A Link State Routing Algorithm for the Configuration of Interconnected HIPPI Switches

Ken Hardwick
Joel M. Halpern
W. R. Franta

Network Systems Corporation

Abstract

The 800 megabit HIPPI data channel has been defined by ANSI X3T9.3 for the purpose of connecting supercomputers, large scale peripheral systems, high performance workstations, and network layer routers. In addition to being a capable point to point medium, the HIPPI standard permits traffic to be routed through a series of intermediate crossbar switches, similar to telephone switching technology. The presence of crossbar switches requires a technique to control the selection of intermediate "trunks" between systems. The authors describe an implementation of link state routing algorithms which can be practically employed in this environment.

1: HIPPI and HIPPI switches

1.1: HIPPI Summary

The HIPPI (High Performance Parallel Interface) data channel is a standard defined by the ANSI X3T9.3 committee[1], which is dedicated to the concept of standardizing interconnections between devices in the gigabit per second range.

It can briefly be characterized by:

- A 32 bit I-field used to signal the downstream device. It is this wide selection field which makes the use of crossbar switch networks of HIPPI channels possible.

From the start, the HIPPI architecture was designed with the idea that source and destination hosts could be connected via one or more crossbar switches. These switches would examine the selection word on an incoming connect request and electrically switch the request to one of several output channel ports depending on the destination implied by the selection field. Sub-microsecond switching speeds within a crossbar switch would make this technology feasible for both peripheral and data communications traffic.

In the X3T9.3 document dealing with Switch Control[2], two important modes of addressing were specified: a source addressing mode where the host specifies the output ports on each switch in succession, and a destination addressing mode in which a 12 bit address of the destination was provided to the switch.

Because HIPPI switches can be scaled up to interconnected sets of channels and "fabrics" of trunk channels and switches, there is a natural tendency to think of it as a network in the conventional sense of the term.

HIPPI switches. Electronic crossbars permit an N port switch to sustain up to N concurrent gigabit connections. Series of switches can be used in larger networks, at the risk of internal blocking when all trunks between switches are busy.
1.2: Differences from classic LAN/WAN's.

The problem of interconnecting a fabric of HIPPI switches is very similar to the problems experienced with the routing of a network layer protocol such as IP or OSI CLNS; similar problems are also encountered in building configurations of MAC layer bridges. As will be seen, many of the solutions found for the problem of interconnecting LAN's with broadcast networks and point to point links also serve us in the HIPPI environment. However, the HIPPI crossbar and simplex channel environment bring new factors into the network topology:

- **Busy paths.** Unlike the messages of a packet switched network which can occupy a transmission medium for a carefully limited period of time, HIPPI connections can potentially last for wildly variable lengths of time, as there is no limit on the duration of a connection, the amount of data sent on a connection, or the net rate at which data is transferred.

- **Low forwarding penalty.** There is virtually no penalty in latency when additional hops are added; additional delay is only provided by the electronic propagation delay through the additional channels, and through the switch's internal passthrough electronics.

Although HIPPI provides a measure of distance insensitivity once data transfer commences, the connection and disconnection process requires signals to propagate between source and destination. Thus we wish to avoid paths which needlessly add geographic distance to the connection.

- **Simplex paths and channel pairing.** Popular LAN routing algorithms assume that all links are full duplex in the sense that receiving a message from a node implies the capability to transmit (and forward) to that node on the received message path. The architecture of HIPPI fabrics clearly does not require this: theoretically channels are unidirectional and a switching fabric can be constructed which arbitrarily connects switches and endnodes though unidirectional paths.

- **Limited address space.** Since HIPPI uses dedicated paths between nodes for the duration of a connection, it is not considered suitable as the sole medium for very large networks, its addressability is limited to 4096 nodes. We do not need the hierarchical addressing conventions found in network layer protocols or the incomplete routing information found in bridging (spanning tree learning bridge) protocols.

- **Unintelligent end nodes.** Due to HIPPI's nature as a data channel, we cannot presume a LAN model of the network which requires some participation on the part of end nodes. It must be possible to manually configure an end node so that no configuration traffic be sent to it or required of the end node.

- **No Inherent Broadcast or Multicast.** The HIPPI fabric is a web of point to point links between switch nodes. The propagation of routing information through multiple switches becomes a problem for the update process and not the medium.

