The University of Kansas

Information and Telecommunication Technology Center

Technical Report of the
Networking and Distributed systems Laboratory

A KU-PNNI User's Manual Version 2.0

Phongsak Prasithsangaree, Gowri Dhandapani,
Bhavani Shanmugam, Kamalesh Kalarickal,
Venuprakash Barathan, and Douglas Niehaus

ITTC-FY2001-TR-18833-13

November 2000

Project Sponsor:
Sprint Corporation

Copyright © 2000:
The University of Kansas Center for Research, Inc.,
2335 Irving Hill Road, Lawrence, KS 66045;
and Sprint Corporation.
All rights reserved.

Contents

Chapter 1

Introduction

This user manual describes the KU PNNI simulation tool. It has a common user interface to setup a simulation experiment. This interface is designed to facilitate the user in specifying the minimal required parameters to setup a network and run experiments of varying complexity and magnitude. Its intention is to allow the user to specify the input script describing an experiment with ease. The tool parses the user input script, completes the necessary additional information left unspecified in the user input, and then performs the simulation.

This document is an attempt to make the user familiar with the abilities, limitations, and idiosynchracies of this tool. Chapter 2 provides a brief description of PNNI. Chapter 3 talks about the implementation of Multiple Peer Groups in our simulator. Chapter 4 explains about the AT-M addressing scheme. Chpater 5 deals with the Information Database structure. 6 discusses the architecture of the simulation software. Chapter 7 explains the crankback and alternate routing mechanism. Chapter 8 summarizes the call generator used to make calls, i.e. generate the traffic in the experiment. Chapters 9 and 10 describe the structure and details of the interface.

Several example input scripts and the output produced for them detailed in Chapter 11. Language details are provided in Appendix A and the BNF grammar is chronicled in Appendix B.

Chapter 2

PNNI

2.1 Introduction to PNNI

PNNI is an acronym for Private Network-Node Interface or Private Network-Network Interface. It is a topology state routing protocol in which nodes flood quality of service and reachability information to all other nodes so that each node obtains complete information about the topology of the network and its current state. In an effort to reduce the complexity of such a scheme, PNNI uses a hierarchical model to divide nodes into peer groups.

Features of PNNI include the following:

- It has the capability to automatically configure itself when the address structure reflects the topology of the network.
- The hierarchical model allows it to scale well for large networks.
- Crankback feature allows it to reroute around failed components during call setup.
- Connection follows the same path as the setup message used.
- Uses many routing metrics including cell transfer delay, cell delay variation, current and average peak load.

2.2 PNNI Routing

In general, PNNI is an ATM routing protocol that provides Quality of Service. Because ATM is connection oriented, a connection must be routed from the source to the destination during setup. To ensure QoS for the user, an efficient and adaptive method for setting up calls is essential.

PNNI divides a network up into groups of related nodes called *Peer Groups*. Within a peer group, nodes exchange topology state information with each other so that each node obtains a complete picture of the topology of the peer group and its state. One hierarchical level up, a logical node represents each peer group and summarizes state and reachability information for the peer group that it represents. Each peer group has a leader who communicates with the logical node to exchange information about the network.

2.2.1 Peer Group

A peer group consists of a group of network nodes that usually share a common trait - physical proximity, similar addresses, etc. The nodes are connected either by physical connections or logical connections. They share information with each other to assure that all nodes in the peer group have the same assessment of the network.

2.2.2 HELLO

Each node in the peer group periodically exchanges HELLO packets with its neighbors to determine its local state. A HELLO packet is an advertisement of one's address that requires the recipient to reply back with its address. It also includes some metric measurements about the state of the connection.

2.2.3 Node Peer FSM and Topology Database Exchange

After two nodes exchange HELLOs, if they determine that they are in the same peer group they begin to exchange topology databases. When the HELLO packets are exchanged and nodes synchronize with each other, they advertise the topology information they have in summary packets. When a node receives a summary packet it makes a list of topology information it needs and sends the request through request packets. Required topology information packets are created by bundling up all of the requested topology information related to a node in one group called *PNNI Topology State Element*. The PTSEs of different nodes thus created are put together into a packet called a *PNNI Topology State Packet*, PTSP, and sent to the peer node. This results in two nodes having identical network information. This only occurs between neighbors - flooding distributes information to the other nodes.

The PTSEs contain attributes concerning the state of the links and nodes in the peer group. This may include metrics such as delay, attributes such as security or capacity restrictions, or parameters such as cell transfer delay, cell delay variation, or cell loss ratio. They may also contain reachability information - i.e., the list of destinations which may be reached through a node.

While exchanging the topology databases, each node first informs the other of the 'version' of its PNNI topology State Elements. Any PTSEs that are newer than those currently held by the node are updated. The others are discarded. Once the databases have been synchronized, the two nodes advertise the link between them through flooding.

2.2.4 Flooding

The flooding of information between nodes in the peer group allows each peer group to accumulate a complete picture of the database, rather than just information about its immediate neighbors.

Each node packages its PTSEs up into a PTSP (PNNI Topology State Packet) and sends it out to all neighboring nodes. If the PTSEs contained in the PTSP are new or more recent than those held by the node, the PTSEs are updated and the PTSP is sent out to all neighboring nodes of the recipient except the PTSP originator. This propagates the information throughout the peer group.

Because the recipient acknowledges PTSPs, they are assured to be delivered. This is important because this guarantees reliable exchange of topology information, which is essential in such a scheme. If a PTSP is not acknowledged within a certain time frame, it is sent again. In addition, PTSEs have a life span. If they are not updated regularly, they are discarded.

Updates may happen periodically or be event driven. For example, many times the PTSE is updated with any change in the administration weight. The default flooding period is 30 minutes as defined by the ATM Forum [1]. Nodes have a hold-down time to keep flooding from happening too often and congesting the network.

2.2.5 Addressing and Reachability

Each node has an ATM address, this is a 22 byte address, the first byte being the level indicator, this gives the number of bits of common prefix within a peer group, the next byte indicates if it is a logical node or a physical node, these two bytes are followed by a prefix, a hardware address and the selector byte.

Because it is not feasible or desirable for a switch to explicitly advertise each individual reachable address, PNNI uses address prefixes to determine reachability.

When a switch learns of a connection to a reachable address, it checks to see if it has an entry for a summary address that is a prefix of the new address. If so, it advertises the summary address and not the individual address. If not, the address must be advertised individually and is called a foreign address.

A summary address might look like 39.9.1.1.X. This means that any address beginning with 39.9.1.1 may be reached through a particular node.

2.2.6 Source Routing and Call Admission Control

PNNI signaling is based on source routing which means that the source switch computes the path for the entire transmission. It enumerates the path in a *DTL (Designated Transition List)*. A DTL is a list of nodes and/or peer groups to traverse. Every source node follows a *Generic Call Admission Control (GCAC)* algorithm for pruning out those links that cannot support the call requirements. After that a path is computed using some node specific *Routing Policy*. Finally, at each Node-to-Node interface, the specific *Call Admission Control (CAC)* policy determines whether that path can support the requested traffic.

Because routing decisions are not necessary at every intermediate switch, source routing improves network performance. Additionally, this allows different switches to use different routing algorithms.

A routing function performs differently depending on what criteria are critical. In our simulator, single and multiple QoS routing functions are implemented. The Dijkstra's Single source shortest path algorithm is used for computing the route.

2.2.6.1 Single QoS Routing

Our simulation is able to find route which follows the criterion below

- minimum hop count
- minimum delay
- minimum administrative weight
- maximum bandwidth

When hop count is a criterion, the route with the minimum number of hops is returned. If there are more than one routes which have the minimum number of hops, the one to be returned is randomly selected.

When delay is a criterion, the route with the minimum amount of delay is returned. If there are more than one route are available, the one to be returned is randomly selected.

When administrative weight is a criterion, the route with the minimum administrative weight is returned. If there are more than one routes are available, the one to be returned is randomly selected.

When bandwidth is a criterion, the route with the maximum bandwidth is returned. If there are more than one routes that have maximum bandwidth , the one to be returned is randomly selected.

2.2.6.2 Multiple QoS Routing

To find a route which is optimum is a key of routing function. Sometimes routing with only one QoS constraint does not always return a route that best fits a connection requirement. Multiple QoS routing is

The multiple QoS routing function we were required to implement are listed below:

- Minimum Administrative Weight - Shortest Routing Algorithm

  The notion of "min-adweight-shortest" means that is link delay the first criterion and administrative weight is the second criterion. For all routes connected from a source node to a destination, it finds the route with the minimum delay. If there are more than one routes found, it finds the route with the minimum administrative weight from that set.

- Shortest-Minimum Administrative Weight Routing Algorithm

  The notion of "shortest-min-adweight" means that administrative weight is the first criterion and link delay is the second criterion. For all routes connected from a source node to a destination, it finds the route with the minimum administrative weight. If there are more than one routes found, it finds the route with the minimum delay from that set.

- Shortest-Widest Routing Algorithm

  The notion of "shortest-widest" means that link bandwidth is the first criterion and link delay is the second criterion. For all routes connected from a source node to a destination, it finds the route with the maximum bandwidth. If there are more than one routes found, it finds the route with the minimum delay from that set.

- Widest-Minimum Hop Routing Algorithm

  The notion of "widest-min-hop" means that hop count is the first criterion and link bandwidth is the second criterion. For all routes connected from a source node to a destination, it finds the route with the minimum hop count. If there are more than one routes found, it finds the route with the maximum bandwidth from that set.

2.2.7 PNNI Hierarchy

2.2.7.1 Peer Group Leader

Each peer group in PNNI has a leader which is little more than a communication point between the peer group and the logical node one level up in the hierarchy that represents the peer group. Other than this role in information distribution, the PGL behaves the same as the other nodes in the peer group.

There can be at most one leader per peer group and it is selected through *Peer Group Leader Elections (PGLE)*. The node with the highest leadership priority is elected leader. Note that this is an ongoing process, so the leader can change temporally.

2.2.7.2 Logical Nodes

A logical node has the responsibility of representing the summary information of its child peer group in its own peer group. The logical node must accumulate information about its child peer group and distribute it to the nodes in its own peer group. It also is obligated to relay information about its own peer group down to its child peer group.

2.2.7.3 Information Exchange Between Levels of the Hierarchy

The peer group leader is responsible for accumulating the complete topology state information for its peer group (as are all members of the peer group). The group leader can then summarize the information and pass it on to the logical node representing it.

The group leader passes reachability information and topology aggregation up the hierarchy. Topology aggregation is the accumulation of topology information that includes nodes that are not in your peer group through summaries of topology. PTSEs are never transferred up the hierarchy.

The logical node has the associated task of bundling up this summary information and distributing it within its peer group. Upon exchanging information with its peer group members and amassing its own topology information, the logical node passes all PTSEs that it has received down the hierarchy to the group leader. The group leader then floods it to all other members of the peer group. This information is necessary to allow nodes in the lower levels to route to all destinations in the routing domain. Through this exchange, the lowest level nodes learn of the topology (at least in summary) of the rest of the network.

2.3 PNNI Signaling

ATM signaling is a protocol used to set up, maintain and clear switched virtual connections between two ATM end users over private or public UNIs. The protocol is, in fact, an exchange of messages that take place between the ATM end user and adjacent ATM switch. The messages contain information that is used to build, maintain or clear the connection. The messages are segmented into cells at the signaling AAL and then transported over a standard signaling channel, VPI=0, VCI=5.

ATM signaling messages can be grouped into one of three functional categories: call establishment, call status, and call clearing operations.

Call establishment messages consist of the following:

SETUP Message: Sent by calling, or source ATM end-user, to network (defined here as nearest ATM switch connected to ATM end user over UNI) and from network (defined here as nearest ATM switch connected to destination ATM end-user over UNI) to called, or destination ATM end-user. Used to initiate connection setup. Contains information such as destination ATM address, traffic descriptors and QoS.

CALL PROCEEDING Message: Sent by destination end user to the network and by network to source ATM end user to indicate that call establishment has been initiated.

CONNECT Message: Sent by destination ATM end user to network and by network to source ATM end user to indicate that destination ATM end user accepts connection request.

CONNECT ACKNOWLEDGE Message: Sent by network to destination ATM end user to indicate call is accepted. May also flow from source ATM end user to network to maintain symmetrical call control procedures.

Call status messages are:

STATUS Message: Sent by ATM end-user or network in response to a STATUS ENQUIRY message.

STATUS ENQUIRY Message: Sent by ATM end-user or network to solicit STATUS message.

NOTIFY Message: Sent by ATM end-user or network to indicate information pertaining to a call/connection.

Call clearing messages consist of the following:

RELEASE Message: Sent by an ATM end-user to request the network to clear the end-to-end connection or is sent by the network to indicate that the VCC is cleared and that the receiving ATM end user should release the virtual channel and prepare to release the call reference after sending a RELEASE COMPLETE.

RELEASE COMPLETE Message: Sent by an ATM end-user or network to indicate that virtual channel and call reference have been released and that the entity receiving the message should release the call reference.

The following figure shows the flow of messages for a point-to-point connection.

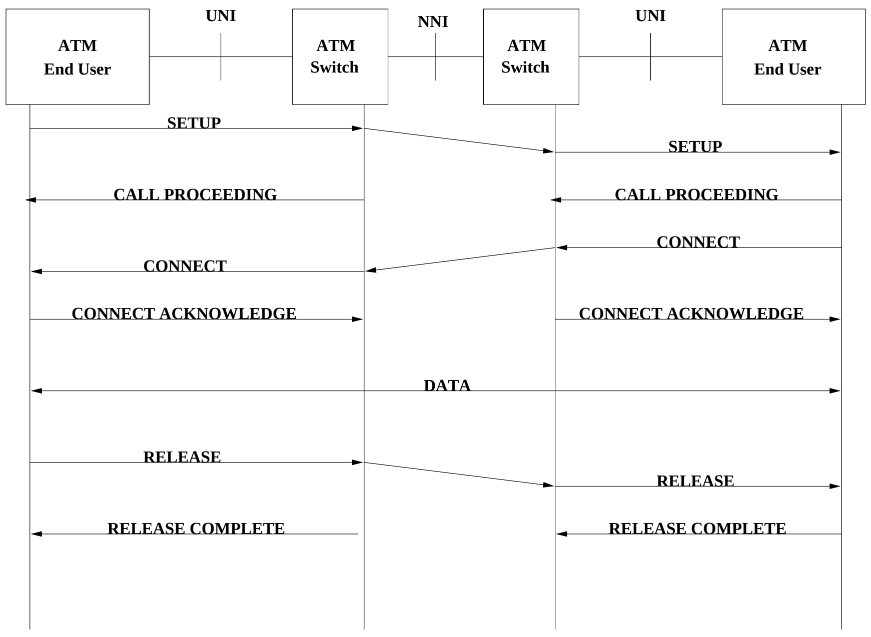

Figure 2.1: ATM Point-to-Point connection Message Flow

The ATM Forum has defined signaling standards UNI 3.0, 3.1 and 4.0. The signaling component of PNNI is used to forward the SVC request through the network of switches until it reaches the destination. PNNI signaling makes use of the network topology, resource, and reachability information provided by PNNI routing to progress an VSC request through the network. It is based on UNI 3.1/4.0 signaling but has been enhanced with several extensions specific for the PNNI environment. PNNI signaling supports new capabilities: specific QoS parameters, ATM anycast addressing and scoping, and ABR. Additionally PNNI uses two other techniques designated transit lists (DTL) and crankback with alternate routing, to successfully complete the SVC request and connection setup. The DTL enumerates a path from source to destination through the hierarchy and is computed by the source because PNNI uses source routing. Crankback with alternate routing is used to contend with network problems or inaccurately aggregated information.

2.3.1 Point to Point Connection

A point to point connection is a full duplex connection between two ATM endpoints. To establish such a connection, several steps are required. First, a path from source to destination must be computed. Second, the call must be established, preparing for failure and coping with it. Finally, the call must be released.

What follows is a brief description of each of the first two steps described above.

2.3.2 Designated Transit Lists

A designated transit list is a vector of information that defines a complete path from the source node to the destination node across a peer group in the routing hierarchy. A DTL is computed by the source node or first node in a peer group to receive an SVC request. Based on the source node's knowledge of the network, it computes a path to the destination that will satisfy the QoS objectives of the request. Nodes then simply obey the DTL and forward the SVC request through the network.

A DTL is implemented much like a stack as seen in Figure 2.2 [2]. The border, or entry node to each peer group specifies a path across that peer group. Figure 2.2 is an example of a DTL and an associated network.

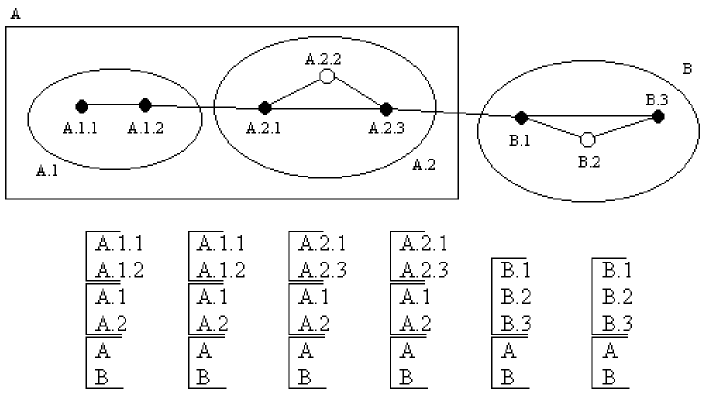
A
A.2.2
B
B.3
A.1.1
A.1.2
A.2.1
A.2.3
B.1
A.1
A.2
B.2
A.1.1 A.1.2 | A.1 A.2 | A B
A.1.1 A.1.2 | A.1 A.2 | A B
A.2.1 A.2.3 | A.1 A.2 | A B
A.2.1 A.2.3 | A.1 A.2 | A B
B.1 B.2 B.3 | A B
B.1 B.2 B.3 | A B

Figure 2.2: A DTL Example

In this example, we are trying to get from A.1.1 to B.3 in our network. The path is enumerated as shown in the first column (each column is a the DTL at each stop along its path. The message is sent from A.1.1 to A.1.2 as shown in the DTL. At A.1.2 it sees that it is the destination for level 1 of the hierarchy. It then moves down the DTL and sees that the new destination is PG A.2. The message is then sent to a border node on A.2 (A.2.1). Because A.2.1 is an entry point, it enumerates the path through its peer group and adds it to the DTL as seen in the third column. The message is then sent to A.2.3 as prescribed in the DTL. At A.2.3, it sees that it is the destination for level 1 of the hierarchy. It then moves down the DTL and sees that its peer group is the destination for level 2 of the hierarchy. It then moves down the DTL and sees that the destination is PG B. The message is then sent to a border node on B (B.1). Because B.1 is an entry point, it enumerates the path through its peer group and adds it to the DTL as seen in column 5. The message is then sent to B.2 and then to B.3 where it sees that it is the destination for level 1 of the hierarchy. It then moves down the DTL and sees that its peer group is the destination for level 3 of the hierarchy as well. This is the end of the DTL, so the node knows that message has arrived and that it is the recipient.

2.3.3 Crankback with Alternate Routing

When a DTL is created, it uses the currently available information about the network. For many reasons though, this information may be incorrect. Nodes or links may go down at any time or resources may become unavailable because of network congestion causing the DTL to route to an undesirable node. This may cause the SVC request to be blocked, short of going all the way back to the source to reestablish the connection, PNNI invokes a technique called crankback with alternate routing.

When the SVC request cannot be forwarded according to the DTL, it is cleared back to the originator of the DTL with an indication of the problem. This is the crankback mechanism. When failure first occurs, the message is sent back to the originator of the top entry of the DTL. At that point a new DTL (alternate route) may be constructed that bypasses the nodes or links that blocked the SVC request but which must match the higher-level DTLs which are further down in the DTL stack. If no path can be found, then the request is cranked back to the previous DTL originator. Ideally, it would not have to be routed all the way back to the source, but at some point on its way back, a better route can be found and the message could continue on its way to the destination with minimal delay. But if the source node is reached, then the crankback message is translated into a REJECT and the source must attempt another connection request.

In a single peer group scenario, if there is a failure at some point, the message is cranked back to the source node with an indication of the problem and an alternate route is established bypassing the failed nodes or links. In a multiple peer group scenario, if there is a failure in a particular peer group then the call is cranked back to the ingress border node of that peer group. The ingress border node then tries to find another route for transiting this peer group. The signaling message flow is as shown in Figure 2.3.

For example, imagine in our example of DTL routing in figure 2.2 above that node B.2 became unavailable, unbeknownst to the source, before our message began its journey. When it reached B.1 and realized that B.2 was unavailable, it would crankback the message. The message is then propagated back and any node that added an entry to the DTL can change the DTL. For example, A.2.3 would merely pass it along because it did not add an entry to the DTL. However, A.2.1 did. So did B.1. When B.1 decides to crankback, it would be the first to try and reroute the message because it was the last to modify the DTL. If B.3 is available, B.1 may change the DTL to reflect the new route from B.1 to B.3, bypassing B.2, but is not obligated to. If B.1 could not reroute the message or chose not to reroute the message due to some restrictions of its own, (in this case, kind of a dead end) then B.1 would send the message back to A.2.3 and so on and so forth until a new

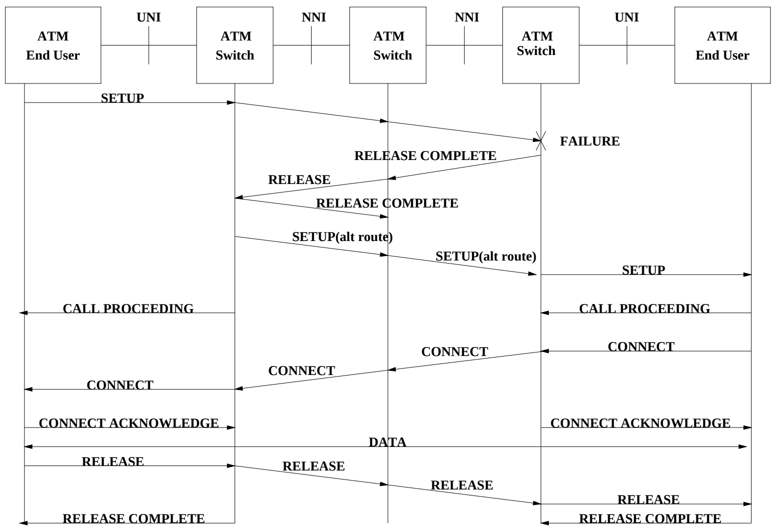

Figure 2.3: Message flow with Crankback and Alternate Routing

path is found.

This is PNNI's dynamic routing capability.

2.4 Summary

PNNI, the Private Network-Node Interface is a hierarchical routing protocol which uses topology state information to keep all nodes in the network aware of the network's state. There is an initial exchange of information, as well as upon significant changes in the network state. In this way, PNNI can provide many features absent in other routing protocols.

Chapter 3

Multiple Peer Groups in KUPNNI

3.1 Introduction to PNNI Hierarchy

PNNI is an ATM routing protocol that uses a hierarchical organization of nodes and links, and summarizes reachability information between levels in the hierarchy. This allows ATM networks to scale to a very large number of ATM switches, since it reduces the amount of topology and state information that needs to flow in the network. Each level of the hierarchy consists of nodes interconnected by links. Multiple nodes at one level can be summarized and represented as a logical node at a higher level. At the lowest level of the hierarchy, each node is a physical switching device and each link is a physical link. At higher levels, nodes and links are logical representations. At the lowest level of the hierarchy, each ATM switch is assigned a *peergroup ID* which identifies the peer group to which the switch belongs. A *peer group* is a collection of nodes, each of which exchanges full link state information with each other and obtain the same *topology database*, so that they share the same view of the world.

Figure 3.1 shows a network of 17 ATM switches that are organized into five peer groups. These peer groups are called lowest level peer groups and form the base of the hierarchy. A link that connects two nodes in the same peer group is called a *horizontal link*. A node attached to a link that crosses a peer group boundary is called a *border node*. Links that connect border nodes are called *outside links*.

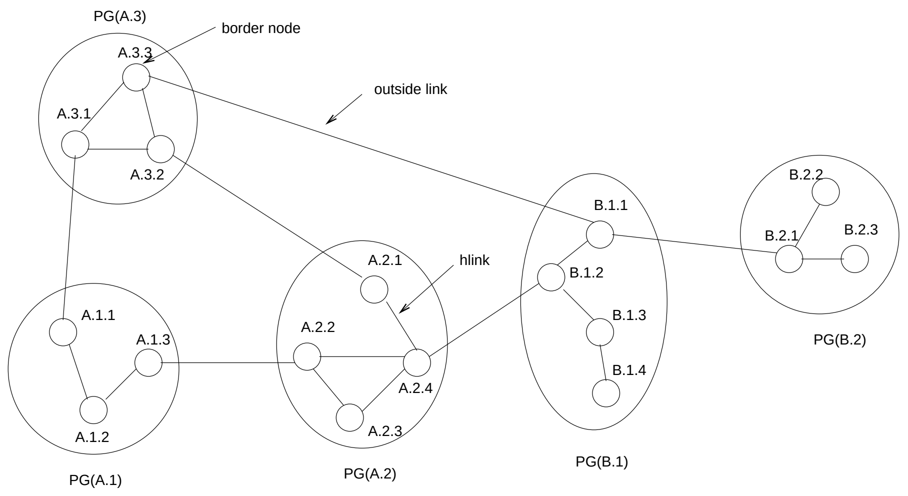

Figure 3.1: Lowest level of ATM Network

PNNI is a topology state protocol which is similar to a link state protocol. PNNI nodes flood information about themselves and their neighbors to all other members of their peer group. Information is distributed as *PNNI Topology State Elements (PTSEs)* within packets called PNNI Topology State Packets (PTSPs). Nodes assemble this information into a topology database which they use to compute routes. Nodes distribute the following types of information.

- *Nodal information* describes a node's identity and reachability.
- *Link State information* describes characteristics of horizontal links.
- *Uplink information* represents the characteristics of outside links.
- *Nodal State information* contains the representation of the internal details of a peer group

3.1.1 Forming the hierarchy

The nodes in each peer group elect one of their members to be the *Peer Group Leader (PGL)*. The ATM switch that hosts the PGL activates another logical PNNI node called the *Logical Group Node (LGN)* which represents it at the next level of the hierarchy. The LGN summarizes the peer group called its *child peer group* by performing *nodal and link aggregation*. At the next level, LGNs form their own peer groups based on the peer group IDs with which they have been configured. The peer group to which an LGN belongs is the *parent peer group* of the nodes that the LGN represents.

