1 Question

1.1 Goals

- To understand the PNNI routing protocol in ATM networks.
- To implement a simplified PNNI routing protocol.

1.2 Introduction

PNNI is an ATM protocol specification for connecting either private ATM nodes (switches) or private ATM networks (groups of switches). Switches in a private network can “trust” one another and can exchange detailed information about the topology and operational state of the network.

PNNI includes two categories of protocols:

1. **Routing protocol**: A protocol for distributing topology and state information between switches and clusters of switches. This information is used to compute paths through the network. A hierarchy mechanism ensures that the protocol scales well for large, wide area ATM networks. A key feature of the PNNI hierarchy is its ability to automatically configure itself for networks in which the address structure reflects the topology. PNNI topology and routing are based on the link-state routing technique.

2. **Signaling protocol**: A second protocol is defined for signaling during connection establishment. This protocol uses the information provided by the routing protocol to establish point-to-point and point-to-multipoint connections across the entire network.

1.3 The PNNI Routing Protocol

In this lab we will focus on the first protocol, i.e., we will simulate a *simplified* version of PNNI routing.

In order to plan a route through the network, the originating switch must have the following information available:
- A network topology map to enable the switch to calculate possible routes to the destination endpoint.
- Up-to-date state information to enable the switch to determine if required QoS characteristics can be met on any particular route under consideration. This information is part of the topology map.

In the PNNI protocol, each switch maintains its own database. This database is based on the hierarchical topology of the network. The routing protocol is used for building this database and for keeping the database current.

The functions of PNNI routing implemented in this lab include:

1. Discovery of neighbor and link status information using the **Hello protocol**. Hello messages are exchanged between neighbors in each level of the PNNI hierarchy.
2. Maintaining PNNI topology databases. This is done by flooding **PNNI Topology State Packet (PTSP) messages** throughout a node’s peer group at each level.
3. Summarizing the topology state information, and constructing the routing hierarchy. This operation is performed by the **peer group leaders (PGL)** at each level of the hierarchy.

### 1.3.1 The PNNI Routing Hierarchy

The key to understanding PNNI topology services is the principle that a **switching system** can be either a real switch (one physical box) or it can be a “virtual” switching system composed of an interconnected group of switching systems. In this case the definition of the virtual switch is recursive. Consequently, a virtual switch can also be a group of groups of switches and so on. A virtual switch is called a logical group node (LGN). Figure 1 illustrates a network consisting of 26 interconnected physical level (called level 0 in the lab) nodes. Users are directly connected to these level 0 nodes. Data passes through lowest-level nodes to other lowest-level nodes until it reaches the destination user.

![Figure 1: ATM network with 26 switching systems and 33 bi-directional links](image)

Figure 1: ATM network with 26 switching systems and 33 bi-directional links
Each node in the figure represents a physical switching system in the network. The arcs in Figure 1 represent physical links connecting two switching systems. Each physical ATM switch is uniquely identified by its ATM address. The 20-byte ATM address is taken as a bit string and structured into a 13-byte (104-bit) address prefix followed by 6-byte end system identifier and a 1-byte selector (subaddress). The address prefix uniquely identifies a physical switch and all its connected end devices. In this lab, we will use dotted character strings to denote addresses. The notation is illustrated in Figure 2.

Each logical group node (LGN) is composed of a peer group (PG) of LGNs. A peer group is a group of interconnected nodes with an arbitrary number of high-order (leftmost) ATM address bits in common. Each node within a PG exchanges information with other members of the group, so that all the members of the PG maintain an identical view of the network topology. At the lowest level, a peer group is a number of connected switches with some common high-order address bits. In Figure 2 the example network is organized into 7 peer groups at the lowest level. A peer group is identified by its peer group identifier (PGID). The peer groups ID’s in level 0 of Figure 2 are A.1, A.2, A.3, A.4, B.1, B.2, and C.1. The highest level peer group is called the root peer group.

A peer group can be abstracted as a single node at the next higher level. This abstraction of a group of nodes by a single node is thus a logical group node (LGN). Numbering (or addressing or naming) logical group nodes reflects the hierarchical routing structure. For example the node denoted by A.3.2 is located in PG(A.3),

At the lowest level, individual switches are interconnected by real, physical ATM links (or pre-established
VP/VC connections). At higher levels, however, LGNs are interconnected by logical links which may or may not map to a single physical link. Logical links inside a peer group are horizontal links whereas links that connect nodes in different peer groups are outside links.

