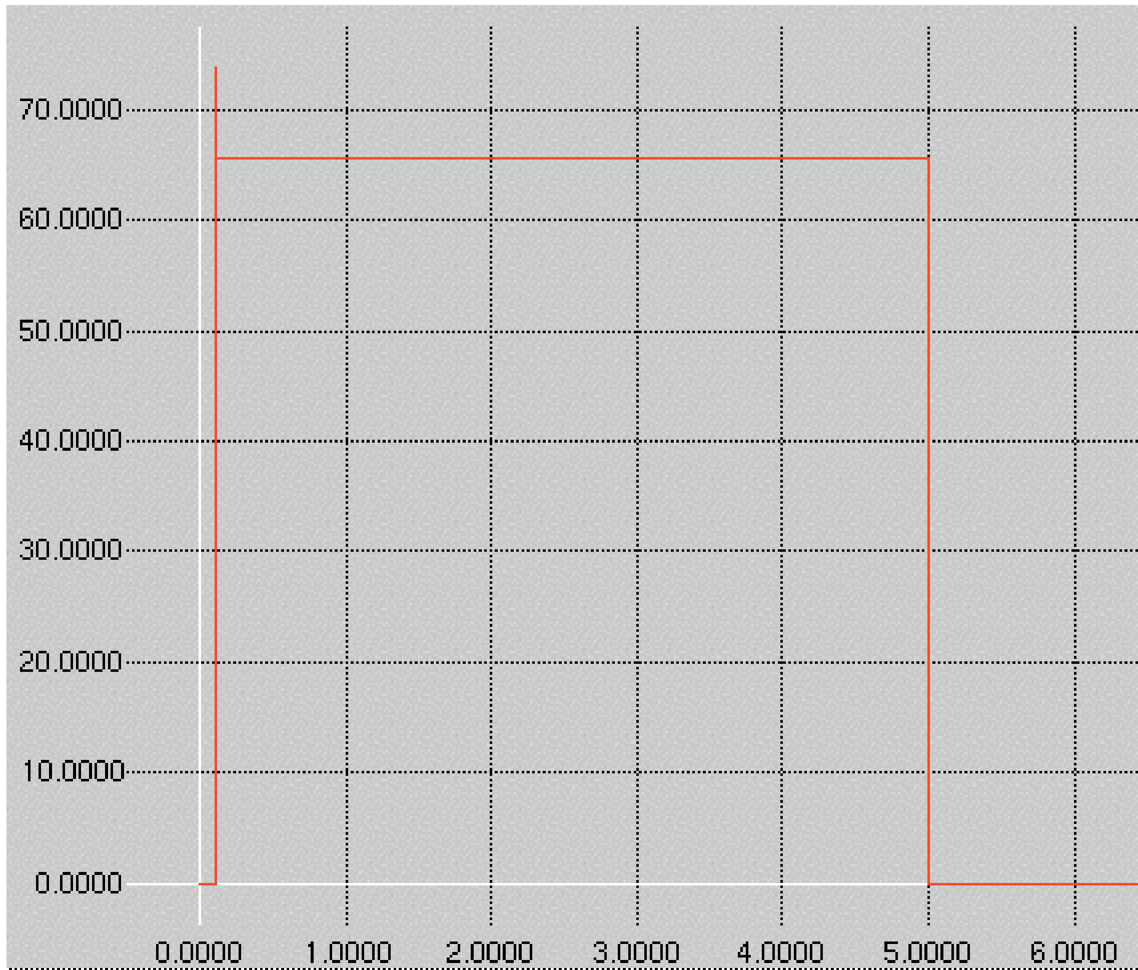

Figure 5 - Graph of bandwidth utilization (Y axis in Mbps) for a 2 isochronous node network, from 0 secs to 6.0 secs (X axis).

4.4 - both

The following list shows the max bandwidth utilization for 2, 3, 5, 7, and 8 nodes with half the nodes transmitting asynchronous packets of size 512 and the other half transmitting isochronous packets of size of 1024.

2 nodes ==> 57.344000 Mbps
3 nodes ==> 65.536000 Mbps
5 nodes ==> 65.536000 Mbps
7 nodes ==> 65.536000 Mbps
8 nodes ==> 65.536000 Mbps

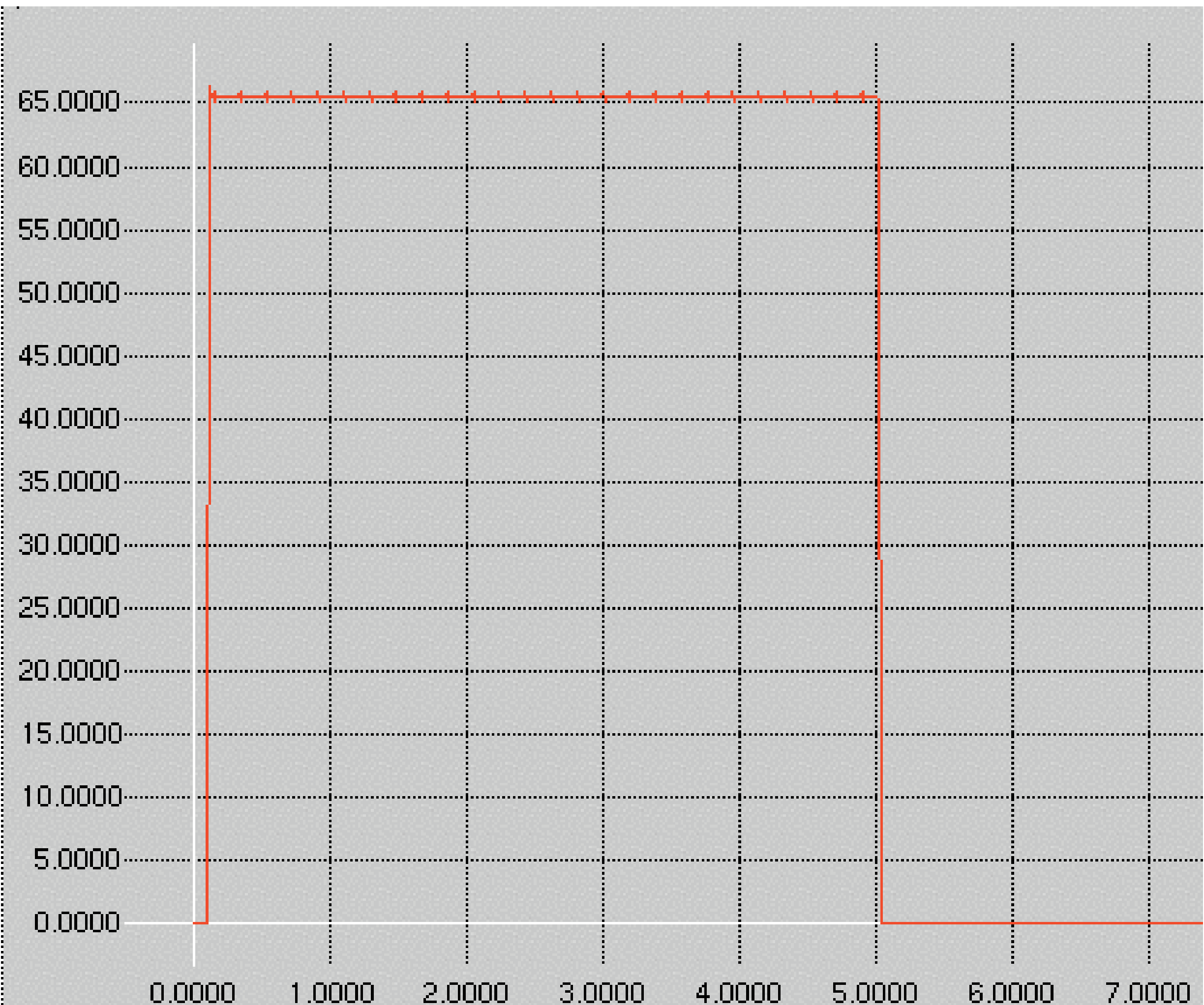

Figure 5 - Graph of bandwidth utilization (Y) in Mbps .vs. time in seconds (X) for an 8 node lan with half isochronous and half asynchronous agents. The sampling interval was 0.01 seconds.

4.5 - analysis

As expected, isochronous transfers more efficiently use the available bandwidth than asynchronous transfers. In part this may be attributed to not limiting the isochronous transfers to 80% of the bandwidth as the protocol specifies.

In both asynchronous and isochronous transfers, we see that the more nodes involved in transfers, the greater the bandwidth utilization. This predominately appears for asynchronous transfers, since the fairness interval includes a single transmission from each node, and the more nodes, the longer the fairness interval, so the fairness interval gap occurs less often. Isochronous transfer times also show some increase, though this may result from the fixed isochronous fairness interval, the cycle time. The cause of the anomalous dip in bandwidth utilization is not known. Contention issues

should not play a part of this; however it is possible that some nodes are never winning the arbitrations, because the cycle restarts before they have a chance.

The previously calculated bandwidth utilizations for 7 asynchronous nodes at 60.941257 Mbps and 7 isochronous nodes at 85.226799 Mbps, compare well with the simulated values of 65.536000 Mbps and 87.384064 Mbps respectively.

5 - Conclusion:

IEEE 1394 provides a third type contention management: the arbitration cycle. It removes the possibility of collision, unlike Ethernet, but like TokenRing. It can easily regenerate after a node failure, since the key to arbitration is in the idle times.

IEEE 1394 also support two distinct transfer modes, asynchronous and isochronous, each well suited to particular applications.

I have implemented a basic simulation of the IEEE 1394 protocol, including these two transmission modes, in ns2, a network simulator traditionally used for IP network simulation.

The results indicate that both transfer modes encounter significant overhead, but that the isochronous transfer mode is more efficient than the asynchronous transfer mode. The bandwidth utilization improves slightly with an increasing number of asynchronous nodes, since the fairness interval increases, and the relatively lengthy fairness interval gap occurs less often.

For future work it might be interesting to extend the isochronous protocol at the link or transaction layer such that acknowledges are also transmitted isochronously. In this way we could use the shorter isochronous idle gap to initiate arbitration sooner.

6 - References:

• FireWire System Architecture, 2nd Edition (IEEE 1394a). Anderson, Don. MindShare Inc. Addison-Wesley Inc. ISBN: 0-201-48535-4.

- P1394b Draft Standard for a High Performance Serial Bus: p1394b1-33.pdf
- What's New About 1394b: ppt1.pdf
- Isochronous Resource Management: br062r00.pdf
- New Technology for 1394 (overview): 1394ABoverview.pdf
- IEEE 1394-1995 High Performance Serial Bus (overview): 1394overview.pdf

- NS-2 Documentation (for implementation purposes): ns_doc.pdf

• Isochronous bandwidth utilization improvement in distributed queue dual bus-based personal communication networks Yang Y.; Lai T.-H.; Liu M.-T. Computer Communications, 15 October 1998, vol. 21, no. 16, pp. 1420-1433(14) Elsevier Science.

- Some additional sources of Information are as follows (tested 14/02/2002):
- The 1394 trade association:
  http://www.1394ta.org/
- 1394 specifications from 1394 trade association:
  http://www.1394ta.org/Technology/Specifications/index.htm
  http://www.1394ta.org/Technology/Specifications/specifications.htm
- The meeting notes of the IEEE committee for 1394a:
  http://grouper.ieee.org/groups/1394/1/Documents/
- The meeting notes of the IEEE committee for 1394b:
  http://www.zayante.com/p1394b/
- Some overviews of 1394:
  http://www.iol.unh.edu/training/1394.html
- NS Tutorials (including extending):
  http://nile.wpi.edu/NS/
  http://www.isi.edu/nsnam/ns/tutorial/index.html

6 - Appendix:

6.1 - diffs for `Makefile.in`:

```
***************
*** 190,196 ****
      parentnode.o \
      basetrace.o \
      simulator.o \
-     ieee-1394/mac-1394.o ieee-1394/packet-1394.o ieee-1394/agent-1394.o ieee-1394/link-1394.o\
      @V_STLOBJ@
```

Note: The hyphen (-) indicates the lines added to `Makefile.in`, and all other lines are only shown for context.

6.2 - diffs for `packet.h`:

```
***************
*** 142,151 ****
        PT_PUSHBACK,

        // insert new packet types here
-
-         // ieee 1394 packet
-         PT_IEEE1394,
-
```

```
        PT_NTYPE // This MUST be the LAST one
  };

--- 142,147 ----
***************
*** 219,227 ****

                //pushback
                name_[PT_PUSHBACK] = "pushback";
-
-                 //ieee 1394
-                 name_[PT_IEEE1394] = "ieee1394";

                name_[PT_NTYPE]= "undefined";
        }
--- 215,220 ----
```

Note: The hyphen (-) indicates the lines added to `packet.h` , and all other lines are only shown for context.

6.3 - diffs for `ns-default.tcl`:

```
***************
*** 908,917 ****
  #RtModule/Hier set classifier_ [new Classifier/Hier]
  #RtModule/Manual set classifier_ [new Classifier/Hash/Dest 2]
  #RtModule/VC set classifier_ [new Classifier/Virtual]
-
- # IEEE 1394
- Agent/ieee1394/Async set packetSize_ 512
- Agent/ieee1394/Isoch set packetSize_ 1024
- Mac/ieee1394 set trace_ false
-
-
--- 908,910 ----
```

Note: The hyphen (-) indicates the lines added to `ns-default.tcl` , and all other lines are only shown for context.

