RC 20922 (07/15/97)
Computer Science/Mathematics

IBM Research Report

A Hybrid Spanning Tree Algorithm For Efficient Topology Distribution In PNNI

Erol Basturk

IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Paul Stirpe

Reuters
88 Parkway South
Hauppage, NY 11788

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

A Hybrid Spanning Tree Algorithm for Efficient Topology Distribution in PNNI

Erol Basturk

IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10562
phone: 914-784-6625
Fax: 914-784-6205
basturk@watson.ibm.com

Paul Stirpe

Reuters
88 Parkway Drive South
Hauppauge, NY 11788
phone: 516-851-6170
Fax: 516-233-6916
paul.stirpe@reuters.com

Abstract

The ATM Forum's PNNI standard specifies a flooding mechanism to distribute topological state information amongst nodes participating in a PNNI network. While the flooding mechanism provides robust topology distribution, we show by simulation, that it can disproportionately overload lower-bandwidth links and, due to its inherent redundancy, can generate considerable computational overhead due to the processing of redundant topology updates. To address these issues, we introduce the Hybrid Spanning Tree algorithm, a spanning tree-based topology distribution mechanism that has low computational maintenance and can support policy that restricts topology distribution control traffic from being carried over lower-bandwidth links. Unlike other spanning tree proposals, the Hybrid Spanning Tree algorithm provides a simple and practical migration path to smoothly transition PNNI nodes executing the flooding algorithm to that of the Hybrid Spanning Tree algorithm.

1. Introduction

In the ATM Forum's PNNI specification [1], each node in a PNNI network maintains a fully replicated topology database specifying the state of each link for all nodes within the network cluster boundary. The PNNI specification specifies a hierarchical topology representation of a network. Network clusters are groups of network nodes that typically reside in the same administrative domain. Each node within a given network cluster maintains a full topology representation of all nodes and links within its cluster. The topology of nodes and links outside a cluster is abstractly represented by an aggregation mechanism, and subsequently provided to other network clusters in its abstract form. Such information is less exact, but provides a mechanism for the PNNI architecture to scale to large networks. While it is not known at this time how well the PNNI architecture scales, it has been architected to allow for many levels in the PNNI hierarchy.

Many of the services provided by the PNNI standard directly utilize the topology database. For example, route computation is carried-out using the link state algorithm that relies on the content of the topology database. A call initiated at a node may require bandwidth reservation and/or quality of service (QoS) guarantees. Route computation is performed using the topology database in order to find a path that meets the specified connection characteristics. The integrity and accuracy of the topology database is therefore paramount in computing accurate routes and in meeting the connection characteristics of calls.

1.1 What's Wrong with Flooding

The PNNI specification utilizes a flooding algorithm for standard topology distribution. Topology state information is contained in a PNNI Topology State Element (PTSE) which is carried in a PNNI Topology State Packet (PTSP). When a node receives or generates a topology state update that is new or of more recent origin than the node's current copy, the node floods the PTSE to all its links, except for the link on which the information was received. A PTSE sent to a neighboring node is retransmitted until acknowledged. A PTSE contains information about the state of a given link. A node may group several PTSEs into a PTSP, prior to forwarding the PTSP to a neighboring node. From here onward, we interchangeably use the PTSP/PTSE terminology with the understanding that one or more PTSEs may be contained in a PTSP, and a PTSP is the fundamental unit of information sent between nodes for topology distribution.

Flooding has several good properties. It provides a good fault-tolerant mechanism that can aid in reducing the amount of synchronization between nodes following a network failure. Flooding is simple to implement, robust and works well in networks where link failures are common, justifying the use of a more heavy-weight solution.

However, the inherent redundancy of the flooding distribution algorithm comes as a disadvantage by creating extra traffic and processor interruption at each node in the cluster level network. While we like to think of bandwidth as relatively *free* in high speed networks, the economic reality is that bandwidth is still expensive in wide-area networks (WAN), often accounting for the majority of the overall network cost [5]. Furthermore, it is thought that extraneous processor interrupts can place a heavy burden on the control processors that execute the PNNI control point.

The traffic and processor load imposed on each node by the flooding topology distribution algorithm is a function of the frequency with which the topology state information changes. Topology state information consists of static and dynamic components. For example, a policy related attribute for a given link may remain unchanged for the life of a network. Alternatively, the metric link state information may change more frequently depending on the changing load imposed on a given link. A PTSE may be triggered periodically or may be event driven as a result of meeting a threshold. For example, if the bandwidth allocation on a given link does not change for a long period of time, a periodic PTSE may be generated to refresh all the nodes within the given cluster as to the state of a link. Alternatively, if the bandwidth on the link is rapidly increasing and decreasing as the result of fluctuations in the link call loading, a new PTSE may be generated frequently (each time a threshold is crossed). This provides the other network nodes with the most recent bandwidth information on the given link, and thus the ability to make more intelligent and successful routing decisions.

Several observations may be drawn from the above discussion. First, the *static* components of the topology state information has little loading effect on the network in the steady state and is not a significant source of control traffic and processor utilization. Once the topology database is synchronized, the static state information changes infrequently. Therefore, the loading effect on the network may, for all practical purposes, be ignored. Second, the network performance is a function of the *dynamic* components of the topology state information that are specified by the periodic PTSE generation rate for each link, the PTSE trigger thresholds for each link, the call arrival and departure rates on each link, and the call connection characteristics of arrivals and departures. As the PTSE generation rate is increased and trigger thresholds tightened, the accuracy of the topology database is improved (updates are more frequently generated).

1.2 Spanning Tree Approaches for Topology Distribution

Instead of using flooding, a spanning tree connecting the nodes in a cluster can be used to distribute the PNNI topology information more efficiently. The advantages compared to flooding is the reduction of control traffic and processor utilization, but at the cost of maintaining the spanning tree and assuring consistency of the topology in case of spanning tree failures. Also, a spanning tree approach provides the ability to customize the tree to use preferred links (i.e. higher speed, less loaded), potentially leading to an improvement in network utilization and more rapid topology distribution. A spanning tree solution works best in relatively stable networks, where link failure is uncommon, reducing the cost of tree maintenance and topology synchronization.

Several hardware-assisted spanning tree based approaches for topology distribution have been proposed [3][4], which provide reasonable solutions for improving the efficiency of topology distribution. However, we believe that a solution, whatever it may be, should meet all of the following basic requirements.

- It must provide the same semantics as that of the flooding mechanism.
- It must be able to be phased into the current PNNI standard and enable a smooth migration path from the current flooding mechanism to a spanning tree based approach. That is, inter-operation of a flooding node and a spanning tree node must be possible for intra and inter cluster topology distribution[1].
- It must provide a marked improvement in link utilization and/or processor utilization for PTSE processing.
- It should incur a computational overhead that is similar or less than that of the flooding topology distribution mechanism.