Each peer group (except the root peer group) has a peer group leader (PGL) that can be any node in a peer group. The PGL is selected based on a priority system by a distributed leader election process. In our lab, we will not simulate the election process of the PGL. PGL’s will be assigned at configuration time. The PGL has no special privileges within the group. It performs the functions of the LGNs at the higher levels of the hierarchy. Thus, the PGL participates in the topology exchange protocols within the higher-level peer group and represents the nodes in the lower level peer group. This means that the PGL is a single LGN in the higher-level peer group. In this way, it passes (summarized) information up and down the hierarchy. For example, in Figure 2, node A.1.3 performs the functions of A.1 in PG(A), while A.2 performs the functions of A.2 in PG(A) and A in the root PG.

1.3.2 Neighbor Discovery: The Hello Protocol

When a switching system becomes operational, it periodically sends Hello packets to its neighbors. The hello packets contain the node’s ATM end system address, (node ID), PGID and link status information. When a node receives a Hello packet from its neighbor, it compares the peer group ID received with its own PGID to determine whether this adjacent node is part of its peer group or a border node in an adjacent peer group. The link and node information in the hello message are stored as part of receiving node’s topology database.

The exchange of hello messages initially takes place at the level 0 (the physical level). When the further topology exchange results in the formation of higher levels, then the hello messages are exchanged similarly at each higher level. Physically, the hello messages are exchanged between the leaders that represent the node in the hierarchy. In this way, each LGN discovers its neighbors and link status.

For this lab, we will assume that all physical nodes become operational simultaneously, and a round of hello messages follows. The exchange of hello messages is followed by a synchronization of topology databases between neighbors. In this lab, we update topology databases using PTSP’s as described in section 3.3.

1.3.3 Topology Aggregation: Flooding of PTSP’s

After the hello messages are exchanged, the node builds a PNNI Topology State Packet (PTSP) message that lists all its adjacent links and their characteristics (such as the ATM address of the node to which the link is attached). In fact, the PTSP message contains the nodes entire PG_view structure as described in the code. The PTSP message represents the node’s present view of the PNNI routing domain. This PTSP message is then reliably flooded (broadcast) to all adjacent members of its peer group. When a node receives a PTSP message, it broadcasts the message to all its neighbors except to the neighbor from which it received the PTSP (this avoids infinite flooding). In the PNNI protocol, PTSP’s messages are acknowledged hop-by-hop. In this lab, we do not acknowledge PTSP messages, and assume that all links are reliable.

Every node receiving a PTSP message checks if this message is a duplicate (this can be implemented by assigning sequence numbers to messages). In this lab, we assume that a node does not send duplicate PTSP messages. If a node receives a PTSP message originating from the same node and forwarded by two different neighbors, the receiving node simply keeps the first message and discards the rest. This is explained further in the code.

The flooding of PTSP’s enables each node in a peer group to maintain an identical view of the network topology. The detailed PTSP information about resources in a particular peer group is never propagated to nodes outside the peer group. Only the summarized topology of the PG is propagated throughout the higher-level PG. At higher levels, all of the detailed information about lower level nodes is not necessary and is hidden to ensure scalability. Information about a peer group is aggregated to form a logical image of the peer group as a single LGN.
1.3.4 Forming the Routing Hierarchy

The logical group nodes in Figure 2 are organized into peer groups. For example, logical nodes A.1, A.2, A.3 and A.4 are organized into peer group A. Each LGN in peer group A represents a separate lower level peer group. Peer group A has a peer group leader (logical group node A.2). Note the functions that define peer group leader of A are located in node A.2, which is in turn implemented on the switching system containing lowest-level node A.2.3. Peer group A is called the “parent peer group” of peer groups A.1, A.2, A.3 and A.4. Conversely, peer groups A.1, A.2, A.3 and A.4 are called “child peer groups” of peer group A.