- **Raw Speed.** The speed of the medium eliminates the need to economize on bookkeeping overhead.

2: Switch routing protocol objectives

Our principal objective can be stated as follows: given a set of end systems connected to a fabric of multiple HIPPI switches, how can a management facility determine the currently operational web of HIPPI connections which interconnect the switches, and how can it determine and utilize the best available paths between any source and destination end system?

A secondary objective could be to provide host end systems with information on their local addresses and potential network connectivity. We have documented such a scheme, but will ignore the problems of end system interactions in this paper.

2.1: Distributed solution

One approach to this problem consists of a central management station which will somehow survey the topology of the HIPPI fabric from its point of attachment to the fabric. Once the topology is determined, it then downloads every switch with a set of routing tables suitable for the transfer of data between nodes. The other alternative is a distributed solution where every switch in the fabric of HIPPI switches has the capability to independently decide how arriving messages will be routed based on information automatically provided by its neighboring switches.

We feel that a distributed solution has overwhelming technical merit, due to its greater robustness than a single management node.

2.2: Routing algorithms

The problem of route selection in a graph of distributed forwarding nodes is commonly addressed in MAC layer bridges and network layer routers. Many algorithms to resolve this problem exist; the three most popular approaches are:

- The spanning tree algorithm[3], now a standard for IEEE 802.1d MAC layer bridges. In fact, the spanning tree approach has been used for a network very similar to HIPPI in Autonet[4].

- Bellman-Ford algorithms which operate by each node being aware of its immediate routing neighbors and
the list of nodes declared to be reachable in turn by those neighbors.

- Link State algorithms which operate by having each forwarding node in the network becoming aware of all other forwarding nodes and the connectivity they are perceived to have.

We feel that the characteristics of the link state algorithm are particularly well suited to the problems of interconnecting HIPPI switches, due to its capability to calculate multiple paths based on a complete knowledge of the network topology.

### 2.3: Path Selection

A routing protocol must implicitly determine what treatment to give multiple paths in redundant networks. The simplest alternative is to select a single, working path between source and destination, and ignore all others. Link state protocols such as OSI IS-IS[3] and OSPF[4] take the stronger approach that traffic can be forwarded in one of several different directions, provided that all potential paths have equal cost to the ultimate destination, where “cost” is defined as a numeric value, or “metric” representing the expense of using a specific path. The choice of such multiple paths can in fact result in the out of order delivery of messages, but this condition is acceptable for most connectionless network layer protocols.

<table>
<thead>
<tr>
<th>Paths to reach Destination D</th>
<th>Node</th>
<th>Best 2nd 3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>From switch α</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>From switch β</td>
<td>E</td>
<td>F A</td>
</tr>
<tr>
<td>From switch γ</td>
<td>C</td>
<td>F B</td>
</tr>
<tr>
<td>From switch δ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Redundant switch paths and unequal cost forwarding. If δ is temporarily unable to accept messages from γ or β, a store and forward network would circulate packets between αβγ until they expire of “old age.” On a circuit switching network all paths ABF would be seized until it was determined that no further connections are possible, at which point the connection would expire.

Unequal cost path forwarding operates on the concept that a forwarding node has an ordered list of possible next hops towards a message’s destination. Given any message, it attempts to transmit it on the lowest cost path which is available or uncongested at the current time.

If only paths of equal merit are chosen in routing decisions, it is guaranteed that any forwarding action will move the message closer to the destination. If unequal cost paths are used, it raises the possibility that messages will backtrack to the point of being reintroduced on intermediate nodes through which they have already passed. This results in routing loops in store and forward networks.

In a connection oriented network such as HIPPI the effects of unequal cost paths are more benign. Although connections can proceed back to a switch through which the connection already passes, loops are not possible as the channels remain in use while the connection attempt is being made.

### 2.4 Switch management requirements

Discussion of a routing protocol between switches presume certain switch capabilities:

- **Unique ID.** The switch must have some reproducible means of referring itself to all other switches in the network. Since the HIPPI-LE specification[7] contains room for an IEEE administered unique address, it is suggested that every switch have an IEEE 48 bit address assigned to it.