Although we initially believed that all of the above requirements could be met by the Hybrid Spanning Tree

1. This implies that new hardware assisted addressing modes should *not* be required by the ATM switches.

algorithm, it was discovered that for relatively high speed networks with *homogeneous* speed links, the potential benefits of a spanning tree distribution are not as significant in terms of bandwidth utilization for control traffic. However, in networks that have heterogeneous link speeds, a spanning tree topology distribution mechanism can provide the necessary policy to restrict control traffic to be carried only over the links that have the adequate capacity. Furthermore, it is shown that the spanning tree approach does provide significant savings in messaging overhead, thus dramatically reducing the number of messaged processed by PNNI nodes.

1.3 The Hybrid Spanning Tree Topology Distribution Algorithm

The Hybrid Spanning Tree (HST) algorithm provides the same semantics as that of the flooding algorithm. HST is characterized as a hybrid algorithm because it allows nodes that utilize the spanning tree for topology distribution to interoperate with nodes that continue to utilize flooding. Furthermore, the interaction is transparent to the nodes that utilize flooding. Thus, a smooth migration path is established in which a PNNI node within a cluster can be "loaded" with the HST code and brought on-line independent of the other nodes in the cluster level network[1]. The HST algorithm may operate at the bottom level cluster or amongst peer group leaders (PGL) at any level in the PNNI hierarchy. We provide a method by which the HST algorithm may be implemented, without requiring adoption by the ATM Forum as a standard[2]. This is accomplished by use of the implementation "hooks" provided within the PNNI standard for proprietary extensions. The HST algorithm meets the requirement for improved link utilization and reduced processor computational load, as do most of the spanning tree based approaches.

In Section 2, the HST spanning tree algorithm is introduced. Migration from flooding to the HST algorithm is detailed. Furthermore, two optimization of the HST algorithm, Minimal Isolated Spanning Tree and Minimal Isolated Spanning Subtree are introduced. In Section 3, we present the performance analysis of flooding versus spanning tree for topology distribution, including a description of the simulation model, results and conclusions.

1. Migration is often overlooked and as a result can incur a high cost in terms of practical installation of such a control point improvement.

2. Although, adoption by the ATM Forum of the HST topology distribution algorithm is encouraged.

2. A Hybrid Spanning Tree

We specify the HST algorithm which allows an easy migration to a spanning tree distribution mechanism and provides improved performance within a heterogeneous cluster of standard PNNI nodes and HST nodes. This heterogeneity could be encountered at level in the PNNI hierarchy where standard PNNI logical nodes that utilize flooding are intermixed with HST nodes.

The basic idea of HST is as follows: using a simple migration strategy (Section 2.1), a PNNI node is upgraded to a HST node. This process is performed independently at each node and imposes minimal disruption on the PNNI network. As a result of this migration, an enhanced distribution mechanism, compatible with regular PNNI nodes, is performed by each HST node upon receipt of topology updates (Section 2.2). To provide this new service, the topology database needs to be extended (Section 2.3) with a new type-length-value (TLV) to allow each HST node to advertise its ability to participate in the HST algorithm. Based on the content of this extended database, each HST node determines to which unique minimal spanning tree or subtree it belongs. Two versions of the HST algorithm are presented (Section 2.4). The maintenance of the minimal spanning tree(s) is triggered by different conditions (Section 2.5). Depending on the assumptions made on the type of network, one algorithm may be preferable for implementation (Section 2.6).

2.1 Migration to HST

Minimal disruption of the network is at the core of the migration strategy and greatly influences the design of the hybrid spanning tree. The migration proceeds by taking down one node at a time, upgrading the node with the new software (HST algorithm), and then the node is brought back on-line. Besides the normal topology updates generated by neighboring PNNI nodes, no other disruption is imposed on the rest of the network. A PNNI (non-HST) neighbor node need not perform any new actions upon receipt of updates from the HST nodes other than those already prescribed by the PNNI standard.

2.2 Hybrid Distribution Mechanism at HST Nodes

The purpose of building a spanning tree is to allow efficient topology distribution to take place in an heterogeneous environment of standard PNNI nodes and spanning tree-capable PNNI nodes (or HST node). To allow

such a selective distribution capability, an HST node, upon completion of the spanning tree computation, can classify its immediate links into three categories:

1. *A flooding link* will denote a link where the adjacent node is not a HST node.
2. *An active spanning tree (or ST) link* is an edge of the spanning tree.
3. *A potential spanning tree (or ST) link* is a link connecting to another HST node but not on the current spanning tree.

This differentiation having been made, we now describe how the standard flooding mechanism can be made more efficient by selecting where to forward topology updates. Upon receipt of a topology state packet (PTSP), a HST node acts exactly like a PNNI node, with the exception of the forwarding function. In both a PNNI node and a HST node, only the subset of outgoing links excluding the link from where the update was received, is considered by the forwarding function. In PNNI, new updates are forwarded indiscriminately onto those links. In a HST node, however, they are forwarded only onto the active ST links and, if any, onto the remaining flooding links. With this *hybrid distribution* mechanism, a HST node defaults to standard PNNI flooding if it has only flooding links, or to spanning tree if it has only HST links. An example of a PNNI cluster hierarchy using such an enhanced distribution mechanism is given on Figure 1. As shown, the different types of links to forward on are present at different level of the hierarchy.