It is important to notice that the PGL performs primarily one function - that of a “gateway” for passing topology information about the PG up the hierarchy and topology information about the network down the hierarchy. Each switch in the network keeps a hierarchical view of network topology and uses this to construct routes through the network.

Logical group nodes at higher levels behave similar to lower level nodes. They exchange hello messages, between neighbors (that means that one or more links must exist between their child peer group members). They also flood PTSP messages. Note that information specific to child peer groups is not part of the higher level databases. The PGLs also pass PTSP messages to their child peer groups. These PTSP’s contain summarized network topology of the total network obtained by the higher level LGN. This enables each switch on the lowest level to know enough about the overall network to construct a route.

Figure 2 shows the complete network hierarchy. Completion is achieved by creating ever higher levels of peer of peer groups until the entire network is encompassed in a single highest level peer group. In the example in Figure 2, this is achieved by configuring a PG containing logical group nodes A, B and C. Node A represents peer group A which in turn represents peer groups A.1, A.2, A.3, A.4 and so on.

1.3.5 A Single Node’s View

Figure 3 shows the network perspective of a lowest-level node in PG(A.3). Note that all nodes in PG(A.3) have the same network perspective. This view shown includes all ancestor peer groups of PG(A.3), i.e. peer group A and the highest level peer group. In general, a node’s view of the network consists of detailed information about its lowest level peer group and summarized reachability information about higher level peer groups.

Figure 3 show links between LGN’s of different hierarchical levels. These are the links between A.3.1 and A.2, A.3.4 and A.4, A.3.2 and B, and A.3 and B. These links are called uplinks. Uplinks denote the existance of a physical links between members of the respective PG’s. Thus, uplink A.3.4 to A.4 represents the physical link between A.3.4 and A.4.6, uplink A.3.2 to B and uplink A.3 to B represent the physical link between A.3.2 and B.1.1.

This network view provides sufficient information for a node to find a route to any other node in the entire network.

1.4 Lab Requirements

This lab will consist of two parts:

1. Experiment with the PNNI algorithm as shown in the demo program.

2. Implement the following three functionalities of a simplified PNNI model:

   (a) Sending a hello message.

   (b) Receiving a hello message.

   (c) Sending a PTSP message.

1.5 Methodology

This section first describes the data structures provided to you for implementing the simplified PNNI protocol. Second, it provides a declaration of the functions you need to implement.
1.5.1 PNNI data structures

Each node is of type PNNINode which has a field node_dbase containing all the relevant network topology information for the node.

typedef struct _PNNINode {
    / * Stuff deleted */
    / * Do not modify this structure */
    Node_Type node_dbase; / * the valuable information of this node is */
         / * stored here */
} PNNINode;

The node_dbase is of type Node_Type which is defined as:

typedef struct _Node_Type {
    char address[MAXADDR];        /* Physical Address of the node */
    int leader;                  /* the leadership value of this node */
        /* = 0 ==> non-leader node */
        /* i ==> leader upto level i-1 */
} Node_Type;
int n_physical_neighbors; /* Number of physical neighbors */
char physical_neighbor_address[MAXADDR][NNODE]; /* physical address of */
Component *physical_neighbor_ptr[NNODE]; /* Pointer to neighbor structure */
/* This is for internal use */

int nBorders; /* Number of physical borders nodes */
char border_address[MAXADDR][NNODE]; /* physical address of border */
Component *border_ptr[NNODE]; /* Pointer to border structure */
/* This is for internal use */

PeerGroupView PG_view[NLEVELS]; /* This node’s view to the whole network */

Note that:

• All addresses are at most MAXADDR long and are defined as strings in dotted character form as shown in the examples.

• There are at most NNODE nodes in the network. NNODE is thus an upper bound to the number of neighbors and borders.

• There are at most NLEVELS in the network hierarchy.