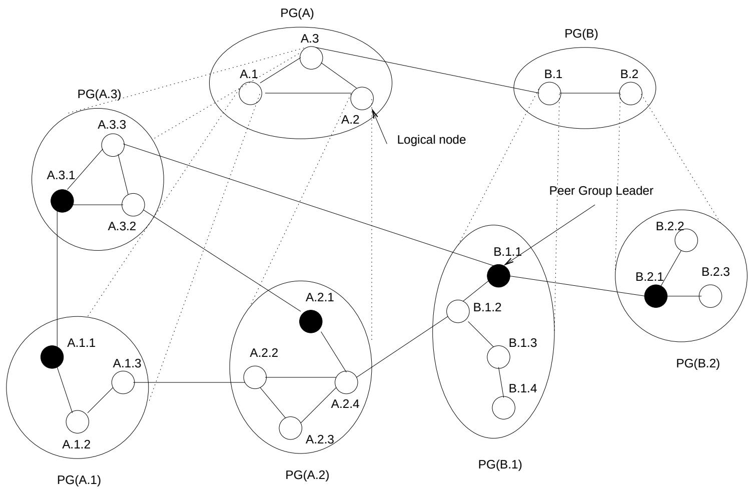

Figure 3.2: Second level of hierarchy

Figure 3.2 shows five peer groups, each peer group has elected a PGL. Each PGL has activated a LGN to represent its peer group at the next level of the hierarchy. The LGNs have also formed their own peer groups - PG(A) and PG(B) at their level. In this example, PG(A.2) is the child peer group of LGN A.2 and PG(A) is its parent peer group.

Limited routing information flows both up and down the hierarchy. The LGN aggregates topology state information and summarizes reachability information about its child peer group and floods this information throughout the parent peer group. The aggregation and summarization that occurs here reduces the amount of routing information that must be distributed and that each node must maintain, giving hierarchical PNNI its scalability. An LGN collects information from the parent peer group and injects the information into its child peer group. In this way, lower level nodes learn summarized information about the network outside their own peer group.

From the flooding of information throughout a peer group and from the summarized information down the hierarchy, each PNNI node constructs a topology database describing its own peer group and all ancestor peer groups. In order to route a connection request, a node must know which nodes within its peer group have connections to an ancestor peer group. A border node provides this information by advertising an *uplink* that connects the border node to a higher level peer group.

Figure 3.3 shows uplinks for PG A.1. In addition to connecting peer groups at different levels, uplinks enable LGNs to construct horizontal links to other LGNs.

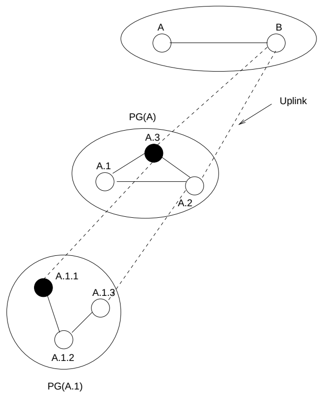

Figure 3.3: Uplinks

The process of building the next level of hierarchy continues until the entire network is contained in a single highest level peer group, illustrated in Figure 3.4. When the hierarchy is complete, each lowest level PNNI node has a complete topology database which enables it to construct a connected graph representing its logical view of the network.

3.1.2 Nodal and Link Aggregation

PNNI achieves scalability by hiding state information. One mechanism used to hide state information is the representation of a peer group as a logical group node (LGN). This type of information hiding is called node aggregation. A source node contemplating routes through an LGN needs some notion of the state of the logical node. The PNNI Specification offers two options:

- simple node representation
- complex node representation

A simple representation assumes that traversal of a node affects the end-to-end parameter values insignificantly, such as when the node represents a physical node. In this case, there is no cost, delay, or loss associated with transiting a node. When the simple node representation is used for an LGN, the entire LGN is treated as a point, with no resource constraints.

A complex node representation is useful when transiting a node is significant with respect to the connection's QoS requirements. This will naturally be true when a node is an LGN, representing a lowest level peer group and transiting the node really means transiting multiple physical hops. The interior reference point of a logical node is referred to as the *nucleus*. A border reference point of a logical node is known as *port*. A logical connectivity between the nucleus and a port is referred to as a *spoke* with a *radius*. *Exceptions* can be used to represent particular ports whose connectivity to the nucleus is significantly different from the default radius. Two ports may also

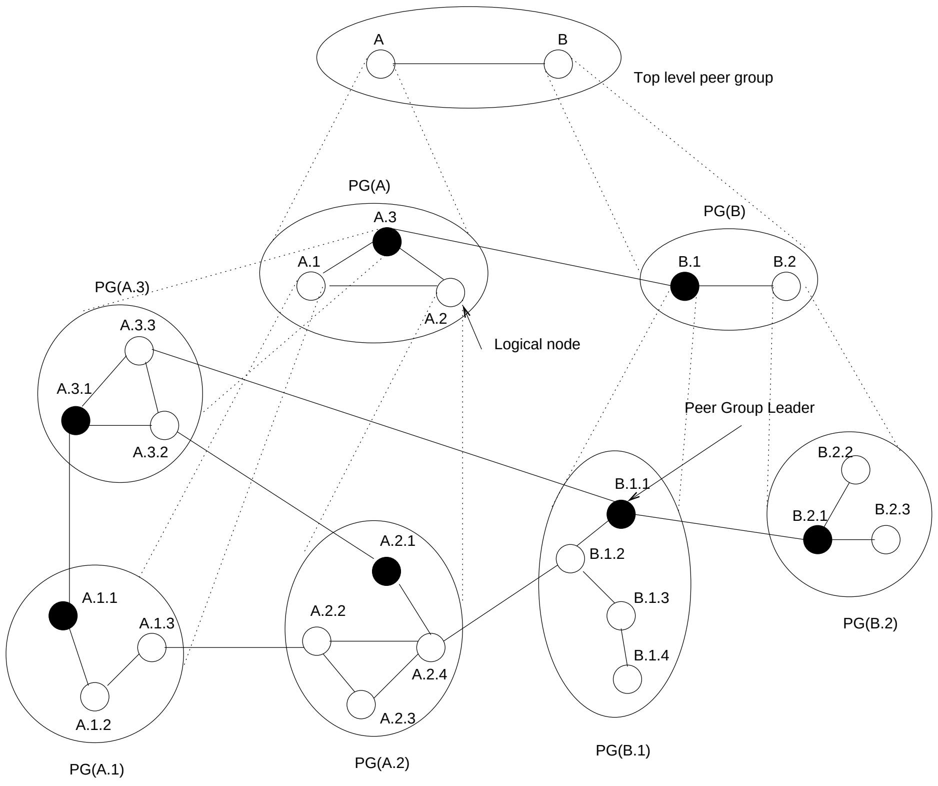

Figure 3.4: Complete hierarchy

be connected via a *bypass*. A router outside the domain makes routing decisions using only the routing costs between different ports of the domain, termed *port-to-port distances*. The complex node representation is shown in Figure 3.5.

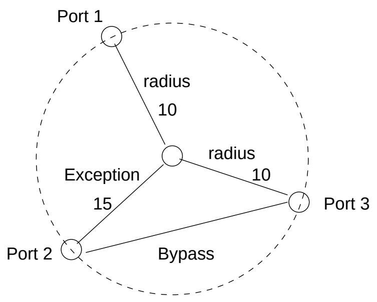

Figure 3.5: Complex Node representation

Topology aggregation usually consists of the following four steps:

- group or partition the network into domains and form the routing hierarchy
- derive the port-to-port distances in each domain
- represent the port-to-port distances in a compact manner
- exchange the aggregated information among the domains

Network partitioning depends both on administrative constraints and performance considerations. Exchanging information between domains is similar to exchanging information between network nodes inside a domain, and can be accomplished by flooding. Deriving the port-to-port distances and representing them in a compact manner are critical for routing performance and scalability and are discussed in Section 3.2.

Link Aggregation is part of constructing the PNNI hierarchy. When two peer groups have multiple links between them, link aggregation determines whether the common higher level peer group represents those links individually as parallel links between LGNs or combines them into a single link. The border nodes at the ends of each link exchange their *aggregation tokens* which are configured by the network manager. Links which have the same aggregation token are aggregated to form a single link at the next level of hierarchy. The state parameters of the logical link are derived from the underlying physical link depending on the policy chosen, the details of which are explained in Section 3.3

3.2 Nodal Aggregation Policies

The various nodal aggregation schemes that are supported by our simulator are

3.2.1 Full mesh

In this mesh representation, we representthe port-to-port distances by a matrix, with one entry per port pair. Thus, a domain with N ports has a matrix of size N^2. The information lost in full mesh aggregation, compared to the flat non-hierarchical routing is mainly the correlation between port-pair distances, since the paths between different port-pairs may share one or more common links. However, this information is not used by most routing algorithms. In general, the full mesh representation is accurate, but does not scale well with an increase in peer group size.

3.2.2 Symmetric Star aggregation

In this approach, the assumption is that the topology of the domain is symmetric, i.e., the distances are the same between any two ports. Thus, the distances are represented by a single parameter, the *radius* of the domain. This is much more scalable than the full mesh, since it reduces the routing information representation size complexity, the price it pays is reduced routing inaccuracy. The radius is calculated as the average of all port-to-port distances. The full mesh and the star representation are shown in Figure 3.6, note that all the spokes in the star representation have the same bandwidth value. For a*metric* parameter such as the *delay, administrative weight*, the radius is half the value of the diameter, however for an *attribute* such as the *bandwidth*, the radius is the same as the diameter.

3.2.3 Asymmetric Star aggregation

As a modification to the symmetric star approach, the distance of the border node from the nucleus is computed taking into account only its incoming distances or its outgoing distances rather than taking all port-to-port distances. The incoming distances to a particular node are the distances to reach this node from other nodes where as the outgoing distances are the distances to reach other nodes from this particular node. Each of these distances is represented as an exception in the star. This is expected to be a better approximation of the real distances than the symmetric star approach. There are a few alternatives for calculating these exceptions, using the average, the

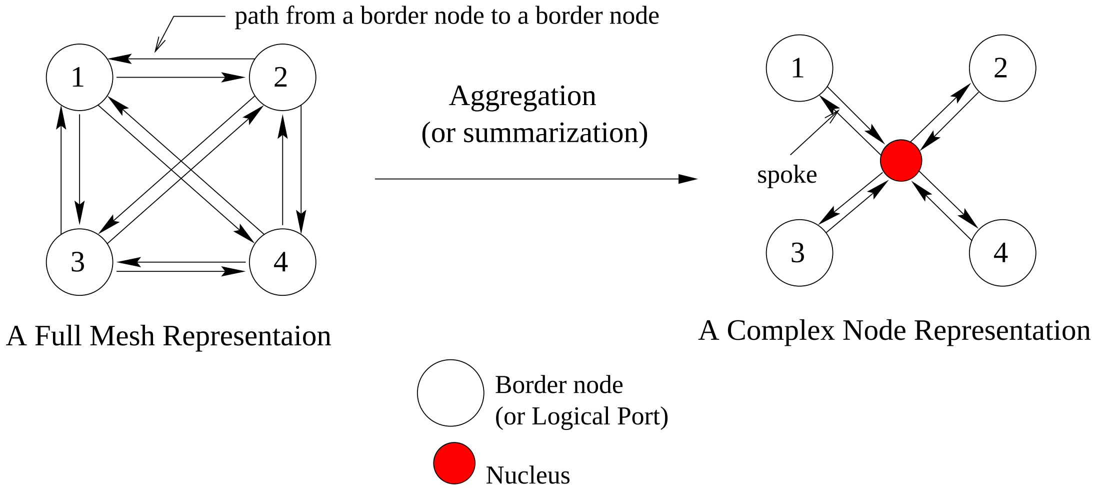
path from a border node to a border node
1
2
3
4
Aggregation
(or summarization)
A Full Mesh Representaion
spoke
A Complex Node Representation
Border node
(or Logical Port)
Nucleus

Figure 3.6: Nodal aggregation

optimistic, or the pessimistic approximations of the port-to-port distances. Optimistic distance means taking the maximum value for an attribute and the minimum value for a metric. Taking the optimistic distance would predict more resources than are actually available, this might cause a larger number of crankbacks. The pessimistic distances advertise the minimum of attributes and the maximum of metrics. So, predicting the worst distance would be underestimating the resources available in the peer group, causing unnecessary call rejects. Hence, using the average approximation seems to be a reasonable choice.

From Figure 3.7, showing the optimistic approximation for an *attribute* such as the bandwidth, we see that the bandwidth values for outgoing paths from *port 1* are *10, 15 and 20*, the maximum value *20* is assigned to the spoke from *port 1* to the *nucleus*. Similarly we can observe that the incoming distances to *port 1* are *10, 15 and 25*, the value of the spoke from the *nucleus* to *port 1* is assigned *25*. Also, note that since the figure shows attribute values, the radius is equal to the diameter.

Figure 3.8 shows the pessimistic approximation for a metric parameter such as the delay, the delay values for outgoing paths from *port 1* to the *nucleus* are *10, 15 and 20*. The pessimistic approximation chooses the maximum of the metric values and hence the exception from *port 1* to the nucleus has a value of $20/2 = 10$. Note that the value of the spoke is half that of the diameter because delay is an *additive metric*. Figure 3.9 shows the average approximation for delay values, the outgoing distances from *port 1* are *10, 15 and 20*, so the spoke from *port 1* to the *nucleus* is calculated as the average of the values $(10 + 15 + 20)/3/2 = 7.5$.

3.3 Link Aggregation Policies

Link aggregation is used to summarize the outside links between two peer groups. If two outside links to the same peer group have the same aggregation token, they are aggregated and represented in the next level as a single logical link. The state parameters for the advertised logical link could be computed based on any of the following algorithms:

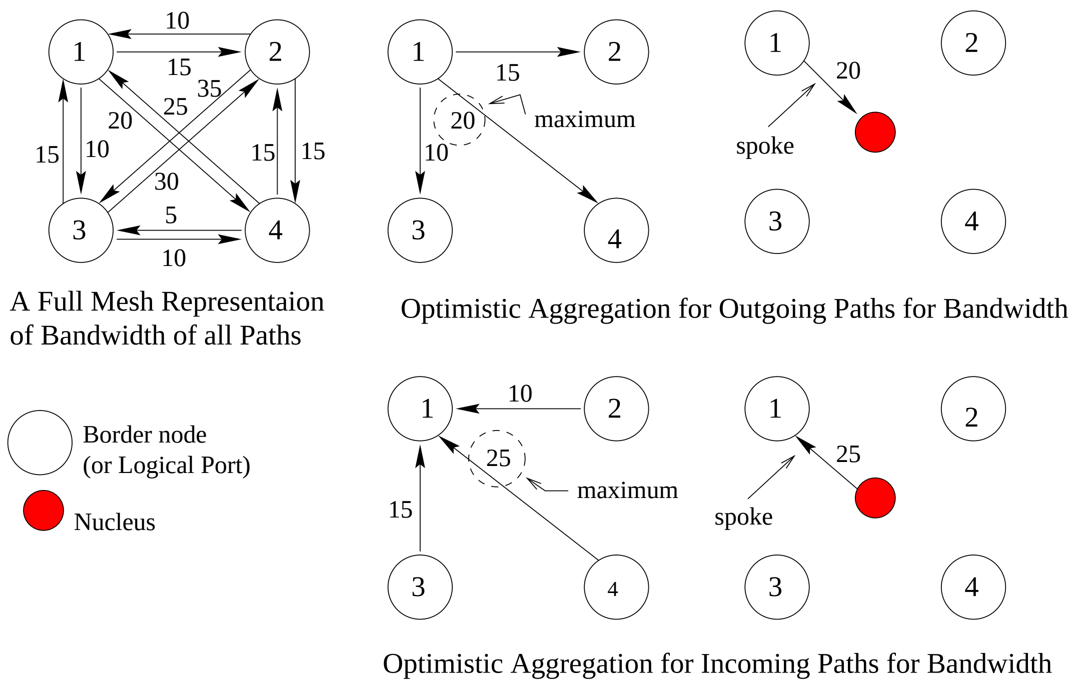
1
2
3
4
10
15
35
25
20
30
5
maximum
spoke
A Full Mesh Representaion of Bandwidth of all Paths
Optimistic Aggregation for Outgoing Paths for Bandwidth
Border node (or Logical Port)
Nucleus
Optimistic Aggregation for Incoming Paths for Bandwidth

Figure 3.7: Optimistic Nodal Aggregation for Bandwidth Values

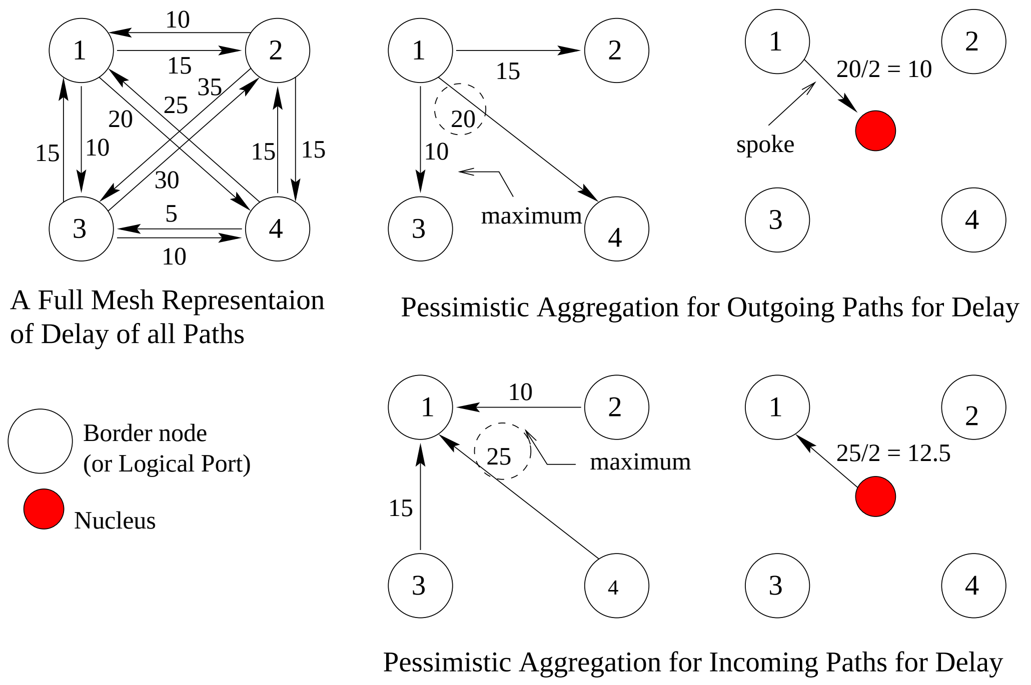
1
2
3
4
10
15
35
25
20
30
5
maximum
spoke
20/2 = 10
25/2 = 12.5
A Full Mesh Representaion of Delay of all Paths
Pessimistic Aggregation for Outgoing Paths for Delay
Border node (or Logical Port)
Nucleus
Pessimistic Aggregation for Incoming Paths for Delay

Figure 3.8: Pessimistic Nodal Aggregation for Delay Values

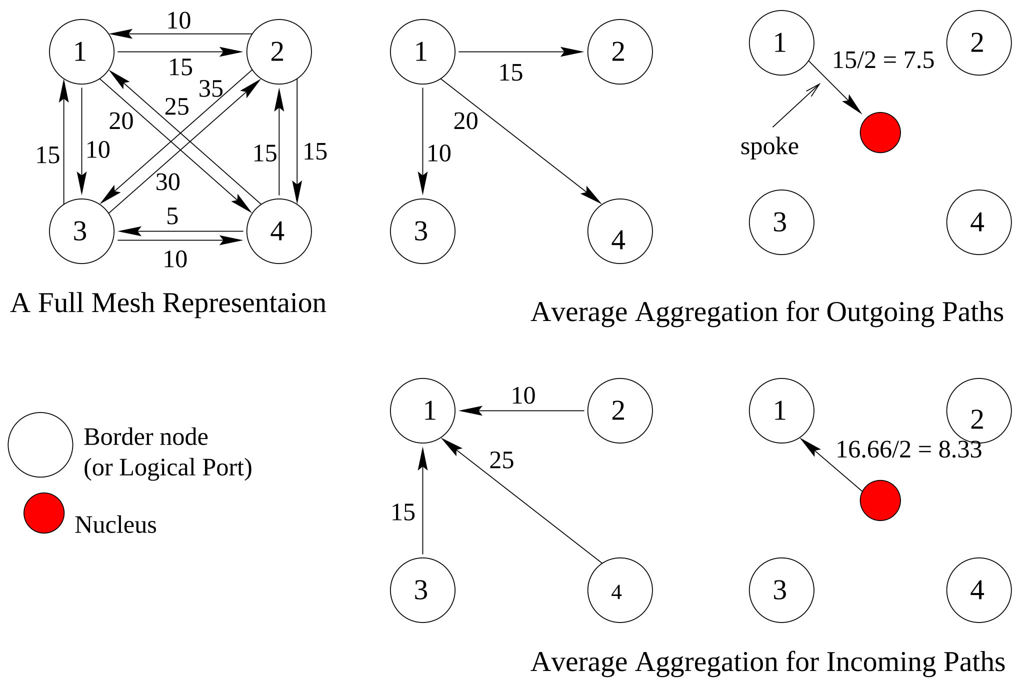
10
15
35
25
20
15
10
15
15
30
5
10
A Full Mesh Representaion
15
20
10
15/2 = 7.5
spoke
Average Aggregation for Outgoing Paths
Border node
(or Logical Port)
Nucleus
10
25
15
16.66/2 = 8.33
Average Aggregation for Incoming Paths

Figure 3.9: Average Nodal Aggregation for Delay Values

3.3.1 Optimistic link aggregation

This policy advertises the *best* QoS parameters of all the outside links. The best of the costs of links can be interpreted in two different ways, maximum and minimum. For a *metric* parameter, the best is the *minimum* cost of all outside links, for example, link delay. On the other hand, for an *attribute* such as the bandwidth of the link, the best choice would be the *maximum*. An example of optimistic link aggregation for bandwidth values is shown in Figure 3.10, observe that there are two outside linksbetween peer groups *B and C* with the same aggregation token. These two outside links are represented as one link in the next level, taking the optimistic distance for an attribute such as the bandwidth would mean the maximum of *150 and 300* and hence the value is *300*.

3.3.2 Pessimistic aggregation

This policy summarizes the link information by finding the *worst* QoS parameter values of the outside links. The worst of the costs will be the *maximum* of the metrics and the *minimum* of the attributes. An example of pessimistic link aggregation for delay values is shown in Figure 3.11, observe that there are two outside links between peer groups *B and C* with the same aggregation token. These two outside links are represented as one link in the next level, taking the pessimistic distance for a metric such as the delay would mean the maximum of *15 and 30* and hence the value is *30*.

3.3.3 Average aggregation

This policy computes the average of the QoS metrics and attributes for all the outside links having the same aggregation token and represents these values at the next level. An example for

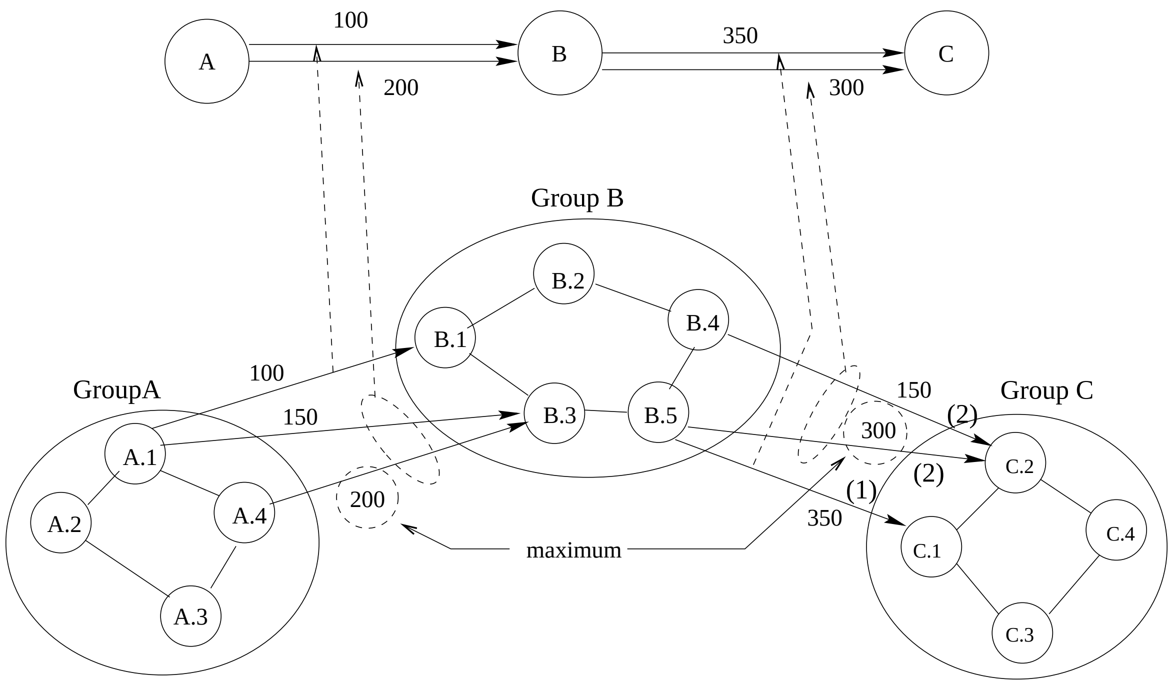

Figure 3.10: Optimistic Link Aggregation for Bandwidth Values

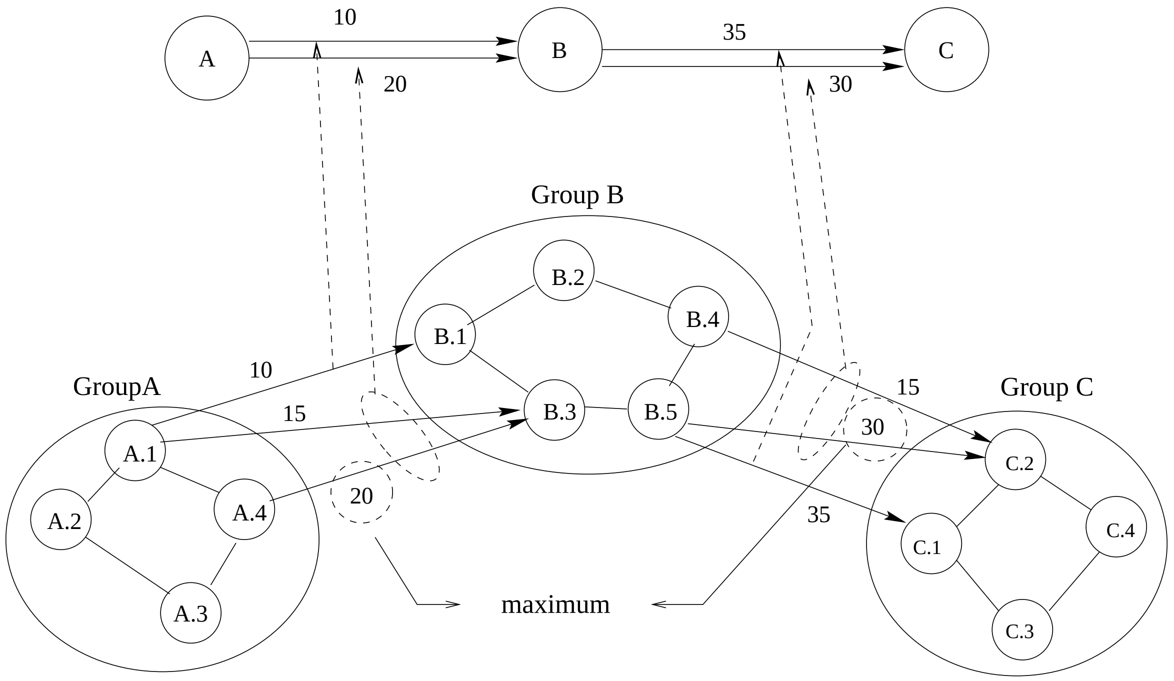

Figure 3.11: Pessimistic Link Aggregation for Delay Values

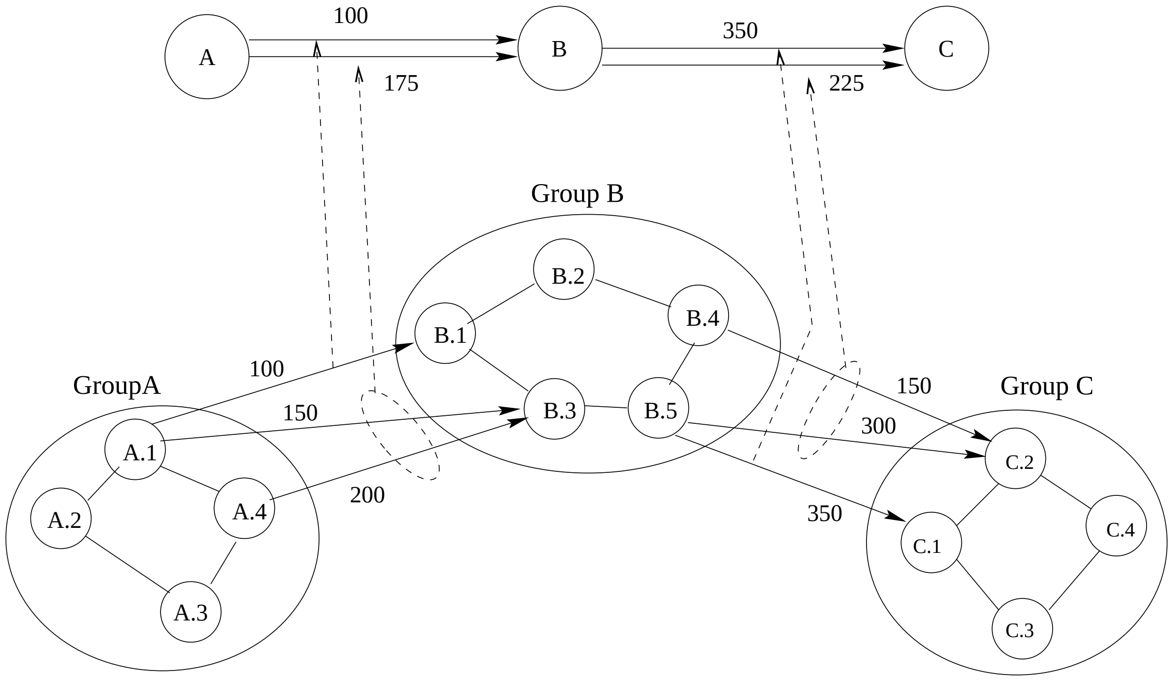

Figure 3.12: Average Link Aggregation for Bandwidth Values

bandwidth values is shown in Figure 3.12, the two outside links from peer group *B to C* are aggregated and represented as one link and the attribute value is calculated as the average of the two $(300 + 150)/2 = 225$.