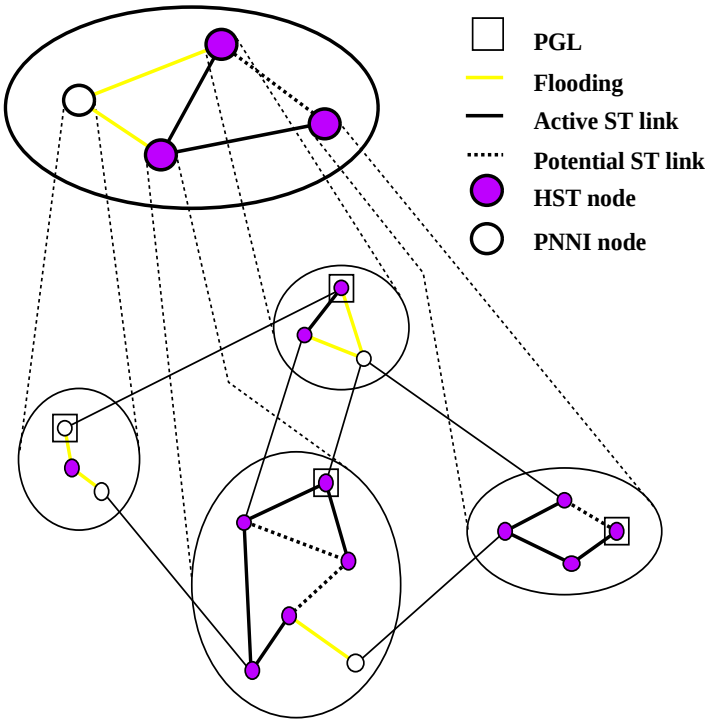

Figure 1. Two layers of hybrid clusters

2.3 Extending the Topology Database

To allow HST nodes to discover information about other node's capabilities, the content of the topology database needs to be extended. During normal PNNI topology database exchange and distribution, the additional information can be *piggybacked* in a type-length-value (TLV) embedded in a "Systems Capabilities" field of the PTSE messages. This field is marked with an IEEE organization unit and can actually be included in any other type of PNNI packet, including Hellos. It would indicate that a particular node in the cluster has the ability to execute the HST algorithm. Thus, the set of nodes, links and the link costs would be provided as input to the HST algorithm executing at each HST capable node[1]. Each HST nodes includes the following information in the PTSE that it originates:

- An indicator of HST spanning tree support.
- For each outgoing link being advertised, a unidirectional cost parameter, and possibly a port identifier to which this link is connected (see Appendix A).

The amount of additional information included in the TLV of a PTSE is relatively small. Furthermore, it is conveyed to other HST nodes without any impact on regular PNNI nodes.

2.4 Minimal Spanning Tree Algorithm

As presented in Section 2.2, the distribution of topology updates is improved by letting the forwarding function select links of appropriate status. The status is assigned and maintained based on the output of the HST minimal spanning tree algorithm that is run independently on each HST node. The algorithm on a particular node uses the additional TVL information contained in

1. In our model, a node owns its outgoing links, which are unidirectional, and assigns a cost to each of them. The cost of the bidirectional link is a function of the two unidirectional costs and is determined by the minimal spanning tree algorithm (see Section 2.4 and Appendix A).

its topology database to determine the configuration of the minimal spanning tree to which this node belongs. This configuration is directly used to assign a status to the outgoing links. Assuming a consistent topology and a set of unit link costs, the algorithm on every node of a specific tree will produce the same result. Even though the topology may not be consistent at all instances, the algorithm will ultimately converge because each new computation will allow the diffusion of new updates, gradually reducing the topology inconsistency.

We present in Figure 2 a slight variation of Prim's algorithm [2] to build a spanning tree of minimal cost. The reason for the specified change is to accommodate the presence of nodes in the graph that are not eligible to join the spanning tree (non-HST capable nodes), which could potentially create several disjoint trees.

```
MIST-mPrim(G, w, r)
1     P ← {r}
2     ω[r] ← 0
3     π[r] ← NIL
4     T ← ∅
5   while P ≠ ∅ do
6       P ← P − { u | ω[u] = min_{x ∈ P} ω[x] }
7       T ← T ∪ {u}
8       for each v ∈ Adj(u) and v ∈ C and v ∉ T do
9         if v ∈ P then
10            if w(u, v) < ω[v] then
11                ω[v] ← w(u, v)
12                π[v] ← u
13            end if
14        else
15              P ← P ∪ {v}
16              ω[v] ← w(u, v)
17              π[v] ← u
18        end if
19  end while
```

Figure 2. Minimal Spanning Tree Algorithm

2.4.1 Motivation

According to [2], there are 2 major classes of algorithms to build a minimal spanning tree. Kruskal's algorithm and Prim's algorithm. Kruskal's algorithm sorts the edges and builds the minimal tree by forming separate trees and successively joining them together. Prim's algorithm grows one tree from a root vertex, successively adding nodes until the spanning tree is complete.

If we denote by $G(E, V)$, a graph with a set E of edges and a set V of vertices, both algorithms can be shown to yield similar performances, of the order $O(E\log E)$. However, Prim's algorithm can be more efficiently implemented using Fibonacci heaps for sorting, yielding $O(E + V\log V)$.

In both Kruskal's and Prim's algorithms, the assumption is made that all the graph vertices are to become part of the spanning tree. In our specific context, we define a subset $C \subseteq V$ of vertices that are allowed to join the tree (these are the HST nodes; other standard PNNI nodes can coexist but will not participate in the tree). Prim's algorithm seems better suited for our problem, because each node needs only know on which spanning tree it is connected and does not need to know about other disjointed spanning trees (Kruskal's algorithm would produce a list of disjoint minimal spanning trees); by growing a minimal spanning tree with itself as root, each node can potentially save some unnecessary computations.

2.4.2 First Optimization: Minimal Isolated Spanning Tree (MIST)

We now described a modified version of Prim's algorithm that operates on the subset C (see Figure 2).In the original algorithm, a priority queue is initialized with all the graph vertices. In our approach, the priority queue is initialized only with candidate (HST) nodes. The result is a set of vertices $T \subseteq C$ that form the tree and, associated with each vertex (except the root) $x \in T$, a parent $\pi[x] \in T$. This tree is the minimal isolated spanning tree that includes the starting root (node computing the HST algorithm) of the tree. Possible other disjoint trees in the graph are not computed because they don't affect the status assignment to outgoing links of a specific node.

Let r be the root of the tree, initialized as the node running the algorithm, $P \in C$ the set of candidate vertices to join the tree. These vertices are connected to one of the vertices of the tree via a *light edge* of cost $\omega[x]$,

where $x \in P$. A light edge of minimum cost is selected at each step of the algorithm and a candidate vertex is added to the tree. Adjacent vertices to this vertex are added to the list of candidate vertices. The function $\text{Adj}(x)$ returns the set of adjacent vertices to x. The cost of the edge between two nodes x and y is $w(x, y)$. The cost function, which must be unique throughout the cluster, is detailed in Appendix A.

2.4.3 Second Optimization: Minimal Isolated Spanning Subtree (MISS)

The tree produced by the HST algorithm is only used by a HST node to update the status of its outgoing links. Therefore, the algorithm could stop whenever all the adjacent HST nodes have been included in the growing minimal spanning tree. This could provide significant computational savings, because the tree is grown from the root, and could potentially terminate long before examining all HST nodes $P \in C$. The resulting MISS algorithm, MISS-mPrim(G, w, r) is defined relatively to the MIST algorithm of Figure 2, by adding a few lines.

We define a new set R of all adjacent nodes to the root and contained in subset C.