- **Management Processor.** In order to execute a distributed routing algorithm, each switch must have an internal or directly attached supervisory processor which is capable of receiving information about neighboring nodes, accepting manual configurations for end nodes, and computing best paths based on that information.

- **Internal HIPPI connection.** Each management processor must have a built in HIPPI read and write connection, and the logic of the switch must permit the management processor to be addressable through a HIPPI connection from its neighbors in the HIPPI fabric. Furthermore, the management processor must have the capability to emit a HIPPI message on the output channel of its choice, and detect on which input channel an incoming frame addressed to it has arrived.

Control processors in adjacent HIPPI switches can communicate with each other using source routed addressing. By convention, the management processor is reachable through port 0 of its switch.
• **Channel control.** A management processor would provide poor quality routing information if it were unaware of the state of the channel connections in its switch. At a minimum, the management processor must be able to set the forwarding tables for each incoming port on the switch and determine if a connection physically exists on that port.

2.5: **Forwarding requirements**

Since performance of HIPPI strongly suggests that the process of output channel selection to be implemented in hardware, the forwarding process must necessarily remain simple. Since there are only 4096 possible destinations, the forwarding information for each can be obtained by simple table lookup keyed by destination.

If the only acceptable paths between a source and destination are ones with an equal (and lowest) cost, then the information needed in a forwarding entry is simply a mask of the HIPPI output channels which are eligible to forward the message to the destination from the path. Presuming that a non-busy channel can be selected from the candidates in a trivial length of time, the algorithm used (first available, random choice, etc) to determine which channel should be used is unimportant.

If we want to go further and choose paths which do not have the lowest known cost in the fabric, additional hardware forwarding capability is required, namely:

- The list of output channels suitable for forwarding to a given destination must be a prioritized list, so that paths with lower costs will be preferred over higher cost ones.
- Although no HIPPI I-field with destination addressing contains a history of its previous path through the network, it is certainly possible to avoid paths which will result in utilizing a channel leading directly back to the switch from which it arrived. One simple mechanism for this is to simply have a separate table of 4096 entries for each incoming channel; the RAM requirements remain modest, and directly returning paths can be removed for any input channel.

Note that this mechanism cannot in itself avoid "wrapped" paths involving three more more switches. It is the the routing protocol's job to eliminate such paths.

2.6: **Destination address allocation**

It is not possible to allocate destination addresses dynamically to end systems at initialization time; many data channel oriented protocols will need destination addresses which will remain fixed through network reconfigurations.

We propose that the destination address be considered a logical resource used to identify the facility desired by the requesting source node. They are manually assigned by the network administrator. Tables of these destination addresses can be configured into the software of host and peripheral systems, or automatically determined through multicast mechanisms and discovery protocols such as the TCP/IP ARP protocol.

When this facility is used by higher layer networking protocols, HIPPI destination addresses are manually associated with the network layer addresses of other network layer routers. This makes HIPPI topologically a "nonbroadcast LAN", similar to X.25, SMDS, or Frame Relay.

2.6.1: **Multihoming**

Many HIPPI nodes such as supercomputers and large scale peripheral systems are capable of servicing and sustaining several HIPPI channels at once. Similarly, several independent nodes on the same HIPPI network may be equally capable of servicing an atomic transaction from a requesting host. Since each of these channels to an endnode may be busy, the source node shouldn't care which of several equivalent channels are used at the destination node.

Multihoming can work beautifully with unequal cost path routing to provide fully redundant connections between hosts which shift traffic over to available destination channels whenever outages occur. The only restriction on a multihomed host appears to be that the switches providing access to a destination address must be no more than two hops apart for all paths to work. At greater distances traffic would be directed back to the busy or inoperative channel rather than forwarded to an alternative destination.

Since a requesting system has no mechanism to know which set of channels are currently available at the destination, it is desirable to have the HIPPI fabric solve this problem.
2.6.2: Aliasing

So long as addresses are administratively assigned, it is a great convenience to permit an end system to be the recipient of several destination addresses. This extends the concept of resource addressing to include the concept that a single destination may comprise more than one resource.

A secondary purpose of aliasing is that it permits the manually administered interconnection of HIPPI switch fabrics which may use other internal protocols for HIPPI address routing. These two subfabrics may be connected simply by manually assigning the set of destination addresses owned by the foreign fabric to all output channels leading to the foreign fabric.