Chapter 4

ATM Addressing Scheme

4.1 Introduction

This document gives an overview about the ATM addressing formats in general and how these addresses are used in the KUPNNI simulator.

4.2 ATM Addressing Formats

For the purposes of switched virtual connections, an ATM endsystem address uniquely identifies an ATM endpoint. The format of an ATM address for endpoints in private ATM networks is modelled after the format of an OSI Network Service Access Point (NSAP), as specified in ISO 8348 and ITU-T X.213.

The structure of the ATM Address can have three different formats as shown in Figure 4.1

Each of the address fields in the formats are explained below.

- Initial Domain Part (IDP) The Initial Domain Part (IDP) specifies an administrative authority which has the responsibility for allocating and assigning values of the Domain Specific Part (DSP). The IDP consists of two fields, the Authority and Format Identifier (AFI) and the Initial Domain Identifier (IDI).

  The AFI identifies the authority allocating the Data Country Code, International Code Deignator, or E.164 number; the format of the IDI, and the syntax of the remainder of the address. The length of this field is 1 octet. The following codes are specified.

  ```
  AFI             Format of IDI and DSP

  39              DCC ATM Format
  47              ICD ATM Format
  45              E.164 ATM Format
  ```

- Data Country Code (DCC) The Data Country Code specifies the country in which the address is registered. The length of this field is two octets. The codes will be left justified and padded on the right with the hexadecimal value 'F' to fill the two octets.

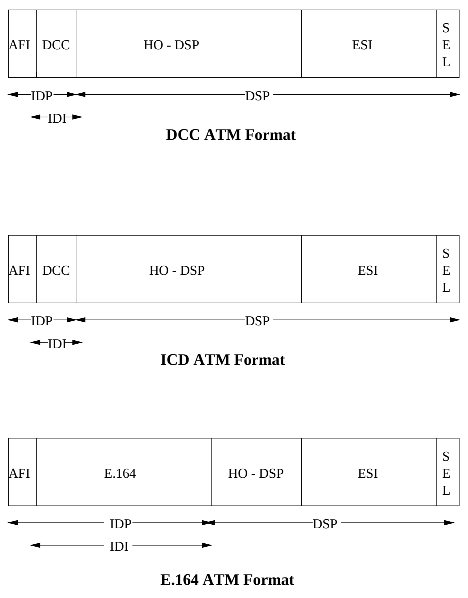

Figure 4.1: ATM Addressing Format

- International Code Designator (ICD) The International Code Designator specifies an international organisation. The length of this field is two octets. The codes will be left justified and padded on the right with the hexadecimal value 'F' to fill the two octets.
- E.164 (E.164) E.164 specifies Integrated Services Digital Network Numbers. These numbers include telephone numbers. The international format of these numbers will be used. The length of this field is 8 octets.
- Domain Specific Part (DSP) The Domain Specific Part is subdivided into the High Order DSP (HO-DSP) and low order part which consists of the End System Identifier (ESI) and Selector (SEL).
- HO-DSP The coding of this field is specified by the authority identified by the IDP. The authority determines how identifiers will be assigned and interpreted within that domain. The contents of this field not only describes the hierarchy of addressing authority, but also topological significance. That is, the HO-DSP should be constructed in such a way that routing through interconnected ATM subnetworks is facilitated.
- End System Identifier (ESI) The end system identifier identifies an end system. This identifier must be unique within a particular value of the IDP + HO-DSP. In addition, to ensure the ability of an end system to autoconfigure its address, this end system identifier can be a globally unique identifier specified by an IEEE MAC address. The length of this field is 6 octets.
- Selector The selector is not used for ATM routing, but may be used by end systems. The length of this field is 1 octet.

4.2.1 Example

As an example an ICD Format ATM Address would look like this.

```
          AFI ICD ---------HO DSP----- -----ESI---- SEL
      96:160:47.0005.80ffe1000000f21a2e06.0020481a2e06.00
```

4.3 Addressing in PNNI

Addressing and identification of components of the PNNI routing hierarchy are based on the use of ATM End System Addresses and/or prefixes applied to ATM End System Addresses. ATM End System Addresses are inturn modelled after NSAP addresses as explained above.

ATM End System Addresses are 20 octets long. PNNI routing operates on the first 19 octets of the ATM address. The selector(20th) octet has only local significance to the end system and is ignored by PNNI routing.

PNNI operates in a topologically hierarchical environment. The structure of the hierarchy is defined by the peer group IDs used in the routind domain. Address assignment has a hierarchy that should generally correspond to topological hierarchy, for proper scaling. This allows address summarization where an address prefix represents reachability to all addresses that begin with the stated prefix.

4.3.1 Level Indicator

PNNI entities(nodes, links and peer groups) occur at various hierarchical levels. The level specifies a bit string length and reanges from 0 to 104. Given two entities, where one is an ancestor of the other, the ancestor is a higher level entity and will have a smaller level indicator than the other. The level indicator is absolute in the sense that it specifies the exact number of significant bits used for the peer group ID. PNNI levels are not dense, in the sense that not all levels will be used in any specific topology. A peer group with an ID of length 'n' bits, may have a parent peer group whose identifier ranges anywhere from 0 - n-1 bits in length. Similarly a peer group with an ID of length 'm' bits may have a child peer group whose identifier ranges from m+1 to 104 bits in length.

4.3.2 Peer Group Identifiers

A peer group identifier is a string of bits between zero and 104 bits (13 octets) in length. Peer group identifiers must be prefixes of ATM End System Addresses such that the organisation that administers the peer group has assignment authority over that prefix. For example, if an organisation is given an n-bit prefix, it may assign peer group identifiers with length n or greater, but not less than n.

Peer group identifiers are encoded using 14 octets; a 1 octet (8 bit) level indicator followed by 13 octets of identifier information. The value of the level indicator must be between zero and 104 (bits). The value sent in the identifier information field must be encoded with the 104-n right most bits set to zero, where n is the level.

4.3.3 Node Identifiers

The Node Identifier is twenty-two octets in length, and consists of a one octet level indicator followed by a twenty one octet value which must be unique within the routing domain. The level of a node is the same as the level of its containing peer group. Two methods for creating nodeIDs are currently defined.

For nodes which do not represent a peer group:

- the level indicator specifies the level of the node's containing peer group.
- the second octet takes the value 160; this helps distinguish this case from the case below since an encoded peer group ID cannot begin with this value.
- the remainder of the node ID contains the twenty one octet ATM End System Address of the system represented by the node.

For a logial node which represents a child peer group say A.2 in its parent peer group A:

- the level indicator specifies the level of the peer group containing the LGN (i.e., the level of peer group A)
- the next 14 octets are the encoded peer group ID of the child peer group A.2
- the next 6 octets contain the End System Identifier (ESI) of the physical system implementing the logical group node functionality.
- the last octet of the nodeID is zero.

4.3.4 Host Identifiers

The Host Identifier is twenty octets in length, it consists of the 13 octet switch prefix and the 6 octet ESI and the SEL 1 octet.

4.3.5 A Hierarchical example

The example shown in Figure 4.2 illustrates how the addresses could be assigned, however this example is not complete in all respects.

A.1.1 and A.1.2 are the peer groups at the lowest level and the addresses of the nodes in these peer groups are as shown,

```
Peergroup A.1.1
A.1.1.1             96:160:47000580ffe1000000f21a2e06.08002009f5f4.00
A.1.1.2             96:160:47000580ffe1000000f21a2e05.00e02924787f.00

Peergroup A.1.2
A.1.2.1             96:160:47000580ffe1000000f21a2f06.006097109b31.00
A.1.2.2             96:160:47000580ffe1000000f21a2f05.aa000400fdc3.00
```

Note that the above peer groups have a level indicator of 96 which means that the first 12 bytes of all the nodes in the peer group will be the same. The hosts connected to these nodes will inherit the switch prefix from the node they are connected to and they will have their own ESI, their addresses could be

```
Host A.1.1.1.1      47000580ffe1000000f21a2e06.000400d88b8d.00
Host A.1.1.1.2      47000580ffe1000000f21a2e06.08002b39c847.00

Host A.1.2.2.1      47000580ffe1000000f21a2f05.08002b9c0d5d.00
```

Now these peer groups are represented by the logical nodes A.1.1 and A.1.2 in the next hierarchical level. Their addresses are

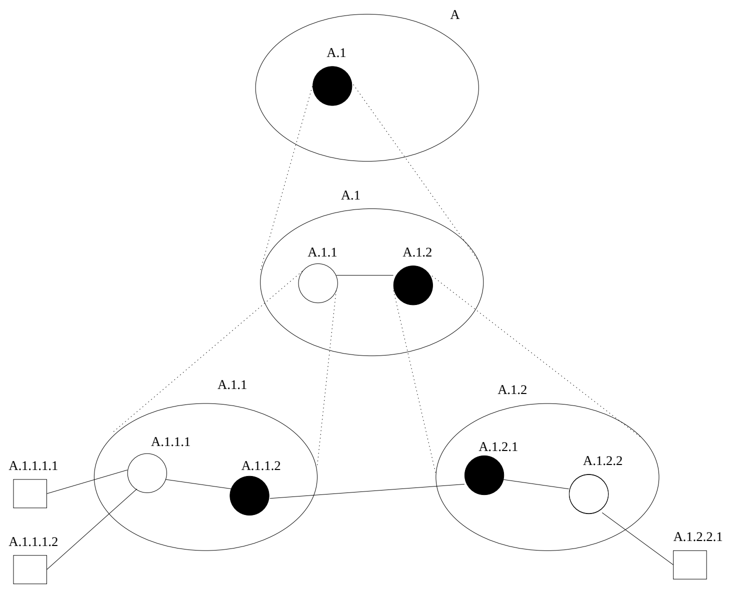

Figure 4.2: An Hierarchical Example

```
Peergroup A.1
LGN A.1.1          88:96:47000580ffe1000000f21a2e00.00e02924787f.00
LGN A.1.2          88:96:47000580ffe1000000f21a2f00.006097109b31.00
```

These logical group nodes have a level of 88 which means the first 11 octets of all the nodes in this peergroup will be the same. The logical group nodes have an encoded 14 octet peer group identifier as explained above. The ESI for these nodes will be that of the physical system implementing the logical group node functionality.

At the next level of hierarchy, the peer group A.1 is represented in peer group A by the logical group node A.1, the address of A.1 is

```
Peergroup A
LGN A.1            80:88:47000580ffe1000000f21a0000.006097109b31.00
```

The above example explains how addresses could be assigned in a multiple peer group hierarchical model.

Chapter 5

A Glance of Information Database Structure

5.1 How Information Database is constructed ?

5.1.1 What is Flooding Mechanism ?

The flooding mechanism is the way that a node discovers the resource information of other n-odes. Every node generates its own PTSEs with the initial information, and floods them out to its neighbor node in the form of packet called PTSP.

The PTSP packet includes these following elements:

- Originating Node ID
- originating Peer group ID
- a list of PTSEs

The PTSP is received by another node. The node determines whether it should update its database using resource information contained in the PTSE. After that, the PTSE is flooded again to another neighbor node. The flooding still continues until all the nodes have the same database.

5.1.2 How the Database is constructed ?

The database is constructed by receiving PTSEs from the flooding mechanism and adding the resource availability information which is extracted from PTSEs into its database.

5.1.3 When the resource availability information in the Database is invalid ?

In order to guarantee that a call connection is successful, the resource information used by the router must be valid. To assure the validity of the information, the Database should be updated or refreshed at a particular time. The database will be updated when there is a significant change of the resource. Otherwise, the database should be refreshed by again flooding out its resource availability information similar to that in the initialization phase. The latter case is periodically in-voked and the period is specified by the product of PTSELifetimeFactor and PTSERefreshInterval parameters.

5.2 What is inside the Information Database ?

The Information Database is a collection of PTSE!

5.2.1 What are inside the PTSE ?

- Type
  It indicates which restricted information groups are allowed to appear inside of the PTSE (see Section 5.14.9 in PNNI specification for details)
- Identifier
  When a node originates multiple PTSEs, each describes different pieces of the node's environment.
- Sequence Number
  It indicates which PTSE is "more recent" when multiple PTSEs simultaneously exist.
- Checksum
  It indicates whether the received PTSE is corrupted.
- Age (Time to Live)
  It is specified by the product of PTSERefreshInterval and PTSELifetimeFactor
- List of Resource Available Information Group
  It contains all link and nodal information groups.

(for more details, see section 5.8.2.2 in PNNI v1.0 specification)

5.3 When the Database must be updated.

The database can be updated with these events:

- Significant change of resource availability
- Expiration of Database

5.3.1 The Significant Change of Resource Availability Information

The significant change detection is the method to assure the sufficient resource availability of the connection when it is established.

Since every node has its own database which is similar to every node after the initialization phase. When the node make a call connection, its resource availability is decreased. However, the resource availability information in others nodes is not changed. If the availability is significantly changed, the call routed which the "old" information will be failed by the connection admission control (CAC). So the PNNI is handled this situation by flooding a new PTSE with a new PTSE containing a new resource availability information in order to update the Database of other nodes.

Note that it also happens in the case that a node releases a call connection. Its resource availability is increased, and it verifies whether the change is significant or not.

The significant change is detected before and after allocating or deallocating the resource. If the node finds that its resource availability is significantly changed, it will create a new PTSE containing new information represented in an Resource Availability Information Group (RAIG) element. Then, the new PTSE is flooded out to the network.

5.3.2 The Expiration of Database

The database of a node should not exist for such a long time when there is no significant change in the node. Therefore, the database should be updated periodically.

The PNNI v1.0 specification gives us two parameters to specify the ”age” of database, PTSERefreshInterval and PTSELifetimeFactor.

PTSERefreshInterval is the time between re-origination of a self-originated PTSE in the absence of triggered updates. The node will re-originate its PTSEs at this rate to prevent flushing of these PTSEs by other nodes.

PTSELifetimeFactor is a multipliable factor used to calculate the initial lifetime of self-originated PTSEs. The initial time is set to the product of the PTSERefreshInterval and the PTSELifetimeFactor.

The defaults of PTSERefreshInterval and PTSELifetimeFactor are 1800 seconds and 200equals to the product of those two parameters. Therefore, the default value of the database age is 3600 seconds. Note those time are simulation time.

5.4 How the Database is updated

5.4.1 What to do when a node receives a new PTSE ?

- Look if the node already has that PTSE in its Database.
- If yes, then look at the PTSE age.
  - If the PTSE age is expired, delete the existing PTSE from the Database.
  - If no, (this indicates an re-originated PTSE). delete the existing PTSE from the Database and insert a new one into its Database. After that, flooding the new PTSE to its neighbor node(s).
- If no, (this indicates the first PTSE it receives.) create an entry for the new PTSE and insert the entry into the database. After that, flood the new PTSE to neighbor node.

5.4.2 What to do when the Database is expired ?

- Delete the ”old” PTSE and tell other nodes to delete it also by flooding a ”dummy” PTSE with the expired age.

5.4.3 What to do when the Database needs to be refreshed ?

- Retrieve each PTSEs in the Database.
- Create a copy of the PTSE retrieved from the database. However, this new PTSE has the higher sequence number of PTSE and the new age which is specified by the product of PTSELifetimeFactor and PTSERefreshInterval parameters.
- Flood the new PTSE out

Chapter 6

Architecture

The KU PNNI network simulation architecture has been developed on Bellcore's Q.port software. For simulations, Q.port's real time based scheduling mechanism is modified to support a virtual time based scheduling mechanism. In the Q.port software Q.93B signaling module is extended to support an additional PNNI information element, Designated Transition Lists (DTL). Naval Research Laboratory (NRL)'s ProuST PNNI architecture, is interfaced with Q.Port for providing the PNNI routing subsystem module. Detailed models of simulation model is provided in the section 6.1.

6.1 Simulation

The Q.Port signaling software's scheduling architecture was modified to provide the simulator capability. In this architecture, all of the switch and host modules involved in the simulation register with a single instance of a *core-reactor* class as shown in the Figure 6.1.

The functions the core reactor class offers are as follows:

- *Registering and dispatching multiple timer events:* The Reactor module holds a pointer to the timer manager. All the modules associated in a switch or host when request for registering a timer, request the reactor through the *schedule-timer* interface it provides. The reactor forwards these requests to the *timer-manager* which it owns.

- *Posting multiple Q.Port internal messages:* The Reactor supports the posting of the internal messaging between different Q.Port modules. When the Q.Port modules need to post messages, they call the *post ticket* interface of the reactor. The reactor forwards this to the *ticket-dispatcher* which it owns.

Since all the Q.Port modules of different switches now report to one common reactor scheduler, Q.Port can support a single schedule oriented, discrete event simulator. In Figure 6.1 a new class *SimKernel* is added to schedule the events to either a specified duration of time or until the simulation ends with no more events to be scheduled. The *SimKernel* class maintains virtual time which is updated whenever a timer event is scheduled. The ticket dispatcher is a class which schedules the tickets registered. The tickets are events which are either the timer events or the inter Q.Port module messaging events. The Input/Output (I/O) manager registers a ticket with the ticket dispatcher for scheduling timer events and the I/O events. Since in the simulation there is no interprocess communication involved, the I/O manager is modified to avoid checking for the input data from the other processes and all the I/O manager tickets are used to service the simulation timer events. The timer events are scheduled by the timer manager, which calls the

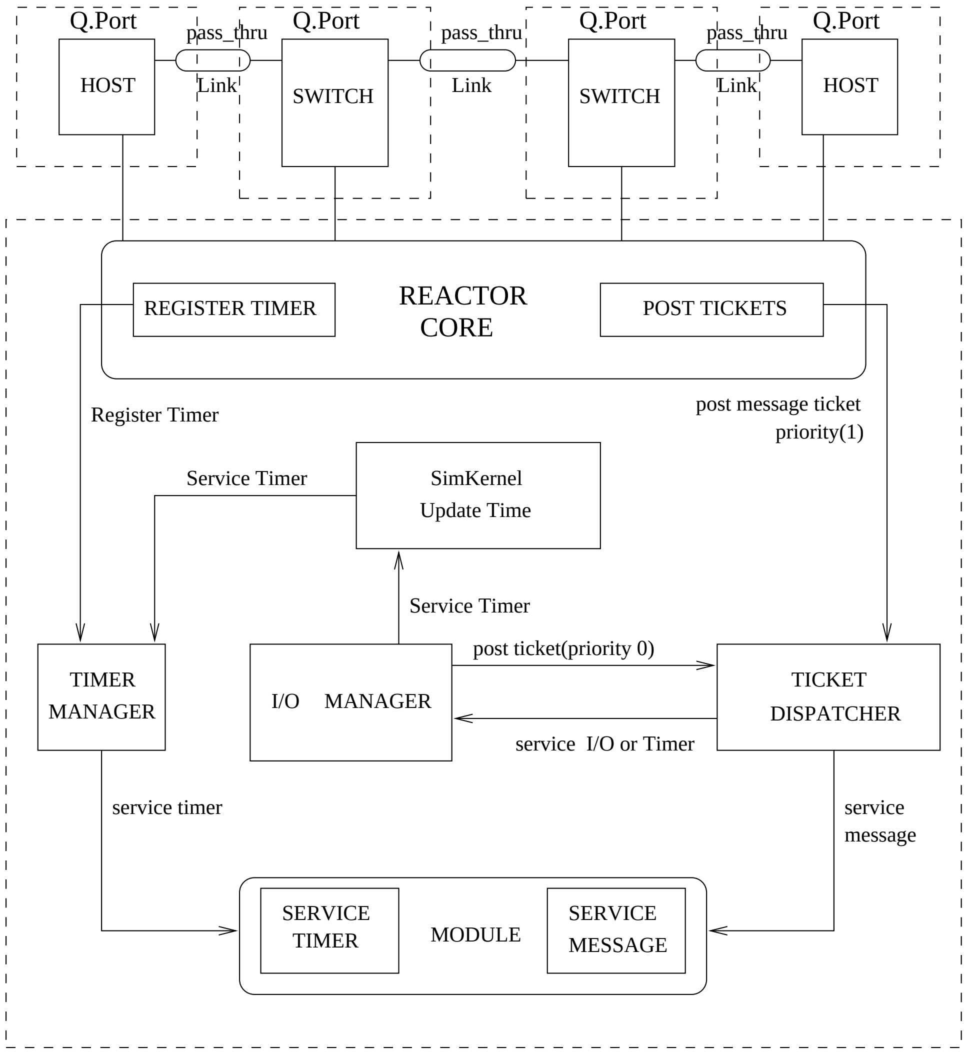

Figure 6.1: KU PNNI Simulator Model

handle-timeout interface of the module which registered the timer. The message events are handled by the the *Ticket-Dispatcher* which calls the *process* interface of the message ticket which was posted.

6.1.1 Simulation Kernel

A simulation kernel class *SimKernel* is added which is used to schedule simulator events. It interacts with the timer manager for scheduling timer events. It maintains virtual time for the simulation. This virtual time is used by the other modules while registering new timer events. The *I/O Manager* class is interfaced with the class *SimKernel* to schedule timer events when the reactor is running in the *simulation* mode. The *SimKernel*'s service routine is used to schedule the timer events. It schedules the next event if the next event's scheduled virtual time is less than the duration of the simulation. It updates the current time to the time, the current event is scheduled to be run.

6.1.2 Priority of Servicing

Two levels of priority are used for scheduling the events. They are:

- *Priority 1:* The events generated by inter Q.Port module messaging are assigned level 1 priority which is the highest priority. The messaging between the Q.Port modules Switch Call Control(SCC), PNNI Routing Service (PNNI RS), Static Router, and the Fabric are modified to suit this priority.
- *Priority 0:* This is the low level priority assigned to the servicing of the timer events.

With this priority setup, the tickets posted for messages are first cleared and only when this queue is empty, the queue for timers is serviced. This helps in processing an incoming message completely through different protocol stacks. Timer interrupts are secondary to processing a message in hand. There is provision for adding additional priorities if required.

6.1.3 Simulation of Link and Queueing Delays

The Simulation model includes simulation of links between the ports of nodes and hosts. Simulation of link delays, queueing delays, and queue length is also included.

6.1.3.1 Simulation of Links Between Ports

The ATM Adaptation Layer 5 (AAL5) module of the Q.Port is the bottom of the signaling stack and represents the port which can be connected to the peer AAL5. This module is modified for the simulator to know before hand, the object pointer of the peer AAL5 object. To each AAL5, the peer AAL5 object pointer is specified during booting up of the configuration. When a data needs to be transferred to the peer AAL5, the *SendToPeerAal5* method passing the data to the peer entity is called. This connectivity of links between simulator ports is labelled as *pass_thru* link.

6.1.3.2 LinkDelays and Queuing delays

The simulation model supports queueing of packets. These packets could be the Q.93B signaling packets encoded within the *Qsaal* PDUs or PNNI Route Control Channel(RCC) packets. The *AAL5* class has been modified to support a *queue length* which could be specified during configuration. Packets which arrive after the queue count exceeds the queue length are dropped. The *queuing delay* on the packets is also supported. This is done by multiplying the *queue size* in the switch's ports at the time of arrival of a packet by the *mean processing delay*. This *processing delay* can be configured. A measure of the processing delay is obtained by measuring the average time taken over processing of a large number of switch events. The link delay is added to the queuing delay, making the total delay before the packet is taken up for processing. The link delay is configurable and the AAL5 module is modified to include this information. Hence when a packet comes to the AAL5 module from the other side of the link the cumulative delays (Queuing delay and Link Delay) are measured and a timer is started with this time. When the timer expires, the AAL5 module takes up the packet for processing, giving it to the next stack in the protocol hierarchy.

6.1.4 Control Flow within Simulation Kernel

The working of the simulation kernel is best described by looking at the flow of control wihthin the various modules described above. We will describe the basic nature of event driven virtual time simulation and enumerate the stages of its execution within the simulation kernel. This is shown in Figure 6.2.