1a $\quad R \leftarrow \{v \mid v \in \text{Adj}(r), v \in C\}$

The termination condition for the algorithm is when the subset is empty (all the immediate neighbors have been included in the spanning tree).

5 **while** $R \neq \varnothing$ **do**

A vertex of R is removed from the set when it is included in the minimal spanning tree.

7a **if** $u \in R$ **then**
7b $\quad R \leftarrow R - \{u\}$
7c **end if**

While this optimization improves the speed of the computation, it could potentially cause the algorithm to be triggered more often, as will be detailed in Section 2.5.

2.4.4 Link Cost and Uniqueness of the Isolated Tree

Every HST node executes the HST algorithm building a minimal spanning tree rooted at its node. Since the algorithm is distributed, a valid concern is to question whether the each node converges to the same spanning tree, even though each node executes the algorithm independently, building the spanning tree starting with itself as the root node. The convergence property is guaranteed by Prim's algorithm provided that at any time there is no conflict in extracting a light edge of minimum cost. This conflict arises when several edges of equal minimum cost become candidates. To remove this conflict, every link must have a unique cost throughout the graph. One method to perform this link cost assignment in an ATM-like network is proposed in Appendix A.

2.5 HST Node Action Upon Receipt of Link Update.

Having provided the details of the HST algorithm, we now illustrate the actions taken by a HST node upon receipt of a link update (PTSE). Upon receipt of an update notifying the existence of a new link, a link failure or a change of link cost, each HST node must determine if the update affects the status of its outgoing links and for that matter, must run the tree determination algorithm (specified in Table 1) as appropriate. As specified, the conditions triggering a spanning tree recomputation will be slightly different depending on whether the MISS or MIST algorithm is used. The actions taken by a HST node upon receipt of updates that trigger the tree determination algorithm are:

1. Run the tree determination algorithm if appropriate, depending on the trigger (see Table 1).
2. Based on the output of the algorithm, reset the status of the affected outgoing links, if necessary (flooding, potential spanning tree or active spanning tree).
3. If a node's potential spanning tree link(s) becomes active and the update that triggered the node's link to become an active HST link had specified a link failure, the failed link could have been an edge that had joined two spanning tree partitions. The tree has been reformed over this potential spanning tree link. Since, during this interval, other updates may not have reached all members of the tree, the node must re-synchronize its database with the neighboring node on the new active link.

In a stable network, the type of updates that trigger the tree determination algorithm are less frequent. This also applies to the database exchange possibly following a link failure. In a steady state, tree maintenance is a minimal operation.

Update Type	MIST	MISS
New HST link (link between 2 HST nodes)	1) If both nodes are already part of the existing spanning tree, then rerun the algorithm. 2) If only one node is part of the existing spanning tree, insert other node in the set of candidate vertices and resume algorithm from existing spanning tree (this is a tree merge). 3) If both nodes are not part of the existing spanning tree, then do nothing (a different tree is affected)	Rerun algorithm
Active link failure	Tree partition. Choose a light edge among potential links between the 2 partitions. If none, discard partition that does not include the root (node executing the algorithm).	Rerun algorithm
Other HST failure (potential or in a different spanning tree)	Do nothing (the tree is not affected).	Do nothing
Link cost change	If link is already in list of potential or active links, rerun the algorithm	Rerun algorithm

TABLE 1. Actions taken by a PNNI node running MIST or MISS upon receipt of an update

2.6 Choice of optimization

Depending on the structure of the cluster, one optimization method may be more appropriate than the other. The advantage of MISS is that it executes faster than MIST, and is of lower complexity($O(1)$). The disadvantage is that due to its local knowledge of the tree, any HST link failure or new HST link will trigger the tree recomputation. MIST, on the other hand, can exploit its knowledge of the entire isolated spanning tree, and will trigger a tree recomputation only on selected events. The computation is however more expensive and of higher complexity, mainly $O(E' + V' \log V')$.

In a relatively stable network cluster, one could argue that new link updates will be roughly equal to link failure updates and that the new links will more often than not be between two HST nodes of an existing spanning tree[1]. If the cluster if composed of many small isolated spanning trees, the MIST optimization will run faster and will be triggered less often than MISS. However, if it is likely that the cluster will be composed of a few (if not only one) bigger isolated spanning trees; MISS's faster tree computation will outweight the larger number of triggers of the algorithm. We believe that, in general, most network clusters will consist of one or a few spanning trees and therefore recommend using the MISS optimization of the HST algorithm.

Figure 3 illustrates this distinction for the MIST and MISS optimizations, respectively. In this scenario, the migration has not yet been completed and a standard PNNI node **F** divides the cluster in two distinct spanning trees. A HST node **A** is used as a common reference. If the node at **A** executes the MIST optimization, **A** has knowledge of the complete spanning tree to which it belongs. This means that a relevant[2] update on link **a** will trigger recomputation, while updates on the potential spanning tree link **b** or the "external" link **o** will not have any effect. On the other hand, if **A** executes the MISS optimization, it has only a partial knowledge of the spanning tree to which it belongs. In this case, any relevant update from link **a**, **b** or **o** will trigger tree recomputation. The tree recomputation time and expected frequency of updates can be used to determine which algorithm is best.

3. Performance Analysis of Flooding versus Spanning Tree

In this section, a performance analysis of the flooding versus the spanning tree topology distribution methods is provided. The two main costs that we model are bandwidth utilization of topology control traffic, and the interrupt processing for sending/receiving PTSP messages. First, we consider a *worst case* analysis where we compare flooding in a fully connected graph to the spanning

1. We assume that the network cluster is in the process of migrating to provide each node with the HST algorithm.

2. That is, the type of updates that have the potential to generate tree computation, as described in Table 1.

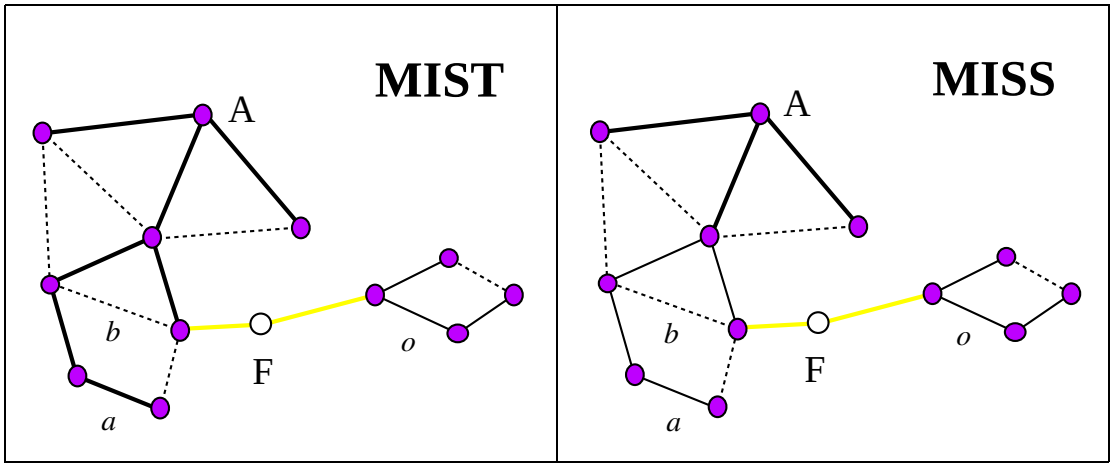

Figure 3. Sample scenario of hybrid spanning trees built with MIST and MISS algorithms