6.4 - diffs for `ns-packet.tcl`:

```
***************
*** 144,150 ****
        GAF
        UMP
        PUSHBACK
-         ieee1394
        NV
  } {
        add-packet-header $prot
--- 144,149 ----
```

Note: The hyphen (-) indicates the lines added to `ns-packet.tcl` , and all other lines

are only shown for context.

6.5 - source for `firewire-lan.tcl`:

```
set ns [new Simulator]

proc finish {} {
        global ns nf tf bwf nf_name tf_name bwf_name
        $ns flush-trace
        close $nf
        close $tf
        close $bwf
        exec more $bwf_name
#        exec  ~/Development/NS/ns-allinone-2.1b8a/nam-1.0a10/nam $nf_name &
#               exec ~/Development/NS/ns-allinone-2.1b8a/xgraph-12.1/xgraph $bwf_name -geometry
800x400 &
        exit 0
}

proc create-trace-files {num_nodes pkt_size delay prefix trace} {
        global ns nf tf bwf nf_name tf_name bwf_name

    set nf_name "results/$prefix-$num_nodes-$pkt_size-$delay-$trace.nam"
    set tf_name "results/$prefix-$num_nodes-$pkt_size-$delay-$trace.tr"
    set bwf_name "results/$prefix-$num_nodes-$pkt_size-$delay.bw"
    set nf [open $nf_name w]
    set tf [open $tf_name w]
    set bwf [open $bwf_name w]
#    $ns namtrace-all $nf
#    $ns trace-all $tf
}

proc create_ieee1394LanNode {bw delay trace} {
    global ns

    set lan [eval new LanNode $ns -bw $bw -delay $delay \
                -llType LL/ieee1394 -ifqType Queue/DropTail -macType Mac/ieee1394 \
                -chanType Channel -phyType Phy/WiredPhy -mactrace $trace]

 return $lan
}

proc create_ieee1394Node_fromNode {lan node bw delay trace} {
    global ns
    $ns instvar link_

    set nif [new LanIface $node $lan \
               -ifqType Queue/DropTail \
               -llType  LL/ieee1394 \
               -macType Mac/ieee1394 \
               -phyType Phy/WiredPhy \
               -mactrace $trace ]

    set ll [$nif set ll_]
    $ll set delay_ $delay
#    $ll varp [$lan set varp_]
#    [$lan set varp_] mac-addr [[$nif set node_] id] [[$nif set mac_] id]
```

```
#    puts "LL downtarget = [[$ll down-target] info vars]"

    set phy [$nif set phy_]
    $phy node $node
    $phy set bandwidth_ $bw

    $lan set lanIface_($node) $nif
    $node add-neighbor $lan

    set vlinkcost [expr [$lan set cost_] / 2.0]

    set nid [$node id]
    set lid [$lan set id_]

    set link1 [new Vlink $ns $lan $node $lan $bw 0]
    set link2 [new Vlink $ns $lan $lan $node $bw 0]

    set link_($nid:$lid) $link1
    set link_($lid:$nid) $link2

    $node add-oif [$link_($nid:$lid) head]  $link_($nid:$lid)
    $node add-iif [[$nif set iface_] label] $link_($lid:$nid)
    [$link_($nid:$lid) head] set link_ $link_($nid:$lid)

    $link_($nid:$lid) queue [$nif set ifq_]
    $link_($lid:$nid) queue [$nif set ifq_]

    $link_($nid:$lid) set iif_ [$nif set iface_]
    $link_($lid:$nid) set iif_ [$nif set iface_]

    $link_($nid:$lid) cost $vlinkcost
    $link_($lid:$nid) cost $vlinkcost

    return $nif
}

proc connect_ieee1394Nodes {parent child bw delay} {

    set channel [new Channel]
    $channel set delay_ $delay

    #puts "[$channel set delay_]"

    set mac_p [$parent set mac_]
    set mac_c [$child set mac_]

    set phy_p [new Phy/WiredPhy]
    set phy_c [new Phy/WiredPhy]

    set recv_p [new Mac/ieee1394/HandlerRecv $mac_p]
    $mac_p add-phy $phy_p $recv_p 0
    $phy_p up-target $recv_p
    $mac_p netif $phy_p

    set recv_c [new Mac/ieee1394/HandlerRecv $mac_c]
    $mac_c add-phy $phy_c $recv_c 1
    $phy_c up-target $recv_c
```

```
    $mac_c netif $phy_c

#    set phy_p [$parent set phy_]
#    set phy_c [$child set phy_]

    $phy_p node [$parent set node_]
    $phy_c node [$child set node_]

    $phy_p channel $channel
    $phy_c channel $channel

    $phy_p set bandwidth_ $bw
    $phy_c set bandwidth_ $bw

    $channel addif $phy_p
    $channel addif $phy_c
}

proc create-ieee1394-lan {num_nodes start_time end_time interval pkt_size delay trace prefix} {
    global ns sink

    set bw 100Mb

    set lan [eval create_ieee1394LanNode $bw $delay $trace]

    # create nodes
    for {set i 0} {$i < $num_nodes} {incr i} {
               set node($i) [$ns node]
    }

    # create iflan nodes
    for {set i 0} {$i < $num_nodes} {incr i} {
               set iface($i) [eval create_ieee1394Node_fromNode $lan $node($i) $bw $delay $trace]
    }

    # connect iflan nodes -- in a list
    set num_children 2
    for {set i 1} {$i < $num_nodes} {incr i} {
               connect_ieee1394Nodes $iface([expr int(floor(($i - 1) / $num_children))])
$iface($i) $bw $delay
    }

    # add receiver agents
    for {set i 0} {$i < $num_nodes} {incr i} {
        set sink($i) [new Agent/LossMonitor]
        $ns attach-agent $node($i) $sink($i)
    }

    # add agents and applications
    for {set i 0} {$i < $num_nodes} {incr i} {
        # create agent and attach it to node i

        if { [string equal $prefix "a"] } {
            set agt($i) [new Agent/ieee1394/Async]
        } elseif { [string equal $prefix "i"] } {
            set agt($i) [new Agent/ieee1394/Isoch]
        } elseif { [string equal $prefix "b"] } {
            if { [expr $i % 2] } {
```

```
                set agt($i) [new Agent/ieee1394/Async]
            } else {
                set agt($i) [new Agent/ieee1394/Isoch]
            }
        } else {
            puts "ERROR: prefix should be: [a, i, b]"
        }

        $ns attach-agent $node($i) $agt($i)

        # create CBR source and attach it to udp
        set app($i) [new Application/Traffic/CBR]
        $app($i) set packetSize_ $pkt_size
        $app($i) set interval_ $interval
        $app($i) attach-agent $agt($i)

        # connect agents
        set distance 1
        set j [expr (($i + $distance) % $num_nodes)]
        $ns connect $agt($i) $sink($j)

        $ns at $start_time "$app($i) start"
        $ns at $end_time "$app($i) stop"
    }
}

proc record-bandwidth {num_nodes} {
    global ns sink bwf

    set time_interval 1
    set bw0 0
    set np0 0

    for {set i 0} {$i < $num_nodes} {incr i} {

        # bytes received by sink
        set bw0 [expr $bw0 + [$sink($i) set bytes_]]
        set np0 [expr $np0 + [$sink($i) set npkts_]]

        #puts "bytes recv: $i bw:$bw0 np:$np0"

        # reset bytes_ values of traffic sinks
        $sink($i) set bytes_ 0
    }

    set now [$ns now]

    # calculate bandwidth (in MBit/s) and write it out
#    puts $bwf "$now [expr $bw0/$time_interval*8.0/1000000.0]"
    puts "$now [expr $bw0/$time_interval*8.0/1000000.0] $bw0 $np0"

    # re-schedule
    $ns at [expr $now+$time_interval] "record-bandwidth $num_nodes"
}

proc create-sim {num_nodes start_time end_time interval pkt_size delay prefix trace} {
    global ns
```

```
    $ns at 0.0 "record-bandwidth $num_nodes"

    create-trace-files $num_nodes $pkt_size $delay $prefix $trace
    create-ieee1394-lan $num_nodes $start_time $end_time $interval $pkt_size $delay $trace
$prefix
}

create-sim 6 0.1 5.0 0.0001 1024 0.1us "b" false

$ns at 10.0 "finish"

$ns run
```

`create-sim` is the function that creates the lan. The key parameters are: # of nodes, data chunk size (for the application), channel delay, and the start and end times for data transmission. It invokes `create-ieee1394-lan`, passing these arguments to it.

`create_ieee1394LanNode` creates a single LanNode which contains a LL/ieee1394 (link Layer) instance, a Mac/ieee1394 instance, and a Phy/WiredPhy instance. The LL and Mac are contected via a Queue/DropTail. Normally, a single LanNode is used to connect all the nodes in a LAN together; however, for our simulation there is one of the above LanNodes for each Firewire node in the LAN.

`create_ieee1394Node_fromNode` combines a given Node and LanNode into a single ieee1394 LanNode, connected via a Queue/DropTail interface.

`connect_ieee1394Nodes` takes two ieee1394 LanNodes, (a parent and a child) and connects them together with a Channel. The direction of the connection, parent to child, determines the tree structure of the network.

`create-ieee1394-lan` creates the actual ieee1394 Lan. It uses the helper function above to create the ieee1394 LanNodes, then adds the appropriate source agents, (either Agent/ieee1394/Async or Agent/ieee1394/Isoch), a data collecting Agent/LossMonitor sink agent, and a CBR (constant bit rate) Application/CBR to each node. The lan topology is a tree, with the `num_children` variable determining the branching factor.

6.6 - source for `packet-1394`:

```
/*
 *  packet-1394.h
 *
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 */

#ifndef __packet_1394_h__
#define __packet_1394_h__
```

```
#include "tclcl.h"
#include "packet.h"
#include "address.h"

struct hdr_ieee1394
{
 enum packet_type {ACK, ASYNC_PAK, ISOCH_PAK, REQ_ARB, GRANT_ARB, CYCLE_RESTART};

 packet_type   trans_code_;   // transaction code

 static int offset_;

 inline static int                &offset() { return offset_; }
 inline static hdr_ieee1394 *access(const Packet* p) {return (hdr_ieee1394*) p->access(offset_);}

 packet_type   &trans_code() {return trans_code_;}
};

class ieee1394Packet
{
 public:

 static Packet                *alloc_packet();

 protected:

 static int uidcnt_;
};

#endif

/*
 *  packet-1394.cpp
 *
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 */

#include "packet-1394.h"

int hdr_ieee1394::offset_;
int ieee1394Packet::uidcnt_ = 0;

static class ieee1394HeaderClass : public PacketHeaderClass
{
 public:

        ieee1394HeaderClass() : PacketHeaderClass("PacketHeader/ieee1394",sizeof(hdr_ieee1394))
        {
            bind_offset(&hdr_ieee1394::offset_);
        }

 void export_offsets()
 {
  field_offset("trans_code_", OFFSET(hdr_ieee1394, trans_code_));
 }
} class_ieee1394hdr;
```

```
Packet                          *ieee1394Packet::alloc_packet()
{Packet        *p = Packet::alloc();
 hdr_cmn       *ch = hdr_cmn::access(p);

 ch->uid() = uidcnt_++;
 ch->ptype() = PT_IEEE1394;
 ch->size() = 0;
 ch->timestamp() = Scheduler::instance().clock();
 ch->iface() = UNKN_IFACE.value(); // from packet.h
 ch->direction() = hdr_cmn::NONE;
 ch->ref_count() = 0;
 ch->error() = 0;

 return p;
}
```

Note: Unnecessary comments were removed from the code.

`hdr_ieee1394` declares the header information for ieee1394 packets. Other header information is shared with `hdr_cmn` (the common header) and `hdr_ip` (used for source and destination addresses).

`ieee1394Packet` is used to create Packets with the `hdr_ieee1394` information properly filled out.