6.1.4.1 Event Driven Virtual Time Simulation

A discrete event-driven simulation is a popular simulation technique. Objects in the simulation model objects in the real world, and are programmed to react as much as possible as the real objects would react. A priority queue is used to store a representation of ”events” that are waiting to happen. This queue is stored in order, based on the time the event should occur, so the smallest element will always be the next event to be modeled. As an event occurs, it can spawn other events. These subsequent events are placed into the queue as well. Execution continues until all events have been processed.

The Q.Port reactor has been re-structured to support the above notion of events queue and virtual time of event execution.

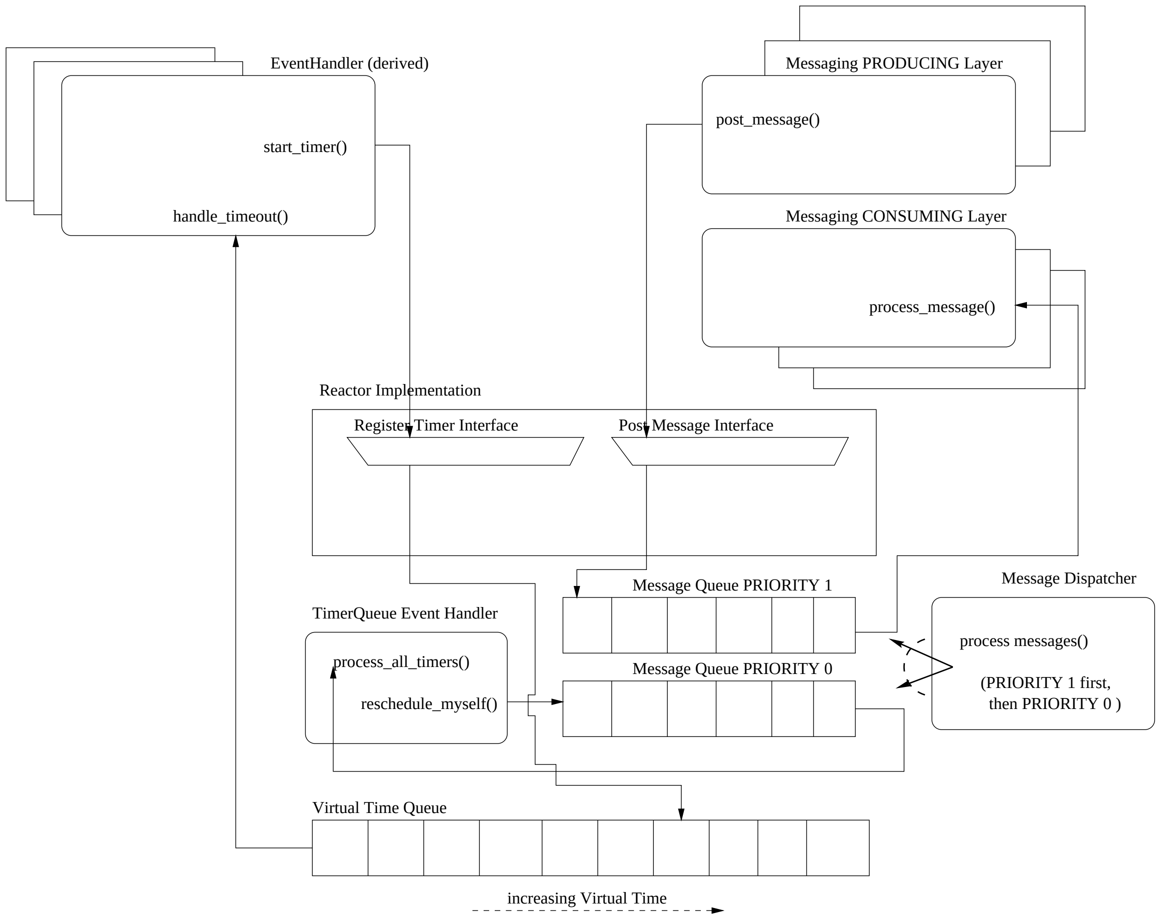

Figure 6.2: Control Flow within Simulation Kernel

6.1.4.2 Walk Through

We begin by describing virtual time model. It consists of:

1. Reactor Implementation: This has two interfaces that allow registering of timer events and message events. These are used by the modules to register timers and pass messages to each other.

2. Post Message Interface: This is interface by which software modules talk with each other. For instance, whenever the Switch Call Control module wants to invoke the Router module, it would pass a message to the router module using the Post Message interface.

3. Register Timer Interface: This is used by modules to instantiate timers. These modules derive from the EventHandler base class which has the functionality to start_timer().

4. Message Dispatcher: This is where the MAIN thread of execution runs. It has several message queues which are categorized on the message PRIORITY. In this imlementaiton, we have two message priorities (0 and 1). The Message Dispatcher runs in a loop that picks up events from the Message Queues, in order of decreasing priority and calls the appropriate "process_message()" function.

5. Timer Queue: This is the timer event queue that gets tagged with timer events registered using the Register Timer Interface. Processing of this queue involves invoking the handle_timeout() function present in the EventHandler that registered the timer in the first place.

6. TimerQueue Manager: This is the module that manages the virtual Timer Queue. It has the concept of Virtual Time and represents the virtual time heart beat of the simulator. When the simulation kernel is instantiated, an initial seed message is posted from this moduel to the Message Queue. This message is intended for the timer manager itself. In this way, it reschedules itself. The process_message funciton of the TimerQueue Manager does the following:

   - find the next expiration from the timer queue.
   - if the expiration time is beyond the STOP_SIMULATION time, stop the simulation
   - else update the current SIMULATION_TIME with the next expiration value and process_all_times().
   - the process_all_timers function invokes the call back functionality for all timers that expire at the current simulation time
   - finally the TimerQueue Manager reschedules itself by posting a message to itself in the PRIORITY 0 Message queue.

7. Derived EventHandlers : These are the software modules that use the simulation kernel. They inherit the virtual functions allowing them to "start_timer()" and "handle_timeout()" in the manner best suited to their functionality.

8. Message PRODUCING Layer : This is the best way to describe the software module that calls "post_message" to send a message to another layer. This message could well be to itself.

9. Message CONSUMING Layer : This is the best way to describe the software module whose "process_message" function is called when ever it receives a message from another layer (which could be itself !!).

6.1.4.3 Distinction between instant and delayed messages

The above flow of control does not consider whether the Message PRODUCING Layer is logically on the same "simulated entity" as the Message CONSUMING Layer. In other words, the produced message is delivered in ZERO virtual time to the consuming layer.

In order to introduce the concept of "simulated link delays", the messages that flow between PROCUDER layer on one simulated entity (a switch) to its peer CONSUMER layer on another simulate entity (another switch/host), are first encapsulated within Timer Events. The way this works is as follows:

Consider the standard situation where the AAL5 layer on one switch wants to communicate to the AAL5 layer on another switch.

- first the PRODUCER AAL5 layer encapsulates the message within a TimerEvent.
- the TimerEvent is tagger with the delay on the link between the PRODUCER and CONSUMER and then registered using the start_timer call.
- the TimerEvent is delayed by the the virtual time it takes to reach the destination, and when that timer pops, the handle_timeout function is called.
- the PRODUCER AAL5 handle_timeout function now passes the message to the destination AAL5 module.

6.1.4.4 Simulating Other Switch Processing Delays

The virtual time model needs to be accounted for in terms of delays induced due to processing within the switch. In order to do this, we use a technique similar to the AAL5 message passing between peer AAL5 modules on different switches.

Take the case of Routing Delays. Here, we have to account for the delay in getting the result of a routing request. The way this is done is as follows:

- when the router gets the request we take a time stamp of real (machine) time.
- the router processes the request.
- we take another time stamp and calculate the real routing delay.
- we also take in a user defined "routing_time" from the input script.
- if the user has defined a routing_time, we start a TimerEvent and seed it with the routing time. Else, we seed the TimerEvent by the real routing delay.
- the TimerEvent is scheduled and it pops and calls the handle_timeout function after the "routing_time" duration.
- the handle_timeout function now passes the result of the routing decision to the module that requested the route.

6.1.5 Q.Port Switch Call Control

The Switch Call Control is a module within Q.port. Every instance of a switch there is an SCC instance, which is aware of the Switch Fabric and also the Q.93b stacks running on each port of the switch. In addition the SCC interacts with the Router module (which can be a static, table lookup router, or a PNNI router module) to get information regarding the next hop for every call that comes in from one of the in-ports. All in all, the SCC is the heart of the switch instance, and controls the call activity within the switch.

The Figure 6.3 shows the different states that exist for a call within the switch along with the transitions that occur between the states when messages are posted to the SCC.

6.1.5.1 SCC Finite State Machine

The SCC has a Finite State Machine that models the call/connection within the switch. In addition, each call is associated with a call record that retains information about the call state. This is the method by which the SCC is able to control and coordinate the calls/connections.

States and their meaning:

1. **IDLE** state - represents the condition when a call connection request arrives into the SCC and is allocated a callrecord. It also represents the state that the call returns to after call clearing, and before the callrecord is destroyed.

2. **PENDING ROUTE** state - signifies that the SCC has contacted the Router module and given it called party information. There can be two router types:

   - Static Router: is contacted to parse the called party information, when the SCC finds that the called party number is reachable without having to go across a NNI interface.
   - Pnni Router: is contacted either when the SCC realises that to reach the called party number the NNI is to be used, or when the call progresses from one NNI to another.

   SCC waits for a acknowledge message from the router module in which is contained the port that the call/connection needs to be forwarded to for the next hop to reach the final called number.

3. **PENDING RESOURCE RESERVE** state - is reached by a call when the SCC sends a resource reservation request to the Fabric class for the next hop that this call must proceed to.

   The SCC waits for a acknowledge message from the fabric class containing the handle to the connection object within the fabric. This handle contains information about the VPI/VCI used for the outgoing call request and is used for future interactions for the call connection in the fabric. Now, the SCC propagates the call request, via the Q.93B stack of the port that is connected to the next hop/switch.

4. **PENDING TERM CONFIRM** state - is reached after the call has been forwarded onto the next hop/node and before that node sends a response back. The response can be either:

   - accepts the connection request by sending a call proceeding or a call connect message
   - rejects the connection request with a call release message.

5. **PENDING RESOURCE CONNECTED** state - is reached by the call after the an accept is received from the terminator. The fabric has contacted for finalizing the connection, by connecting the originating and terminating connection objects.

6. **PENDING ORIG CONFIRM** state - is specified only for switches that uses Interim Inter-Switch Signalling Protocol (IISP). We use the PNNI protocol for inter-switch signalling. So we do not discuss it here.

7. **ACTIVE** state - is reached once the call connect message arrives and the fabric acknowledges that the originating and terminating sides are connected. Now, the SVC is set up and call can proceed.

8. **CLEARING RESOURCE** state - occurs during the tear down of a connection or error cases, when the resources allocated in the fabric for this call request need to be deallocated. The connection objects of the two sides of the connection are destroyed by the fabric and acknowledgment sent to the SCC.

9. **CLEARING PEERS** state - is the intermediate state that is reached for clearing that side of the call that did NOT initiate the tear down. In other words, depending on which side initiated clearing, the opposite side is torn down and bookkeeping and deallocation is done. This arises because of the asynchronous nature of the connection states/messages.

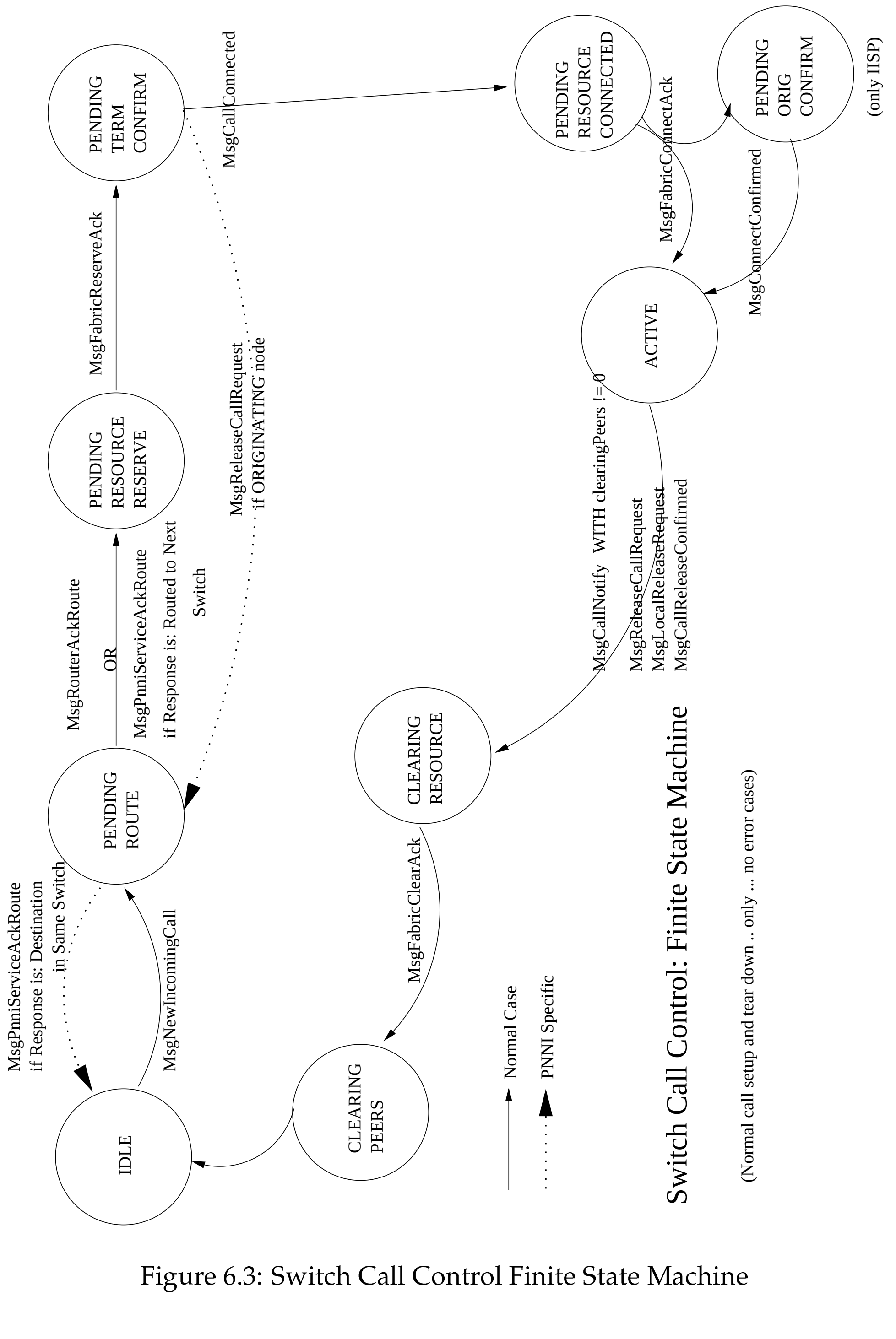

Figure 6.3: Switch Call Control Finite State Machine

Chapter 7

Crankback and Alternate Routing

When the call is failed to route to the destination, *crankback* mechanism is executed to handle the call failure. In addition, the alternate routing mechanism is executed to find an alternate route in corporation with the crankback mechanism. In this chapter, we discuss the process of crankback to handle the call failure in Section 7.1, and the alternate routing mechanism is described in Section 7.2.

7.1 Crankback Mechanism

When the call request is unable to route to the destination host, the call is *crank-backed*. When the crankback occurs at the intermediate node, the call request is released. The node sends the RELEASE or RELEASE COMPLETE message back the previous node from which the call request is received. If the crankback occurs at the destination host, the RELEASE COMPLETE message with Crankback object is sent back to the source node. If the crankback occurs at the intermediate node, the RELEASE message with Crankback object is sent back to the source node along the routed path.

The release message returns to the source node using the path that call request is previously routed. When the release message arrives at the source node, the alternate routing mechanism is performed. The alternate routing mechanism is described in Section 7.2.

A call can be rejected by two reasons. First, the reject can be because of the failure of node. The failure can be that the node is failed to support the call request on its incoming port, or the node is down. Therefore, the call is blocked. This type of blocking is called *BlockedNode* [1]. Below shows the code how to create the Crankback object to collect the failure data when the call is failed because the intermediate node blocks the call.

```
CrankbackElement * crankback =
     new CrankbackElement (
                 peer_level,                // the level of crankback
                 BlockedNode,               // blocked transit types
                 copyOfNodeID,              // the blocked node id.
                 TransitNetworkUnreachable  //cause of crankback
                        );
```

Second, the call can be rejected because the node cannot support the call request on its outgoing port. This could be that the link between this node and the next destination node is failed to support the call request. The failure can be that the link is down, or the link does not have enough resource to support the call request. Therefore, this type of blocking is called *BlockedLink* [1].

Below shows the code how to create the Crankback object to collect the failure data when the call is failed because the link is blocked.

```
CrankbackElement * crankback =
  new CrankbackElement(
               peer_level,  // the level of crankback
               BlockedLink, // blocked transit types
               precNodeId,  // the preceding Node
               blockedPort, // the blocked port of the link to succNode
               succNodeId,  // the succeeding Node
               TransitNetworkUnreachable // cause of crankback
                      );
```

7.2 Alternate Routing Mechanism

When the release arrives at the source node, the release message is processed at the Switch Call Control (SCC) module. First, SCC checks whether the call can be routed again using alternate routing mechanism or not by determining the alternate route retry number that is setup by the network operator. We describe the alternate routing mechanism in two cases. Section 7.2.1 shows the alternate routing mechanism when the call can be routed again. Section 7.2.2 describes when the call cannot be routed because the number of route retries exceeds a limit. In Section 7.2.3, the alternate routing at the intermediate node is described.

7.2.1 Source Node Does Alternate Routing

After checking the alternate route retry number, the call can be routed again if the number does not exceed the limit. The limit is set by the user using *crankback_retries* parameters, and it can be specify in the input script. See more details of how to use this parameter in the Appendix A.

After the route retry number is checked, and the source node receive the release message from the terminating side, SCC sends the *clear_req* message to Fabric module to release the call request. Then SCC goes to TRY_ALTERNATE_ROUTE state waiting for the response from the Fabric. After get the response *clear_ack* from Fabric, SCC prepares the *route_req* in order to request an alternate route from the PNNI router module. Then SCC goes to PENDING_ROUTE state waiting for the response from the PNNI route module. The response from the PNNI router can be categorized into two cases for alternate routing.

In the first case, the PNNI router is able to find an alternate route for the call. Therefore, the PNNI router sends the *route_ack* message with the *AlternateRoute* response to SCC as shown in Figure 7.1. Then the SCC sends the *reserve_ack* message to Fabric module to reserve the resource for this call and to request VPI/VCI for the alternate route. Then SCC goes to PENDINGRESOURCERESERVATION state waiting for the confirmation from Fabric.

If the reserve is successful, Fabric will send the *reserve_ack* message back to SCC as shown in Figure 7.1. And then SCC increases the number of alternate route retry and sends the *setup_req* message to the terminating side to try the alternate route again, and it goes to PENDINGTERMCONFIRMATION state. However, SCC does not send the *callproc_req* to the originating side because this is an alternate routing for the call. The *callproc_req* message is already sent at the first time of the call setup before the call is failed.

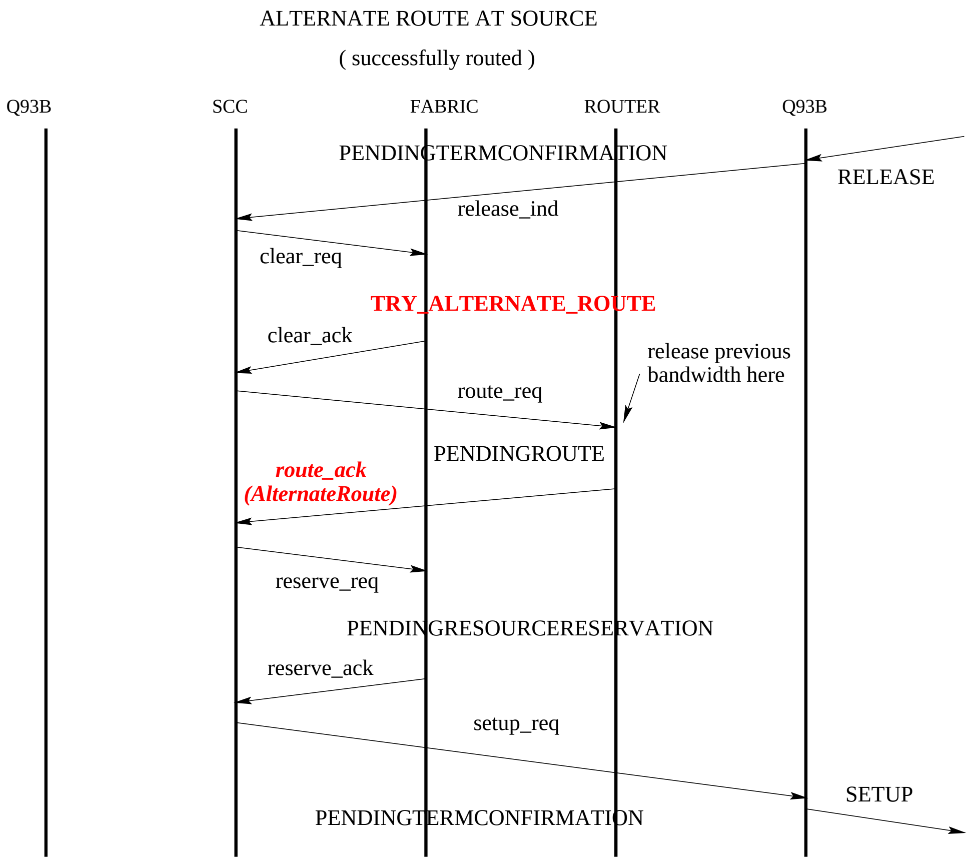

Figure 7.1: The SCC event trace when the alternate routing is successful at source node

Otherwise, the fabric reserve is failed, the Fabric will send the *route_nak* to SCC as shown in Figure 7.2. Then the call will be released as regular release procedure. The SCC sends the *release_req* message to the originating side and the *release_resp* message to the terminating side and goes to CLEARINGPEERS state. After the SCC receives the *release_conf* message, it sends out the *release_resp* message and goes to IDLE state.

In the second case, when SCC receives the failed call, it tries to find an alternate route by sending the new route request to the PNNI router module. However, there can be no route available for the alternate routing. Therefore, the PNNI router module will send the *route_ack* message with the *SourceCrankback* response as shown in Figure 7.3. Then SCC needs to release this call. It sends the *release_req* message to the originating side and the *release_resp* message to the terminating side and goes to CLEARINGPEERS state. After the SCC receives the *release_conf* message, it sends out the *release_resp* message and goes to IDLE state.

7.2.2 Source Node Rejects the Failed Call

In this case, the call is rejected because the number of route retries exceeds the limit. We do not want to call to keep trying forever so we need to limit the number of the route retries. This number can be set by the network operator and specified in the simulation input script.

The call reject procedure is shown in Figure 7.4. After the SCC receives the *clear_ack* message from Fabric module, the SCC checks the route retry number. If exceeds, then the call have to be released. It sends the *release_req* message to the originating side and the *release_resp* message to

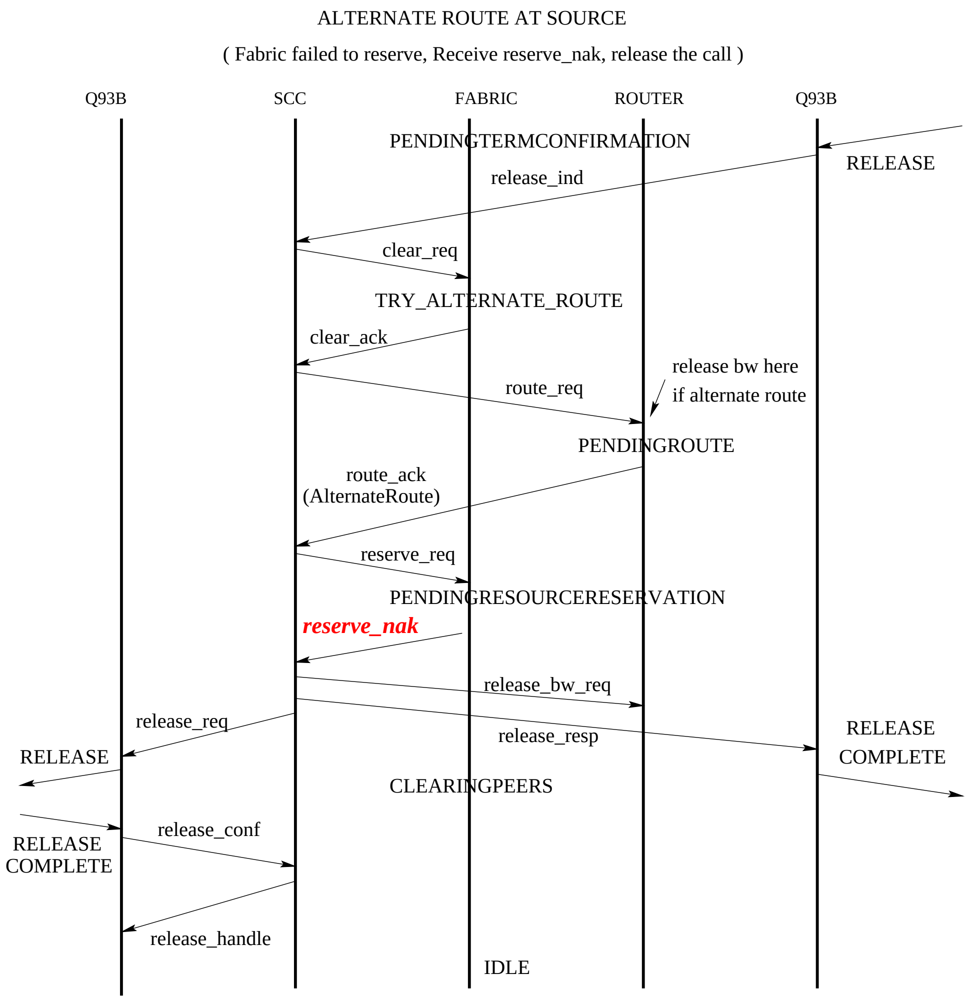

Figure 7.2: The SCC event trace when fabric reserve fails at source node

the terminating side and goes to CLEARINGPEERS state. After the SCC receives the *release_conf* message, it sends out the *release_resp* message and goes to IDLE state.

7.2.3 Intermediate Node Actions

Figure 7.5 shows the procedure of releasing a call and setting up an alternate call if available. This procedure is the same as the regular procedure of releasing and setting up a call.