tree approach for the same graph. A linear graph exemplifies the degenerate case where flooding and the spanning tree methods become equivalent. Next, we provide some simulation results that illustrates the costs for flooding versus the spanning tree approach in networks with more general topologies.

3.1 Worst Case Exact Analysis

We define the worst case flooding scenario as one in which every node encounters the same processing and forwarding overhead for each PTSP containing a utilization update. An exact analysis can be carried-out for a fully connected network, consisting of N nodes, where each node may originate a PTSE and flood its corresponding PTSP to its neighbors. A node originates a PTSP and floods it to its N-1 neighbor nodes. Those nodes, subsequently, and in the worst case, flood the PTSP to there N-2 neighbor node, not including the originating node. The worst case, total number of PTSP messages sent and received by all nodes in the network as a result of one PTSP is (N-1)+(N-1)(N-2).

Alternatively, a spanning tree based algorithm results in N-1 PTSP update messages carried by the network as a result of one PTSP update. The number of PTSP update messages sent and received by all the nodes in the network is N-1.

Thus, N-1 messages are generated by a spanning tree method, and the additional (N-1)(N-2) messages are generated, in the worst case, for the flooding approach. One can consider the latter as the additional cost due to flooding.

While this analysis is artificial in that it assumes a fully connected network, that all nodes impose equal delay in processing PTSP messages, and that all links impose equal delay, it motivates the investigation of alternative approaches to flooding. To provide a more realistic study, we implemented a simulation model that compares the efficiency of topology distribution for the flooding versus spanning tree approaches. The simulation accounts for message processing and control traffic bandwidth utilization for flooding and spanning tree cases.

3.2 Simulation of Flooding versus Spanning Tree

The simulation models the behavior of the steady state topology interactions for either the PNNI flooding mechanism or the HST algorithm. Furthermore, the simulation models the control load imposed by the respective topology distribution method, accounting for the control traffic link utilization and number of messages (processor interrupts) received and sent by all nodes.

The simulation accounts for the line utilization resulting from two main sources of topology distribution control traffic; PTSEs that are asynchronously generated by a node for those links owned by the node, and PTSEs that are received from neighbor nodes, processed by the PNNI control point, and subsequently forwarded to the appropriate neighbor nodes. A PTSE is flooded to all immediate neighbor nodes, which subsequently examines their database for duplicates. If the PTSE has not already been received by the node, it is forwarded on all immediate links, except on the link that the update was received. Several PTSEs may be packed into a single PTSP, or the node may send one PTSP per PTSE. Each PTSP is

acknowledged by the receiving node with a PTSP acknowledgment message.

The simulation counts the number of PTSP and PTSP acknowledgment messages sent and received by each node in the network. One could utilize this information to understand the potential processor load for PTSP message handling.

The simulation does not model PNNI database synchronization, as we consider this to not be a steady state behavior and not be an overall significant source of control traffic. It is assumed that all the PNNI topology databases at each node start in a consistent state (empty). Furthermore, topology aggregation is not modeled and therefore is assumed to not utilize link bandwidth or incur processor loading. The aggregation mechanisms utilized by PNNI are relatively new and the amount of information and the frequency with which it must traverse cluster boundaries is not well defined at this time.

The simulation does not account for triggered PTSE updates generated as a result of a significant event, *e.g.* crossing a bandwidth threshold for a given link. To account for this type of PTSE update would require many assumptions on the traffic arrival and departure rates, and the characterization of the traffic itself. Our approach is to simulate networks with a high periodic PTSE generation rate in steady state, with the belief that the triggered threshold updates consist of a much smaller percentage of control traffic load. Furthermore, we can grossly compensate for the triggered threshold traffic by increasing the periodic update rate. The simulation does not model processing of a peer retransmission list nor processing associated with the premature aging of links.

The simulation receives as input the topology (number of nodes, links and their connectivity), the duration of the simulation (in abstract units of time), the update period (units that a node waits before sending the next PTSE for all of its links), a spanning tree or flooding mode flag, and the PTSP and PTSP acknowledgment message sizes (bytes). The PTSP message size is specified by PTSP Packet = PTSP_header_size + #PTSE*(PTSE_header_size + #RAIG*RAIG), which in bytes is:

PTSP Message size = 44 + #PTSE(20+#RAIG*44),(EQ 1)

where PTSP_header_size is the number of bytes in the PTSP header, #PTSE is the number of PTSEs in the PTSP message, PTSE_header_size is the size, in bytes, of the PTSE header, #RAIG is the number of Resource Information Groups contained in the PTSE, and RAIG is the size, in bytes, of the Resource Information Group. Each PTSP is acknowledged by an acknowledge message specified (in bytes) as follows:

PSTP Ack size = 36 + #PTSE_Ack*12, (EQ 2)

where #PTSP_Ack is the number of multiple nodal PTSE information groups being acknowledged.

The simulation model accounts for the ATM AAL5 segmentation/reassembly and padding overhead on the link.

The simulation results include statistical information on the number of messages that are sent and received on each link of each node. The total message statistics for the network are provided, including per node statistics like number of messages received and sent by the node, the number of bytes sent and received on each of the node's links, and, in the case of flooding, the number of messages received unnecessarily (PTSE updates that have already been received).

3.3 Results and Discussion

A random graph generator was built and utilized to produce graphs on which to execute the simulation. The PTSE update interval (MinPTSEInterval) was set at one unit of time (for simplicity, we assume one unit equals one second). This update interval represents the default *highest* rate that a node may generate PTSE, according to the PNNI specification and acts to reduce the possibility of rapidly exhausting the PTSE sequence number space.

The percent of link utilization for topology distribution control traffic is calculated using Equation 3. The total number of bytes that traverse a uni-directional link over the course of the simulation is counted. The mean, maximum, and minimum bytes transmitted, obtained from the simulation results, are used to calculate the mean, maximum and minimum percent control traffic link utilization, respectively.