2.7: Multicasting

Since HIPPI networks are physically a net of separate point to point connections, it is clear that any technique to send a message from a single source to multiple destinations will be nontrivial.

Through sufficiently ingenious switch control, it is possible to imagine a signalling implementation of multicasting service while remaining within the current HIPPI media standards.

We propose that the switch processors provide a multicast dissemination service to all interested nodes for name resolution purposes; such a solution can be implemented in switch software using straightforward HIPPI hardware. Vendors wishing to add value with more hardware intensive multicasting mechanisms may do so without affecting the logic of host software to process multicast messages.

3: A Link State Routing Protocol

3.1: The routing process model

Link state routing protocols use a four stage model to implement routing:

- An adjacency process to discover connectivity to neighbors. Nodes discover their attachments to other network devices by sending "Hello" messages to advertise their presence on its attached media, and listen for Hello messages from other devices. The adjacency process maintains the local link state, a list of networks and devices that are directly reachable from the local node and their general characteristics.

- An update process which detects and transmits changes in the local link state to other Intermediate Systems in the network, and receives link state updates from other nodes. If the network topology requires that action be taken to forward these updates to all points in the network, the update process does so.

- A decision process which uses the topology information acquired by the update process to determine how any given message arriving at the local node will be forwarded.

- A forwarding process which performs the mechanics of relaying messages given the information produced by the decision process. The forwarding process must be exercised at the time when a connection signal arrives on an incoming data channel on a switch.

3.2: Adjacency -- Determining local link state

Local link state is a data structure which contains all known and useful information about the HIPPI nodes adjacent to a switch. Principally it consists of the ID of the switch producing the message, the addresses of all end node systems which are directly reachable through one of the switch’s output channels, and information on the ports of other switches which are connected to its input channels.

3.2.1: Manually configured information

The presence of unintelligent end nodes requires more manually configured information than exists in most Local Area Networks. In our model, we assume that the data base of manually configured information is maintained local to the switch. In an operating network, remote management facilities could be used to maintain this set of data bases from a central location.

Switch input channels require no manual configuration from the vantage of switch routing. If a message is received from an input channel addressed to the switch
controlling process, it can then be presumed to be from another switch advertising its existence. All other messages originating from the input channel will be forwarded by the switch based on its current routing knowledge.

Output channels from the switch often require administrative information, namely the destination addresses assigned to reach the end system through this channel. The set of destination addresses may be null, indicating an end system which is only reachable through source routed addressing.

All channels not configured to lead to an identifiable End System are presumed to lead to another switch. Provision should be made for the administrator to provide a metric for that connection between switches.

3.2.2: IS Hello's

The Intermediate System Hello Message is the mechanism whereby switches identify their presence to one another, and ensure that they are in agreement as to link states.

All HIPPI switches participating in this scheme send an IS HELLO message periodically on all output channels which are both physically connected and not configured to lead to an End System. All these message use the source routed form of addressing in the I-field, and the message is directed to be sent to port zero of the next node encountered in the switch fabric; thus only the switch management processors of neighboring HIPPI switches should be reached.

Conversely, the switch should be prepared to receive IS Hello messages on all its input channels at any time. When addressed to the control processor, they will be used both to mark that the input channel is the downstream end of a connection to another switch, and to record the identity of the switch which sent the message.

IS Hello messages contain the following information:
- The domainID of the HIPPI routing domain. The domain ID is used to designate an independently configured switch fabric at a site which may contain several such fabrics. Checking of domain ID's will ensure that chaos does not ensue if two independently configured switch fabrics become inadvertently connected.
- The switchID of the switch originating the message.
- The port number of the output channel on which the transmitting switch sent the message.
- A table of all known HIPPI switches (including the originator) participating in this routing protocol, along with the last link state sequence number received from each.

3.3 Update -- Propagating link state

Once the local switch has correctly surveyed its immediate vicinity, it now has the task of sending that information to other nodes. It initiates such transmissions under four circumstances.
- At long, periodic intervals (on the order of several minutes).
- Whenever an adjacent systems comes into service. Intermediate systems and intelligent end systems are brought into service when the first Hello message is received from them. Unintelligent end systems are brought into service if the Interconnect channel signal from the attached node is received on the output channel to the node.
- Whenever an adjacent node goes out of service. This occurs if the Interconnect signal is dropped on the output channel to an end system, or the input channel from another intermediate system. Switch interconnections are also taken out of service if a Hello message is not received from them in a multiple of the Hello message transmission interval.
- Whenever an administrative change is made that affects the link state.