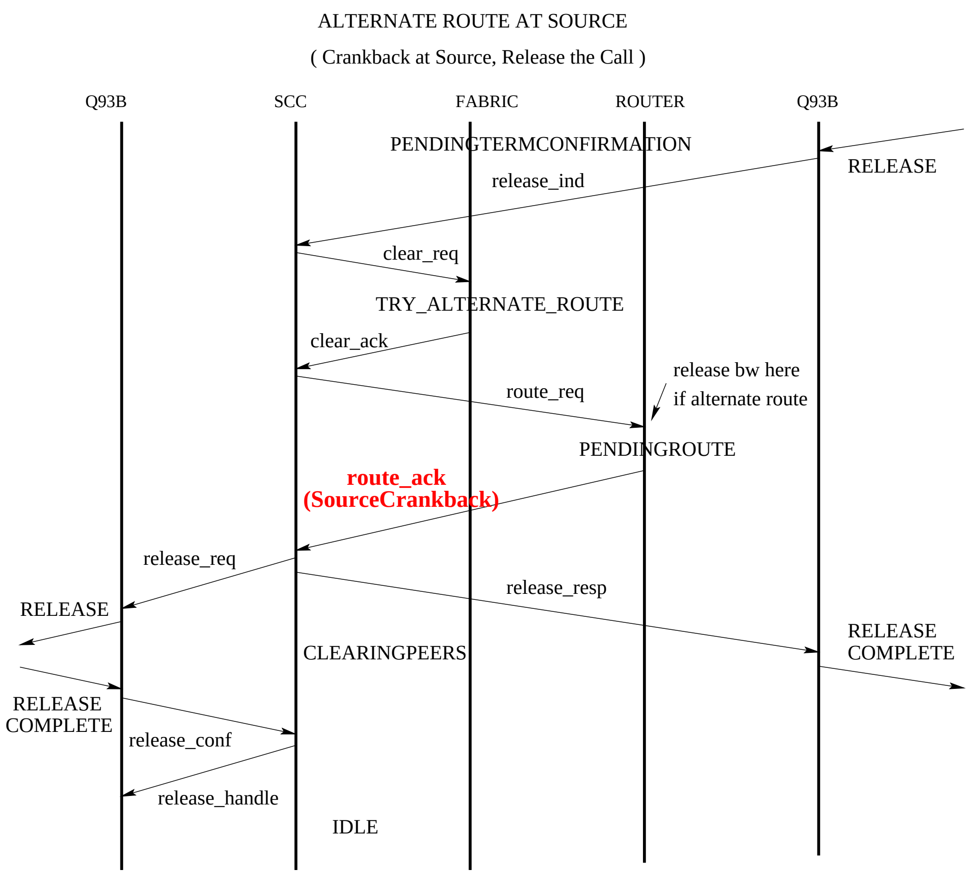

Figure 7.3: The SCC Event Trace when Crankback happens at Source Node

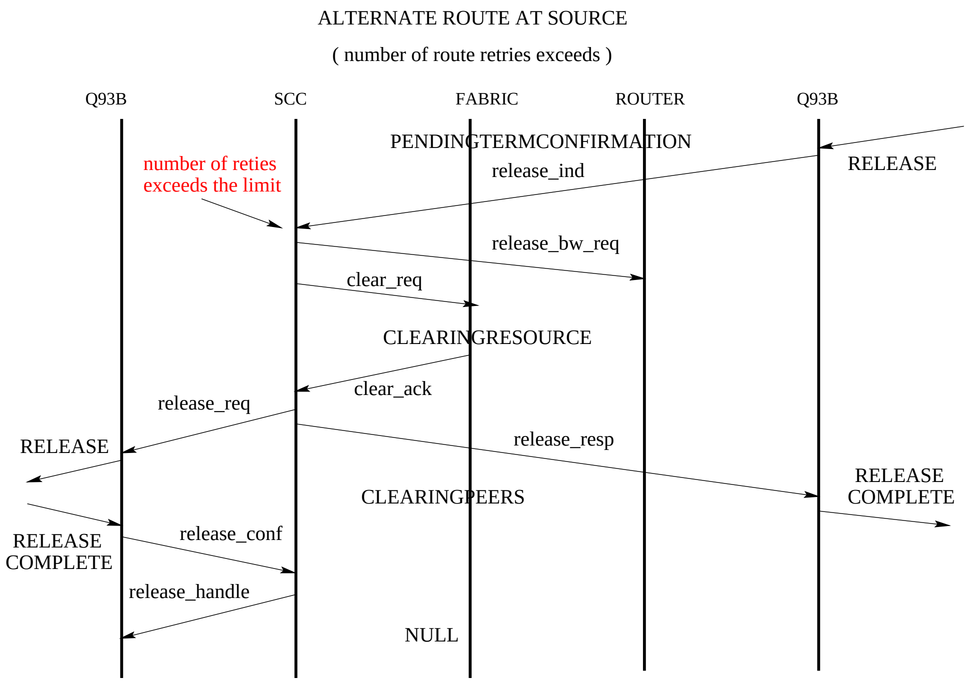

Figure 7.4: The SCC event trace when the route-retry number exceeds the limit at source node

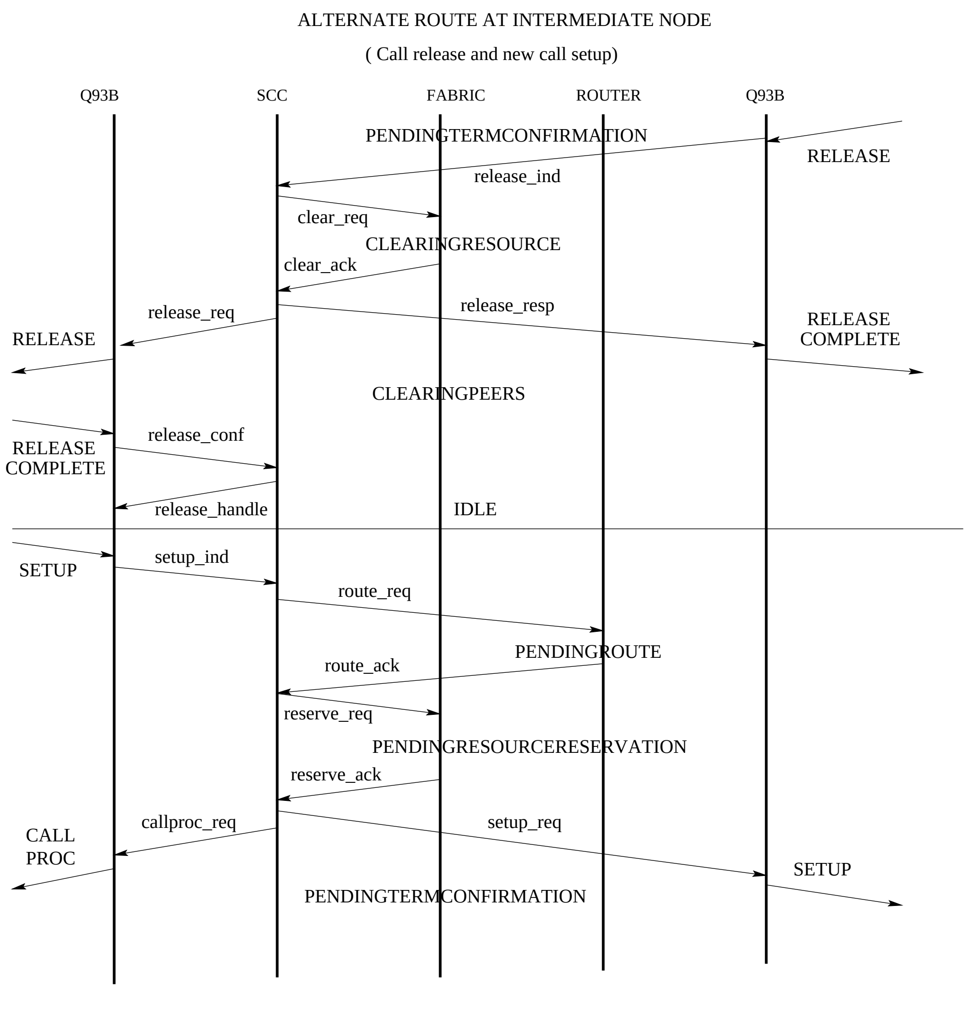

Figure 7.5: The SCC event trace shows the call failure and new (alternate) call setup at an intermediate node

Chapter 8

CallGenerator

The call generator enables the hosts involved in the experimentation to generate calls. The total number of calls to be generated has to be specified. However, the calls can be also generated without any upper limit on the number of calls by providing the value for the host component parameter *calls* as *unspecified*. This would enable call generation for the duration of the experiment.

The following call arrival distributions are supported:

- Poisson Distribution
- Uniform Distribution
- Periodic Distribution

The following additional call generation types are supported:

- Tear Down: In this type when a call connection is established, it is immediately torn down and the the next call is attempted.
- Bursty: In this type, a new call connection is attempted, whenever a previous call gets established.

The following call duration distributions are supported:

- Poisson distribution
- Uniform distribution
- Periodic distribution

The following call types are supported:

- CBR: Constant Bit Rate
- ABR: Available Bit Rate
- RTVBR: Real Time Variable Bit Rate
- NRTVBR: Non Real Time Variable bit rate
- UBR: Unspecified bit rate

The following QoS parameters for connection request are supported:

- PCR: Peak Cell Rate
- SCR: Sustainable Cell Rate
- CTD: Cell Transfer Delay
- CDV: Cell Delay Variation
- CLR: Cell Loss Ratio

The call connections can be made to explicit destinations be defining the probabilities for each destination or by giving an option *uniform_any* which makes calls to all hosts defined in the topology with uniform probability.

The call generator also supports multiple traffic sources with different QoS requirements as part of the total number of the calls generated. QoS requirements are specified for individual sources as shown in Appendix A, section A.4. The calls generated from each traffic can also be specified as a percentage share of the total number of calls generated. When multiple traffic sources are available, the calls are attempted with a common arrival distribution, and with individual duration distributions.

Chapter 9

User Interface

9.1 Components of the PNNI ATM Network

The user interface language is designed based on the following different components that could be present in an ATM network.

- *Node:* A node is an ATM switch which has ability to admit call connection requests and provide PNNI dynamic routing.
- *Host:* A host is an ATM end system that provides call generation capability using a call generator. Hosts connect to nodes.
- *Links:* Links provide connectivity between any two of the three components of an ATM network mentioned above.
- *Load:* Load defines the traffic characteristics of the host.
- *Logical Node:* This provides the information about the logical nodes at the higher levels in the PNNI hierarchy
- *Logical Connections:* The connections between logical nodes are provided as Logical Connections

9.2 The User Interface

The simulator is provided with a common user input script interface. Using this script interface, any required network topology can be specified. The design for this utility is as shown in figure 9.1

When the user specifies the network, a parser is used to parse the input and store the input data into a data structure. There may be some default parameters left out by the user and these are filled in by the `complete_user_ input()` function. When the input configuration is complete, the data structure is given to a simulation boot up program, which sets up the simulation.

The user interface for this tool is a simple language whose grammar and details are chronicled in Appendices A and B. The user specifies an experiment in a text file like the sample input script in program 9.1 that is parsed and processed by the simulation tool.

The following simple user input file is given to help the reader follow the generic explanation in this section. More detailed descriptions will be specified in later sections.

Program 9.1 A Sample User Input Script

```
# Parameter blocks
parameter_block node spark {
        prop_constant           =       50,
        flooding_threshold      =       25,
        calltrace               =       true,
        routing_policy          =       max_bw,
        numports                =       14,
        process_time            =       1.0,
        queuesize               =       5000,
        util_log_period         =       0
};

parameter_block host newton{
        duration                =       100,
        calltype                =       cbr,
        bw                      =       100,
        calls                   =       0,
        arrival_period          =       5,
        duration_period         =       10,
        queuesize               =       5000,
        host_process_time       =       3.0
        };

# Node definitions
node Amy{
        parameter_block spark,
        address = 0x4705ffef560000000000000011000000000000000
        };

node Joe{
        parameter_block spark,
        address = 0x4705ffef560000000000000022000000000000000
        };

node Fred{
        parameter_block spark,
        address = 0x4705ffef560000000000000033000000000000000
        };

node Pam{
        parameter_block spark,
        address = 0x4705ffef560000000000000044000000000000000
        };

host Ken{
        parameter_block newton,
        address = 0x4705ffef56000000000000001100ec301100aa00
        };
```

```
host Bess{
        parameter_block newton,
        address = 0x4705ffef56000000000000004400ea120022bb00
        };

# Load definition
load Bess{
        calls = unspecified,
        arrival_distribution    = periodic,
        arrival_period          = 10,
        numdestinations = 1,
        destinations = [Ken],
        destn_prob = [1.0]
        };

port genericport {bw=OC12, delay=10};

# Node - Node connections
connection Amy->Joe{bw=300, ad_weight = 80};
connection Joe->Fred{bw=310, ad_weight = 90};
connection Fred->Pam{bw=320};

# Node - Host connections
connection Ken->Amy{bw=300, ad_weight = 80};
connection Pam->Bess{bw=300, ad_weight = 80};

# Schedule of the experiment
schedule{
        duration        =   550
        };
```

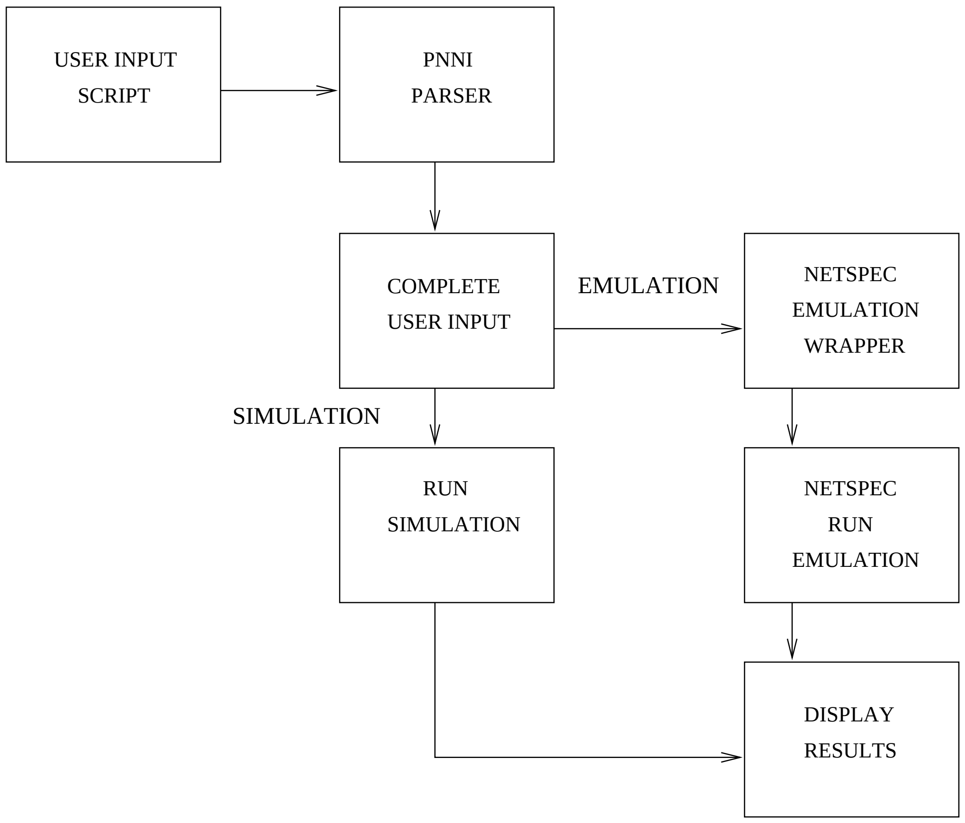

Figure 9.1: Design of a User Interface to the Simulator

9.2.1 Components of User Input

The following components constitute the user input.

- *Parameter blocks:* This is generic information about various node parameters or host parameters that can be used as building blocks to build nodes or hosts in the experiment. Programmed default values will be used for any unspecified values in the script.
- *Individual Node Information:* This specifies the individual node information in addition to, or different from, the generic node information specified previously using the parameter blocks. Any parameter specified here will override the corresponding generic parameter specified above for this node only. A node definition should contain the name and address as mandatory parameters.
- *Generic Port Information:* This specifies the generic connection information shared by all links. This includes bandwidth, administrative weight, link delay for the link associated with the port and any other port-related parameters.
- *Individual Host Information:* This specifies the host information in addition to, or different from, the generic host information specified previously. Any parameter specified here will override the corresponding generic parameter specified above for this host only.
- *Connectivity Information:* This specifies the link between the network entities host and node.

The connectivity information optionally specifies the parameters related to the link between the two sides. Any parameter specified with the connectivity information will override the value for corresponding parameter specified under the generic port information.

- *Load Characteristics:* This specifies the traffic characteristcis of a host, the information about the calls to be generated.
- *Logical Nodes:* This specifies the level of the logical node and the node in the immediate lower level in the hierarchy which the LGN represents.
- *Logical Connections:* This specifies the properties of the logical connections between the logical nodes.
- *Schedule of events:* This specifies the duration of the experiment and the events we'd like to simulate in the experiment namely node failure or link failure. This also includes the random seed which is used for the random number generator in the simulator and the simple or complex representation to be used for Nodal Representation in Hierarchical PNNI.

9.2.2 User Input Restrictions

Required Parameters: The following parameters *must* be specified in a user script.

- Individual node Information
- Individual node to node connectivity information
- Logical node information for hierarchical PNNI
- Logical connectivity information for hierarchical PNNI
- The schedule of the experiment with the duration parameter as mandatory

Optional Parameters: The following components are *optionally* specified in a user script.

- Parameter blocks for defining the general node or host parameters
- The generic port information
- Individual host information
- Individual load information for a host
- Individual host to node connectivity information

Parameter Order: The parser allows parameters to come in any order in the script to make it easier for the user. However, the following restrictions apply (*Note: These restrictions are not reflected in the BNF grammar in Appendix B*).

- The required order of components, if included, should be: definition of nodes, hosts, ports, connections in any order but followed by a schedule for the experiment.
- In a simulation, do not include hostpool information
- The paramater block information for node must precede any individual node information

- The paramater block information for host must precede any individual host information
- The generic port information must precede any individual connection information
- The load information for any host should be given only after the definition of the host itself.
- In host information, (both generic and individual) you can specify multiple sources. In doing so, you *must* specify the number of sources before giving any source-specific information. This is necessary because this information is used to allocate the required memory to store the source-specific information.
- The logical nodes and logical connections should come after the physical nodes and connections and in the decreasing order of their levels.

Unimplemented Features and Limitations: The following features have the noted limitations or are as of yet unimplemented.

- ATM PVC and SVC (unimplemented)

Chapter 10

Nomenclature, Keywords and Syntax

This section describes the keywords and syntax to further help the user get familiar with the interface. Some of the keywords which are used but unexplained in the descriptions should be ignored until they are explained later.

10.1 Naming Conventions

The names of nodes and hosts can be arbitarary strings, we have moved away from conventions C1, H1 etc. Also we can have multiple hosts connected to a node. According to specifications, the nodes within a peer group should have the same prefix and all hosts connected to a switch share the same prefix with only their hardware addresses different. The definition of a node or a host should contain the node name and the address as mandatory parameters.

10.2 Statement

A statement assigns the component specific details to a component group. The format of a statement and several examples are given below. The white spacing between the words of a statement is ignored and each statement is ended by a semicolon.

component group component (parameters);

Examples:

```
1. physicalhosts hostpool (host1, host2, ... hostN);
2. parameter_block node spark (parameter1, parameter2, ... parameterN);
3. parameter_block host newton (parameter1, parameter2, ... parameterN);
4. port genericPortInfo (parameter1, parameter2, ... parameterN);
5. node node1(parameter1, parameter2, ... parameterN);
6. host host1(parameter1, parameter2, ... parameterN);
7. connection node1->host1 (parameter1, parameter2, ... parameterN);
8. load host1(parameter1, parameter2, ... parameterN);
9. logicalnode A.1 (parameter1, parameter2, ... parameterN);
10. logicalconnection A.1->A.2{ parameter1, parameter2 };
```

10.3 Component Group

This specifies the group of components to which this statement belongs. The component groups specified by the following keywords.

- *node:* The node group includes the parameter block information and individual node information. These parameter blocks represent a set of parameters that nodes have in common, so that this common information does not need to given in the definition of every node, instead this block can be used as the base building block. Examples for each are shown below:

  Parameter block for node

  ```
  parameter_block node spark {
          prop_constant           =       50,
          flooding_threshold      =       25,
          calltrace               =       true,
          routing_policy          =       max_bw,
          numports                =       14,
          process_time            =       1.0,
          queuesize               =       5000,
          util_log_period         =       0
  };
  ```

  Individual node information

  ```
  node node1{
          parameter_block spark,
          address = 0x4705ffef5600000000000000011000000000000000
  };
  ```

- *port:* The port group includes the generic port information. An example is shown below:

  Generic port information

  ```
     port = genericPortInfo (bw=OC12, ad_weight=3);
  ```

- *host:* The host group includes the parameter block information and individual host information. Examples for each are shown below.

  Parameter block for host

  ```
  parameter_block host newton{
          duration                =       100,
          calltype                =       cbr,
          bw                      =       100,
          calls                   =       0,
          arrival_period          =       5,
          duration_period         =       10,
          queuesize               =       5000,
          host_process_time       =       3.0
          };
  ```

Individual host information

```
host host1{
        parameter_block newton,
        address = 0x4705ffef56000000000000001100ec301100aa00
        };
```

- *connection:* The connection group includes connectivity information which could be either a connection between two nodes or a connection between a host and node. For the topology, all node-node connections and host-node connections have to be given in the script.Examples of each are shown below.

  Connection between two nodes

  ```
     connection = node1->node2(bw=50, delay=5);
  ```

  Connection between a host and an node

  ```
     connection = node1->host1(delay=50);
  ```

- *Load characteristics:* The traffic characteristics for each host can be defined as follows.

  load information

  ```
  load host2{
          calls                       = unspecified,
          arrival_distribution        = periodic,
          arrival_period              = 10,
          numdestinations             = 1,
          destinations                = [host1],
          destn_prob                  = [1.0]
          };
  ```

- *logicalnode* The logical node definition consists of the child peer group leader that the Logical Group Node represents and the level of the logical node, a sample definition is provided here

  ```
  logicalnode A.1{
          level = 88,
          child = A.1.2
          };
  ```

- logicalconnection The Logical connection consists of the delay of the logical link and the number of connections that the logical link represents.

  ```
  logicalconnection A.1->A.2{ delay = 25 };
  ```

- *schedule of events:* The duration of the experiment, the events like node and link failures can be given here. Also, the random seed and the nodal representation to be used for aggregation are provided here, the 'mpg' flag has to be defined here for running Multiple Peer Group experiments. *schedule*

```
schedule{
    duration    =   550,
    seed        =   1234,
    nodal_represent = complex,
    mpg = true,
    nodefail node1
    };
```

10.4 Additional Syntax

Some additional syntax information associated with the language is given in the table 10.1 below.

syntax	Explanation
{	begins component parameters
}	ends component parameters
,	separates parameters
=	assigns values to parameters or component groups
;	concludes a component statement
->	links two nodes or a host and an node together

Table 10.1: Syntax table

Chapter 11

Examples

11.1 A Ring Topology Configuration

11.1.1 Simulation

Figure 11.1 depicts a eight node ring topology network. In the figure, `Denver` to `LosAngeles` are nodes, and `Host4` to `Host7` are the hosts connected to the nodes. The user script to obtain the above topology is shown in program 11.1. Some of the important parameters in `parameter_block node` are explained here. For details please refer to Appendix A.

- *prop_constant:* This specifies the value for the PNNI proportional multiplier which determines the significant change in link topology information.
- *flooding_threshold:* This determines the minimum threshold of bandwidth for flooding link topology information.
- *process_time:* This is the average time in milliseconds in which an event is processed in the node.
- *duration:* This specifies the duration of the simulations in seconds.

The parameter_block node is named as `spark` and all nodes inherit the properties of this block. The individual properties of the node must be explicitly specified in addition to these parameters.

The generic information related to the hosts are provided with the `parameter_block host` identifier. The explanations for some of the important parameters are provided below.

- *calls:* This specifies the number of calls attempted (15).
- *arrival_period:* This indicates the inter-call-arrival duration (5 seconds).
- *bw:* This specifies the bandwidth requested per call (10 KB), the bandwidth could have a distribution like fixed, uniform or poisson.
- *duration_period:* This indicates the call duration (10 seconds).

Note that the parameters specified with the `parameter_block host` keyword are generic to all of the hosts in the simulation, i.e., to hosts `Host4` to `Host7` in this topology. To explain explicitly, Host4 makes 15 calls to Host6 and Host5 generates 15 calls to Host7 with an inter-arrival call period of 5 seconds and call duration of 10 seconds. Calls could be made to multiple destinations also.

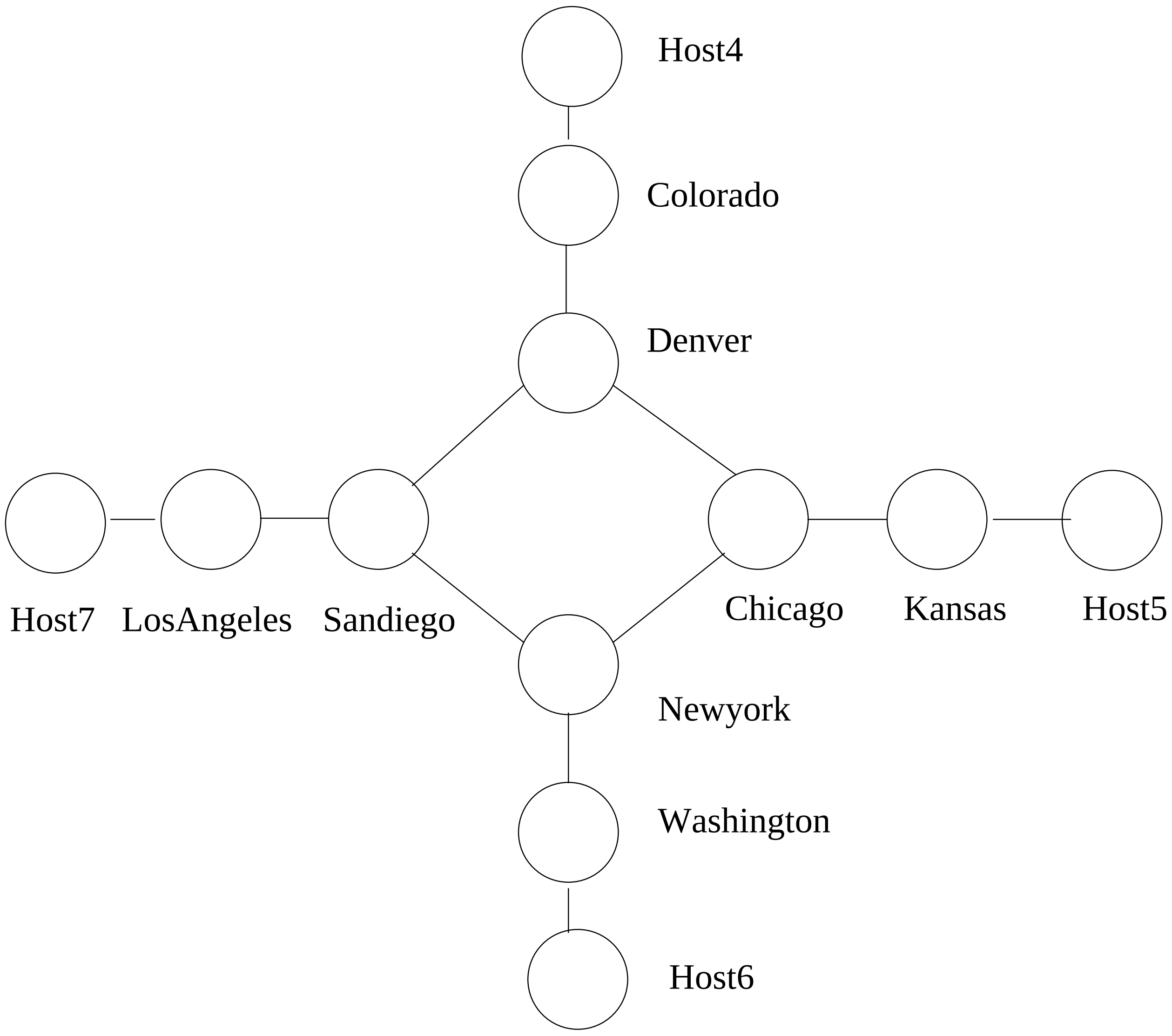

Figure 11.1: An Eight Node Ring topology

The `genericport` information specifies the generic connectivity parameters for links between ports of PNNI network entities.