% BW Utilization = (bytes transmitted*8)/(link capacity*simulation time) (EQ 3)

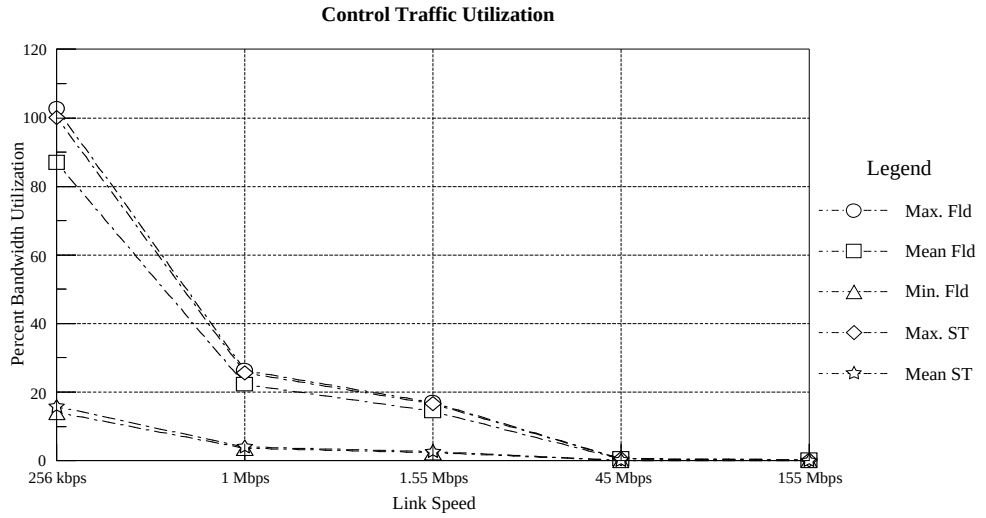

Figure 4. Bandwidth utilization for 20 nodes, degree of 6 links per node

Simulations were executed for several different topologies, varying the node degree, number of nodes in the cluster and the simulation duration.

Figure 4 shows a comparison of the flooding and spanning tree based topology distribution algorithms for a randomly generated network with 20 nodes, where the average degree (number of unidirectional links) per node is six, and the simulation executes for a duration of 1000 seconds (simulation units). The indices for the x axis correspond to link speeds as follows: 1=256kbps, 2=1Mbps, 3=T1, 4=T3, 5=OC3. The semantics of maximum, mean and minimum in the flooding and spanning tree simulations require definition for the flooding and spanning tree cases. In the case of flooding, the maximum, mean and minimum graph labels correspond to the respective percent bandwidth utilization for the most heavily loaded link, the mean loaded link, and the lightest loaded link. In the spanning tree case, the maximum and mean load correspond to *only* the spanning tree links; non-spanning tree link control traffic utilization is zero.

In both the flooding and spanning tree cases, the maximum percent utilization is approximately equal, while the mean percent utilization is significantly less for the spanning tree case. As one would suspect, the number of bytes utilized by the flooding algorithm, is significantly greater than the spanning tree case. This results in an absolute bandwidth savings in the spanning tree case, although for the higher speed links, the relative savings becomes less significant. For example, the difference in mean percent utilization for a network consisting of 256 Kbps link for the flooding and spanning tree case is 86% and 16%, respectively. However, for a network of homogeneous T1 links the mean percent utilization for a 256 Kbps link for the flooding and spanning tree cases is 14% and 3% respectively. As the link speed is increased, the benefits provided by the spanning tree algorithm in homogeneous speed networks are reduced.

The figures do not show the results for 64Kbps links because the maximum percent control traffic utilization is in upwards of 400% for flooding and the spanning tree algorithms[1]. This indicates that in the *worst* steady state case (MinPTSEInterval=1), topology distribution is infeasible in low bandwidth homogeneous networks. However, in heterogeneous networks consisting of at least one high speed link between neighbor nodes, the spanning tree algorithm could provide a superior solution. That is, the spanning tree could be configured to restrict topology distribution to that of high speed links.

For example, consider a heterogeneous speed network consisting of at least one E1 link between every neighbor node, and 256 kbps speed links interconnecting nodes. Using the spanning tree algorithm, the topology control traffic could be restricted to the E1 links. Thus, the maximum control traffic load (from Figure 4) on the E1 links would be approximately 28% rather than over 100% of the 256 kbps link capacity required by flooding.

1. As a rule of thumb, control traffic bandwidth utilization should not exceed 20% in the average case. Any percent bandwidth utilization greater than 100% indicates infeasibility.

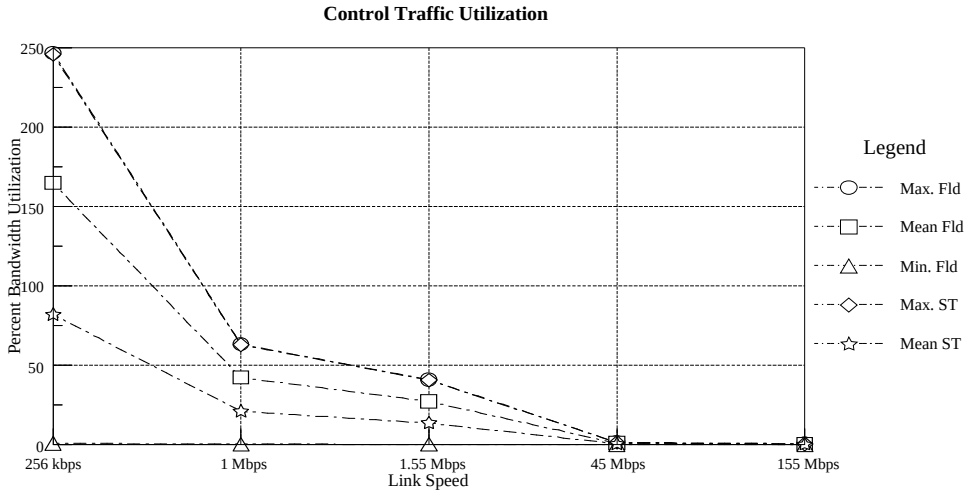

Figure 5. Bandwidth utilization for 100 nodes, degree of 3 links per node

Figures 5 shows a comparison of the flooding and spanning tree based topology distribution mechanisms for a randomly generated 100 node network cluster, with average degree three unidirectional links, and a simulation time of 1000 seconds. Again, the maximum link utilization is approximately the same for the flooding and the spanning tree cases for a homogeneous speed network. The mean link utilization is significantly lower in the spanning tree case. However, as the link speed is increased, the significance becomes less important. The mean percent utilization for a network consisting of 256 Kbps links for the flooding and spanning tree case is 164% and 81%, respectively. The mean percent utilization for a network of T1 link for the flooding and spanning tree cases is 27% and 13.5% respectively.