The link state message consists of the information discovered via the adjacency process, and includes a sequence number so its recipients can ignore duplicate receipts of the message.
The Link State message processing begins by transmitting it to all output channels which are potentially leading to intermediate systems, and which have received previously transmitted IS Hello messages without a physical channel error. This begins the flooding process.

3.3.1: Flooding

Flooding is the process whereby link state messages from every node are sent throughout the switch fabric. The flooding algorithm works as follows.

- The switch management processor receives a Link State Update message from one of its input channels.
- It compares the sequence number of the update message with the sequence number last received from that originating node. If the sequence number of the newly arrived message is not greater than a previously received message, the update message is discarded.
- If the link update contains new material, it is added to the switch's link state data base, then the message is relayed out all output channels believed to have a switch at the other end.

In the flooding process, redundant copies of messages will be sent to switches with redundant connections to the rest of the network. Since no switch will propagate the same message twice, the possibility of looping is avoided.

Since HIPPI connections between switches tend to be highly redundant, it is useful heuristically to only flood one copy of a message to a switch which is known to be the next destination of several of the switch's output channels. Since we are propagating a link state away from its source of change, it is safe to rely on the network topology acquired before the link state update arrived to propagate it further.

3.3.2: Synchronization

Flooding will perform the task of delivering topology changes to all active nodes in the network in a short interval of time. However, it will not handle the case of switches coming into service after installation, reset or extended outage.

To solve this problem, the Link State Query facility is provided. It consists of a request for the latest link state update from a switch, and a response containing that information. The Query is made to a neighboring switch. Thus any switch must be able to construct a Link State Update for every other (known) switch on the HIPPI fabric on demand.

Whenever a switch receives an IS Hello message on an input channel from one of its neighbors, it will receive a list of all routing switches and the last Link State Update sequence number received from each in that message.

Should the receiving switch find itself out of date, it will direct a Link State Query message giving the desired switch ID and sequence number on all output channels that may lead to an active switch.

Any switch receiving a Link State Query message will generate a response when:

- It knows the link state of the switch identified in the query, and has a sequence numbered version greater than or equal to the one requested.
- According to its link state data base, one of its output channels is connected to an input channel of the requesting switch. This will generally be known when the requesting switch sent a Link State Update to the current node via the same channel as the query, providing information learned when the current node generated an IS Hello on the channel.

Note that if a switch receives a query for a version of link state of which it is not aware, it should not in turn attempt a query for that version. Queries should only be made if some node asserts it possesses the new link state; otherwise requests for rumored but nonexistent link states will endlessly flood the network if a single node generates an erroneous query.

3.3.3: Link State Response Flooding

In general topology networks the link state query process is straightforward; on HIPPI switches interconnected with simplex channels life complicates because a switch with incomplete network topology does not know how to send to its neighbors simply because it

![Diagram](image)

Link Response Flooding. In this slightly pathological case, if $\gamma$ finds itself out of synchronization it cannot execute a transaction with any single switch to resynchronize itself. Accordingly, the query is flooded so that $\alpha$ will detect the query and return a response on the channel $\epsilon$. 

367
has received an IS Hello from them. This problem is handled through a flooding process:

- Using the multicast capability defined in §3.6, the switch initiating the query internally generates a multicast packet directed to the multicast group of "all switches." It transmits the properly encapsulated multicast message on all active output channels not configured to lead to an end system.
- Neighboring switches receiving this message use normal multicast mechanisms to deliver it to all switches. However, since it is addressed to "all switches" each retains a single copy for processing by the switch management software.
- When the switch software recognizes the query message, it floods the Response on all output channels which are active and lead to either an unknown destination, or which are known to lead to the querying switch. The response contains not just the link state for a single switch, but the state for all switches known to the sender. Due to the volume of information, this may be delivered in the form of several, independent Response messages.