The PeerGroupView structure type defines the view of each node at each level. The PeerGroupView and PeerGroupMember structures are defined as follows:

typedef struct _PeerGroupView {
  int n_PG_members; /* How many members we have in this PG */
  PeerGroupMember PG_member[NNODE]; /* Structure describing member’s view */
  /* of the world */
  /* PG_member[0] is me */
} PeerGroupView;

typedef struct _PeerGroupMember {
  char PG_member_address[MAXADDR]; /* LGN Addr of this PG member */
  char physical_address[MAXADDR]; /* Physical address of this member */
  NODE_ENTITY_TYPE_PTR PG_member_ptr; /* A pointer to this member’s main */
  /* data structure */
  /* This is an internal structure */
  int n_uplinks; /* Number of uplinks this member has */
  Uplinkto up_neighbor[NNODE]; /* Uplinks of this member */
  int n_neighbors; /* Number of neighbors this member has */
  char neighbor[MAXADDR][NNODE]; /* This member’s neighbor’s addresses */
  Component *neighbor_ptr[NNODE]; /* Pointers to the above neighbors */
  /* This is an internal structure */
} PeerGroupMember;

Uplinkto is a structure describing links between levels.
typedef struct _UPLINKTO {
    char address[MAXADDR];    /* LGN address of the destination */
    char physical_address[MAXADDR]; /* Physical address of the destination */
    Component * ptr; /* A pointer to the destination. */
    /* This is internal to the simulator */
} _UPLINKTO;

Hello and PTSP messages are defined below:

/* The PTSP message */
typedef struct _PTSPMessage {
    int source_level; /* the level of the sender */
    int dest_level; /* the level of the receiver */
    /* Ignore this field for now */
    char source_address[MAXADDR]; /* Source’s LGN address */
    char source_physical_address[MAXADDR]; /* Source’s physical address */
    Component *source_ptr; /* A pointer to source’s whole data structure */
    PeerGroupView PG_view[NLEVELS]; /* the sender’s current view to the PGs */
    /* of every level. */
    /* PG_view.PG_member[0] is copied by the */
    /* receiver of the PTSP */
} PTSPMessage;

/* The Hello message */
typedef struct _HelloMessage {
    int level; /* The level at which Hello is sent */
    char address[MAXADDR]; /* Sender’s LGN address */
    char physical_address[MAXADDR]; /* Physical address of the sender */
    Component *node_ptr; /* A pointer to the data structure of the sender */
    int response; /* 1 if this a response to a hello, 0 otherwise */
} HelloMessage;

NOTE: Although the data structures in this lab are complex, this is indeed a simple representation of the real networking code in most current systems. You need to study and manipulate these structures carefully, and incrementally test your implementation; otherwise, network and operating systems programming can easily turn into an exercise in futility.

1.5.2 A Simplified PNNI Implementation

For this lab, we will assume that messages are exchanged in rounds. Thus the simulation of PNNI consists of the following steps:

- Exchange all hello messages at level 0.
- Exchange all PTSP messages at level 0.
- Leaders at level 0 form level 1.
- Exchange all hello messages at level 1.
- Exchange all PTSP messages at level 1.
- Propagate information down to level 0.
- Leaders at level 1 form level 2, and so on...

The triggering of each step will be controlled by the simulator. You will be required to write the functions that modify the data structures of a message and node.
- **cn_send_hello(cn, cn2, level, response)**: Send a hello message from component cn to cn2 at level of the hierarchy. Use cn_transmit_hello() to transmit to neighbors. Refer to the code for details.

- **cn_receive_hello(cn, cn2, msg)**: Process a hello message received.

- **cn_send_ptsp(cn, l)**: Form and send a PTSP message. Use cn_transmit_ptsp(). l is a pointer to the source level and destination level fields of the ptsp.

### 1.5.3 A list of available functions

- **int samePG(char * physical_address1, char * physical_address2, int level)**: Returns 1 if the two physical addresses belong to the same PG at the given level, 0 otherwise. For example, A.1.1 and A.2.1 are in the same PG at level 1 but not in level 0 etc.

- **int find_distance(char * physical_address1, char * physical_address2)**: Finds the hierarchical distance between two physical addresses. For example, the distance between is A.1.1 and A.2.1 is 1, and the distance between A.1.1 and A.1.2 is 0 etc.