In the worst case, where each node is generating PTSEs at the highest rate (MinPTSEInterval=1), one observes that the mean link utilization for the spanning tree case is approximately 100% better than that of the flooding case. The maximum loaded link in either case is equally loaded and no savings is achieved by using the spanning tree approach for a homogenous speed network. Again, in a heterogeneous speed network, the spanning tree algorithm could provide a superior solution.

Next we examine the number of messages that each node must process in order to support the flooding or spanning tree topology distribution methods. The number of messages received could translate to the number of interrupts that the node must process. OPENET [4] has suggested that one way to reduce the number of interrupts is to implement a scheduling algorithm that accumulate several inbound PTSEs prior to interrupting the node processor. In both the flooding and spanning tree cases, perfect reception of all messages is assumed (retry is never required). For the flooding and spanning cases, the following statistics are obtained over the course of the simulation:

- The number of PTSPs originated by all the nodes in the network per second,
- The number of PTSPs forwarded by all node in the network per second,
- The number of PTSP acknowledgments sent/ received by all nodes in the network per second,
- The number of PTSPs received by all the nodes in the network that were actually applied to the PNNI database because they are more recent than what was already contained in the database (database hit),
- The number of PTSPs received by all the nodes in the network that were found to contain older information than that contained in the PNNI database, and thus were discarded (database miss).

Table 2 shows the number of messages sent and received by each node corresponding to the same networks used Figure 4 and Figure 5 respectively. The first set of numbers corresponds to the 20 node, degree 6 network, whereas the latter set of numbers corresponds to the 100 node, degree 3 network. For the 20 node network over the course of the 1000 unit (second) simulation, the number of updates received by all nodes and subsequently discarded because they were out-of -date, accounted for approximately 81% of the total PTSP message traffic (not including the PTSP ack). In the spanning tree case, the number of missed updates is zero because all update flow along a tree (loops cannot occur in a stable network). The number of PTSP messages forwarded and originated in the flooding case is over five times

Method Type	# Originated	# Forwarded	# Acks	# Hit updates	# Miss updates
Flooding	866/1178	12150/58077	13016/59255	2355/29477	10655/29770
Spanning Tree	245/748	2109/28727	2354/29475	2345/29474	0/0

TABLE 2. Message count for 2 configurations: (20 nodes, 6 links per node)/(100 nodes, 3 links per node)

greater than in the spanning tree case.

For the 100 node network, the number of missed updates in approximately 50% of the total number of PTSPs originated and forwarded by all nodes. There are approximately twice the number of PTSP messages forwarded and originated in the flooding case compared to the spanning tree case.

The number of missed updates, which corresponds to extraneous CPU cycles in the node processors and wasted bandwidth utilization on the links is mainly a function of the number of total links in the network, given a fixed update frequency. Figure 6 shows a comparison of the messaging overhead of flooding versus the spanning method. Several different graphs are represented in increasing order of total number of unidirectional links, along the abscissa axis. The ordinate axis shows the total number of originated plus forwarded messages sent during the simulation of each respective graph. Each graph is simulated for a duration of 1000 units, with MinPTSEInterval=1, and one PTSE per PTSP. As the size of the graph is increased (in total number of links), the overhead associated with flooding messaging becomes greater. Secondly, for graphs of approximately the same number of total links, the graph with the lower number of nodes and thus higher degree per node, will exhibit a significantly greater difference in the flooding and spanning tree methods. For example, in Figure 6 two graphs with approximately 300 links demonstrate distinct behavior for the spanning tree and flooding cases. For the graph with 100 nodes and 3 links per node (300 links), the flooding message approximately two times that for the spanning tree case. For the graph with 50 nodes and 6 links per node (300 links), the flooding message is over five time greater than that for the spanning tree case. This behavior is also observed for the graph containing 20 nodes and 6 links per node (120 links) compared with the graph containing 50 nodes and 3 links per node (150 links). In Figure 6, the graph containing 20 nodes and 6 links per node depicts a significantly greater difference in messaging for the flooding and spanning cases, as compared to the graph with 50 nodes and only 3 links per node.

The number of messages generated over the 1000 unit (second) period can be utilized to estimate the CPU overhead incurred by a real ATM switch that is executing the PNNI control point. The time required to process each PTSE per node is given by equation 4.

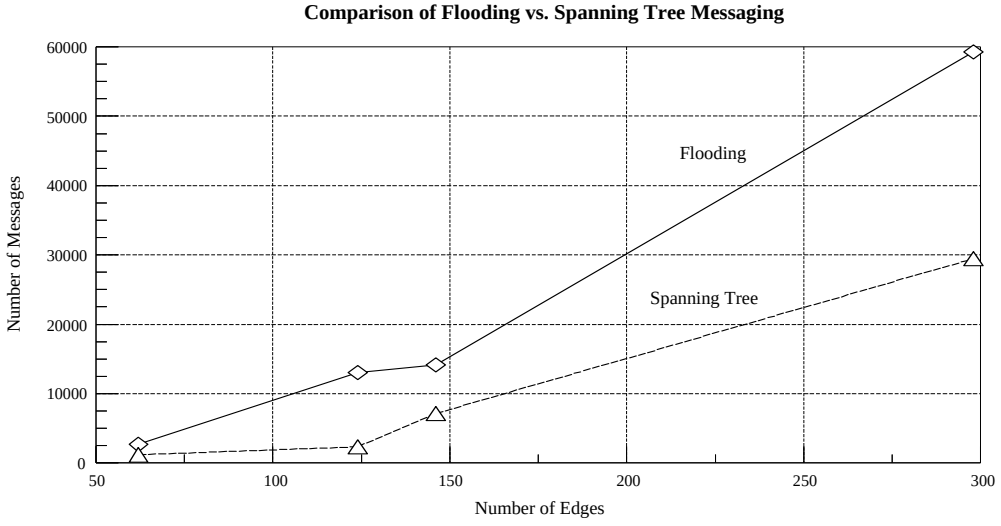

Figure 6. Messaging comparison of Flooding and Spanning Tree

Process_time_per_node = (#messages/ #nodes)*process_time_per_PTSE (EQ 4)

For example, if processing a hit update requires 1 milliseconds and processing a missed update requires 0.5 milliseconds, then the graph containing 50 nodes and 6 link per node would require approximately 0.30 seconds per node for the spanning tree case [(14,883/50)*.001, hits only] and approximately 0.9 seconds per node for the flooding case [((14,883/50)*.001) + ((62570/50)*.0005)), hits plus misses].