6.7 - source for `agent-1394`:

```
/*
 *  agent-1394.h
 *
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 */

#ifndef __agent_1394_h__
#define __agent_1394_h__

#include "tclcl.h"
#include "agent.h"
#include "packet-1394.h"

class ieee1394Agent : public Agent
{
 public:

        ieee1394Agent();

 int                          command(int argc, const char*const* argv);

 virtual void sendmsg(int nbytes, const char *flags = 0);

 protected:
```

```
 virtual Packet               *init_packet(Packet *p);
};

class ieee1394AgentAsync : public ieee1394Agent
{
 public:

        ieee1394AgentAsync();

 protected:

 virtual Packet               *init_packet(Packet *p);
};

class ieee1394AgentIsoch : public ieee1394Agent
{
 public:

        ieee1394AgentIsoch();

 protected:

 virtual Packet               *init_packet(Packet *p);
};

#endif

/*
 *  agent-1394.cpp
 *
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 */

#include "agent-1394.h"
#include "ip.h"

static class ieee1394AgentAsyncClass : public TclClass
{
 public:

        ieee1394AgentAsyncClass() : TclClass("Agent/ieee1394/Async") {}

 TclObject     *create(int, const char*const*)
 {
  return (new ieee1394AgentAsync());
 }
} class_ieee1394AgentAsync;

static class ieee1394AgentIsochClass : public TclClass
{
 public:

        ieee1394AgentIsochClass() : TclClass("Agent/ieee1394/Isoch") {}

 TclObject     *create(int, const char*const*)
 {
  return (new ieee1394AgentIsoch());
```

```
 }
} class_ieee1394AgentIsoch;

//****************** ieee1394Agent ************************

ieee1394Agent::ieee1394Agent() :
    Agent(PT_IEEE1394)
{
 bind("packetSize_", &size_);
}

int                           ieee1394Agent::command(int argc, const char*const* argv)
{
  if (argc == 2)
  {
    if (strcmp(argv[1], "send_async") == 0)
    {Packet                   *pkt = init_packet(ieee1394Packet::alloc_packet());
     hdr_ieee1394             *hdr = hdr_ieee1394::access(pkt);

     hdr->trans_code_ = hdr_ieee1394::ASYNC_PAK;

     send(pkt, 0);

     printf("sending ASYNC_PAK...\n");

     return (TCL_OK);
    }
    if (strcmp(argv[1], "send_isoch") == 0)
    {Packet                   *pkt = init_packet(ieee1394Packet::alloc_packet());
     hdr_ieee1394             *hdr = hdr_ieee1394::access(pkt);

     hdr->trans_code_ = hdr_ieee1394::ISOCH_PAK;

     send(pkt, 0);

     printf("sending ISOCH_PAK...\n");

     return (TCL_OK);
    }
  }

  return (Agent::command(argc, argv));
}

// based on Agent/UDP
void                          ieee1394Agent::sendmsg(int nbytes, const char* flags)
{double        local_time = Scheduler::instance().clock();
 int                          n;

 if (size_)
  n = nbytes / size_;
 else
  printf("Error: ieee1394 Packet size = 0\n");

 if (nbytes == -1)
 {
  printf("Error:  sendmsg() for UDP should not be -1\n");
  return;
```

```
 }

 while (n-- > 0)
 {Packet                      *pkt = init_packet(ieee1394Packet::alloc_packet());

  target_->recv(pkt);
 }

 n = nbytes % size_;
 if (n > 0)
 {Packet                      *pkt = init_packet(ieee1394Packet::alloc_packet());

  hdr_cmn::access(pkt)->size() = n;
  target_->recv(pkt);
 }
 idle();
}

Packet                        *ieee1394Agent::init_packet(Packet *p)
{hdr_cmn       *ch = hdr_cmn::access(p);
 hdr_ip        *iph = hdr_ip::access(p);

 ch->size() = size_;

 iph->saddr() = here_.addr_;
 iph->sport() = here_.port_;
 iph->daddr() = dst_.addr_;
 iph->dport() = dst_.port_;

 return p;
}

//***************** ieee1394AgentAsync ************************

ieee1394AgentAsync::ieee1394AgentAsync() :
    ieee1394Agent()
{
}

Packet         *ieee1394AgentAsync::init_packet(Packet *p)
{Packet        *p2 = ieee1394Agent::init_packet(p);

 hdr_ieee1394::access(p2)->trans_code_ = hdr_ieee1394::ASYNC_PAK;

 return p2;
}

//***************** ieee1394AgentIsoch ************************

ieee1394AgentIsoch::ieee1394AgentIsoch() :
    ieee1394Agent()
{
}

Packet                        *ieee1394AgentIsoch::init_packet(Packet *p)
{Packet        *p2 = ieee1394Agent::init_packet(p);

 hdr_ieee1394::access(p2)->trans_code_ = hdr_ieee1394::ISOCH_PAK;
```

```
 return p2;
}
```

Note: Unnecessary comments were removed from the code.

`ieee1394Agent` is a base class for the ieee1394 agents. It implements the `sendmsg` member function, which is responsible for creating a packet in response to an Application request, and sending it to the link layer. The `ieee1394AgentAsync` and `ieee1394AgentIsoch`, classes override the `init_packet` function to create the appropriate type of packet.

6.8 - source for `link-1394`:

```
/*
 *  link-1394.h
 *
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 */

#ifndef __link_1394_h__
#define __link_1394_h__

#include "ll.h"
#include "tclcl.h"
#include "node.h"
#include "channel.h"
#include "phy.h"
#include "queue.h"
#include "net-interface.h"
#include "timer-handler.h"

class ieee1394LL : public LL
{
 public:

        ieee1394LL();

 virtual void  sendDown(Packet* p);
 virtual void  sendUp(Packet* p);
 virtual void  recv(Packet* p, Handler* h);

 Channel       *channel(); // helper

 protected:

};

#endif

/*
 *  link-1394.cpp
 *
```

```
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 */

#include "link-1394.h"
#include "mac-1394.h"
#include "packet-1394.h"

static class ieee1394LLClass : public TclClass
{
 public:

        ieee1394LLClass() : TclClass("LL/ieee1394") {}

 TclObject     *create(int, const char*const*)
 {
  return (new ieee1394LL());
 }
} class_ieee1394LL;


ieee1394LL::ieee1394LL() :
    LL()
{
}

void                          ieee1394LL::sendDown(Packet* p)
{hdr_cmn       *ch = HDR_CMN(p);
 hdr_ll        *llh = HDR_LL(p);
 hdr_mac       *mh = HDR_MAC(p);
 hdr_ieee1394  *hdr = (hdr_ieee1394*)p->access(hdr_ieee1394::offset_);
 int                          tx = 0;

 {Scheduler    &s = Scheduler::instance();

  s.schedule(downtarget_, p, delay_);
 }
}

void                          ieee1394LL::sendUp(Packet* p)
{Scheduler     &s = Scheduler::instance();

 if (hdr_cmn::access(p)->error() > 0)
 {
  printf("ieee1394LL: send up -- drop \n");
  drop(p);
 }
 else
  s.schedule(uptarget_, p, delay_);
}

void                          ieee1394LL::recv(Packet *pkt,Handler *)
{hdr_cmn *ch = HDR_CMN(pkt);

 assert(initialized());

 if(pkt->incoming)
  pkt->incoming = 0;
```

```
 // If direction = UP, then pass it up, else, set direction to DOWN and pass it down the stack
 if(ch->direction() == hdr_cmn::UP)
 {
  if (uptarget_)
  {
   sendUp(pkt);
  }
  else
   drop(pkt);
 }
 else
 {
  ch->direction() = hdr_cmn::DOWN;
  sendDown(pkt);
 }
}

Channel                       *ieee1394LL::channel()
{Phy           *phy_ = (Phy*) mac_->downtarget();

 return phy_->channel();
}
```

Note: Unnecessary comments were removed from the code.

`ieee1394Link` is the simple ieee1394 LL (link layer) class which schedules packet delivery between the Agent and the Mac layer.