### 1.6 Deliverables

- In the directory `/home/grad/classes/ece6609/PNNI_LAB/Files` you will find the following files:
  1. node.h, node.c. This is where you will implement the PNNI algorithm. node.h is the header file that defines the PNNI data structures. node.c contains skeletons of the functions cn_send_hello(), cn_receive_hello() and cn_send_ptsp() for you to code.
  2. pnni.config. The configuration file for the demo and testing your program. The configuration file specifies the configuration described in this handout.
  3. Makefile. To compile the source code.
  4. lab2_demo. The demo executable that you will use for the first part of the lab.
  5. node_wrappers.c. This file is needed for compilation. **Do not modify this file.**

- Copy all the files in `/home/grad/classes/ece6609/PNNI_LAB/Files` to your working directory.

- Run `lab2_demo` with the configuration file. You can simply pass the config file name as an argument to `lab2_demo` or you can load the file from the **File** menu of the simulator.

- **Part 1**: Run the simulator for the configuration file, and answer the following questions.
  1. Describe briefly how the messages are exchanged at each level, and between levels?
  2. Click on the node C.1.2. What are its uplinks? What do these uplinks represent?

- **Part 2**: Write the C code for the functions cn_send_hello(), cn_receive_hello() and cn_send_ptsp(). Your results should graphically match with the demo, i.e., at the end of the simulation, the screen should contain the same picture. Use a unix utility like `xview` or `xgrab` to capture the final result.

### 1.7 Submissions

You must submit a soft copy of the following by email to infocom@ece.gatech.edu:

- Your source code file node.c. Please rename node.c to your_name.c, e.g., node.c to tricha.c, before sending it.

You must submit hard copies of the following:
• A short summary of the lab, and answers to the questions in part 1.
• Your source code file node.c.
• An image of your simulation result.

1.8 Running your program: The Graphical User Interface

Login to a Solaris Unix machine. You could find one in the CoC building (third floor). At your unix prompt, type `lab2_demo node.config` in the directory where you have copied the files. A windows showing the network configuration will open on your terminal. You can use the information in the plot window to debug your program and answer the questions in part 1.

The top of the window has a menu bar that has the following selections.

<table>
<thead>
<tr>
<th>Menu</th>
<th>Submenu</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Load</td>
<td>Load config file</td>
</tr>
<tr>
<td></td>
<td>Exit</td>
<td>Stop the program and exit</td>
</tr>
<tr>
<td>Edit</td>
<td>Raise</td>
<td>Raise the plot window to the front of main window. Useful when you cannot see the graph.</td>
</tr>
<tr>
<td>Run</td>
<td>Pause</td>
<td>Pause the simulation.</td>
</tr>
<tr>
<td></td>
<td>Resume</td>
<td>Resume the simulation.</td>
</tr>
<tr>
<td></td>
<td>Delay</td>
<td>Set delay between events. The default is 50.</td>
</tr>
<tr>
<td></td>
<td>Stop Time</td>
<td>The simulation time to stop.</td>
</tr>
<tr>
<td></td>
<td>Inc/Dec Debug Level</td>
<td>Increment/Decrement Debug Level, which is used in dprintf().</td>
</tr>
<tr>
<td>Help</td>
<td>About</td>
<td>About the CISE Project.</td>
</tr>
<tr>
<td></td>
<td>How to use</td>
<td>Help text.</td>
</tr>
</tbody>
</table>

The bottom of the window has the status bar with the following information.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>The configuration file name.</td>
</tr>
<tr>
<td>Stop Time</td>
<td>The end time for the simulation</td>
</tr>
<tr>
<td>Delay</td>
<td>Delay between events.</td>
</tr>
<tr>
<td>DebugLevel</td>
<td>DebugLevel used in dprintf().</td>
</tr>
<tr>
<td>Simulation time</td>
<td>The clock in the simulator.</td>
</tr>
</tbody>
</table>

Familiarize yourself with the simulator and use `lab2_demo` to do the following:

• First, load a config file. This can be done by File/Load or by putting the config file at the command line of `lab2_demo`.
• Then use Run/Run to start your simulation.
• Change the delay to make it run faster/slower. Use “Run/Single step” and space bar to see it step by step.
• Change Debug Level (from 0–3) and use `dprintf(int debug_level, "format", variables)` in your program to print out debug information.
2 Miscellaneous Notes

- For questions, please send email to infocom@ece.gatech.edu.

- For really important questions relating to possible bugs in the simulator or the handout, also send e-mail to infocom@ece.gatech.edu.