The ring topology is created by connecting the nodes using the `connection` component group identifier as specified in `connection = Denver->Colorado` lines. Note that since all these connections are specified with no optional parameters, they use the parameters specified with the `genericport` identifier. This means that the connections have a link bandwidth of 622 Mbps (OC12 link rate) and a link delay of 10 milliseconds.

The results produced by the ring topology simulation are shown in section 11.1.1.1. The sequence of the results is explained below. Please relate it to the output shown.

- *Individual Host Call Records:* Here for each host, a log is generated which shows the statistics including the call type (ex: cbr), bandwidth requested (ex: 808.992 kbps), call start time, call connect time, the call setup time, results of the call attempt (ex: success), and cause of failure if any. These logs also show the percentage call success, the bandwidth rejection (the sum of

Program 11.1 An Eight Node Ring Topology Simulation Script

```
# simple ring topology

parameter_block node spark {
        prop_constant           =       25,
        flooding_threshold      =       5,
        calltrace               =       true,
        routing_policy          =       max_bw,
        numports                =       14,
        process_time            =       1.0,
        util_log_period         =       10,
        queuesize               =       5000
};

parameter_block host newton{
        duration                =       100,
        calltype                =       cbr,
        bw                      =       10,
        arrival_period          =       5,
        duration_period         =       10,
        queuesize               =       5000,
        host_process_time       =       3.0
        };

node Denver{
        parameter_block spark,
        address = 0x4705ffef56000000000000001100000000000000
        };

node Chicago{
        parameter_block spark,
        address = 0x4705ffef56000000000000002200000000000000
};

node NewYork{
        parameter_block spark,
        address = 0x4705ffef56000000000000003300000000000000
        };

node SanDiego{
        parameter_block spark,
        address = 0x4705ffef56000000000000004400000000000000
        };

node Colorado{
        parameter_block spark,
        address = 0x4705ffef56000000000000005500000000000000
        };
```

```
node Kansas{
        parameter_block spark,
        address = 0x4705ffef56000000000000006600000000000000
        };

node Washington{
        parameter_block spark,
        address = 0x4705ffef56000000000000007700000000000000
        };

node LosAngeles{
        parameter_block spark,
        address = 0x4705ffef56000000000000008800000000000000
        };

host Host4{
        parameter_block newton,
        address = 0x4705ffef560000000000000005500ec3011001100
        };
host Host5{
        parameter_block newton,
        address = 0x4705ffef560000000000000006600ec3011002200
        };
host Host6{
        parameter_block newton,
        address = 0x4705ffef560000000000000007700ec3011003300
        };
host Host7{
        parameter_block newton,
        address = 0x4705ffef560000000000000008800ec3011004400
        };

load  Host4{
        calls = 15,
        numdestinations = 1,
        destinations = [Host6],
        destn_prob = [1.0]
        };

load  Host5{
        calls = 15,
        numdestinations = 1,
        destinations = [Host7],
        destn_prob = [1.0]
        };
```

```
port genericport {bw=OC12, delay=10};


connection Denver->Colorado { bw = OC12 };
connection Denver->Chicago { bw = OC12 };

connection Chicago->Kansas { bw = OC12 };
connection Chicago->NewYork { bw = OC12 };

connection NewYork->Washington { bw = OC12 };
connection NewYork->SanDiego { bw = OC12 };

connection SanDiego->LosAngeles { bw = OC12 };
connection SanDiego->Denver { bw = OC12 };

connection Host4->Colorado { bw = OC12 };
connection Host5->Kansas { bw = OC12 };
connection Host6->Washington { bw = OC12 };
connection Host7->LosAngeles { bw = OC12 };

schedule {
duration = 150
};
```

the bandwidth requested by the rejected calls) and the average call setup time.

- *Average Host Call Records:* Here the average results statistics for calls attempted from all the hosts in the experiment are shown.
- *Individual Node Records:* Here the individual node related logs are shown. They are:
  - *convergence time:* This indicates the time when a node obtains all the initial topology information generated by the other nodes in the topology.
  - *total floods:* This specifies the total number of PNNI topology information messages generated by a node.
  - *total wasted floods:* This specifies the total number of redundant topology information received at a node. The topology messages are considered redundant when the topology information which they carry is already present in a node and hence ignored.
  - *avg hops:* The average number of links traversed by a connection request for a successful call.
  - *source failed calls:* The calls which failed at the source node due to non availability of a route which satisfies the call bandwidth requirements.
  - *intermediate hop failed calls:* The calls which failed at an intermediate node due to non-availability of bandwidth at the next hop.
  - *average routing time:* The average *clock time* taken to compute a route using a routing algorithm.
  - *pnni data sent:* The PNNI total topology data bytes sent from a node.
  - *utilization logs:* This shows the link utilization at each of the links.
- *Average Node Records:* This specifies the average node results statistics calculated over all of the nodes in the experiment.

11.1.1.1 Simulation Results

- First we present the standard output that is printed when the simulator is run.

```
% ./kupnni

Usage: kupnni [OPTIONS]... scriptfile

KU PNNI Simulator, University of Kansas

OPTIONS:
   -l, --log            output an event-level log into individual files
   -q, --quiet          suppress printing of event-level log onto the screer
   -d, --debug=[LEVEL]  debug information
   -c, --copyright      copyright information
   -v, --version        version information
   -h, --help           print this message and exit

LEVEL controls the nature of debugging information
```

```
values: [none | brief | intermediate | full]

By default, debug level is none, results are printed to stdout,
and no log files are produced.

Report bugs to pnni@ittc.ukans.edu
```

- Next, if the script presented above were in the file `ring.script`, the output of the simulator for the following command line would be:

```
% ./kupnni ring.script


 ---- W E L C O M E   T O   K U   P N N I   S I M U L A T O R ----


     Information and Telecommunication Technology Center (ITTC)


        University of Kansas Center for Research, Inc.


                        Copyright (C) 1998



       by the University of Kansas Center for Research, Inc.


This software was developed by the Information and Telecommunication
Technology Center (ITTC) at the University of Kansas. ITTC does not
accept liability whatsoever for this product. Please see the detailed
COPYRIGHT notice within the distribution. This version is for Sprint
internal research use only. It is NOT to be redistributed.

For enquires, please contact:
            Dr. Douglas Niehaus <niehaus@ittc.ukans.edu>
                            or
                KU-PNNI Group <pnni@ittc.ukans.edu>
               -------------------------------------

The Q.port(TM) signaling software, herein known as the "Software", is
the copyrighted work of Bellcore. The Software has been modified by
the University of Kansas Center for Research, Inc. Information and
Telecommunication Technology Center for use with the Virtual ATM
(VATM) switch for ATM on Linux which is copyrighted work of University
of Kansas Center for Research, Inc. Information and Telecommunication
```

```
Technology Center (ITTC). The modified Software is made available for
downloading solely for use by end users. Reverse engineering of the
modified Software or any other type of tampering and/or infringement
is expressly prohibited by law, and may result in severe civil and
criminal penalties.

               Q.port is a trademark of Bellcore.

               ----------------------------------

SimKernel: Simulation ended in sim time 000150.000000
Duration of simulation is 3 seconds
Type CTRL-C to end the process. Thanks.
Avg  PROCESSING TIME 000000.000259
Over Total events 9829

HostRecordFile.output contains Host information
SwitchRecordFile.output contains Switch information
LinkRecordFile.output contains Link information
DetailedLinkRecordFile.output contains Detailed Link information
CalltraceRecordFile.output contains Calltraceinformation information
NetworkRecordFile.output contains Network information

***** CALL SETUP LOGS START ******

-- Host4 host record begins ------------------------------

No. calltype bw(kbps) starttime   stoptime    setuptime   result  cause
  1 cbr    26.9664 00:00:10.000  00:00:10.250  000.250000  setup
  2 cbr    26.9664 00:00:15.000  00:00:15.136  000.136000  setup
  3 cbr    26.9664 00:00:20.000  00:00:20.139  000.139000  setup
  4 cbr    26.9664 00:00:25.000  00:00:25.136  000.136000  setup
  5 cbr    26.9664 00:00:30.000  00:00:30.139  000.139000  setup
  6 cbr    26.9664 00:00:35.000  00:00:35.136  000.136000  setup
  7 cbr    26.9664 00:00:40.000  00:00:40.139  000.139000  setup
  8 cbr    26.9664 00:00:45.000  00:00:45.136  000.136000  setup
  9 cbr    26.9664 00:00:50.000  00:00:50.139  000.139000  setup
 10 cbr    26.9664 00:00:55.000  00:00:55.136  000.136000  setup
 11 cbr    26.9664 00:01:00.000  00:01:00.139  000.139000  setup
 12 cbr    26.9664 00:01:05.000  00:01:05.136  000.136000  setup
 13 cbr    26.9664 00:01:10.000  00:01:10.139  000.139000  setup
 14 cbr    26.9664 00:01:15.000  00:01:15.136  000.136000  setup
 15 cbr    26.9664 00:01:20.000  00:01:20.139  000.139000  setup

total cbr calls            : 15
% successfull cbr calls    : 100
total cbr bw request (MB)  : 0.404496
cbr bw rejected (MB)       : 0
```

```
mean callsetup time         : 000.144999

-- Host4 host record ends ----------------------------

-- Host5 host record begins ------------------------------

No. calltype bw(kbps) starttime   stoptime    setuptime   result  cause
  1 cbr    26.9664 00:00:10.000  00:00:10.272  000.272000  setup
  2 cbr    26.9664 00:00:15.000  00:00:15.136  000.136000  setup
  3 cbr    26.9664 00:00:20.000  00:00:20.139  000.139000  setup
  4 cbr    26.9664 00:00:25.000  00:00:25.136  000.136000  setup
  5 cbr    26.9664 00:00:30.000  00:00:30.139  000.139000  setup
  6 cbr    26.9664 00:00:35.000  00:00:35.136  000.136000  setup
  7 cbr    26.9664 00:00:40.000  00:00:40.139  000.139000  setup
  8 cbr    26.9664 00:00:45.000  00:00:45.136  000.136000  setup
  9 cbr    26.9664 00:00:50.000  00:00:50.139  000.139000  setup
 10 cbr    26.9664 00:00:55.000  00:00:55.136  000.136000  setup
 11 cbr    26.9664 00:01:00.000  00:01:00.139  000.139000  setup
 12 cbr    26.9664 00:01:05.000  00:01:05.136  000.136000  setup
 13 cbr    26.9664 00:01:10.000  00:01:10.139  000.139000  setup
 14 cbr    26.9664 00:01:15.000  00:01:15.136  000.136000  setup
 15 cbr    26.9664 00:01:20.000  00:01:20.139  000.139000  setup

total cbr calls             : 15
% successfull cbr calls     : 100
total cbr bw request (MB)   : 0.404496
cbr bw rejected (MB)        : 0
mean callsetup time         : 000.146466

-- Host5 host record ends ----------------------------

AVG RESULTS OF ALL CALLS

total cbr calls             : 30
% successfull cbr calls     : 100
total cbr bw request(MB)    : 0.808992
cbr bw rejected (MB)        : 0
mean callsetup time         : 000.145733

***** CALL SETUP LOGS END **********

##### NODE INSTRUMENTATION LOGS START #######

-- Denver node record begins ------------------------------

convergence time              : 000000.090000
total floods                  : 184
total wasted floods           : 54
```

```
avg hops                      : 0
source failed calls           : 0
calls routed successfully     : 15
Intermediate hop failed calls : 0
calls confirmed from dest     : 0
CRANKBACKS                    : 0
avg routing time              : 000000.000000
pnni data sent(kbytes)        : 24.664

---utilization logs start ----

Total core bandwidth  : 1800

time            used_bw(mbps)
Denver->Chicago TotalBw 1200

000010.000000  0             0
000020.000000  0.1272        0.000106
000030.000000  0.1272        0.000106
000040.000000  0.1272        0.000106
000050.000000  0.1272        0.000106
000060.000000  0.1272        0.000106
000070.000000  0.1272        0.000106
000080.000000  0.1272        0.000106
000090.000000  0.0636        5.3e-05
000100.000000  0             0
000110.000000  0             0
000120.000000  0             0
000130.000000  0             0
000140.000000  0             0
000150.000000  0             0

Denver->Chicago avg utilization : 9.9375e-05
Denver->SanDiego TotalBw 1800

000010.000000  0             0
000020.000000  0             0
000030.000000  0             0
000040.000000  0             0
000050.000000  0             0
000060.000000  0             0
000070.000000  0             0
000080.000000  0             0
000090.000000  0             0
000100.000000  0             0
000110.000000  0             0
000120.000000  0             0
000130.000000  0             0
```

```
000140.000000  0              0
000150.000000  0              0
Denver->Colorado TotalBw 600

000010.000000  0              0
000020.000000  0.1272         0.000212
000030.000000  0.1272         0.000212
000040.000000  0.1272         0.000212
000050.000000  0.1272         0.000212
000060.000000  0.1272         0.000212
000070.000000  0.1272         0.000212
000080.000000  0.1272         0.000212
000090.000000  0.0636         0.000106
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

Denver->Colorado avg utilization : 0.00019875

avg utilization  : 0.0001325
----utilization logs end----

-- Denver node record ends --------------------------------

-- Chicago node record begins ------------------------------

convergence time              : 000000.091000
total floods                  : 186
total wasted floods           : 55
avg hops                      : 0
source failed calls           : 0
calls routed successfully     : 30
Intermediate hop failed calls : 0
calls confirmed from dest     : 0
CRANKBACKS                    : 0
avg routing time              : 000000.000000
pnni data sent(kbytes)        : 25.376

---utilization logs start ----

Total core bandwidth  : 1800

time            used_bw(mbps)
Chicago->Denver TotalBw 600
```

```
000010.000000  0              0
000020.000000  0.1272         0.000212
000030.000000  0.1272         0.000212
000040.000000  0.1272         0.000212
000050.000000  0.1272         0.000212
000060.000000  0.1272         0.000212
000070.000000  0.1272         0.000212
000080.000000  0.1272         0.000212
000090.000000  0.0636         0.000106
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

Chicago->Denver avg utilization : 0.00019875
Chicago->NewYork TotalBw 1800

000010.000000  0              0
000020.000000  0.2544         0.000141333
000030.000000  0.2544         0.000141333
000040.000000  0.2544         0.000141333
000050.000000  0.2544         0.000141333
000060.000000  0.2544         0.000141333
000070.000000  0.2544         0.000141333
000080.000000  0.2544         0.000141333
000090.000000  0.1272         7.06667e-05
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

Chicago->NewYork avg utilization : 0.0001325
Chicago->Kansas TotalBw 1200

000010.000000  0              0
000020.000000  0.1272         0.000106
000030.000000  0.1272         0.000106
000040.000000  0.1272         0.000106
000050.000000  0.1272         0.000106
000060.000000  0.1272         0.000106
000070.000000  0.1272         0.000106
000080.000000  0.1272         0.000106
000090.000000  0.0636         5.3e-05
000100.000000  0              0
```

```
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

Chicago->Kansas avg utilization : 9.9375e-05

avg utilization  : 0.000265
----utilization logs end----

-- Chicago node record ends --------------------------------

-- NewYork node record begins ------------------------------

convergence time               : 000000.091000
total floods                   : 193
total wasted floods            : 49
avg hops                       : 0
source failed calls            : 0
calls routed successfully      : 30
Intermediate hop failed calls : 0
calls confirmed from dest      : 0
CRANKBACKS                     : 0
avg routing time               : 000000.000000
pnni data sent(kbytes)         : 25.936

---utilization logs start ----

Total core bandwidth  : 1800

time            used_bw(mbps)
NewYork->Chicago TotalBw 600

000010.000000  0              0
000020.000000  0.2544         0.000424
000030.000000  0.2544         0.000424
000040.000000  0.2544         0.000424
000050.000000  0.2544         0.000424
000060.000000  0.2544         0.000424
000070.000000  0.2544         0.000424
000080.000000  0.2544         0.000424
000090.000000  0.1272         0.000212
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
```

```
000150.000000  0              0

NewYork->Chicago avg utilization : 0.0003975
NewYork->SanDiego TotalBw 1800

000010.000000  0              0
000020.000000  0.1272         7.06667e-05
000030.000000  0.1272         7.06667e-05
000040.000000  0.1272         7.06667e-05
000050.000000  0.1272         7.06667e-05
000060.000000  0.1272         7.06667e-05
000070.000000  0.1272         7.06667e-05
000080.000000  0.1272         7.06667e-05
000090.000000  0.0636         3.53333e-05
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

NewYork->SanDiego avg utilization : 6.625e-05
NewYork->Washington TotalBw 1200

000010.000000  0              0
000020.000000  0.1272         0.000106
000030.000000  0.1272         0.000106
000040.000000  0.1272         0.000106
000050.000000  0.1272         0.000106
000060.000000  0.1272         0.000106
000070.000000  0.1272         0.000106
000080.000000  0.1272         0.000106
000090.000000  0.0636         5.3e-05
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

NewYork->Washington avg utilization : 9.9375e-05

avg utilization  : 0.000265
----utilization logs end----

-- NewYork node record ends --------------------------------

-- SanDiego node record begins ------------------------------
```

```
convergence time               : 000000.092000
total floods                   : 192
total wasted floods            : 46
avg hops                       : 0
source failed calls            : 0
calls routed successfully      : 15
Intermediate hop failed calls : 0
calls confirmed from dest      : 0
CRANKBACKS                     : 0
avg routing time               : 000000.000000
pnni data sent(kbytes)         : 26.412

---utilization logs start ----

Total core bandwidth  : 1800

time            used_bw(mbps)
SanDiego->Denver TotalBw 1800

000010.000000  0              0
000020.000000  0              0
000030.000000  0              0
000040.000000  0              0
000050.000000  0              0
000060.000000  0              0
000070.000000  0              0
000080.000000  0              0
000090.000000  0              0
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0
SanDiego->NewYork TotalBw 600

000010.000000  0              0
000020.000000  0.1272         0.000212
000030.000000  0.1272         0.000212
000040.000000  0.1272         0.000212
000050.000000  0.1272         0.000212
000060.000000  0.1272         0.000212
000070.000000  0.1272         0.000212
000080.000000  0.1272         0.000212
000090.000000  0.0636         0.000106
000100.000000  0              0
000110.000000  0              0
```

```
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

SanDiego->NewYork avg utilization : 0.00019875
SanDiego->LosAngeles TotalBw 1200

000010.000000  0              0
000020.000000  0.1272         0.000106
000030.000000  0.1272         0.000106
000040.000000  0.1272         0.000106
000050.000000  0.1272         0.000106
000060.000000  0.1272         0.000106
000070.000000  0.1272         0.000106
000080.000000  0.1272         0.000106
000090.000000  0.0636         5.3e-05
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

SanDiego->LosAngeles avg utilization : 9.9375e-05

avg utilization  : 0.0001325
----utilization logs end----

-- SanDiego node record ends -------------------------------

-- Colorado node record begins -----------------------------

convergence time              : 000000.097000
total floods                  : 31
total wasted floods           : 19
avg hops                      : 4
source failed calls           : 0
calls routed successfully     : 15
Intermediate hop failed calls : 0
calls confirmed from dest     : 0
CRANKBACKS                    : 0
avg routing time              : 000000.002538
pnni data sent(kbytes)        : 3.372

---utilization logs start ----

Total core bandwidth  : 600
```

```
time            used_bw(mbps)
Colorado->Denver TotalBw 600

000010.000000  0              0
000020.000000  0.1272         0.000212
000030.000000  0.1272         0.000212
000040.000000  0.1272         0.000212
000050.000000  0.1272         0.000212
000060.000000  0.1272         0.000212
000070.000000  0.1272         0.000212
000080.000000  0.1272         0.000212
000090.000000  0.0636         0.000106
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

Colorado->Denver avg utilization : 0.00019875

avg utilization  : 0.00019875
----utilization logs end----
---CallTracing starts----

Call No: Trace by Node Number

        1Colorado.Denver.Chicago.NewYork.Washington.

        2Colorado.Denver.Chicago.NewYork.Washington.

        3Colorado.Denver.Chicago.NewYork.Washington.

        4Colorado.Denver.Chicago.NewYork.Washington.

        5Colorado.Denver.Chicago.NewYork.Washington.

        6Colorado.Denver.Chicago.NewYork.Washington.

        7Colorado.Denver.Chicago.NewYork.Washington.

        8Colorado.Denver.Chicago.NewYork.Washington.

        9Colorado.Denver.Chicago.NewYork.Washington.

       10Colorado.Denver.Chicago.NewYork.Washington.
       11Colorado.Denver.Chicago.NewYork.Washington.

       12Colorado.Denver.Chicago.NewYork.Washington.

       13Colorado.Denver.Chicago.NewYork.Washington.

       14Colorado.Denver.Chicago.NewYork.Washington.

       15Colorado.Denver.Chicago.NewYork.Washington.

---CallTracing ends----


-- Colorado node record ends --------------------------------

-- Kansas node record begins ------------------------------

convergence time             : 000000.103000
total floods                 : 33
total wasted floods          : 21
avg hops                     : 4
source failed calls          : 0
calls routed successfully    : 15
Intermediate hop failed calls : 0
calls confirmed from dest    : 0
CRANKBACKS                   : 0
avg routing time             : 000000.002903
pnni data sent(kbytes)       : 3.468

---utilization logs start ----

Total core bandwidth  : 600

time            used_bw(mbps)
Kansas->Chicago TotalBw 600

000010.000000  0              0
000020.000000  0.1272         0.000212
000030.000000  0.1272         0.000212
000040.000000  0.1272         0.000212
000050.000000  0.1272         0.000212
000060.000000  0.1272         0.000212
000070.000000  0.1272         0.000212
000080.000000  0.1272         0.000212
000090.000000  0.0636         0.000106
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
```

```
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

Kansas->Chicago avg utilization : 0.00019875

avg utilization  : 0.00019875
----utilization logs end----
---CallTracing starts----

Call No: Trace by Node Number

        1Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        2Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        3Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        4Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        5Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        6Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        7Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        8Kansas.Chicago.NewYork.SanDiego.LosAngeles.

        9Kansas.Chicago.NewYork.SanDiego.LosAngeles.

       10Kansas.Chicago.NewYork.SanDiego.LosAngeles.

       11Kansas.Chicago.NewYork.SanDiego.LosAngeles.

       12Kansas.Chicago.NewYork.SanDiego.LosAngeles.

       13Kansas.Chicago.NewYork.SanDiego.LosAngeles.

       14Kansas.Chicago.NewYork.SanDiego.LosAngeles.

       15Kansas.Chicago.NewYork.SanDiego.LosAngeles.

---CallTracing ends----


-- Kansas node record ends --------------------------------

-- Washington node record begins ------------------------------
```

```
convergence time              : 000000.104000
total floods                  : 31
total wasted floods           : 20
avg hops                      : 0
source failed calls           : 0
calls routed successfully     : 15
Intermediate hop failed calls : 0
calls confirmed from dest     : 0
CRANKBACKS                    : 0
avg routing time              : 000000.000000
pnni data sent(kbytes)        : 2.984

---utilization logs start ----

Total core bandwidth  : 600

time            used_bw(mbps)
Washington->NewYork TotalBw 600

000010.000000  0             0
000020.000000  0.1272        0.000212
000030.000000  0.1272        0.000212
000040.000000  0.1272        0.000212
000050.000000  0.1272        0.000212
000060.000000  0.1272        0.000212
000070.000000  0.1272        0.000212
000080.000000  0.1272        0.000212
000090.000000  0.0636        0.000106
000100.000000  0             0
000110.000000  0             0
000120.000000  0             0
000130.000000  0             0
000140.000000  0             0
000150.000000  0             0

Washington->NewYork avg utilization : 0.00019875

avg utilization  : 0.00019875
----utilization logs end----

-- Washington node record ends ------------------------------

-- LosAngeles node record begins ------------------------------

convergence time              : 000000.105000
total floods                  : 32
total wasted floods           : 21
```

```
avg hops                      : 0
source failed calls           : 0
calls routed successfully     : 15
Intermediate hop failed calls : 0
calls confirmed from dest     : 0
CRANKBACKS                    : 0
avg routing time              : 000000.000000
pnni data sent(kbytes)        : 3.032

---utilization logs start ----

Total core bandwidth  : 600

time            used_bw(mbps)
LosAngeles->SanDiego TotalBw 600

000010.000000  0              0
000020.000000  0.1272         0.000212
000030.000000  0.1272         0.000212
000040.000000  0.1272         0.000212
000050.000000  0.1272         0.000212
000060.000000  0.1272         0.000212
000070.000000  0.1272         0.000212
000080.000000  0.1272         0.000212
000090.000000  0.0636         0.000106
000100.000000  0              0
000110.000000  0              0
000120.000000  0              0
000130.000000  0              0
000140.000000  0              0
000150.000000  0              0

LosAngeles->SanDiego avg utilization : 0.00019875

avg utilization  : 0.00019875
----utilization logs end----

-- LosAngeles node record ends ---------------------------------


AVG NODE RECORDS

convergence time  low         : 000000.090000
convergence time high         : 000000.105000
avg hops                      : 4
total floods                  : 882
total wastedfloods            : 285
pnni bw low                   : 2.984
pnni bw high                  : 26.412
total pnni data(kbytes)       : 115.244

### NODE INSTRUMENTATION LOGS END ######
```

- Finally, with the `-l` option, the simulator will create the following files with the results segregated into: Note: the output has been modified to format/fit a printable page.