6.9 - source for `mac-1394`:

```
/*
 *  mac-1394.h
 *
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 *  Based on mac-802_3 from ns2
 *
 */

#ifndef __mac_1394_h__
#define __mac_1394_h__

#include <assert.h>
#include "mac.h"

#define IEEE_1394_MAXFRAME    1518                          // bytes
#define IEEE_1394_MINFRAME    64                            // bytes
#define IEEE_1394_HDR_LEN     20                            // bytes
#define IEEE_1394_ARB_SIZE    1                             // bytes

class ieee1394Mac;

class ieee1394MacHandler : public Handler
{
```

```
 public:

              ieee1394MacHandler(ieee1394Mac *m) : mac(m) {};

 virtual void                handle(Event *e) = 0;
 virtual inline void         cancel();
 double                      expire() { return intr.time_; }

 protected:

 ieee1394Mac                 *mac;
 Event                       intr;
};

/**
* Invoked to simulate the sending delay time between the first bit of a packet
* and the last bit of the packet.
*
* schedule(p,t)              just started sending packet p, so delay for time t.
* handle(e)                  the packet from the last schedule just finished transmitting, so
*                            notify the associated mac that the transmit was completed.
* busy()                     true if a packet is in transit.
*/
class ieee1394MacHandlerSend : public ieee1394MacHandler
{
 public:
              ieee1394MacHandlerSend(ieee1394Mac *m);

 virtual void                handle(Event *e);
 virtual void                schedule(Packet *p, double t);
 virtual void                cancel();
 virtual bool                busy() const { return p_ != NULL;};

 virtual void                setDelay(bool d) { delay = d;};

 virtual const Packet        *packet() const { return p_; }

 protected:

 class delayedCompleted : public ieee1394MacHandler
 {
  public:

              delayedCompleted(ieee1394Mac *m);

  virtual void                              handle(Event *e);
  virtual void                              schedule();
 };

 class delayedCanceled : public ieee1394MacHandler
 {
  public:

              delayedCanceled(ieee1394Mac *m);

  virtual void                              handle(Event *e);
  virtual void                              schedule();
 };
```

```
 Packet                       *p_;

 delayedCompleted             completed;
 delayedCanceled              canceled;

 bool                         delay;
};

/**
* Invoked to simulate the delay time between the arrival of the first bit of a packet
* and the last bit of the packet.  Also keeps track of the PHY connectors.
*
* schedule(p,t)               packet p just started arriving, so delay for time t.
* handle(e)                   the packet from the last schedule just finished arriving, so
*                             notify the associated mac that the receive was completed.
* busy()                      true if a packet is in transit.
* is_downtarget_to_parent()   true if PHY target goes toward root.
*/
class ieee1394MacHandlerRecv : public ieee1394MacHandler, public NsObject
{
 friend class ieee1394Mac;

 public:
              ieee1394MacHandlerRecv(ieee1394Mac *m);

 virtual int                  command(int argc, const char*const* argv);
 virtual void                 handle(Event *e);
 virtual void                 schedule(Packet *p, double t);
 virtual void                 cancel();
 virtual bool                 busy() const { return p_ != NULL;};

 virtual void                 setDelay(bool d) { delay = d;};

 virtual const Packet         *packet() const { return p_; }

 virtual NsObject             *downtarget() {return downtarget_;};

 // for receiving from downtarget_ only, sending should be done directly via downtarget_
 virtual void                 recv(Packet *p, Handler *callback = 0);

 virtual bool                 is_downtarget_to_parent() const { return is_route_to_parent; };

 virtual void                 deliverPacket(Packet *p);

 protected:

 class delayedCompleted : public ieee1394MacHandler
 {
  public:

              delayedCompleted(ieee1394Mac *m, ieee1394MacHandlerRecv *r);

  virtual void                            handle(Event *e);
  virtual void                            cancel();
  virtual void                            schedule(Packet *p);

  protected:
```

```
  ieee1394MacHandlerRecv      *recv;
  Packet                      *pkt;
 };

 Packet        *p_;
 NsObject      *downtarget_;
 bool          is_route_to_parent; // true if the downtarget_ goes towards the parent.

 delayedCompleted             completed;
 bool                         delay;
};

/**
* Manages recurring cycle_restart interupt.  .
*
* schedule(t)  set interupt to occur in time t.
* handle(e)    a cycle_restart is due -- automatically reschedules if mac is the root.
*/
class ieee1394MacHandlerCycleTimer : public ieee1394MacHandler
{
 public:
               ieee1394MacHandlerCycleTimer(ieee1394Mac *m,double t = 0.00125) :
ieee1394MacHandler(m), time_(t) {};

 virtual void                 handle(Event *e);
 virtual void                 schedule(double t);
 virtual void                 schedule();

 protected:

 double        time_;
};

/**
* Determines when gaps of a pre-determined length have occured.
*
* schedule(t)                 set interupt to occur in time t.
* handle(e)                   a gap has occured.
* cancel()                    a transmission was seen, so cancel gap timing
*/
class ieee1394MacHandlerSubactionGapTimer : public ieee1394MacHandler
{
 public:
               ieee1394MacHandlerSubactionGapTimer(ieee1394Mac *m,double t=0.0000001);

 virtual void                 handle(Event *e);
 virtual void                 schedule(double t);
 virtual void                 schedule();
 virtual void                 cancel();

 virtual double               getTime() const {return time_;};

 protected:

 double        time_;
};
```

```
/**
* Determines when gaps of a pre-determined length have occured.
*
* schedule(t)                 set interupt to occur in time t.
* handle(e)                   a gap has occured.
* cancel()                    a transmission was seen, so cancel gap timing
*/
class ieee1394MacHandlerIsochGapTimer : public ieee1394MacHandler
{
 public:
               ieee1394MacHandlerIsochGapTimer(ieee1394Mac *m,double t=0.00000005);

 virtual void                 handle(Event *e);
 virtual void                 schedule(double t);
 virtual void                 schedule();
 virtual void                 cancel();

 virtual double               getTime() const {return time_;};

 protected:

 double        time_;
};

/**
* Determines when gaps of a pre-determined length have occured.
*
* schedule(t)                 set interupt to occur in time t.
* handle(e)                   a gap has occured.
* cancel()                    a transmission was seen, so cancel gap timing
*/
class ieee1394MacHandlerFairnessGapTimer : public ieee1394MacHandler
{
 public:
               ieee1394MacHandlerFairnessGapTimer(ieee1394Mac *m,double t=0.0000005);

 virtual void                 handle(Event *e);
 virtual void                 schedule(double t);
 virtual void                 schedule();
 virtual void                 cancel();

 virtual double               getTime() const {return time_;};

 protected:

 double        time_;
};

/**
* Base state class: represents a state, the events it can respond to, and the behaviours and
* transitions for the state.
*
* eventBecomeCurrentState()   called after this instance becomes the current state.
* eventRequestSendData(p,h)   called when the LL wants to send a packet p with handler h.
* eventSendCompleted()        called after a sent packet (any type) is delivered.
* eventSendCanceled()         called after a sent packet (any type) is canceled.
* eventRecvCompleted()        called after a recv packet (any type) arrives.
* eventRecvCanceled()         called after a recv packet (any type) is canceled.
```

```
* eventIncomingData(p,r)      called when receiving a packet p via interface r.
* eventReceivedData(p,r)      called after receiving a packet p via interface r.
* eventIncomingArbReq(p,r)    called when receiving a packet p via interface r.
* eventReceivedArbReq(p,r)    called after receiving a packet p via interface r.
* eventIncomingArbGrant(p,r)  called when receiving a packet p via interface r.
* eventReceivedArbGrant(p,r)  called after receiving a packet p via interface r.
* eventIncomingAck(p,r)       called when receiving a packet p via interface r.
* eventReceivedAck(p,r)       called after receiving a packet p via interface r.
* eventIncomingCycleRestart(p,r)             called when receiving a packet p via interface r.
* eventReceivedCycleRestart(p,r)             called after receiving a packet p via interface r.
* eventSubactionGap()         called after a gap length for async.
* eventFairnessGap()          called after a gap length for the fairness interval.
* eventIsochGap()             called after a gap length for isoch.
* eventTimeToRestartCycle()   called after cycle length timer expires.
*/
class ieee1394MacState
{
 public:

        ieee1394MacState(ieee1394Mac *m);

 virtual void                 eventBecameCurrentState();
 virtual void                 eventRequestSendData(Packet *p,Handler *h);
 virtual void                 eventSendCompleted();
 virtual void                 eventSendCanceled();
 virtual void                 eventRecvCompleted();
 virtual void                 eventRecvCanceled();
 virtual void                 eventIncomingData(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedData(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingCycleRestart(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedCycleRestart(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventSubactionGap();
 virtual void                 eventFairnessGap();
 virtual void                 eventIsochGap();
 virtual void                 eventTimeToRestartCycle();

 protected:

 // broadcasts copies of packet p to all connections except those connected via r
 // (if r is NULL, then send to all). returns true if any packets were sent.
 // invokes the mhSend_ timer if any packets sent.
 // packet p will be freed (either immediately or later).
 // returns true if any packets were sent.  false if none were sent.
 virtual bool                 broadcastPacket(Packet *p,ieee1394MacHandlerRecv *r);

 // broadcasts copies of packet p to the up connection.
 // returns true if any packets were sent, false if this is a root node.
 // invokes the mhSend_ timer if any packets sent.
 // packet p will be freed (either immediately or later).
 // returns true if any packets were sent.  false if none were sent.
 // err is true if the packet wasn't set because of a collision.
 // we are the root only if nothing was sent, and there was no error.
```

```
 virtual bool                 sendPacketUp(Packet *p,bool &err);

 ieee1394Mac   *mac;
};

class ieee1394MacStateTransitionSend : public ieee1394MacState
{
 public:

        ieee1394MacStateTransitionSend(ieee1394Mac *m);

 virtual void                 eventSendCompleted();
 virtual void                 eventSendCanceled();
 virtual void                 eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbGrant(Packet *p,ieee1394MacHandlerRecv *r);

 virtual void                 setNextState(ieee1394MacState *ns);

 protected:

 ieee1394MacState             *next_state;
};

class ieee1394MacStateIdle : public ieee1394MacState
{
 public:

        ieee1394MacStateIdle(ieee1394Mac *m);

 virtual void                 eventBecameCurrentState();
 virtual void                 eventRequestSendData(Packet *p,Handler *h);
 virtual void                 eventSendCompleted();
 virtual void                 eventSendCanceled();
 virtual void                 eventIncomingData(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedData(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingCycleRestart(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedCycleRestart(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventTimeToRestartCycle();
};

class ieee1394MacStateWaitToRequestArb : public ieee1394MacStateIdle
{
 public:

        ieee1394MacStateWaitToRequestArb(ieee1394Mac *m);

 virtual void                 eventBecameCurrentState();
 virtual void                 eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventSubactionGap();
 virtual void                 eventFairnessGap();
 virtual void                 eventIsochGap();
```

```
 protected:

 virtual bool                canRequestArb() const;

 // canRequestArb must be true to call this function
 virtual void                requestArb();

 // no packets should be in transit when this is called
 // packet p is only used to identify a possible requester and is not for later use.
 virtual void                grantArb(Packet *req_p);
};

class ieee1394MacStateWaitForGrantArb : public ieee1394MacState
{
 public:

        ieee1394MacStateWaitForGrantArb(ieee1394Mac *m);

 virtual void                eventBecameCurrentState();
 virtual void                eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventReceivedArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventIncomingArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventReceivedArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventSubactionGap();
 virtual void                eventFairnessGap();
 virtual void                eventIsochGap();

 protected:

 virtual ieee1394MacState    *acceptArb();
};

class ieee1394MacStateWaitForArbCompletion : public ieee1394MacState
{
 public:

        ieee1394MacStateWaitForArbCompletion(ieee1394Mac *m);

 virtual void                eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventReceivedArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventIncomingArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventReceivedArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                eventSubactionGap();
 virtual void                eventFairnessGap();
 virtual void                eventIsochGap();
};

class ieee1394MacStateWaitForActionCompletion : public ieee1394MacStateIdle
{
 public:

        ieee1394MacStateWaitForActionCompletion(ieee1394Mac *m);

 virtual void                eventBecameCurrentState();
 virtual void                eventRequestSendData(Packet *p,Handler *h);