3.4: Decision -- Determining the best path

We envision the process of computing the best paths for switches in these three separate steps:

- Produce a usable graph of the HIPPI switch topology, with information on all paths outbound from every switch contained in the link state database.
- Use that graph to compute the N best paths to every other switch from our switch, where N is the number of possible paths supported by our forwarding process.
- Use the best interswitch paths and the information on end systems attached to each switch in the link state data base to construct a forwarding table for use by the forwarding process for every input channel on the switch.

3.4.1: Ingraph vs. Outgraph

Many graph theory algorithms have been written to extract desirable information from an arbitrary graph of nodes connected by paths of varying costs; however, most algorithms presume a list of simplex paths leading out of each node on the graph. In our case, our link state data base is a modification of this concept, in that with each node we have information about incoming intermediate systems, and outgoing end systems.

Our first task is to take \( D[S,p] \), the link state for input port \( p \) on switch \( S \), and produce the graph \( G[T,q] \) which contains the switch directly reachable from output port \( q \) on switch \( T \). Since we have the inbound connections for every switch in the fabric, this is a fairly trivial process and produces a working graph free of information extraneous to the path calculation process.

3.4.2: Metrics

So far we have not defined the criteria used to pick the "best" path among the (possibly) many candidates, or the further criteria for ranking alternative paths in order of merit. Link state algorithms use the concept of a metric, a dimensionless quantity defining the net cost of a multihop path through the network.

In principle, a metric can represent any real world quantity one pleases, so long as it follows the criterion that the total metric for a multihop path is the sum of the metrics for the individual hops. Multiple metrics may be exchanged between intermediate systems so long as calculation for any given path uses a consistent set of metrics. Possible candidates for metrics on HIPPI are:

- Hop Count. The cost of a path to a destination is simply the sum of the number of channel connections in the path. Channel connections considered expensive may have higher integral values assigned to them.
- Delay. Theoretically hardware on a HIPPI switch could determine the time needed to for each port to connect and compute metrics based on connection delay.

We intend to proceed with a hop count metric at the current time; it is straightforward to calculate and permits the network administrator to easily set policy for the use of intermediate links on their HIPPI fabric.

3.4.3: N best paths algorithm

In the second stage, given the previous graph \( G \), and the set of end systems reachable from each switch, we wish to compute the array \( \text{Route}[E, idx] \), where each \( idx \) row consists of an ordered list of the N best output channels to be selected to reach the end system \( E \), as well as the array \( \text{Cost}[E, idx] \) giving the cost of reaching \( E \) via the port indicated in \( \text{Route}[E, idx] \).

The algorithm used is a generalization of the Dijkstra Shortest Path First[8] algorithm, and will produce the N shortest paths to all end systems and all switches. The switches are computed to help prune the search.

The Dijkstra algorithm uses a "Tent" to hold tentative paths, and a final result is produced successively from this. In the Dijkstra algorithm, only one route to a given destination is in the tent at a time, since the basic approach only requires one path. For our algorithm, instead of pruning if there is a single path, we continue to put entries into the tent unless there are too many final
results. This check for final results must be done both when putting things into the tent, and when removing things from the tent.

The first step in the algorithm is to initialize the "Tent" with all directly reachable end systems and switches. Then, one repeatedly takes out the smallest entry in the tent and examines it, until the tent is empty.

If the entry removed is an end system and the route to it is considered legitimate, add this route to the Route and Cost tables. In all cases, legitimate routes are those which haven’t exceeded the N entries we’re looking for, don’t exceed a maximum cost, and are not looping back on themselves.

If the entry removed is an intermediate system and the path to it is still legitimate, add this route to it to the record of switch routes. Then, take every legitimate route from this intermediate system to all other intermediate systems and add them to the tent.

This algorithm will result in processing all paths to intermediate or end systems, in Cost order. It will prune useless branches as soon as practical. It is possible to add a refinement which prunes paths out of the Tent when adding paths to it, but this is probably an excessive complication.

3.4.3: Forwarding table construction

At this point, we have an ordered list of the output channels needed to reach each switch in the network, and the cost associated with reaching that switch. The task is now to compute the forwarding table array Fwd[in,addr,out], a table providing an ordered list of output channel numbers for each of the 4096 addr destination addresses, separately computed for every input port.

This is also pretty straightforward with the information at hand:

For every addr 0..4095:

Find the set of switches D which has end system addr directly attached to them. Produce Dest[T,p] which is identical to Route[S,p] save that Dest contains only those switches T ∈ D.