   1. HostRecordFile.output contains Host information

```
Host CoS    Total_Calls Successful_call(%) Total_BW_Req(mbps) BW_Rejecte
Host4 CBR     15          100                0.954              0
Host5 CBR     15          100                0.954              0

Host mean_call_set_up_time(msec)
Host4       000.146466
Host5       000.144999
```

   2. SwitchRecordFile.output contains Switch information

```
Switch      Conv_Time  T_Flood T_W_Flood Avg_Hops Calls_Requested Src_Fa
Denver      000.900000 184     54        0        15              0
Chicago     000.910000 186     55        0        30              0
NewYork     000.910000 193     49        0        30              0
SanDiego    000.920000 192     46        0        15              0
Colorado    000.970000 31      19        4        15              0
Kansas      000.103000 33      21        4        15              0
Washington  000.104000 31      20        0        15              0
LosAngeles  000.105000 32      21        0        15              0

Switch Calls_Routed Inter_Fail  Crankbacks Alt.Routed Calls_Confirmed
Denver        15           0           0          0          0
Chicago       30           0           0          0          0
NewYork       30           0           0          0          0
SanDiego      15           0           0          0          0
Colorado      15           0           0          0          0
Kansas        15           0           0          0          0
Washington    15           0           0          0          0
LosAngeles    15           0           0          0          0

Switch Avg_Routing_Time PNNI_data_sent Avg_ Utilization
Denver      000.000000       24.664         0.01325
Chicago     000.000000       25.376         0.0265
NewYork     000.000000       25.936         0.0265
SanDiego    000.000000       26.412         0.01325
Colorado    000.253800       3.372          0.019875
Kansas      000.290300       3.468          0.019875
Washington  000.000000       2.984          0.019875
```

```
LosAngeles  000.000000       3.032          0.019875
```

Legend: Switch represents the node for which the report is being generated
Conv_Time is the time when a node obtains all the initial topology information generated by the other nodes in the topology
T_Flood is total number of PNNI topology information messages generated
T_W_Flood is total number of REDUNDANT PNNI topology information messages generated
Avg_Hops is the the average number of links traversed by a connection request for a successful call
Calls_Requested is count of number of call requests that the node got
Source_Fail is count of number of call requests that got rejected at this node
Calls_Routed is count of number of call requests that this node successfully forwarded
Inter_Fail is count of number of call requests that got rejected
further down in the network by subsequent nodes
Crankbacks is count of number of crankbacks that this node serviced
Alt.Routed is count of number of Alternate Routes that were generated at this node
Calls_Confirmed is count of number of call requests that were confirmed to have been setup
Avg_Routing_Time is average time taken to compute a route using a routing algorithm in seconds
PNNI_data_sent is the total PNNI topology data bytes sent from a node
Avg_Utilization shows the link utilization averaged over all links out of this node

3. LinkRecordFile.output contains Link information

```
Link                       Total Peak Utilization Avg_Utilization(%)

Denver->Chicago            1200  0.1272           9.9375e-05

Denver->SanDiego           1800  0                0

Denver->Colorado           600   0.1272           0.00019875

Chicago->Denver            600   0.1272           0.00019875

Chicago->NewYork           1800  0.2544           0.0001325

Chicago->Kansas            1200  0.1272           9.9375e-05

NewYork->Chicago           600   0.2544           0.0003975

NewYork->SanDiego          1800  0.1272           6.625e-05

NewYork->Washington        1200  0.1272           9.9375e-05

SanDiego->Denver           1800  0                0

SanDiego->NewYork          600   0.1272           0.00019875
```

```
SanDiego->LosAngeles       1200  0.1272           9.9375e-05

Colorado->Denver           600   0.1272           0.00019875

Kansas->Chicago            600   0.1272           0.00019875

Washington->NewYork        600   0.1272           0.00019875

LosAngeles->SanDiego       600   0.1272           0.00019875
```

4. CalltraceRecordFile.output contains Calltraceinformation information

```
---CallTracing starts----

Calls from Node Colorado

Call No:  Trace by Node Number

1        Colorado.Denver.Chicago.NewYork.Washington.

2        Colorado.Denver.Chicago.NewYork.Washington.

3        Colorado.Denver.Chicago.NewYork.Washington.

4        Colorado.Denver.Chicago.NewYork.Washington.

5        Colorado.Denver.Chicago.NewYork.Washington.

6        Colorado.Denver.Chicago.NewYork.Washington.

7        Colorado.Denver.Chicago.NewYork.Washington.

8        Colorado.Denver.Chicago.NewYork.Washington.

9        Colorado.Denver.Chicago.NewYork.Washington.

10       Colorado.Denver.Chicago.NewYork.Washington.

11       Colorado.Denver.Chicago.NewYork.Washington.

12       Colorado.Denver.Chicago.NewYork.Washington.

13       Colorado.Denver.Chicago.NewYork.Washington.

14       Colorado.Denver.Chicago.NewYork.Washington.

15       Colorado.Denver.Chicago.NewYork.Washington.

Calls from Node Kansas
```

```
Call No:  Trace by Node Number

1        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

2        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

3        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

4        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

5        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

6        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

7        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

8        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

9        Kansas.Chicago.NewYork.SanDiego.LosAngeles.

10       Kansas.Chicago.NewYork.SanDiego.LosAngeles.

11       Kansas.Chicago.NewYork.SanDiego.LosAngeles.

12       Kansas.Chicago.NewYork.SanDiego.LosAngeles.

13       Kansas.Chicago.NewYork.SanDiego.LosAngeles.

14       Kansas.Chicago.NewYork.SanDiego.LosAngeles.

15       Kansas.Chicago.NewYork.SanDiego.LosAngeles.

---CallTracing ends----
```

5. NetworkRecordFile.output contains Network information

```
***** CALL SETUP LOGS START ******

-- Detailed Host4 host record begins ---------------------------
---

calltype  bw(kbps)  starttime     stoptime      setuptime   result   cau
CBR           63.6 00:00:10.000  00:00:10.250  000.250000  setup
CBR           63.6 00:00:15.000  00:00:15.136  000.136000  setup
CBR           63.6 00:00:20.000  00:00:20.139  000.139000  setup
CBR           63.6 00:00:25.000  00:00:25.136  000.136000  setup
CBR           63.6 00:00:30.000  00:00:30.139  000.139000  setup
CBR           63.6 00:00:35.000  00:00:35.136  000.136000  setup
CBR           63.6 00:00:40.000  00:00:40.139  000.139000  setup
CBR            63.6 00:00:45.000  00:00:45.136  000.136000  setup
CBR            63.6 00:00:50.000  00:00:50.139  000.139000  setup
CBR            63.6 00:00:55.000  00:00:55.136  000.136000  setup
CBR            63.6 00:01:00.000  00:01:00.139  000.139000  setup
CBR            63.6 00:01:05.000  00:01:05.136  000.136000  setup
CBR            63.6 00:01:10.000  00:01:10.139  000.139000  setup
CBR            63.6 00:01:15.000  00:01:15.136  000.136000  setup
CBR            63.6 00:01:20.000  00:01:20.139  000.139000  setup

-- Host4 host record ends ----------------------------

-- Detailed Host5 host record begins ---------------------------
---

calltype  bw(kbps)  starttime     stoptime      setuptime   result   cau
CBR            63.6 00:00:10.000  00:00:10.272  000.272000  setup
CBR            63.6 00:00:15.000  00:00:15.136  000.136000  setup
CBR            63.6 00:00:20.000  00:00:20.139  000.139000  setup
CBR            63.6 00:00:25.000  00:00:25.136  000.136000  setup
CBR            63.6 00:00:30.000  00:00:30.139  000.139000  setup
CBR            63.6 00:00:35.000  00:00:35.136  000.136000  setup
CBR            63.6 00:00:40.000  00:00:40.139  000.139000  setup
CBR            63.6 00:00:45.000  00:00:45.136  000.136000  setup
CBR            63.6 00:00:50.000  00:00:50.139  000.139000  setup
CBR            63.6 00:00:55.000  00:00:55.136  000.136000  setup
CBR            63.6 00:01:00.000  00:01:00.139  000.139000  setup
CBR            63.6 00:01:05.000  00:01:05.136  000.136000  setup
CBR            63.6 00:01:10.000  00:01:10.139  000.139000  setup
CBR            63.6 00:01:15.000  00:01:15.136  000.136000  setup
CBR            63.6 00:01:20.000  00:01:20.139  000.139000  setup

-- Host5 host record ends ----------------------------

AVG RESULTS OF ALL CALLS
total cbr calls             : 30
% successfull cbr calls     : 100
total cbrbw request(mb)     : 1.908
cbrbwrej(mb)                : 0
mean callsetup time         : 000.145733

***** CALL SETUP LOGS END **********

##### NODE INSTRUMENTATION LOGS START #######

AVG NODE RECORDS

convergence time  low         : 000000.090000
convergence time high         : 000000.105000
```

```
avg hops                      : 4
total floods                  : 882
total wastedfloods            : 285
pnni bw low                   : 2.984
pnni bw high                  : 26.412
total pnni data(kbytes)       : 115.244

### NODE INSTRUMENTATION LOGS END ######
```

6. DetailedLinkRecordFile.output contains Detailed Link information

```
-- Denver node record begins -----------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 1800

time                 used_bw(mbps)   % of total

Denver->Chicago TotalBw 1200

000010.000000              0            0
000020.000000         0.1272       0.0106
000030.000000         0.1272       0.0106
000040.000000         0.1272       0.0106
000050.000000         0.1272       0.0106
000060.000000         0.1272       0.0106
000070.000000         0.1272       0.0106
000080.000000         0.1272       0.0106
000090.000000         0.0636       0.0053
000100.000000              0            0
000110.000000              0            0
000120.000000              0            0
000130.000000              0            0
000140.000000              0            0
000150.000000              0            0

Denver->Chicago avg utilization : 9.9375e-05

Denver->SanDiego TotalBw 1800

000010.000000              0            0
000020.000000              0            0
000030.000000              0            0
000040.000000              0            0
000050.000000              0            0
000060.000000              0            0
000070.000000              0            0
000080.000000              0            0
```

```
000090.000000              0            0
000100.000000              0            0
000110.000000              0            0
000120.000000              0            0
000130.000000              0            0
000140.000000              0            0
000150.000000              0            0

Denver->Colorado TotalBw 600

000010.000000              0            0
000020.000000         0.1272       0.0212
000030.000000         0.1272       0.0212
000040.000000         0.1272       0.0212
000050.000000         0.1272       0.0212
000060.000000         0.1272       0.0212
000070.000000         0.1272       0.0212
000080.000000         0.1272       0.0212
000090.000000         0.0636       0.0106
000100.000000              0            0
000110.000000              0            0
000120.000000              0            0
000130.000000              0            0
000140.000000              0            0
000150.000000              0            0

Denver->Colorado avg utilization : 0.00019875

----utilization logs end----

-- Denver node record ends --------------------------------

-- Chicago node record begins -------------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 1800

time                 used_bw(mbps)   % of total

Chicago->Denver TotalBw 600

000010.000000              0            0
000020.000000         0.1272       0.0212
000030.000000         0.1272       0.0212
000040.000000         0.1272       0.0212
000050.000000         0.1272       0.0212
000060.000000         0.1272       0.0212
```

```
000070.000000          0.1272       0.0212
000080.000000          0.1272       0.0212
000090.000000          0.0636       0.0106
000100.000000               0            0
000110.000000               0            0
000120.000000               0            0
000130.000000               0            0
000140.000000               0            0
000150.000000               0            0

Chicago->Denver avg utilization : 0.00019875

Chicago->NewYork TotalBw 1800

000010.000000               0            0
000020.000000          0.2544    0.0141333
000030.000000          0.2544    0.0141333
000040.000000          0.2544    0.0141333
000050.000000          0.2544    0.0141333
000060.000000          0.2544    0.0141333
000070.000000          0.2544    0.0141333
000080.000000          0.2544    0.0141333
000090.000000          0.1272   0.00706667
000100.000000               0            0
000110.000000               0            0
000120.000000               0            0
000130.000000               0            0
000140.000000               0            0
000150.000000               0            0

Chicago->NewYork avg utilization : 0.0001325

Chicago->Kansas TotalBw 1200

000010.000000               0            0
000020.000000          0.1272       0.0106
000030.000000          0.1272       0.0106
000040.000000          0.1272       0.0106
000050.000000          0.1272       0.0106
000060.000000          0.1272       0.0106
000070.000000          0.1272       0.0106
000080.000000          0.1272       0.0106
000090.000000          0.0636       0.0053
000100.000000               0            0
000110.000000               0            0
000120.000000               0            0
000130.000000               0            0
000140.000000               0            0
```

```
000150.000000              0           0

Chicago->Kansas avg utilization : 9.9375e-05

----utilization logs end----

-- Chicago node record ends ------------------------------

-- NewYork node record begins ----------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 1800

time                 used_bw(mbps)   % of total

NewYork->Chicago TotalBw 600

000010.000000              0           0
000020.000000         0.2544      0.0424
000030.000000         0.2544      0.0424
000040.000000         0.2544      0.0424
000050.000000         0.2544      0.0424
000060.000000         0.2544      0.0424
000070.000000         0.2544      0.0424
000080.000000         0.2544      0.0424
000090.000000         0.1272      0.0212
000100.000000              0           0
000110.000000              0           0
000120.000000              0           0
000130.000000              0           0
000140.000000              0           0
000150.000000              0           0

NewYork->Chicago avg utilization : 0.0003975

NewYork->SanDiego TotalBw 1800

000010.000000              0           0
000020.000000         0.1272  0.00706667
000030.000000         0.1272  0.00706667
000040.000000         0.1272  0.00706667
000050.000000         0.1272  0.00706667
000060.000000         0.1272  0.00706667
000070.000000         0.1272  0.00706667
000080.000000         0.1272  0.00706667
000090.000000         0.0636  0.00353333
000100.000000              0           0
```

```
000110.000000              0            0
000120.000000              0            0
000130.000000              0            0
000140.000000              0            0
000150.000000              0            0

NewYork->SanDiego avg utilization : 6.625e-05

NewYork->Washington TotalBw 1200

000010.000000              0            0
000020.000000         0.1272       0.0106
000030.000000         0.1272       0.0106
000040.000000         0.1272       0.0106
000050.000000         0.1272       0.0106
000060.000000         0.1272       0.0106
000070.000000         0.1272       0.0106
000080.000000         0.1272       0.0106
000090.000000         0.0636       0.0053
000100.000000              0            0
000110.000000              0            0
000120.000000              0            0
000130.000000              0            0
000140.000000              0            0
000150.000000              0            0

NewYork->Washington avg utilization : 9.9375e-05

----utilization logs end----

-- NewYork node record ends --------------------------------

-- SanDiego node record begins ------------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 1800

time                 used_bw(mbps)   % of total

SanDiego->Denver TotalBw 1800

000010.000000              0            0
000020.000000              0            0
000030.000000              0            0
000040.000000              0            0
000050.000000              0            0
000060.000000              0            0
```

```
000070.000000             0            0
000080.000000             0            0
000090.000000             0            0
000100.000000             0            0
000110.000000             0            0
000120.000000             0            0
000130.000000             0            0
000140.000000             0            0
000150.000000             0            0

SanDiego->NewYork TotalBw 600

000010.000000             0            0
000020.000000        0.1272       0.0212
000030.000000        0.1272       0.0212
000040.000000        0.1272       0.0212
000050.000000        0.1272       0.0212
000060.000000        0.1272       0.0212
000070.000000        0.1272       0.0212
000080.000000        0.1272       0.0212
000090.000000        0.0636       0.0106
000100.000000             0            0
000110.000000             0            0
000120.000000             0            0
000130.000000             0            0
000140.000000             0            0
000150.000000             0            0

SanDiego->NewYork avg utilization : 0.00019875

SanDiego->LosAngeles TotalBw 1200

000010.000000             0            0
000020.000000        0.1272       0.0106
000030.000000        0.1272       0.0106
000040.000000        0.1272       0.0106
000050.000000        0.1272       0.0106
000060.000000        0.1272       0.0106
000070.000000        0.1272       0.0106
000080.000000        0.1272       0.0106
000090.000000        0.0636       0.0053
000100.000000             0            0
000110.000000             0            0
000120.000000             0            0
000130.000000             0            0
000140.000000             0            0
000150.000000             0            0
```

```
SanDiego->LosAngeles avg utilization : 9.9375e-05

----utilization logs end----

-- SanDiego node record ends --------------------------------

-- Colorado node record begins ------------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 600

time                 used_bw(mbps)   % of total

Colorado->Denver TotalBw 600

000010.000000              0            0
000020.000000         0.1272       0.0212
000030.000000         0.1272       0.0212
000040.000000         0.1272       0.0212
000050.000000         0.1272       0.0212
000060.000000         0.1272       0.0212
000070.000000         0.1272       0.0212
000080.000000         0.1272       0.0212
000090.000000         0.0636       0.0106
000100.000000              0            0
000110.000000              0            0
000120.000000              0            0
000130.000000              0            0
000140.000000              0            0
000150.000000              0            0

Colorado->Denver avg utilization : 0.00019875

----utilization logs end----

-- Colorado node record ends --------------------------------

-- Kansas node record begins ------------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 600

time                 used_bw(mbps)   % of total

Kansas->Chicago TotalBw 600
```

```
000010.000000               0            0
000020.000000          0.1272       0.0212
000030.000000          0.1272       0.0212
000040.000000          0.1272       0.0212
000050.000000          0.1272       0.0212
000060.000000          0.1272       0.0212
000070.000000          0.1272       0.0212
000080.000000          0.1272       0.0212
000090.000000          0.0636       0.0106
000100.000000               0            0
000110.000000               0            0
000120.000000               0            0
000130.000000               0            0
000140.000000               0            0
000150.000000               0            0

Kansas->Chicago avg utilization : 0.00019875

----utilization logs end----

-- Kansas node record ends --------------------------------

-- Washington node record begins ------------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 600

time                 used_bw(mbps)   % of total

Washington->NewYork TotalBw 600

000010.000000               0            0
000020.000000          0.1272       0.0212
000030.000000          0.1272       0.0212
000040.000000          0.1272       0.0212
000050.000000          0.1272       0.0212
000060.000000          0.1272       0.0212
000070.000000          0.1272       0.0212
000080.000000          0.1272       0.0212
000090.000000          0.0636       0.0106
000100.000000               0            0
000110.000000               0            0
000120.000000               0            0
000130.000000               0            0
000140.000000               0            0
000150.000000               0            0
```

```
Washington->NewYork avg utilization : 0.00019875

----utilization logs end----

-- Washington node record ends --------------------------------

-- LosAngeles node record begins ------------------------------

---Detailed utilization logs start ----

Total core bandwidth  : 600

time                 used_bw(mbps)   % of total

LosAngeles->SanDiego TotalBw 600

000010.000000              0            0
000020.000000         0.1272       0.0212
000030.000000         0.1272       0.0212
000040.000000         0.1272       0.0212
000050.000000         0.1272       0.0212
000060.000000         0.1272       0.0212
000070.000000         0.1272       0.0212
000080.000000         0.1272       0.0212
000090.000000         0.0636       0.0106
000100.000000              0            0
000110.000000              0            0
000120.000000              0            0
000130.000000              0            0
000140.000000              0            0
000150.000000              0            0

LosAngeles->SanDiego avg utilization : 0.00019875

----utilization logs end----

-- LosAngeles node record ends --------------------------------
```

Bibliography

[1] ATM Forum. *"Private Network-Network Interface Specification Version 1.0"* February 1997

[2] Raj Jain, *"PNNI: Routing in ATM Networks"* http://www.cis.ohio-state.edu/ jain

[3] Raif O. Onvural, Rao Cherukuri, *"Signaling in ATM Networks"* Artech House, Massachusetts 1997

[4] ITU Recommendation Q.2931, *"B-ISDN Digital Subscriber Signaling System No 2 User To Network Interface Layer 3 Specification for Basic Call/Connection Control"*, ITU-International T-elecommunication Union, 1995

[5] ATM Forum Technical Committee, *"ATM User Network Interface specification version 3.1"*, ATM FORUM, September 1994

[6] ATM Forum Technical Committee, *"ATM User Network Interface specification version 4.0 (af-sig-0061.000)"*, ATM FORUM, July 1996

Appendix A

Component Specific Details

This section details the ten components of the language and their legal parameters. The parameter set associated with each of the components is described along with the values that they may be assigned.

For any of the component group discussed, all parameters are optional unless specified otherwise below. Any number of parameters are allowed and parameters may be repeated if desired (the last value will be the one retained). However, if no parameters are to be given, then *do not* include any parenthesis. See the declaration for Host or node in the example script in program 9.1.

A.1 Parameter block for Node Information

Below is a listing of currently supported parameters, their default values, and their accepted values. The listing also provides information about the necessity of the parameter. In the case of optional parameters, any parameter not specified by the user will default to the programmed value.

parameter:	**ptse_age_offset**
description:	The random number of the offset value of PTSE lifetime. The offset unit is in percent.
values:	[fixed <value>] or [uniform <low> <high>]
default:	[uniform -25 25]
optional:	Yes
example:	- uniform with low = 20 and high = 30 shown as ptse_age_offset = [uniform 20 30] - fixed with value = 25 shown as ptse_age_offset = [fixed 25]
parameter:	**fabric**
description:	The switch fabric used.
values:	ku, generic, plaid
default:	ku
optional:	Yes

parameter:	**numports**
description:	The number of switch ports including the control port.
values:	Integers from 2 to 64
default:	4
optional:	Yes

parameter:	**routing_policy**
description:	The policy to be used in the routing algorithm.
	- Minimum Hop policy (min_hop) is the distance vector policy which assigns the same cost to all the links.
	- Maximum Bandwidth policy (max_bw) computes a route with maximum bandwidth.
	- Minimum Delay policy (min_delay) computes a route with minimum time to the destination.
	- Minimum Administrative Weight policy (min_adw) finds a path with the minimum administrative weight.
	- Shortest and Minimum Administrative Weight policy (shortest_min_adw) evaluates min_adw routing first; if there is are more than one routes, the min_delay (shortest delay path) routing is applied to these routes to find the final route.
	- Minimum Administrative Weight and Shortest policy (min_adw_shortest) evaluates the routes based on min_delay first. If there are more than one routes, a min_adw routing is done on these set of equal routes to find the final route.
	- Shortest and Widest policy (shortest_widest) evaluates the route for maximizing the bandwidth. If there are more than one routes, min_delay routing policy is used to find the final route.
	- Widest and Minimum Hop policy (widest_min_hop) finds the route with minimum hop count first. If there are more than one possible routes, we select the path with maximum bandwidth.
	- Minimum Administrative Weight and Widest policy (min_adw_widest) finds the route with maximum bandwidth first. If there are more than one possible routes, we select the path that has minimum administrative weight.
values:	min_hop, max_bw, min_delay, min_adw, shortest_min_adw, min_adw_shortest, shortest_widest, widest_min_hop, min_adw_widest
default:	max_bw
optional:	Yes

parameter:	**flooding_significance**
description:	The policy to be used for horizontal link PTSE flooding. Use an explicit threshold of minimum bandwidth or a re-origination function based dynamic_threshold policy.
values:	explicit_threshold, dynamic_threshold
default:	dynamic_threshold
optional:	Yes

parameter:	**flooding_threshold**
description:	Minimum bandwidth threshold to be used for flooding PNNI link information.
values:	Bandwidth in terms of the number of 53 byte ATM cells.
default:	2
optional:	Optional if experimentation does not involve routing. Mandatory in case of experimentation involving routing and the value for the parameter flooding_significance is either explicit_threshold or dynamic_threshold.

parameter:	**prop_constant**
description:	The proportionality constant to be used in the re-origination function for the dynamic_threshold case.
values:	Integers from 0 to 99
default:	10
optional:	Yes

parameter:	**default_flooding_period**
description:	The PNNI default flooding period if the topology information is not re-originated.
values:	Integer value in seconds
default:	1800
optional:	Yes

parameter:	**default_flooding_factor**
description:	A constant multiplied together with the default_flooding_period.
values:	Positive integer
default:	2
optional:	Yes

parameter:	**acac_policy**
description:	Policy used in the connection admission control. In call_packing, if more than one link is available to the next hop, then the routing is based on loading the links one by one. In load_balancing, the load is evenly distributed amongst all of the links.
values:	call_packing, load_balancing
default:	call_packing
optional:	Yes

parameter:	**util_log_period**
description:	This is the interval in seconds to log the link utilization for experimental purposes.
values:	Non-negative integer value in seconds
default:	0
optional:	Yes

parameter:	**calltrace**
description:	This is the switch to enable logging of the route that each successful call makes.
values:	true, false
default:	false
optional:	Yes

parameter:	**queuesize**
description:	Sets the packet queue size in switch ports. Any packet arriving when the queue is full is dropped.
values:	Non-negative integer
default:	200
optional:	Yes

parameter:	**hello_timer**
description:	The PNNI hello retransmit timer.
values:	Integer value in seconds
default:	15
optional:	Yes

parameter:	**hello_inactivity_factor**
description:	The PNNI hello inactivity factor determines the amount of time in which a hello packet is expected from the peer.
values:	Integer
default:	4
optional:	Yes

parameter:	**summary_timer**
description:	The PNNI summary timer is the timer used for retransmission of topology summary information.
values:	Integer value in seconds
default:	20
optional:	Yes

parameter:	**ptsp_timer**
description:	Timer for retransmission of PTSP packets.
values:	Integer value in seconds
default:	20
optional:	Yes

parameter:	**ack_timer**
description:	Timer for transmission of acknowledgement packets.
values:	Integer value in seconds
default:	5
optional:	Yes

parameter:	**request_timer**
description:	Timer for transmission of request packets.
values:	Integer value in seconds.
default:	20
optional:	Yes

parameter:	**reaggregation_timer**
description:	Timer for reaggregation of peer group state information
values:	Integer value in seconds.
default:	0
optional:	Yes

parameter:	**process_time**
description:	The message process time in nodes.
values:	Real number in milliseconds
default:	2.0
optional:	Yes

A.2 Individual Node Information

Individual node information contains node specific information when it differs from the specified generic node information.