 virtual void                eventSendCompleted();
 virtual void                eventSendCanceled();
```

```
 virtual void                 eventRecvCompleted();
 virtual void                 eventRecvCanceled();
 virtual void                 eventReceivedData(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbGrant(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedCycleRestart(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventSubactionGap();
 virtual void                 eventFairnessGap();
 virtual void                 eventIsochGap();
 virtual void                 eventTimeToRestartCycle();
};

class ieee1394MacStateAsyncSendData : public ieee1394MacState
{
 public:

        ieee1394MacStateAsyncSendData(ieee1394Mac *m);

 virtual void                 eventBecameCurrentState();
};

class ieee1394MacStateIsochSendData : public ieee1394MacState
{
 public:

        ieee1394MacStateIsochSendData(ieee1394Mac *m);

 virtual void                 eventBecameCurrentState();
};

class ieee1394MacStateWaitForAck : public ieee1394MacState
{
 public:

        ieee1394MacStateWaitForAck(ieee1394Mac *m);

 virtual void                 eventIncomingArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedArbReq(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventIncomingAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventReceivedAck(Packet *p,ieee1394MacHandlerRecv *r);
 virtual void                 eventSubactionGap();
 virtual void                 eventFairnessGap();
 virtual void                 eventIsochGap();
};

class ieee1394MacStateCycleRestartSend : public ieee1394MacState
{
 public:

        ieee1394MacStateCycleRestartSend(ieee1394Mac *m);

 virtual void                 eventBecameCurrentState();
};
```

```
// ieee1394 MAC data structure

class ieee1394Mac : public Mac
{
 friend class ieee1394MacHandler;
 friend class ieee1394MacHandlerRecv;
 friend class ieee1394MacHandlerSend;
 friend class ieee1394MacHandlerCycleTimer;
 friend class ieee1394MacHandlerSubactionGapTimer;
 friend class ieee1394MacHandlerIsochGapTimer;
 friend class ieee1394MacHandlerFairnessGapTimer;

 friend class ieee1394MacState;
 friend class ieee1394MacStateTransitionSend;
 friend class ieee1394MacStateIdle;
 friend class ieee1394MacStateWaitToRequestArb;
 friend class ieee1394MacStateWaitForGrantArb;
 friend class ieee1394MacStateWaitForArbCompletion;
 friend class ieee1394MacStateWaitForActionCompletion;
 friend class ieee1394MacStateAsyncSendData;
 friend class ieee1394MacStateWaitForAck;
 friend class ieee1394MacStateIsochSendData;
 friend class ieee1394MacStateCycleRestartSend;

 public:

 enum mac_state {NO_ARB, SENT_ARB_REQ, GRANTED_ARB, SENT_PACKET, SUBACTION_DONE};

            ieee1394Mac();
 virtual    ~ieee1394Mac();

 virtual void                 recv(Packet* p, Handler* h);
 virtual int                  command(int argc, const char*const* argv);

 virtual ieee1394MacState     *getCurrentState() const;
 virtual bool                 is_busy() const;

 virtual void                 cancel_gap_timers();
 virtual void                 schedule_gap_timers();

 protected:

 virtual void                 sendDown(Packet *p, Handler *h);
 virtual void                 sendUp(Packet *p, Handler *h);

 virtual void                 recv_from_phys(Packet *p,ieee1394MacHandlerRecv *r);

 virtual bool                 is_root() const; // true if there is no parent.

 // this should return the address that appears in the packets originating from the
 // attached node.
 virtual nsaddr_t             get_node_address() const;

 virtual void                 changeCurrentState(ieee1394MacState *s);

 // typically ieee1394 nodes would only have two ports for cost reasons...
 static const int             max_receivers = 10;
```

```
 int                          trace_;      // To turn on MAC level collision traces

 ieee1394MacHandlerRecv                      *mhRecv_[max_receivers];
 ieee1394MacHandlerSend                      mhSend_;
 ieee1394MacHandlerCycleTimer                mhCycleTimer_;
 ieee1394MacHandlerSubactionGapTimer         mhSubactionTimer_;
 ieee1394MacHandlerIsochGapTimer             mhIsochTimer_;
 ieee1394MacHandlerFairnessGapTimer          mhFairnessTimer_;

 virtual char                                *get_state_name(ieee1394MacState *c);

 ieee1394MacState                            *current_state;
 ieee1394MacStateTransitionSend              transition_send_state;
 ieee1394MacStateIdle                        idle_state;
 ieee1394MacStateWaitToRequestArb            wait_to_arb_state;
 ieee1394MacStateWaitForGrantArb             wait_for_arb_state;
 ieee1394MacStateWaitForArbCompletion        wait_for_completion_state;
 ieee1394MacStateWaitForActionCompletion     wait_for_action_completion_state;
 ieee1394MacStateAsyncSendData               async_send_state;
 ieee1394MacStateWaitForAck                  async_await_ack_state;
 ieee1394MacStateIsochSendData               isoch_send_state;
 ieee1394MacStateCycleRestartSend            cycle_restart_send_state;

 Packet                                      *data_packet_pending;
 bool                                        restart_cycle_pending;
 bool                                        subaction_gap; // okay to request arb now
 bool                                        isoch_gap; // okay to request arb now
 bool                                        fairness_gap; // okay to reset async_sent now
 bool                                        async_sent; // false if we can arb
 bool                                        isoch_sent; // false if we can arb
 bool                                        won_grant; // true if wa already won the arb grant;
};

#endif

/*
 *  mac-1394.cpp
 *
 *  Created by Glendon Holst on Tue Mar 19 2002.
 *
 *  Based on mac-802_3 from ns2
 *
 */

#include "mac-1394.h"
#include "packet-1394.h"
#include <packet.h>
#include <random.h>
#include <arp.h>
#include <ll.h>

//#define MAC_DEBUG

#ifndef MAC_DEBUG
#define FPRINTF(s, f, t, index, func) do {} while (0)
#else
   static double xtime= 0.0;
#  define FPRINTF(s, f, t, index, func) \
```

```
         do { fprintf(s, f, t, index, func); xtime= t; } while (0)
#endif MAC_DEBUG

inline int min(int f,int s) { return (f < s ? f : s); }

// ******** ieee1394MacClass **********

static class ieee1394MacClass : public TclClass
{
 public:

               ieee1394MacClass() : TclClass("Mac/ieee1394") {}

 TclObject     *create(int, const char*const*) {return (new ieee1394Mac());}
} class_ieee1394Mac;

// ******** ieee1394MacHandlerRecvClass **********

static class ieee1394MacHandlerRecvClass : public TclClass
{
 public:

               ieee1394MacHandlerRecvClass() : TclClass("Mac/ieee1394/HandlerRecv") {}

 TclObject     *create(int argc, const char*const* argv)
 {
  if (argc == 5)
  {TclObject   *obj;

   if ((obj = TclObject::lookup(argv[4])) == 0)
   {
    fprintf(stderr, "%s lookup failed\n", argv[4]);
    return NULL;
   }
   return (new ieee1394MacHandlerRecv((ieee1394Mac*) obj));
  }
  return NULL;
 }
} class_ieee1394MacHandlerRecv;

// ******** ieee1394MacHandler **********

inline void    ieee1394MacHandler::cancel()
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 {Scheduler    &s = Scheduler::instance();

  s.cancel(&intr);
 }
}

// ******** ieee1394MacHandlerSend **********

ieee1394MacHandlerSend::ieee1394MacHandlerSend(ieee1394Mac *m) :
    ieee1394MacHandler(m),
    p_(NULL),
```

```
    completed(m),
    canceled(m),
    delay(false)
{
}

void                          ieee1394MacHandlerSend::handle(Event *)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 assert(p_);

 mac->cancel_gap_timers();

 Packet::free(p_);
 p_ = NULL;

 if (delay)
 {
  completed.schedule();
  delay = false;
 }
 else
 {
  if (!(mac->is_busy()))
   mac->schedule_gap_timers();

  mac->getCurrentState()->eventSendCompleted();
 }
}

void                          ieee1394MacHandlerSend::schedule(Packet *p, double t)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 assert(p_ == NULL);

 mac->cancel_gap_timers();

 {Scheduler    &s = Scheduler::instance();

  if (busy())
  {hdr_ieee1394              *hdr1 = hdr_ieee1394::access(p);
   hdr_ieee1394              *hdr2 = hdr_ieee1394::access(p_);
   hdr_ip                    *ip1 = hdr_ip::access(p);
   hdr_ip                    *ip2 = hdr_ip::access(p_);

   printf("ieee1394MacHandlerSend::schedule - collision error -- should not occur!\n");
   printf("    SEND - addr:%d to_send<%d:%d,%d>, sending:<%d:%d,%d>
t:%.15f\n",mac->get_node_address(),hdr1->trans_code(),ip1->saddr(),ip1->daddr(),hdr2->trans_code(
),ip2->saddr(),ip2->daddr(),Scheduler::instance().clock());

   Packet::free(p);
   cancel();
  }
  else
```

```
  {
   s.schedule(this, &intr, t);
   p_ = p;
  }
 }
}

void                          ieee1394MacHandlerSend::cancel()
{
 ieee1394MacHandler::cancel();

 if (p_ != NULL)
 {
  Packet::free(p_);
  p_ = NULL;

  if (delay)
  {
   canceled.schedule();
   delay = false;
  }
  else
  {
   if (!(mac->is_busy()))
    mac->schedule_gap_timers();

   mac->getCurrentState()->eventSendCanceled();
  }
 }
}

ieee1394MacHandlerSend::delayedCompleted::delayedCompleted(ieee1394Mac *m) :
    ieee1394MacHandler(m)
{
}

void                          ieee1394MacHandlerSend::delayedCompleted::handle(Event *)
{
 if (!(mac->is_busy()))
  mac->schedule_gap_timers();

 mac->getCurrentState()->eventSendCompleted();
}

void                          ieee1394MacHandlerSend::delayedCompleted::schedule()
{
 ieee1394MacHandler::cancel();

 {Scheduler    &s = Scheduler::instance();

  s.schedule(this, &intr, 0.0000000001);
 }
}

ieee1394MacHandlerSend::delayedCanceled::delayedCanceled(ieee1394Mac *m) :
    ieee1394MacHandler(m)
{
}
```

```
void                          ieee1394MacHandlerSend::delayedCanceled::handle(Event *)
{
 if (!(mac->is_busy()))
  mac->schedule_gap_timers();

 mac->getCurrentState()->eventSendCanceled();
}

void                          ieee1394MacHandlerSend::delayedCanceled::schedule()
{
 ieee1394MacHandler::cancel();

 handle(NULL);

 {Scheduler    &s = Scheduler::instance();

//  s.schedule(this, &intr, 0.0000000001);
 }
}

// ******** ieee1394MacHandlerRecv **********

ieee1394MacHandlerRecv::ieee1394MacHandlerRecv(ieee1394Mac *m) :
    ieee1394MacHandler(m),
    p_(NULL),
    downtarget_(NULL),
    is_route_to_parent(false),
    completed(m,this),
    delay(true)
{
}

void                          ieee1394MacHandlerRecv::recv(Packet *p, Handler *)
{double        txtime = mac->netif_->txtime(p) + 0.0000000000001;

 if (hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::GRANT_ARB && mac->mhSend_.busy() &&
     hdr_ieee1394::access(mac->mhSend_.packet())->trans_code() == hdr_ieee1394::REQ_ARB)
 {
  mac->mhSend_.cancel();
 }

 schedule(p,txtime);

 if (p_ != NULL)
 {Packet       *p2 = p_->copy();
  hdr_ieee1394 *hdr = hdr_ieee1394::access(p2);
  bool         is_async = hdr->trans_code() == hdr_ieee1394::ASYNC_PAK;
  bool         is_isoch = hdr->trans_code() == hdr_ieee1394::ISOCH_PAK;
  bool         is_arb_req = hdr->trans_code() == hdr_ieee1394::REQ_ARB;
  bool         is_arb_grant = hdr->trans_code() == hdr_ieee1394::GRANT_ARB;
  bool         is_ack = hdr->trans_code() == hdr_ieee1394::ACK;
  bool         is_cycle_restart = hdr->trans_code() == hdr_ieee1394::CYCLE_RESTART;

  if (is_async || is_isoch)
  {
   mac->current_state->eventIncomingData(p2,this);
  }
```

```
  else if (is_arb_req)
  {
   mac->current_state->eventIncomingArbReq(p2,this);
  }
  else if (is_arb_grant)
  {
   mac->current_state->eventIncomingArbGrant(p2,this);
  }
  else if (is_ack)
  {
   mac->current_state->eventIncomingAck(p2,this);
  }
  else if (is_cycle_restart)
  {
   mac->current_state->eventIncomingCycleRestart(p2,this);
  }
  else
  {
   fprintf(stderr, "ieee1394MacHandlerRecv::recv - invalid packet type\n");
   Packet::free(p2);
  }
 }
}

inline void    ieee1394MacHandlerRecv::cancel()
{
 ieee1394MacHandler::cancel();

 if (p_ != NULL)
 {
  Packet::free(p_);
  p_ = NULL;

  if (!(mac->is_busy()))
   mac->schedule_gap_timers();

  mac->current_state->eventRecvCanceled();
 }
}

void                          ieee1394MacHandlerRecv::handle(Event *)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 mac->cancel_gap_timers();

 {Packet       *p = p_->copy();

  Packet::free(p_);
  p_ = NULL;

  if ((hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::GRANT_ARB ||
       hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::ASYNC_PAK) &&
       hdr_ip::access(p)->daddr() == mac->get_node_address())
  {
   completed.schedule(p);
  }
```

```
  else
  {
   if (!(mac->is_busy()))
    mac->schedule_gap_timers();

   mac->current_state->eventRecvCompleted();
   deliverPacket(p);
  }
 }
}

void                          ieee1394MacHandlerRecv::schedule(Packet *p, double t)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 mac->cancel_gap_timers();

 {
  assert(busy());
  assert(p);

  if (busy())
  {hdr_ieee1394               *hdr1 = hdr_ieee1394::access(p);
   hdr_ieee1394               *hdr2 = hdr_ieee1394::access(p_);
   hdr_ip                     *ip1 = hdr_ip::access(p);
   hdr_ip                     *ip2 = hdr_ip::access(p_);

   printf("ieee1394MacHandlerRecv::schedule - collision error -- should not occur!\n");
   printf("     SCHED - %d <%d:%d,%d>, <%d:%d,%d>
%.15f\n",mac->get_node_address(),hdr1->trans_code(),ip1->saddr(),ip1->daddr(),hdr2->trans_code(),
ip2->saddr(),ip2->daddr(),Scheduler::instance().clock());

   Packet::free(p);
   cancel();
  }
  else
  {Scheduler   &s = Scheduler::instance();

   s.schedule((ieee1394MacHandler*) this, &intr, t);
   p_ = p;
  }
 }
}

void                          ieee1394MacHandlerRecv::deliverPacket(Packet *p)
{
 mac->recv_from_phys(p,this);
}

int                           ieee1394MacHandlerRecv::command(int argc, const char*const* argv)
{Tcl           &tcl = Tcl::instance();

 if (argc == 2)
 {
  if (strcmp(argv[1], "up-target") == 0)
  {
   if (mac != 0)
```

```
    tcl.result(mac->name());
   return (TCL_OK);
  }
  if (strcmp(argv[1], "down-target") == 0)
  {
   if (downtarget_ != 0)
    tcl.result(downtarget_->name());
   return (TCL_OK);
  }
 }
 else if (argc == 3)
 {TclObject    *obj;

  if( (obj = TclObject::lookup(argv[2])) == 0)
  {
   fprintf(stderr, "%s lookup failed\n", argv[1]);
   return TCL_ERROR;
  }