If D is null, then continue with the next addr.

For every switch input channel:

For i ← 1..N or Dest empty:

Find the output channel p in Dest with the lowest cost path shown in Cost[S,p].
If G(H,p) points to the same switch as D(H,in) for our own switch H, then we would be forwarding a message directly back to the switch it came from; skip the next step.
Set Fwd[in, addr,p] to the output channel in Dest with the lowest cost path shown in Cost[S,p].
Remove all entries to channel p in Dest.

This will produce a table of up to N-1 paths for each input port, destination address pair.

3.5: Forwarding -- Forwarding messages

As was mentioned earlier, the forwarding process is very time critical in that a HIPPI switch may potentially be processing hundreds of thousands of connections each second. Much of the thought process that has gone into this paper has been toward producing results (algorithms) in which a hardware implemented forwarding algorithm is possible, practical, and economically non-demanding.

The forwarding data base consists of specialized, shared RAM which is can be written by the switch control processor and is readable by the hardware routing logic.

For each port, a table with 4096 entries exists in forwarding RAM. Whenever a message arrives, the destination destination address is used to index the table for that port. Each entry in the table contains an ordered list of output channel numbers. This list may be null to indicate that it is not possible to forward a message from that address. The maximum number, N, of entries in the list obviously does not need to exceed the number of output channels on the switch; the actual maximum may be less and does not have to agree with the maximum of any other switch in the fabric.

Source routing and I-field validation are part of HIPPI-SC and are not covered here. The steps specific to routing are:

1. Extract the destination address addr from the I-field. Obtain the forwarding table entry Fwd[in, addr,*].
2. For every entry in the list Fwd[in, addr,*], determine if the output channel designated by the entry is available for transmission. If it is, direct the message to that output channel. If it is not, proceed to the next entry in the list. If Single Path Selection is set in the I-field, reject the message if the first entry in the list is busy.
3. If all entries in Fwd[in, addr,*] are exhausted, reject the message.

3.6: Multicasting implementation

Since current HIPPI channel architecture inherently permits only an atomic transmission of a message from a
single source to a single destination, any multicast mechanism must use some flooding technique to disseminate messages to all interested hosts on the fabric. We propose that a multicast service suitable for name resolution only be implemented through the agency of the switch management processors.

As multicast messages are sent in multiple directions in a redundantly connected network, a mechanism must be provided so that these messages do not endlessly loop through the network. The most common solution is to implement a Spanning Tree algorithm so that every switch and host gets exactly one copy of the message. Since this requires an independent protocol to Link State simply to propagate multicasts, we propose a flooding mechanism based on the Link State information:

1. HIPPI-SC switch control reserves certain I-field addresses for multicast groups.
2. All output channels to end systems are profiled to indicate the multicast groups to be received by that host.
3. Whenever an end system wishes to initiate a multicast, it generates an I-field with the correct multicast group.
4. The forwarding mechanism directs all multicast messages to the control processor on the local switch. The message is received in its entirety before further processing commences.
5. On receipt of the message, the control processor stamps it with a sequence number unique to that switch. It encapsulates the message with its switch ID and multicast sequence number and passes it to the Link State Flooding algorithm.
6. Link State flooding sends the encapsulated message to all adjacent switch processors. They determine if they have seen a multicast with this sequence number and switch ID before and ignore it if they have. Otherwise they flood the message further if appropriate nodes exist.
7. When the control process receives a sequenced multicast message for the first time, it decapsulates the message and presents a copy of the original multicast message to all of its output channels which are both connected to an end system and are configured to receive that multicast group.

4: Conclusions

We have taken the principles of general purpose routing protocols and adapted them to the specialized environment of circuit oriented crossbar switches. It should prove applicable to HIPPI whenever multi-switch networks approach any real complexity, and should prove valuable where metropolitan distance extensions add an element of unreliability to inter-switch connections.

The algorithms used take advantage of the fact that paths cannot be re-entered as data progresses from source to destination and are not directly applicable to many upcoming switching technologies such as ATM. However they are applicable to such items as automated video and PBX switching systems where peer-to-peer connectivity between switching systems is preferred over a highly administered hierarchy of switches.

5: Bibliography