There are several limitations of this study that are noted here. The simulations were executed on a small set of randomly generated graphs, although the authors observed that many graphs studied exhibited similar characteristics. The simulations were executed for the worst case scenario, where each node generates PTSEs for all of its links at the highest possible rate (MinPTSEInterval = 1). Since the *average* behavior of a given network is difficult to define, the study was limited as stated. The dynamic behavior of call arrival and departure with different bandwidth and QoS requirements was not modeled. Furthermore, the grouping of multiple PTSEs into one PTSP, which could result in a lower link utilization and message interrupts, was not considered.

4. Some Implementation Considerations

Given that a flooding based PNNI topology distribution is currently standard, we explore the potential interfaces to the topology database that would require modification to implement the HST algorithm. The HST topology distribution algorithm can be integrated into the standard PNNI topology distribution by providing interfaces to the following components:

- **Flooding**: a method to activate/deactivate the participation of individual outgoing links in the flooding algorithm, such that PNNI updates, are only flooded onto outgoing links that have not been deactivated. The deactivated links correspond to potential HST links.
- **Notification of updates**: the PNNI database should offer a notification service to indicate to the HST component when topology updates are received. The types of updates are: link up, link down, change in any parameter involved in the link costs computation.
- **Hooks into the topology database**: some additional information needs to be provided by the elements of the topology, consisting of new attributes (e.g., HST enabled nodes) for links and nodes.
- **Structure of the topology**: the topology database should be organized in such a way as to allow an efficient implementation of basic functions of the minimal spanning tree algorithm. In particular, the function $\mathrm{Adj}[x]$, which returns all the immediate neighbors of a node should be efficiently provided by the topology database structure.
- **Database synchronization**: the database should provide a function that triggers a database exchange with a specified peer node. This is used after a tree partition event, when a potential tree link becomes an active link.

5. Conclusions

A spanning tree algorithm for topology distribution is proposed as an alternative to the flooding mechanism utilized by the PNNI standard. The Hybrid Spanning Tree (HST) algorithm is introduced, and several variants are provided. It is argued that a spanning tree method for topology distribution is superior to flooding. The major advantages of the HST approach, as compared to other spanning tree methods for topology distribution are: HST does not require ATM switch hardware modifications, and can interoperate with standard PNNI nodes that continue to execute the flooding topology distribution algorithm. HST provides for smooth migration of PNNI nodes executing the flooding algorithm to transition to the HST method. Finally, nodes executing the HST algorithm may continue to interoperate with nodes executing flooding, at any level in the PNNI hierarchy.

A performance analysis of flooding versus a spanning tree approach for topology distribution in the Private Network Node Interface (PNNI) standard is performed. A simulation model that implements topology distribution via flooding or spanning tree is implemented. It is observed in a small sample of randomly generated graphs that the reduction in link bandwidth utilized by control traffic is *not* as significant in highly provisioned, *homogeneous* speed networks. However, a spanning tree topology distribution mechanism can provide significant bandwidth savings for heterogeneous link speed networks. Furthermore, a spanning tree algorithm can support a policy that restricts control traffic to the higher capacity, spanning tree links. It is concluded that PNNI topology distribution over low bandwidth links is typically infeasible. Since the spanning tree approach utilizes less bandwidth for topology distribution, for a given control traffic link utilization, it allows for a higher topology update rate, as compared to flooding. Therefore, the

spanning tree approach can provide more accurate topology representation at each node.

While the savings in control traffic bandwidth utilization provided by the spanning tree method is not dramatic in high bandwidth, homogeneous networks, the savings in messaging overhead is significant in all networks. The messaging overhead produced by the flooding topology distribution method is significantly greater (at least twice as much simulation) than that of the spanning tree method. The increased messaging overhead results in increased CPU utilization for topology processing. A method by which the messaging cost reported here can be translated into real control point processor CPU utilization is specified. Limitations of the simulation model are detailed.i

6. References

[1] *Private Network Node Interface Specification Version 1.0*, ATM Forum Technical Committee, af-pnni-0055.000, March, 1996.

[2] T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms,* MIT Press 1990.

[3] Networking BroadBand Services (NBBS) Architecture Tutorial. Document Number GG244486-00. International Technical Support Organization, Raleigh Center, June, 1995.

[4] Cidon, I., Hsiao, T., Khamisy, A., Parekh, A., Rom, R., Sidi, M. *The OPENET Architecture*. Sun Microsystems Laboratories, December, 1995. SMLI TR-95-37.

[5] Cahn, R., *The Deployment of Private WDM Networks.* SPIE Conference on All-Optical Communication Systems: Architecture, Control and Network Issues. Philadelphia, October, 1995.

Appendix A: Link Cost and Tie Breaker

Each outgoing unidirectional link is assigned a *unidirectional link cost* to support a branch of the spanning tree. When two nodes are connected via two unidirectional links, the common bidirectional link cost is defined as the maximum of the two unidirectional link costs:

$$w1(x, y) = \max(\text{link_cost}(x), \text{link_cost}(y))$$

However, the above definition is not sufficient, for it might happen that during the minimal spanning tree algorithm, two or more light edges may have an identical cost. Which one should the algorithm choose? If one edge is randomly chosen, the minimal tree will not be unique. If every node runs the algorithm, we would like them to converge to an identical tree, otherwise loops could be created in the network. We therefore propose to extend the definition of the cost function. We consider that each vertex $x \in V$ has a unique identifier. We extend our definition of the cost function.

$$\forall (x, y) \mid x < y,$$
$$w(x, y) = \text{concat}(w1(x, y), x, \text{Port_ID}(x, y))$$

where x and y denote the globally unique node IDs of the two adjacent nodes, and $\text{Port_ID}(x, y)$ returns the port ID on node x to which is attached the outgoing link going to y.

This definition gives a globally unique cost to every link of the graph. To compare two bidirectional link costs, we use the relation:

$$w(x, y) < w(a, b) \Leftrightarrow (w1(x, y) < w1(a, b)) \vee$$
$$((w1(x, y) \equiv w1(a, b)) \wedge ((x < a) \vee$$
$$((x \equiv a) \wedge (\text{Port_ID}(x, y) < \text{Port_ID}(a, b)))))$$

The definition of minimum and the comparison of costs performed in the modified minimal spanning tree algorithm are based on this definition. Since only a subset of edges might be used by the HST algorithm, it may not be necessary to compute the cost for every edge of the graph. In the minimal spanning tree algorithm, the value of $w(u, v)$ can be computed when needed and cached for further use. However, this quantity should be updated, if cached, whenever a link cost update is received.