  if (strcmp(argv[1], "up-target") == 0)
  {
   if (*argv[2] == '0')
   {
    mac = 0;
    return (TCL_OK);
   }
   mac = (ieee1394Mac*) obj;
   if (mac == 0)
   {
    tcl.resultf("no such object %s", argv[2]);
    return (TCL_ERROR);
   }
   return (TCL_OK);
  }
  if (strcmp(argv[1], "down-target") == 0)
  {
   if (*argv[2] == '0')
   {
    downtarget_ = 0;
    return (TCL_OK);
   }
   downtarget_ = (NsObject*) obj;
   if (downtarget_ == 0)
   {
    tcl.resultf("no such object %s", argv[2]);
    return (TCL_ERROR);
   }
   return (TCL_OK);
  }
 }
 return (NsObject::command(argc, argv));
}

ieee1394MacHandlerRecv::delayedCompleted::delayedCompleted(ieee1394Mac *m, ieee1394MacHandlerRecv
*r) :
    ieee1394MacHandler(m),
    recv(r),
    pkt(NULL)
```

```
{
}

void                          ieee1394MacHandlerRecv::delayedCompleted::handle(Event *)
{Packet        *p = pkt;

 pkt = NULL;

 if (!(mac->is_busy()))
  mac->schedule_gap_timers();

 mac->getCurrentState()->eventRecvCompleted();
 recv->deliverPacket(p);
}

void                          ieee1394MacHandlerRecv::delayedCompleted::cancel()
{
 ieee1394MacHandler::cancel();
 if (pkt != NULL)
 {
  Packet::free(pkt);
  pkt = NULL;
 }
}

void                          ieee1394MacHandlerRecv::delayedCompleted::schedule(Packet *p)
{
 cancel();

 pkt = p;

 {Scheduler    &s = Scheduler::instance();

  s.schedule(this, &intr, 0.0000000001);
 }
}

// ******** ieee1394MacHandlerCycleTimer **********

void                          ieee1394MacHandlerCycleTimer::handle(Event *)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 if (mac->is_root())
 {
  schedule(time_);
  mac->current_state->eventTimeToRestartCycle();
 }
}

void                          ieee1394MacHandlerCycleTimer::schedule(double t)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 {Scheduler    &s = Scheduler::instance();
```

```
  s.schedule(this, &intr, t);
 }
}

void                         ieee1394MacHandlerCycleTimer::schedule()
{
 schedule(time_);
}

// ******** ieee1394MacHandlerSubactionGapTimer **********

ieee1394MacHandlerSubactionGapTimer::ieee1394MacHandlerSubactionGapTimer(ieee1394Mac *m,double t)
:
    ieee1394MacHandler(m),
    time_(t)
{
}

void                         ieee1394MacHandlerSubactionGapTimer::handle(Event *)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 mac->subaction_gap = true;
 mac->current_state->eventSubactionGap();
}

void                         ieee1394MacHandlerSubactionGapTimer::schedule(double t)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 {Scheduler    &s = Scheduler::instance();

  s.schedule(this, &intr, t);
 }
}

void                         ieee1394MacHandlerSubactionGapTimer::schedule()
{
 schedule(time_);
}

void                         ieee1394MacHandlerSubactionGapTimer::cancel()
{
 ieee1394MacHandler::cancel();
 mac->subaction_gap = false;
}

// ******** ieee1394MacHandlerIsochGapTimer **********

ieee1394MacHandlerIsochGapTimer::ieee1394MacHandlerIsochGapTimer(ieee1394Mac *m,double t) :
    ieee1394MacHandler(m),
    time_(t)
{
}
```

```
void                          ieee1394MacHandlerIsochGapTimer::handle(Event *)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 mac->isoch_gap = true;
 mac->current_state->eventIsochGap();
}

void                          ieee1394MacHandlerIsochGapTimer::schedule(double t)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 {Scheduler    &s = Scheduler::instance();

  s.schedule(this, &intr, t);
 }
}

void                          ieee1394MacHandlerIsochGapTimer::schedule()
{
 schedule(time_);
}

void                          ieee1394MacHandlerIsochGapTimer::cancel()
{
 ieee1394MacHandler::cancel();
 mac->isoch_gap = false;
}

// ******** ieee1394MacHandlerFairnessGapTimer **********

ieee1394MacHandlerFairnessGapTimer::ieee1394MacHandlerFairnessGapTimer(ieee1394Mac *m,double t) :

    ieee1394MacHandler(m),
    time_(t)
{
}

void                          ieee1394MacHandlerFairnessGapTimer::handle(Event *)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 mac->fairness_gap = true;
 mac->current_state->eventFairnessGap();
}

void                          ieee1394MacHandlerFairnessGapTimer::schedule(double t)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), mac->index_,
__PRETTY_FUNCTION__);

 {Scheduler    &s = Scheduler::instance();

  s.schedule(this, &intr, t);
 }
```

```
}

void                          ieee1394MacHandlerFairnessGapTimer::schedule()
{
 schedule(time_);
}

void                          ieee1394MacHandlerFairnessGapTimer::cancel()
{
 ieee1394MacHandler::cancel();
 mac->fairness_gap = false;
}

// ******** ieee1394MacState **********

ieee1394MacState::ieee1394MacState(ieee1394Mac *m) :
    mac(m)
{
}

void                          ieee1394MacState::eventBecameCurrentState()
{
}

void                          ieee1394MacState::eventRequestSendData(Packet *p,Handler *h)
{
 if (mac->data_packet_pending == NULL)
 {
  mac->data_packet_pending = p;
  mac->callback_ = h;
 }
 else
  printf("%s::eventRequestSendData - invalid event - another send is already
pending\n",mac->get_state_name(mac->current_state));
}

void                          ieee1394MacState::eventSendCompleted()
{
}

void                          ieee1394MacState::eventSendCanceled()
{
}

void                          ieee1394MacState::eventRecvCompleted()
{
}

void                          ieee1394MacState::eventRecvCanceled()
{
}

void                          ieee1394MacState::eventIncomingData(Packet
*p,ieee1394MacHandlerRecv *r)
{
 printf("%s::eventIncomingData - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
```

```
}

void                          ieee1394MacState::eventReceivedData(Packet
*p,ieee1394MacHandlerRecv *r)
{
 printf("%s::eventReceivedData - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventIncomingArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 printf("%s::eventIncomingArbReq - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventReceivedArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 Packet::free(p);
}

void                          ieee1394MacState::eventIncomingArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 printf("%s::eventIncomingArbGrant - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventReceivedArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 printf("%s::eventReceivedArbGrant - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventIncomingAck(Packet *p,ieee1394MacHandlerRecv
*r)
{
 printf("%s::eventIncomingAck - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventReceivedAck(Packet *p,ieee1394MacHandlerRecv
*r)
{
 printf("%s::eventReceivedAck - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventIncomingCycleRestart(Packet
*p,ieee1394MacHandlerRecv *r)
```

```
{
 printf("%s::eventIncomingCycleRestart - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventReceivedCycleRestart(Packet
*p,ieee1394MacHandlerRecv *r)
{
 printf("%s::eventReceivedCycleRestart - invalid event - not expecting
event\n",mac->get_state_name(mac->current_state));
 Packet::free(p);
}

void                          ieee1394MacState::eventSubactionGap()
{
 mac->subaction_gap = true;
}

void                          ieee1394MacState::eventFairnessGap()
{
 mac->fairness_gap = true;
 mac->async_sent = false;
}

void                          ieee1394MacState::eventIsochGap()
{
 mac->isoch_gap = true;
}

void                          ieee1394MacState::eventTimeToRestartCycle()
{
 mac->restart_cycle_pending = true;
}

bool                          ieee1394MacState::broadcastPacket(Packet *p,ieee1394MacHandlerRecv
*r)
{int                          cnt = 0;

 if (hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::REQ_ARB ||
      hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::GRANT_ARB)
 {
  hdr_cmn::access(p)->size() = IEEE_1394_ARB_SIZE;
 }
 else if (hdr_cmn::access(p)->size() < IEEE_1394_MINFRAME)
 {// pad packet
  hdr_cmn::access(p)->size() = IEEE_1394_MINFRAME;
 }

 for (int i = 0; i < mac->max_receivers; i++)
  if (mac->mhRecv_[i] != NULL && mac->mhRecv_[i] != r)
  {
   if (!mac->mhRecv_[i]->busy())
   {Packet     *p_bcast = p->copy();

    hdr_cmn::access(p_bcast)->direction() = hdr_cmn::DOWN;
    cnt++;
    mac->mhRecv_[i]->downtarget()->recv(p_bcast);
```

```
   }
   else
   {hdr_ieee1394              *hdr_out = hdr_ieee1394::access(p);
    hdr_ieee1394              *hdr_in = hdr_ieee1394::access(mac->mhRecv_[i]->packet());

    if (hdr_in->trans_code() == hdr_ieee1394::REQ_ARB) // cancel, then send
    {Packet    *p_bcast = p->copy();

     mac->mhRecv_[i]->cancel();
     hdr_cmn::access(p_bcast)->direction() = hdr_cmn::DOWN;
     cnt++;
     mac->mhRecv_[i]->downtarget()->recv(p_bcast);
    }
    else if (hdr_out->trans_code() == hdr_ieee1394::REQ_ARB)
    {
     // don't send packet, but don't complain
    }
    else
    {
     printf("%s::broadcastPacket - collision error -- should not
occur!\n",mac->get_state_name(mac->current_state));
    }
   }
  }

 if (cnt > 0)
 {double       txtime = mac->netif_->txtime(p) + 0.0000000000001;
  bool                        delay;

  delay =   (r == NULL) && (hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::GRANT_ARB) &&
                (hdr_ip::access(p)->daddr() == mac->get_node_address());


  mac->mhSend_.schedule(p,txtime);
  mac->mhSend_.setDelay(delay);
 }
 else
 {
  Packet::free(p);
 }

 return (cnt > 0);
}

bool                          ieee1394MacState::sendPacketUp(Packet *p,bool &err)
{int                          cnt = 0;

 err = false;

 if (hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::REQ_ARB ||
      hdr_ieee1394::access(p)->trans_code() == hdr_ieee1394::GRANT_ARB)
 {
  hdr_cmn::access(p)->size() = IEEE_1394_ARB_SIZE;
 }
 else if (hdr_cmn::access(p)->size() < IEEE_1394_MINFRAME)
 {// pad packet
  hdr_cmn::access(p)->size() = IEEE_1394_MINFRAME;
 }
```

```
 for (int i = 0; i < mac->max_receivers; i++)
  if (mac->mhRecv_[i] != NULL && mac->mhRecv_[i]->is_downtarget_to_parent())
  {
   if (!mac->mhRecv_[i]->busy())
   {Packet     *p_sup = p->copy();

    hdr_cmn::access(p_sup)->direction() = hdr_cmn::DOWN;
    cnt++;
    mac->mhRecv_[i]->downtarget()->recv(p_sup);
   }
   else
   {hdr_ieee1394              *hdr_out = hdr_ieee1394::access(p);
    hdr_ieee1394              *hdr_in = hdr_ieee1394::access(mac->mhRecv_[i]->packet());

    if (hdr_out->trans_code() == hdr_ieee1394::REQ_ARB)
    {
     err = true; // cancel sending REQ_ARB because of collision.
    }
    else
    {
     printf("%s::sendPacketUp - collision error -- should not occur!\n",
mac->get_state_name(mac->current_state));
    }
   }
  }

 if (cnt > 0)
 {double       txtime = mac->netif_->txtime(p) + 0.0000000000001;

  mac->mhSend_.schedule(p,txtime);
 }
 else
 {
  Packet::free(p);
 }

 return (cnt > 0);
}

// ******** ieee1394MacStateTransitionSend **********

ieee1394MacStateTransitionSend::ieee1394MacStateTransitionSend(ieee1394Mac *m) :
    ieee1394MacState(m),
    next_state(NULL)
{
}

void                          ieee1394MacStateTransitionSend::eventSendCompleted()
{
 if (next_state != NULL)
 {
  mac->changeCurrentState(next_state);
 }
 else
 {
  mac->changeCurrentState(&(mac->idle_state));
 }
```

```
}

void                          ieee1394MacStateTransitionSend::eventSendCanceled()
{
 if (next_state != NULL)
 {
  mac->changeCurrentState(next_state);
 }
 else
 {
  mac->changeCurrentState(&(mac->idle_state));
 }
}

void                          ieee1394MacStateTransitionSend::eventIncomingArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests if we are sending anything
 if (r != NULL)
 {
  r->cancel();
 }
 Packet::free(p);
}

void                          ieee1394MacStateTransitionSend::eventReceivedArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests if we are sending anything
 Packet::free(p);
}

void                          ieee1394MacStateTransitionSend::eventIncomingArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
// mac->mhSend_.cancel();

printf("ieee1394MacStateTransitionSend::eventIncomingArbGrant - EVENT INCOMING ARG GRANT -
receiver should take care of this\n");

// broadcastPacket(p,r);
 Packet::free(p);
}

void                          ieee1394MacStateTransitionSend::eventReceivedArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // if this grant is for us, then we are in the wrong state -- should be StateWaitForGrantArb
 printf("ieee1394MacStateTransitionSend::eventReceivedArbGrant - EVENT RECEIVED ARG GRANT -
receiver should take care of this\n");
 Packet::free(p);
}

void                          ieee1394MacStateTransitionSend::setNextState(ieee1394MacState *ns)
{
 next_state = ns;
}
```

```
//********* ieee1394MacStateIdle **********

ieee1394MacStateIdle::ieee1394MacStateIdle(ieee1394Mac *m) :
    ieee1394MacState(m)
{
}

void                          ieee1394MacStateIdle::eventBecameCurrentState()
{
 if (!mac->mhSend_.busy() &&
     ((mac->restart_cycle_pending && mac->is_root()) || (mac->data_packet_pending != NULL)))
  mac->changeCurrentState(&(mac->wait_to_arb_state));
}

void                          ieee1394MacStateIdle::eventRequestSendData(Packet *p,Handler *h)
{
 if (mac->data_packet_pending == NULL)
 {
  mac->data_packet_pending = p;
  mac->callback_ = h;
  mac->changeCurrentState(&(mac->wait_to_arb_state));
 }
 else
 {
  printf("ieee1394MacStateIdle::eventRequestSendData - invalid event - idle state should not have
other events pending\n");
  Packet::free(p);
 }
}

void                          ieee1394MacStateIdle::eventSendCompleted()
{
 // ignore event, it would be from passing packets along.
}

void                          ieee1394MacStateIdle::eventSendCanceled()
{
 // ignore event, it would be from passing packets along.
}

void                          ieee1394MacStateIdle::eventIncomingData(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void                          ieee1394MacStateIdle::eventReceivedData(Packet
*p,ieee1394MacHandlerRecv *r)
{hdr_ieee1394                 *p_hdr = hdr_ieee1394::access(p);
 hdr_ip        *p_iph = hdr_ip::access(p);
 nsaddr_t      saddr = p_iph->saddr();
 bool                         for_this_node = (mac->get_node_address() == p_iph->daddr());
 bool                         async_data = p_hdr->trans_code() == hdr_ieee1394::ASYNC_PAK;

 if (for_this_node)
 {Packet       *p_up = p->copy();

  if (async_data)