All of the generic node parameters listed in section A.1 can also be used in individual node information with the following exceptions:

Unavailable parameters:

- ctrl
- link
- duration
- convergence
- process_time

Additional parameters:

parameter:	**parameter_block**
description:	The base building block from which the node can inherit common properties
values:	parameter block name
default:	none.
optional:	If this block is not given, the default parameters are assumed.

parameter:	**level**
description:	The number of bits that need to be same within a peer group for PNNI to function.
values:	Integer ranging from 8 to 104
default:	96, assuming the lowest hierarchy, this is the maximum number of bits that can be used as a prefix. Higher the level indicator, lower is the node in the hierarchy
optional:	Yes

parameter:	**peergroupid**
description:	To represent whether a physical node or logical node
values:	160 for physical node
default:	160
optional:	Yes

parameter:	**address**
description:	The address of the node represented as a 40 byte string
values:	40 byte atm address
optional:	Mandatory

parameter:	**leader**
description:	This is set for a peer group leader
values:	true, false
default:	false
optional:	Yes

parameter:	**bordernode**
description:	This is set for a border node
values:	true, false
default:	false
optional:	Yes

parameter:	**aggr_token**
description:	To uniquely identify a border node, an aggregation token is configured
values:	Any Integer
default:	0
optional:	Mandatory for a border node

parameter:	**nodal_aggr_policy**
description:	The aggregation policy used for Nodal Aggregation
values:	optimistic, pessimistic, average, mesh, sym_average
default:	mesh
optional:	Yes

parameter:	**link_aggr_policy**
description:	The link aggregation policy to be used
values:	optimistic, pessimistic, average
default:	average
optional:	Yes

A.3 Generic Port Information

Below is a listing of currently supported parameters, their default values, and their ranges. The listing also provides information about the necessity of the parameter. In the case of optional parameters, if the parameter is not specified by the user, the programmed value will be used.

parameter:	**bw**
description:	The port bandwidth for all of the ports on the switch.
values:	OC3 (155Mbps), DS3 (45Mbps), OC12 (600 Mbps), DS1 (1.5 Mbps), T1 (1.5 Mbps), or an integer that explicitly specifies a value in Mbps.
default:	OC3
optional:	Yes

parameter:	**delay**
description:	The link delay for links.
values:	Integer value in milliseconds
default:	0
optional:	Yes

parameter:	**cdv**
description:	The cell delay variation suffered at each of the links.
values:	Integer value in millisecond (must be less than the link delay)
default:	0
optional:	Yes

parameter:	**clr**
description:	The cell loss ratio at the links.
values:	Positive integer (The cell loss ratio is $10^{(-\mathrm{clr})}$)
default:	0
optional:	Yes

parameter:	**ad_weight**
description:	The administrative weight of a link.
values:	Integer between 0 and 99.
default:	0
optional:	Yes

A.4 Parameter block for Host Information

Below is a listing of currently supported parameters, their default values, and their ranges. The listing also provides information about the necessity of the parameter. In the case of optional parameters, if the parameter is not specified by the user, the programmed value will be used.

parameter:	**calltype**
description:	This is the service type of the call to be attempted.
values:	CBR for constant bit rate service, ABR for available bit rate service, RTVBR for real time variable bit rate service, VBR for non real time variable bit rate service, or UBR for unspecified bit rate service
default:	CBR
optional:	Yes

parameter:	**calls**
description:	The number of calls to be attempted. The value *unspecified* indicates that calls are attempted for the duration of the simulation.
values:	Integer or unspecified
default:	10
optional:	Yes

parameter:	**arrival_distribution**
description:	The distribution of call arrivals.
values:	periodic, poisson, bursty, tear_down
default:	periodic
optional:	Yes

parameter:	**arrival_period**
description:	In the case of periodic distribution, it's the period of the call arrivals.
values:	Integer value in seconds
default:	10
optional:	Yes

parameter:	**arrival_low**
description:	Lower limit of uniform arrival distribution.
values:	Integer value in seconds
default:	5
optional:	Yes

parameter:	**arrival_high**
description:	Upper limit of uniform arrival distribution.
values:	Integer value in seconds
default:	10
optional:	Yes

parameter:	**arrival_mean**
description:	Mean arrival time of a Poisson distribution.
values:	Integer value in seconds
default:	15
optional:	Yes

parameter:	**duration_distribution**
description:	The distribution of call durations.
values:	periodic, poisson, uniform
default:	periodic
optional:	Yes

parameter:	**duration_period**
description:	In the case of periodic distribution, it's the period of call durations.
values:	Integer value in seconds
default:	40
optional:	Yes

parameter:	**duration_low**
description:	Lower limit of a uniform duration distribution.
values:	Integer value in seconds
default:	5
optional:	Yes

parameter:	**duration_high**
description:	Upper limit of a uniform duration distribution.
values:	Integer value in seconds
default:	10
optional:	Yes

parameter:	**duration_mean**
description:	Mean duration time of a Poisson duration distribution.
values:	Integer value in seconds
default:	15
optional:	Yes

parameter:	**sourcetype**
description:	The type of source option in the host: *single* when we have only one traffic source, *multiple* when we have more than one traffic source.
values:	single, multiple
default:	single
optional:	Yes

parameter:	**numsources**
description:	The number of traffic sources in a host.
values:	Integer more than 0
default:	Not applicable
optional:	Required when sourcetype = multiple

parameter:	**share (followed by the source number)**
description:	The percentage of total calls attempted by a traffic source.
values:	1 to 100
default:	0
optional:	If sourcetype is multiple then for each source type this parameter has to be specified, for example sourcetype=multiple, numsources=3, share1=30, share2=30, share3=40

parameter:	**duration**
description:	The duration of the simulation.
values:	Integer value in seconds
default:	1000
optional:	Yes

parameter:	**queuesize**
description:	Sets the packet queue size in host ports. Any packet arriving when the queue is full is dropped.
values:	Integer
default:	200
optional:	Yes

parameter:	**host_process_time**
description:	The message process time in hosts.
values:	Real number in milliseconds
default:	5.0
optional:	Yes

parameter:	**call_bw**
description:	The bandwidth for call connection requests. The bandwidth could have a distribution - could be fixed, poisson or uniform. For fixed, the constant BW is the next parameter, for uniform, low and high parameters and for poisson, the mean is required. All bandwidth parameters are meansured in Kbps
values:	[fixed <value>] or [poisson <mean>] or [uniform <low> <high>]
default:	[fixed 64]
optional:	Yes

parameter:	**ctd**
description:	The cell transfer delay for the call connection request.
values:	Integer value in milliseconds
default:	0
optional:	Yes

parameter:	**clr**
description:	The cell loss ratio for call connection requests.
values:	Non-negative integer (The cell loss ratio is $10^{(-\mathtt{clr})}$)
default:	0
optional:	Yes

parameter:	**cdv**
description:	The cell delay variation suffered for call connection requests.
values:	Integer value in milliseconds
default:	0
optional:	Yes

If multiple sources are present, then the following parameters can be specified for each by simply adding the source number to the end of the parameter name:

- share (required)
- distribution (duration_distribution)
- duration_period
- duration_low
- duration_high
- duration_mean
- call_bw
- cdv
- ctd
- clr

For example, for source 1, we could specify share1, distribution1, duration_low1, duration_high1, call_bw1, ctd1, clr1, and cdv1. Perhaps for source 2, we specify share2, distribution2, duration_mean2, call_bw2, cdv2, clr2, and ctd2. And so on...

A.5 Individual Host information

Individual host information contains host specific information when it differs from the generic host information.

All of the generic host parameters listed in section A.4 can also be use in individual host information with the following exceptions:

Unavailable parameters:

- link
- duration
- host_process_time

Additional parameters:

parameter:	**parameter_block**
description:	The base building block from which the host can inherit common properties
values:	parameter block name
default:	none.
optional:	If this block is not given, the default parameters are assumed.

parameter:	**address**
description:	The address of the host represented as a 40 byte string, the host should have the prefix from the switch to which it is connected and the MAC address in addition to the prefix.
values:	40 byte atm address
optional:	Mandatory

A.6 Load Information

This section defines the traffic characteristics of every host - call information, destinations and the probabilities associated with them.

parameter:	**numdestinations**
description:	To specify the number of destinations the calls are made when the value of destination parameter is explicit.
values:	Integer
default:	0
optional:	Must be specified when destination parameter is *explicit*.

parameter:	**destination**
description:	Type of destination the calls are attempted to: *explicit* the explicit the destinations. If not specified no calls are made
values:	explicit
default:	invalid
optional:	Yes

parameter:	**destinations**
description:	Specifies the explicit destinations or ould be the keyword uniform_any in which case calls are made to all hosts in the network with equal probability
values:	names of the destination hosts which must separated by blank spaces
default:	Not applicable.
optional:	Must be specified if the value of the destination parameter is *explicit*.

parameter:	**destn_prob**
description:	Specifies the probabilities with which calls have to be made to the explicit destinations.
values:	floating point probabilities of the destination hosts which must separated by blank spaces
default:	Not applicable.
optional:	Must be specified if the value of the destination parameter is *explicit*.

Apart from these parameters, the load also contains information like the arrival and suration distribution of the calls, their mean, low and high values etc.

A.7 Connectivity Information

In connectivity information, the two sides of the connections consist of node-node connections but need not, consist of host to node connections. For each connection, port information can be provided as chronicled in section A.3 above.

Available port parameters:

- *bw*
- *delay*
- *cdv*
- *clr*
- *ad_weight*

Additional parameters:

parameter:	**numconnections**
description:	The number of connections between the two entities.
values:	Integer greater than or equal to 1
default:	1
optional:	Yes

A.8 Logical Node Information

This section consists of logical nodes at higher levels in the PNNI hierarchy.

parameter:	**level**
description:	The level of the logical node.
values:	Integer value greater than 104
optional:	Mandatory

parameter:	**child**
description:	The child node which the Logical Node represents
values:	The leader name at the immediate lower level
optional:	Mandatory

A.9 Logical Connectivity Information

In logical connectivity information, the two sides of the connections consist of logical nodes.

parameter:	**delay**
description:	The delay of the logical link.
values:	Integer value in seconds
optional:	Yes

parameter:	**numconnections**
description:	The number of connections between the two entities.
values:	Integer greater than or equal to 1
default:	1
optional:	Yes

A.10 Schedule of events

The schedule gives the duration of the experiment and events like nodefail, portfail if any.

parameter:	**duration**
description:	The duration of the simulation.
values:	Integer value in seconds
optional:	Has to be given

parameter:	**seed**
description:	The random seed for the simulation.
values:	Any Integer value
default:	Randomly generated in the simulator
optional:	Yes

parameter:	**nodal_represent**
description:	Simple or Complex representation to be used for nodal aggregation
values:	simple, complex
default:	simple
optional:	Yes

parameter:	**mpg**
description:	Flag for running hierarchical PNNI simulations
values:	true, false
default:	false
optional:	Mandatory for Hierarchical PNNI simulations

parameter:	**node_fail**
description:	This is a flag used to intentionally denote that a particular node has failed for all calls coming into it. Thus all calls passing that node will fail and crankback with node failure as cause. This is useful for crankback and alternate routing DEBUGGING purposes. The Nodename of the failure node is given.
values:	<Nodename>
default:	false
optional:	Yes

parameter:	**link_fail**
description:	This is a flag used to intentionally denote a link to have failed. Thus all calls on that link will crank back with link failure as cause value. This is useful for crankback and alternate routing DEBUGGING purposes. Two Nodenames have to be given, saying the port of Nodename1 connecting to Nodename2 is a failure
values:	<Nodename1> <Nodename2>.
default:	0
optional:	Yes

Appendix B

BNF Grammar

Note that the grammar is completely case-insensitive for all keywords, parameters, and values

```
<CHAR>   ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
             n | o | p | q | r | s | t | u | v | w | x | y | z
<DIGIT>  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<INT>    ::= <DIGIT>+
<REAL>   ::= <INT>*.<INT>
<STRING> ::= < <CHAR> | <INT> >*
<EMPTY>  ::= NULL
<ENTITYNAME> ::= <STRING>
<ARROW> ::= ->

/* * * * * * * * * * * * * * * * * * *
 * Start of a script                 *
 * * * * * * * * * * * * * * * * * * */

<program> ::= <EMPTY> | <itemList> ';' <schedule>

<itemList> ::= <itemList> ';' <item>
             | <item>

/* * * * * * * * * * * * * * * * * * *
 * Start of Schedule Options         *
 * * * * * * * * * * * * * * * * * * */

<schedule> ::= schedule '{' <duration> '}' ';'
             | schedule '{' <duration> ',' <eventList> '}' ';'

<duration> ::= duration '=' <INT>

<eventList> ::= <eventList> ',' <event>
              | <event>

<event> ::= <failureOption>
          | <recoveryOption>
```

```
<failureOption> ::= <nodefailOption>
                  | <portfailOption>
                  | <linkfailOption>

<recoveryOption> ::= <EMPTY>

<nodefailOption> ::= nodefail <ENTITYNAME>

<portfailOption> ::= portfail <ENTITYNAME> <ENTITYNAME>

<linkfailOption> ::= linkfail <ENTITYNAME> <ENTITYNAME>


/* * * * * * * * * * * * * * * * * *
 * Start of Item Options           *
 * * * * * * * * * * * * * * * * * */

<item> ::= <parameterblock>
         | <node>
         | <host>
         | <port>
         | <load>
         | <physicalhosts>
         | <connection>

<parameterblock> ::= parameter_block node <ENTITYNAME>
                     '{' <genericNodeOptions> '}'
                   | parameter_block host <ENTITYNAME>
                     '{' <genericHostOptions> '}'

<node> ::= node <ENTITYNAME> '{' <nodeOptions> '}'

<host> ::= host <ENTITYNAME> '{' <hostOptions> '}'

<port> ::= port genericport  '{' <portOptions> '}'

<load> ::= load <ENTITYNAME> '{' <loadOptions> '}'

<physicalhosts> ::= <PHYSICAL_HOSTS> '=' hostpool '(' <machineOptions> ')'

<connection> ::= connection <ENTITYNAME> <ARROW> <ENTITYNAME>
                 '{' <connectionOptions> '}'
               | connection <ENTITYNAME> <ARROW <ENTITYNAME>

/* * * * * * * * * * * * * * * * * *
 * Start of Generic Node Options   *
 * * * * * * * * * * * * * * * * * */
```

```
<genericNodeOptions> ::= <genericNodeOptions> , <genericNodeOption>
                       | <genericNodeOption>

<genericNodeOption> ::= <ctrlOption>
                      | <calltraceOption>
                      | <fabricOption>
                      | <linkOption>
                      | <routingOption>
                      | <floodingSignificantOption>
                      | <floodingThresholdOption>
                      | <propconstantOption>
                      | <defaultFloodingPeriodOption>
                      | <defaultFloodingfactorOption>
                      | <acacPolicyOption>
                      | <numportsOption>
                      | <utillogPeriodOption>
                      | <crankbackRetriesOption>
                      | <queueSizeOption>
                      | <helloTimeOption>
                      | <helloInactivityOption>
                      | <summaryTimerOption>
                      | <ptspTimerOption>
                      | <ackTimerOption>
                      | <requesTimerOption>
                      | <convergenceOption>
                      | <ProcessTimeOption>
                      | <searchModeOption>

<nodeOptions> ::= <nodeOptions> ’,’ <nodeOption>
                | <nodeOption>

<nodeOption> ::= <nodeblockOption>
               | <searchModeOption>
               | <levelOption>
               | <peergroupidOption>
               | <addressOption>
               | <idOption>
               | <calltraceOption>
               | <fabricOption>
               | <routingOption>
               | <floodingSignificantOption>
               | <floodingThresholdOption>
               | <propconstantOption>
               | <defaultFloodingPeriodOption>
               | <defaultFloodingfactorOption>
               | <acacPolicyOption>
               | <numportsOption>
               | <utillogPeriodOption>
               | <crankbackRetriesOption>
               | <queueSizeOption>
               | <helloTimeOption>
               | <helloInactivityOption>
               | <summaryTimerOption>
               | <ptspTimerOption>
               | <ackTimerOption>
               | <requesTimerOption>
               | <nodeWorkstationOption>
               | <fabricvciOption>
               | <ProcessTimeOption>

<nodeblockOption> ::= parameter_block <ENTITYNAME>

<searchModeOption> ::= search_mode '=' dijkstra
                     | search_mode '=' dfs_walk

<levelOption> ::= level '=' <INT>

<peergroupidOption> ::= peergroupid '=' <INT>

<addressOption> ::= address '=' <ATMADDRESS>

<idOption> ::= id '=' <INT>

<ATMADDRESS> ::= '0x'<STRING>*

<calltraceOption> ::= calltrace '=' true
                    | calltrace '=' true

<ctrlOption> ::= ctrl = true
               | ctrl = false

<fabricOption> ::= fabric = generic
                 | fabric = plaid
                 | fabric = ku

<linkOption> ::= link = tcp
               | link = atm_svc
               | link = atm_pvc
               | link = pass_thru

<routingOption> ::= routing_policy = min_hop
                  | routing_policy = max_bw
                  | routing_policy = min_delay
                  | routing_policy = min_bw
                  | routing_policy = min_adw
                  | routing_policy = shortest_widest
                  | routing_policy = widest_shortest
                  | routing_policy = widest_min_hop
                  | routing_policy = min_hop_widest
                  | routing_policy = min_adw_widest
                  | routing_policy = min_adw_shortest
                  | routing_policy = shortest_min_adw
                  | routing_policy = min_adw_widest_shortest
                  | routing_policy = widest_shortest_min_hop
                  | routing_policy = shortest_widest_min_hop
                  | routing_policy = max_bw_dfswalk
                  | routing_policy = min_hop_dfswalk
                  | routing_policy = min_time_dfswalk

<floodingSignificanceOption> ::= flooding_significance = explicit_threshold
                               | flooding_significance = dynamic_threshold

<floodingThresholdOption> ::= flooding_threshold = <INT>

<propconstantOption> ::= prop_constant = <INT>

<defaultFloodingPeriodOption> ::= default_flooding_period = <INT>

<defaultFloodingFactorOption> ::= default_flooding_factor = <INT>

<acacPolicyOption> ::= acac_policy = load_balancing
             | acac_policy = call_packing

<numportsOption> ::= numports = <INT>

<utillogPeriodOption> ::= utiln_log_period = <INT>

<crankbackRetriesOption> ::= crankback_retries = <INT>

<durationOption> ::= duration = <INT>

<queueSizeOption> ::= queuesize = <INT>

<helloTimerOption> ::= hello_timer = <INT>

<helloInactivityOption> ::= hello_inactivity_factor = <INT>

<summaryTimerOption> ::= summary_timer = <INT>

<ptspTimerOption> ::= ptsp_timer = <INT>

<ackTimerOption> ::= ack_timer = <INT>
```

```
<requestTimerOption> ::= request_timer = <INT>

<edgeNodesOption> ::= edge_nodes = <INT>

<coresNodeOption> ::= core_nodes = <INT>

<convergenceOption> ::= convergence = <INT>

<ProcessTimeOption> ::= process_time = <REAL>

<nodeWorkstationOption> ::= workstation = <ENTITYNAME>

<fabricvciOption> ::= fabric_vci = <INT>


/* * * * * * * * * * * * * * * * * *
 * Start of Host Parameter Options *
 * * * * * * * * * * * * * * * * * */

<genericHostOptions> ::= <genericHostOptions> ',' <genericHostOption>
                       | <genericHostOption>

<genericHostOption> ::= <callsOption>
                      | <hostCtrlOption>
                      | <hostDurationOption>
                      | <hostqueueSizeOption>
                      | <hostProcessOption>
                      | <hostLinkOption>
                      | <callTypeOption>
                      | <arrivalDistributionOption>
                      | <durationDistributionOption>
                      | <destinationOption>
                      | <sourceTypeOption>
                      | <arrivalPeriodOption>
                      | <arrivalLowOption>
                      | <arrivalHighOption>
                      | <arrivalMeanOption>
                      | <durationPeriodOption>
                      | <dutationLowOption>
                      | <durationHighOption>
                      | <durationMeanOption>
                      | <numDestinationsOption>
                      | <destinationsOption>
                      | <numSourcesOption>
                      | <multipleSourceMBSOption>
                      | <multipleSourceSCROption>
                      | <multipleSourcePCROption>
                      | <multipleSourceBWOption>
```

```
                        | <multipleSourceCTDOption>
                        | <multipleSourceCLROption>
                        | <multipleSourceCDVOption>
                        | <multipleSourcePercentOption>
                        | <multipleSourceDurationTypeOption>
                        | <multipleSourceDurationPeriodOption>
                        | <multipleSourceDurationLowOption>
                        | <multipleSourceDurationHighOption>
                        | <multipleSourceDurationMeanOption>
                        | <hostSCROption>
                        | <hostPCROption>
                        | <hostMBSOption>
                        | <hostBWOption>
                        | <hostCTDOption>
                        | <hostCLROption>
                        | <hostCDVOption>

<hostOptions> ::= <hostOptions> ',' <hostOption>
                | <hostOption>


<hostOption> ::= <hostblockOption>
               | <hostaddressOption>
               | <hostidOption>
               | <hostWorkstationOption>
               | <hostqueueSizeOption>
               | <sourceTypeOption>
               | <numSourcesOption>
               | <multipleSourceMBSOption>
               | <multipleSourceSCROption>
               | <multipleSourcePCROption>
               | <multipleSourceBWOption>
               | <multipleSourceCTDOption>
               | <multipleSourceCLROption>
               | <multipleSourceCDVOption>
               | <multipleSourcePercentOption>
               | <multipleSourceDurationTypeOption>
               | <multipleSourceDurationPeriodOption>
               | <multipleSourceDurationLowOption>
               | <multipleSourceDurationHighOption>
               | <multipleSourceDurationMeanOption>


<hostblockOption> ::= parameter_block <ENTITYNAME>

<hostaddressOption> ::= <ADDRESS> '=' <ATMADDRESS>
<hostidOption>  ::= <SWITCHID> '=' <INT>
```

```
<hostWorkstationOption> ::= workstation = <SERVERADDR>


<hostCtrlOption> ::= ctrl = true
                   | ctrl = false

<hostDurationOption> ::= duration = <INT>

<hostqueueSizeOption> ::= queuesize = <INT>

<hostProcessOption> ::= host_process_time = <REAL>
                      | host_process_time = <INT>

<hostLinkOption> ::= link = tcp
                   | link = atm_svc
                   | link = atm_pvc
                   | link = pass_thru


<destinationOption> ::= destination = fixed
                      | destination = range
                      | destination = explicit

<sourceTypeOption> ::= sourcetype = single
             | sourcetype = multiple


<numDestinationsOption> ::= numdestinations = <INT>

<destinationsOption> ::= destinations ’=’ ’[’ <stringlist> ’]’
                       | destinations ’=’ uniform_any

<stringlist> ::= <stringlist> <ENTITYNAME>
               | <ENTITYNAME>

<integerList> ::=  <INT> <INT>*

<numSourcesOption> ::= numsources = <INT>

<multipleSourceDurationTypeOption> ::= distribution<INT> = poisson
                                     | distribution<INT> = fixed
                                     | distribution<INT> = uniform

<multipleSourceBWOption> ::= bw<INT> = <INT>

<multipleSourceCTDOption> ::= ctd<INT> = <INT>

<multipleSourceCDVOption> ::= cdv<INT> = <INT>
```

```
<multipleSourceCLROption> ::= clr<INT> = <INT>

<multiplePercentOption> ::= share<INT> = <INT>

<multipleSourceDurationPeriodOption> ::= duration_period<INT> = <INT>

<multipleSourceDurationLowOption> ::= duration_low<INT> = <INT>

<multipleSourceDurationHighOption> ::= duration_high<INT> = <INT>

<multipleSourceDurationMeanOption> ::= duration_mean<INT> = <INT>

<hostBWOption> ::= <BANDWIDTH> = <INT>
<BANDWIDTH> ::= bw | bandwidth

<hostCTDOption> ::= ctd = <INT>

<hostCDVOption> ::= cdv = <INT>

<hostCLROption> ::= clr = <INT>

<port> ::= port = genericport (<portOption> <, <portOption>>*)

<portOption> ::= <bwOption>
               | <delayOption>
               | <cdvOption>
               | <clrOption>
               | <adweightOption>

<bwOption> ::= <BANDWIDTH> = OC3
             | <BANDWIDTH> = DS3
             | <BANDWIDTH> = OC12
             | <BANDWIDTH> = T1
             | <BANDWIDTH> = DS1
             | <BANDWIDTH> = <INT>

<delayOption> ::= delay = <INT>

<cdvOption> ::= cdv = <INT>

<clrOption> ::= clr = <INT>

<adweightOption> ::= as_weight = <INT>



/* * * * * * * * * * * * * * * * * *
```

```
 * Start of Load Parameter         *
 * * * * * * * * * * * * * * * * * */

<loadOptions> ::= <loadOptions> ’,’ <loadOption>
                | <loadOption>

<loadOption> ::= <callsOption>
               | <callTypeOption>
               | <arrivalDistributionOption>
               | <durationDistributionOption>
               | <arrivalPeriodOption>
               | <arrivalLowOption>
               | <arrivalHighOption>
               | <arrivalMeanOption>
               | <durationPeriodOption>
               | <durationLowOption>
               | <durationHighOption>
               | <durationMeanOption>
               | <hostMBSOption>
               | <hostSCROption>
               | <hostPCROption>
               | <hostCallBWDistOption>
               | <hostCTDOption>
               | <hostCLROption>
               | <hostCDVOption>
               | <destinationOption>
               | <numDestinationsOption>
               | <destinationsOption>
               | <destn_probOption>

<callsOption> ::= calls = <INT>
                | calls = unspecified
<callTypeOption> ::= calltype = cbr
                   | calltype = rtvbr
                   | calltype = vbr
                   | calltype = abr
                   | calltype = ubr

<arrivalDistributionOption> ::= arrival_distribution = periodic
                              | arrival_distribution = poisson
                              | arrival_distribution = bursty
                              | arrival_distribution = explicit
                              | arrival_distribution = tear_down
                              | arrival_distribution = uniform
<durationDistributionOption> ::= duration_distribution = periodic
                               | duration_distribution = poisson
                               | duration_distribution = uniform
```

```
<arrivalPeriodOption> ::= arrival_period = <REAL>
                        | arrival_period = <INT>

<arrivalLowOption> ::= arrival_low = <REAL>
                     | arrival_low = <INT>

<arrivalHighOption> ::= arrival_high = <REAL>
                      | arrival_high = <INT>

<arrivalMeanOption> ::= arrival_mean = <REAL>
                      | arrival_mean = <INT>

<durationPeriodOption> ::= duration_period = <INT>

<durationLowOption> ::= duration_low = <INT>

<durationHighOption> ::= duration_high = <INT>

<durationMeanOption> ::= duration_mean = <INT>

<hostMBSOption> ::= mbs ’=’ <INT>

<hostSCROption> ::= scr ’=’ <INT>

<hostPCROption> ::= pcr ’=’ <INT>

<hostCallBWDistOption> ::= call_bw ’=’ ’[’ poisson <INT> ’]’
                         | call_bw ’=’ ’[’ uniform <INT> <INT> ’]’
                         | call_bw ’=’ ’[’ fixed <INT> ’]’

<destn_probOption> ::= destn_prob ’=’ ’[’ <realList> ’]’

<realList> ::= <realList> <REAL>
             | <REAL>


/* * * * * * * * * * * * * * * * * *
 * Start of Machine Options        *
 * * * * * * * * * * * * * * * * * */


<machineOptions> ::= <machineOptions> ’,’ <machineOption>
                   | <machineOption>

<machineOption> ::= <ENTITYNAME>


/* * * * * * * * * * * * * * * * * *
```

```
 * Start of Connection Options     *
 * * * * * * * * * * * * * * * * * */


<connectionOptions> ::= <connectionOptions> ',' <connectionOption>
                      | <connectionOption>

<connectionOption> ::= <portOption>
                     | <numConnectionsOption>
                     | <leftvciOption>
                     | <rightvciOption>
                     | <leftportOption>
                     | <rightportOption>
                     | <serverOption>

<numConnectionsOption> ::= numconnections = <INT>

<leftvciOption> ::= leftvci = <INT>

<rightvciOption> ::= rightvci = <INT>

<leftportOption> ::= leftport = <INT>

<rightportOption> ::= rightport = <INT>

<serverOption> ::= server = <ENTITYNAME>
```