```

```
  {Packet                     *ack_p = ieee1394Packet::alloc_packet();
   hdr_ieee1394               *hdr = hdr_ieee1394::access(ack_p);
   hdr_cmn                    *ch = hdr_cmn::access(ack_p);
   hdr_ip                     *iph = hdr_ip::access(ack_p);

   hdr->trans_code_ = hdr_ieee1394::ACK;
   ch->direction() = hdr_cmn::DOWN;
   iph->saddr() = mac->get_node_address();
   iph->daddr() = saddr;

   broadcastPacket(ack_p,NULL);
  }

  Packet::free(p);
  mac->uptarget_->recv(p_up,(Handler*)NULL);  // assum zero mac-delay for now
 }
 else
  Packet::free(p);
}

void                          ieee1394MacStateIdle::eventIncomingArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{hdr_ip                       *iph = hdr_ip::access(p);
 nsaddr_t                     saddr = iph->saddr();
 bool                                        err;

 if (!sendPacketUp(p,err) && !err) // we are the root node
 {Packet                      *grant_p = ieee1394Packet::alloc_packet();
  hdr_ieee1394                *hdr = hdr_ieee1394::access(grant_p);
  hdr_cmn                     *ch = hdr_cmn::access(grant_p);
  hdr_ip                      *grant_iph = hdr_ip::access(grant_p);

  hdr->trans_code_ = hdr_ieee1394::GRANT_ARB;
  ch->direction() = hdr_cmn::DOWN;
  grant_iph->saddr() = 0;
  grant_iph->daddr() = saddr;

  if (broadcastPacket(grant_p,NULL))
  {
   mac->transition_send_state.setNextState(&(mac->wait_for_action_completion_state));
   mac->changeCurrentState(&(mac->transition_send_state));
  }
  else
  {
   mac->changeCurrentState(&(mac->wait_for_action_completion_state));
  }
 }
 else if (!err)
 {
  mac->changeCurrentState(&(mac->wait_for_completion_state));
 }
 else
 {
 }
}

void                          ieee1394MacStateIdle::eventIncomingArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
```

```
{
 broadcastPacket(p,r);
}

void                          ieee1394MacStateIdle::eventReceivedArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 Packet::free(p);
 mac->changeCurrentState(&(mac->wait_for_action_completion_state));
}

void                          ieee1394MacStateIdle::eventIncomingAck(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void                          ieee1394MacStateIdle::eventReceivedAck(Packet
*p,ieee1394MacHandlerRecv *r)
{
 Packet::free(p);
}

void                          ieee1394MacStateIdle::eventIncomingCycleRestart(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void                          ieee1394MacStateIdle::eventReceivedCycleRestart(Packet
*p,ieee1394MacHandlerRecv *r)
{
 mac->isoch_sent = false; // we can send isoch packets again
 Packet::free(p);
}

void                          ieee1394MacStateIdle::eventTimeToRestartCycle()
{
 ieee1394MacState::eventTimeToRestartCycle();

 if (mac->is_root())
  mac->changeCurrentState(&(mac->wait_to_arb_state));
}

// ******** ieee1394MacStateWaitToRequestArb **********

ieee1394MacStateWaitToRequestArb::ieee1394MacStateWaitToRequestArb(ieee1394Mac *m) :
    ieee1394MacStateIdle(m)
{
}

void                          ieee1394MacStateWaitToRequestArb::eventBecameCurrentState()
{
 if (canRequestArb())
 {
  requestArb();
 }
}
```

```
void                          ieee1394MacStateWaitToRequestArb::eventIncomingArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{Packet        *p2 = p->copy();
 Packet        *p3 = p2->copy();
 bool                         err;

 Packet::free(p);

 if (canRequestArb())
 {hdr_ip                      *iph = hdr_ip::access(p2);

  iph->saddr() = mac->get_node_address();
  iph->daddr() = 0;
 }

 if (!sendPacketUp(p2,err) && !err)
 {// we are the root node

  grantArb(p3);
 }
 else if (!err)
 {// we are not the root
  mac->transition_send_state.setNextState(&(mac->wait_for_arb_state));
  mac->changeCurrentState(&(mac->transition_send_state));
 }
 else // err == true
 {// we are not the root
  mac->changeCurrentState(&(mac->wait_for_arb_state));
 }

 Packet::free(p3);
}

void                          ieee1394MacStateWaitToRequestArb::eventSubactionGap()
{
 ieee1394MacState::eventSubactionGap();
 eventBecameCurrentState();
}

void                          ieee1394MacStateWaitToRequestArb::eventFairnessGap()
{
 ieee1394MacState::eventFairnessGap();
 eventBecameCurrentState();
}

void                          ieee1394MacStateWaitToRequestArb::eventIsochGap()
{
 ieee1394MacState::eventIsochGap();
 eventBecameCurrentState();
}

bool                          ieee1394MacStateWaitToRequestArb::canRequestArb() const
{
 if (mac->restart_cycle_pending && mac->is_root())
  return true;

 if (mac->data_packet_pending != NULL)
```

```
 {hdr_ieee1394                *hdr = hdr_ieee1394::access(mac->data_packet_pending);
  bool                                   is_async = hdr->trans_code() ==
hdr_ieee1394::ASYNC_PAK;
  bool                                   is_isoch = hdr->trans_code() ==
hdr_ieee1394::ISOCH_PAK;

  return ((mac->subaction_gap && is_async && !mac->async_sent) ||
          (mac->isoch_gap && is_isoch && !mac->isoch_sent));
 }

 return false;
}

void                          ieee1394MacStateWaitToRequestArb::requestArb()
{Packet        *p2 = ieee1394Packet::alloc_packet();
 hdr_ieee1394                 *hdr = hdr_ieee1394::access(p2);
 hdr_cmn       *ch = hdr_cmn::access(p2);
 hdr_ip        *iph = hdr_ip::access(p2);
 Packet        *p3 = NULL;
 bool                         err;

 hdr->trans_code_ = hdr_ieee1394::REQ_ARB;
 ch->direction() = hdr_cmn::DOWN;
 iph->saddr() = mac->get_node_address();
 iph->daddr() = 0;

 p3 = p2->copy();

 if (!sendPacketUp(p2,err) && !err)
 {// we are the root node

  grantArb(p3);
 }
 else if (!err)
 {// we are not the root
  mac->transition_send_state.setNextState(&(mac->wait_for_arb_state));
  mac->changeCurrentState(&(mac->transition_send_state));
 }
 else // err == true
 {// we are not the root
  mac->changeCurrentState(&(mac->wait_for_arb_state));
 }

 Packet::free(p3);
}

void                          ieee1394MacStateWaitToRequestArb::grantArb(Packet *req_p)
{Packet                       *grant_p = ieee1394Packet::alloc_packet();
 hdr_ieee1394                                *hdr = hdr_ieee1394::access(grant_p);
 hdr_cmn                      *ch = hdr_cmn::access(grant_p);
 hdr_ip                       *grant_iph = hdr_ip::access(grant_p);

 if (canRequestArb())
 {
  hdr->trans_code_ = hdr_ieee1394::GRANT_ARB;
  ch->direction() = hdr_cmn::DOWN;
  grant_iph->saddr() = 0;
  grant_iph->daddr() = mac->get_node_address();
```

```
  mac->won_grant = true;
 }
 else
 {hdr_ip                      *iph = hdr_ip::access(req_p);

  hdr->trans_code_ = hdr_ieee1394::GRANT_ARB;
  ch->direction() = hdr_cmn::DOWN;
  grant_iph->saddr() = 0;
  grant_iph->daddr() = iph->saddr();
 }

 if (broadcastPacket(grant_p,NULL))
 {
  if (mac->won_grant)
  {
   mac->transition_send_state.setNextState(&(mac->wait_for_arb_state));
   mac->changeCurrentState(&(mac->transition_send_state));
  }
  else
  {
   mac->transition_send_state.setNextState(&(mac->wait_for_action_completion_state));
   mac->changeCurrentState(&(mac->transition_send_state));
  }
 }
 else
 {
  if (mac->won_grant)
  {
   mac->changeCurrentState(&(mac->wait_for_arb_state));
  }
  else
  {
   mac->changeCurrentState(&(mac->wait_for_action_completion_state));
  }
 }
}

// ******** ieee1394MacStateWaitForGrantArb **********

ieee1394MacStateWaitForGrantArb::ieee1394MacStateWaitForGrantArb(ieee1394Mac *m) :
    ieee1394MacState(m)
{
}

void                          ieee1394MacStateWaitForGrantArb::eventBecameCurrentState()
{
 if (mac->won_grant)
 {ieee1394MacState            *next_state = &(mac->idle_state);

  mac->won_grant = false;

  next_state = acceptArb();
  mac->changeCurrentState(next_state);
 }
}

void                          ieee1394MacStateWaitForGrantArb::eventIncomingArbReq(Packet
```

```
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 if (r != NULL)
 {
  r->cancel();
 }
 Packet::free(p);
}

void                          ieee1394MacStateWaitForGrantArb::eventReceivedArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 Packet::free(p);
}

void                          ieee1394MacStateWaitForGrantArb::eventIncomingArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void                          ieee1394MacStateWaitForGrantArb::eventReceivedArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{hdr_ip                       *p_iph = hdr_ip::access(p);
 bool                                    we_won_grant = (mac->get_node_address() ==
p_iph->daddr());
 bool                                    sent;
 ieee1394MacState             *next_state = &(mac->wait_for_action_completion_state);

 Packet::free(p);

 if (we_won_grant)
  next_state = acceptArb();

 mac->changeCurrentState(next_state);
}

void                          ieee1394MacStateWaitForGrantArb::eventSubactionGap()
{
 ieee1394MacState::eventSubactionGap();

 printf("ieee1394MacStateWaitForGrantArb::eventSubactionGap - invalid event - GRANT_ARB appears
lost\n");

 mac->changeCurrentState(&(mac->wait_to_arb_state));
}

void                          ieee1394MacStateWaitForGrantArb::eventFairnessGap()
{
 ieee1394MacState::eventFairnessGap();

 printf("ieee1394MacStateWaitForGrantArb::eventFairnessGap - invalid event - GRANT_ARB appears
lost\n");

 mac->changeCurrentState(&(mac->wait_to_arb_state));
}
```

```
void                          ieee1394MacStateWaitForGrantArb::eventIsochGap()
{
 ieee1394MacState::eventIsochGap();

 printf("ieee1394MacStateWaitForGrantArb::eventIsochGap - invalid event - GRANT_ARB appears
lost\n");

 mac->changeCurrentState(&(mac->wait_to_arb_state));
}

ieee1394MacState              *ieee1394MacStateWaitForGrantArb::acceptArb()
{ieee1394MacState             *next_state = &(mac->idle_state);
 bool                                    is_async = false;
 bool                                    is_isoch = false;

 if (mac->data_packet_pending != NULL)
 {hdr_ieee1394                *hdr = hdr_ieee1394::access(mac->data_packet_pending);

  is_async = hdr->trans_code() == hdr_ieee1394::ASYNC_PAK;
  is_isoch = hdr->trans_code() == hdr_ieee1394::ISOCH_PAK;
 }

 if (mac->restart_cycle_pending)
  next_state = &(mac->cycle_restart_send_state);
 else if (is_async)
  next_state = &(mac->async_send_state);
 else if (is_isoch)
  next_state = &(mac->isoch_send_state);
 else
  next_state = &(mac->wait_for_action_completion_state);

 return next_state;
}

// ******** ieee1394MacStateWaitForArbCompletion **********

ieee1394MacStateWaitForArbCompletion::ieee1394MacStateWaitForArbCompletion(ieee1394Mac *m) :
    ieee1394MacState(m)
{
}

void                          ieee1394MacStateWaitForArbCompletion::eventIncomingArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 if (r != NULL)
 {
  r->cancel();
 }
 Packet::free(p);
}

void                          ieee1394MacStateWaitForArbCompletion::eventReceivedArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 Packet::free(p);
```

```
}

void                          ieee1394MacStateWaitForArbCompletion::eventIncomingArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void                          ieee1394MacStateWaitForArbCompletion::eventReceivedArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 Packet::free(p);
 mac->changeCurrentState(&(mac->wait_for_action_completion_state));
}

void                          ieee1394MacStateWaitForArbCompletion::eventSubactionGap()
{
 ieee1394MacState::eventSubactionGap();

 printf("ieee1394MacStateWaitForArbCompletion::eventSubactionGap - invalid event - GRANT_ARB
appears lost\n");

 mac->changeCurrentState(&(mac->idle_state));
}

void                          ieee1394MacStateWaitForArbCompletion::eventFairnessGap()
{
 ieee1394MacState::eventFairnessGap();

 printf("ieee1394MacStateWaitForArbCompletion::eventFairnessGap - invalid event - GRANT_ARB
appears lost\n");

 mac->changeCurrentState(&(mac->idle_state));
}

void                          ieee1394MacStateWaitForArbCompletion::eventIsochGap()
{
 ieee1394MacState::eventIsochGap();

 printf("ieee1394MacStateWaitForArbCompletion::eventIsochGap - invalid event - GRANT_ARB appears
lost\n");

 mac->changeCurrentState(&(mac->idle_state));
}

// ******** ieee1394MacStateWaitForActionCompletion **********

ieee1394MacStateWaitForActionCompletion::ieee1394MacStateWaitForActionCompletion(ieee1394Mac *m)
:
    ieee1394MacStateIdle(m)
{
}

void                          ieee1394MacStateWaitForActionCompletion::eventBecameCurrentState()
{
 ieee1394MacState::eventBecameCurrentState();
}
```

```
void
               ieee1394MacStateWaitForActionCompletion::eventRequestSendData(Packet *p,Handler
*h)
{
 ieee1394MacState::eventRequestSendData(p,h);
}

void                          ieee1394MacStateWaitForActionCompletion::eventSendCompleted()
{
 ieee1394MacState::eventSendCompleted();
}

void                          ieee1394MacStateWaitForActionCompletion::eventSendCanceled()
{
 ieee1394MacState::eventSendCanceled();
}

void                          ieee1394MacStateWaitForActionCompletion::eventRecvCompleted()
{
 ieee1394MacState::eventRecvCompleted();
}

void                          ieee1394MacStateWaitForActionCompletion::eventRecvCanceled()
{
 ieee1394MacState::eventRecvCanceled();
}

void                          ieee1394MacStateWaitForActionCompletion::eventReceivedData(Packet
*p,ieee1394MacHandlerRecv *r)
{bool                         for_this_node = (mac->get_node_address() ==
hdr_ip::access(p)->daddr());
 bool                         isoch_pkt = (hdr_ieee1394::access(p)->trans_code() ==
hdr_ieee1394::ISOCH_PAK);

 ieee1394MacStateIdle::eventReceivedData(p,r);

 if (for_this_node || isoch_pkt)
 {
  mac->changeCurrentState(&(mac->idle_state));
 }
}

void                          ieee1394MacStateWaitForActionCompletion::eventIncomingArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 if (r != NULL)
 {
  r->cancel();
 }
 Packet::free(p);
}

void                          ieee1394MacStateWaitForActionCompletion::eventReceivedArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 Packet::free(p);
```

```
}

void
		ieee1394MacStateWaitForActionCompletion::eventIncomingArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void
		ieee1394MacStateWaitForActionCompletion::eventReceivedArbGrant(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // if this is for us, we are in the wrong state
 Packet::free(p);
}

void				ieee1394MacStateWaitForActionCompletion::eventIncomingAck(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void				ieee1394MacStateWaitForActionCompletion::eventReceivedAck(Packet
*p,ieee1394MacHandlerRecv *r)
{
 Packet::free(p);
 mac->changeCurrentState(&(mac->idle_state));
}

void
		ieee1394MacStateWaitForActionCompletion::eventReceivedCycleRestart(Packet
*p,ieee1394MacHandlerRecv *r)
{
 ieee1394MacStateIdle::eventReceivedCycleRestart(p,r);
 mac->changeCurrentState(&(mac->idle_state));
}

void				ieee1394MacStateWaitForActionCompletion::eventSubactionGap()
{
 ieee1394MacState::eventSubactionGap();

 mac->changeCurrentState(&(mac->idle_state));
}

void				ieee1394MacStateWaitForActionCompletion::eventFairnessGap()
{
 ieee1394MacState::eventFairnessGap();

 mac->changeCurrentState(&(mac->idle_state));
}

void				ieee1394MacStateWaitForActionCompletion::eventIsochGap()
{
 ieee1394MacState::eventIsochGap();

 mac->changeCurrentState(&(mac->idle_state));
}
```

```
void                          ieee1394MacStateWaitForActionCompletion::eventTimeToRestartCycle()
{
 ieee1394MacState::eventTimeToRestartCycle();
}

// ******** ieee1394MacStateAsyncSendData **********

ieee1394MacStateAsyncSendData::ieee1394MacStateAsyncSendData(ieee1394Mac *m) :
    ieee1394MacState(m)
{
}

void                          ieee1394MacStateAsyncSendData::eventBecameCurrentState()
{
 if (mac->data_packet_pending != NULL)
 {Packet       *pp = mac->data_packet_pending->copy();
  hdr_ip       *iph = hdr_ip::access(pp);

  broadcastPacket(pp,NULL);
  mac->changeCurrentState(&(mac->async_await_ack_state));
 }
 else
 {
  printf("ieee1394MacStateAsyncSendData::eventBecameCurrentState - invalid event - send state
should have a packet to send\n");

  mac->changeCurrentState(&(mac->idle_state));
 }
}

// ******** ieee1394MacStateIsochSendData **********

ieee1394MacStateIsochSendData::ieee1394MacStateIsochSendData(ieee1394Mac *m) :
    ieee1394MacState(m)
{
}

void                          ieee1394MacStateIsochSendData::eventBecameCurrentState()
{
 if (mac->data_packet_pending != NULL)
 {Packet       *pp = mac->data_packet_pending->copy();

  broadcastPacket(pp,NULL);

  Packet::free(mac->data_packet_pending);
  mac->data_packet_pending = NULL;
  mac->isoch_sent = true;

  {Handler     *h = mac->callback_;

   mac->callback_ = 0;
   h->handle(0);
  }

  mac->changeCurrentState(&(mac->wait_for_action_completion_state));
 }
 else
```

```
 {
  printf("ieee1394MacStateIsochSendData::eventBecameCurrentState - invalid event - send state
should have a packet to send\n");

  mac->changeCurrentState(&(mac->idle_state));
 }
}

// ******** ieee1394MacStateWaitForAck **********

ieee1394MacStateWaitForAck::ieee1394MacStateWaitForAck(ieee1394Mac *m) :
    ieee1394MacState(m)
{
}

void                          ieee1394MacStateWaitForAck::eventIncomingArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 if (r != NULL)
 {
  r->cancel();
 }
 Packet::free(p);
}

void                          ieee1394MacStateWaitForAck::eventReceivedArbReq(Packet
*p,ieee1394MacHandlerRecv *r)
{
 // ignore arb requests
 Packet::free(p);
}

void                          ieee1394MacStateWaitForAck::eventIncomingAck(Packet
*p,ieee1394MacHandlerRecv *r)
{
 broadcastPacket(p,r);
}

void                          ieee1394MacStateWaitForAck::eventReceivedAck(Packet
*p,ieee1394MacHandlerRecv *r)
{
 Packet::free(mac->data_packet_pending);
 mac->data_packet_pending = NULL;
 mac->async_sent = true;

 {Handler      *h = mac->callback_;

  mac->callback_ = 0;
  h->handle(0);
 }

 Packet::free(p);
 mac->changeCurrentState(&(mac->idle_state));
}

void                          ieee1394MacStateWaitForAck::eventSubactionGap()
{
```

```
 ieee1394MacState::eventSubactionGap();

 printf("ieee1394MacStateWaitForAck::eventSubactionGap - invalid event - ACK appears lost\n");

 mac->changeCurrentState(&(mac->wait_to_arb_state));
}

void                         ieee1394MacStateWaitForAck::eventFairnessGap()
{
 ieee1394MacState::eventFairnessGap();

 printf("ieee1394MacStateWaitForAck::eventFairnessGap - invalid event - ACK appears lost\n");

 mac->changeCurrentState(&(mac->wait_to_arb_state));
}

void                         ieee1394MacStateWaitForAck::eventIsochGap()
{
 ieee1394MacState::eventIsochGap();
}

// ******** ieee1394MacStateCycleRestartSend **********

ieee1394MacStateCycleRestartSend::ieee1394MacStateCycleRestartSend(ieee1394Mac *m) :
    ieee1394MacState(m)
{
}

void                         ieee1394MacStateCycleRestartSend::eventBecameCurrentState()
{
 if (mac->restart_cycle_pending)
 {Packet                     *cr_p = ieee1394Packet::alloc_packet();
  hdr_ieee1394               *hdr = hdr_ieee1394::access(cr_p);
  hdr_cmn                    *ch = hdr_cmn::access(cr_p);
  hdr_ip                     *iph = hdr_ip::access(cr_p);

  hdr->trans_code_ = hdr_ieee1394::CYCLE_RESTART;
  ch->direction() = hdr_cmn::DOWN;
  iph->saddr() = 0;
  iph->daddr() = 0;

  mac->restart_cycle_pending = false;

  if (broadcastPacket(cr_p,NULL))
  {
   mac->transition_send_state.setNextState(&(mac->idle_state));
   mac->changeCurrentState(&(mac->transition_send_state));
  }
  else
  {
   mac->changeCurrentState(&(mac->idle_state));
  }
 }
 else
 {
  printf("ieee1394MacStateCycleRestartSend::eventBecameCurrentState - invalid event - send state
should have a packet to send\n");
```

```
  mac->changeCurrentState(&(mac->idle_state));
 }
}

// ******** ieee1394Mac **********

ieee1394Mac::ieee1394Mac() :
    Mac(),
    mhSend_(this),
    mhCycleTimer_(this,0.000125),
    mhSubactionTimer_(this,0.00001),//(this,0.0000001),
    mhIsochTimer_(this,0.000005),//(this,0.00000005),
    mhFairnessTimer_(this,0.00002),//(this,0.0000005),
    trace_(0),
    current_state(NULL),
    transition_send_state(this),
    idle_state(this),
    wait_to_arb_state(this),
    wait_for_arb_state(this),
    wait_for_completion_state(this),
    wait_for_action_completion_state(this),
    async_send_state(this),
    async_await_ack_state(this),
    isoch_send_state(this),
    cycle_restart_send_state(this),
    data_packet_pending(NULL),
    restart_cycle_pending(false),
    subaction_gap(false),
    isoch_gap(false),
    async_sent(false),
    isoch_sent(false),
    won_grant(false)
{
 for (int i = 0; i < max_receivers; i++)
 {
  mhRecv_[i] = NULL;
 }

 // Bind mac trace variable
 bind_bool("trace_",&trace_);

 changeCurrentState(&idle_state);

 mhCycleTimer_.schedule();
 mhSubactionTimer_.schedule();
 mhIsochTimer_.schedule();
 mhFairnessTimer_.schedule();
}

ieee1394Mac::~ieee1394Mac()
{
 mhCycleTimer_.cancel();
 mhSubactionTimer_.cancel();
 mhIsochTimer_.cancel();
 mhFairnessTimer_.cancel();

 for (int i = 0; i < max_receivers; i++)
 {
```

```
  delete mhRecv_[i];
 }
}

void                          ieee1394Mac::changeCurrentState(ieee1394MacState *s)
{char          *cs = get_state_name(current_state);
 char          *ns = get_state_name(s);
 int           na = get_node_address();

 current_state = s;
 current_state->eventBecameCurrentState();
}

ieee1394MacState              *ieee1394Mac::getCurrentState() const
{
 return current_state;
}

void                                        ieee1394Mac::cancel_gap_timers()
{
// printf("testing: cancel_gap_timers PRE\n");
 mhSubactionTimer_.cancel();
 mhIsochTimer_.cancel();
 mhFairnessTimer_.cancel();
// printf("testing: cancel_gap_timers POST\n");
}

void                                        ieee1394Mac::schedule_gap_timers()
{
// printf("testing: schedule_gap_timers PRE\n");
 cancel_gap_timers();
 mhSubactionTimer_.schedule();
 mhIsochTimer_.schedule();
 mhFairnessTimer_.schedule();
// printf("testing: schedule_gap_timers POST\n");
}

int                           ieee1394Mac::command(int argc, const char*const* argv)
{Tcl           &tcl = Tcl::instance();

 if (argc == 5)
 {TclObject    *obj1, *obj2;

  if( (obj1 = TclObject::lookup(argv[2])) == 0 || (obj2 = TclObject::lookup(argv[3])) == 0 )
  {
   fprintf(stderr, "%s lookup failed\n", argv[1]);
   return TCL_ERROR;
  }

  if (strcmp(argv[1], "add-phy") == 0)
  {
   for (int i = 0; i < max_receivers; i++)
    if (mhRecv_[i] == NULL)
    {
     mhRecv_[i] = (ieee1394MacHandlerRecv *) obj2;
     mhRecv_[i]->downtarget_ = (NsObject *) obj1;
     if (strcmp(argv[4],"1") == 0)
      mhRecv_[i]->is_route_to_parent = true;
```

```
     return (TCL_OK);
    }

   tcl.result("all PHY connections used. increase max_receivers.");
   return (TCL_ERROR);
  }
 }

 return (Mac::command(argc, argv));
}

void                          ieee1394Mac::recv(Packet* p, Handler* h)
{
 BiConnector::recv(p, h);
}

void                          ieee1394Mac::sendUp(Packet *p, Handler *)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), index_,
__PRETTY_FUNCTION__);

 printf("ieee1394Mac::sendUp - should not get here - ieee1394MacHandlerRecv handles incoming
packets, and calls recv_from_phys()\n");
}

void                          ieee1394Mac::sendDown(Packet *p, Handler *h)
{
 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), index_,
__PRETTY_FUNCTION__);
 assert(initialized());
 assert(h);
 assert(data_packet_pending == NULL);
// assert(netif_->txtime(IEEE_1394_MINFRAME) > 2*netif_->channel()->maxdelay());

 if (data_packet_pending != NULL)
 {
  fprintf(stderr, "ieee1394Mac::sendDown - MAC is busy - already an existing send request.\n");
  exit(1);
 }

 hdr_cmn::access(p)->size() += IEEE_1394_HDR_LEN;

 if (hdr_cmn::access(p)->size() < IEEE_1394_MINFRAME)
 {// pad packet
  hdr_cmn::access(p)->size() = IEEE_1394_MINFRAME;
 }

 if (hdr_cmn::access(p)->size() > IEEE_1394_MAXFRAME)
 {static bool  warnedMAX = false;

  if (!warnedMAX)
  {
   fprintf(stderr, "ieee1394Mac: frame is too big: %d\n", hdr_cmn::access(p)->size());
   warnedMAX = true;
  }
  hdr_cmn::access(p)->size() = IEEE_1394_MAXFRAME;
 }
```

```
 current_state->eventRequestSendData(p,h);
}

void                          ieee1394Mac::recv_from_phys(Packet *p,ieee1394MacHandlerRecv *r)
{hdr_cmn       *ch= HDR_CMN(p);
 hdr_mac       *mh= HDR_MAC(p);

 FPRINTF(stderr, "%.15f : %d : %s\n", Scheduler::instance().clock(), index_,
__PRETTY_FUNCTION__);

 // Strip off the mac header and padding if any
 ch->size() -= (IEEE_1394_HDR_LEN + mh->padding_);

 // in case of transmision error
/* if( ch->error() )
 {
  fprintf(stderr,"\nChecksum error\nDropping packet");
  fflush(stderr);
  Packet::free(p);
 }
 else
*/

 {hdr_ieee1394                *hdr = hdr_ieee1394::access(p);
  bool                                   is_async = hdr->trans_code() ==
hdr_ieee1394::ASYNC_PAK;
  bool                                   is_isoch = hdr->trans_code() ==
hdr_ieee1394::ISOCH_PAK;
  bool                                   is_arb_req = hdr->trans_code() ==
hdr_ieee1394::REQ_ARB;
  bool                                   is_arb_grant = hdr->trans_code() ==
hdr_ieee1394::GRANT_ARB;
  bool                                   is_ack = hdr->trans_code() == hdr_ieee1394::ACK;
  bool                                   is_cycle_restart = hdr->trans_code() ==
hdr_ieee1394::CYCLE_RESTART;

  if (is_async || is_isoch)
  {
   current_state->eventReceivedData(p,r);
  }
  else if (is_arb_req)
  {
   current_state->eventReceivedArbReq(p,r);
  }
  else if (is_arb_grant)
  {
   current_state->eventReceivedArbGrant(p,r);
  }
  else if (is_ack)
  {
   current_state->eventReceivedAck(p,r);
  }
  else if (is_cycle_restart)
  {
   current_state->eventReceivedCycleRestart(p,r);
  }
  else
  {
```

```
   fprintf(stderr, "ieee1394Mac::recv_from_phys - invalid packet type\n");
   Packet::free(p);
  }
 }
}

bool                          ieee1394Mac::is_root() const
{
 for (int i = 0; i < max_receivers; i++)
  if (mhRecv_[i] != NULL && mhRecv_[i]->is_downtarget_to_parent())
   return false;

 return true;
}

nsaddr_t       ieee1394Mac::get_node_address() const
{
 if (netif_ && netif_->node())
  return netif_->node()->address();

 return -1;
}

bool                          ieee1394Mac::is_busy() const
{
 if (mhSend_.busy())
  return true;

 for (int i = 0; i < max_receivers; i++)
  if (mhRecv_[i] != NULL && mhRecv_[i]->busy())
   return true;

 return false;
}

char                          *ieee1394Mac::get_state_name(ieee1394MacState *c)
{
 if (c == &transition_send_state)
  return "ieee1394MacStateTransitionSend";

 if (c == &idle_state)
  return "ieee1394MacStateIdle";

 if (c == &wait_to_arb_state)
  return "ieee1394MacStateWaitToRequestArb";

 if (c == &wait_for_arb_state)
  return "ieee1394MacStateWaitForGrantArb";

 if (c == &wait_for_completion_state)
  return "ieee1394MacStateWaitForArbCompletion";

 if (c == &wait_for_action_completion_state)
  return "ieee1394MacStateWaitForActionCompletion";

 if (c == &async_send_state)
  return "ieee1394MacStateAsyncSendData";
```

```
 if (c == &async_await_ack_state)
  return "ieee1394MacStateWaitForAck";

 if (c == &isoch_send_state)
  return "ieee1394MacStateIsochSendData";

 if (c == &cycle_restart_send_state)
  return "ieee1394MacStateCycleRestartSend";

 return "unknown";
}
```

Note: Unnecessary comments were removed from the code.

`ieee1394Mac` is the central class for the ieee1394 LL and PHY layer (the combined portions we call the Mac). It uses many other classes to function, but these classes can be represented in three groups:

States: `ieee1394MacState` is the base class for 10 states that make up the Mac. The states and their transition diagrams a shown in Section 3. The use of separate objects to represent each state and transition are from the *Design Patterns* text, by Gamma, Helm, Johnson, and Vlissides.

Timers: There are 4 key timer classes; one for cycle restart timing, and three for idle timing to determine when the sub-action, isochronous, and fairness gaps have occurred. Timers are based on the `ieee1394MacHandler` class.

Transceivers: There is one sender instance of `ieee1394MacHandlerSend`, and up to `max_receivers` instances of `ieee1394MacHandlerRecv`. Primarily, these transceivers are used to time the arrival times of packets. Additionally, each receiver manages the incoming packets from an other ieee1394 LanNode, and directs the delivered packet to the ieee1394Mac instance, or to the appropriate ieee1394MacState instance. The sender instance does not manage the sending of packets, but only times them. The sending of packets is done directly via the `down_target` of the receivers.