The University of Kansas

Information and Telecommunication Technology Center

Technical Report of the
Networking and Distributed systems Laboratory

A KU-PNNI Test Suite Manual Version 2.0

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

ITTC-FY2001-TR-18833-12

October 2000

Project Sponsor:
Sprint Corporation

Contents

List of Figures

List of Scripts

Chapter 1

Introduction

The KU PNNI Simulator Test Suite introduces a set of tests several network scenarios for testing the KU PNNI simulator. The purpose of this test suite is to determine the correct function and to evaluate the performance of the KU PNNI simulator features. The test suite manual also guides a user of the simulator to successfully perform an experiment with a network topology.

This document is also designed to make the user familiar with the abilities, limitations, and performance of the KU PNNI simulator. The different test scenarios are provided. These test scenarios will guide a user who wishes to create a custom test scenario and shows examples of how to run the KU PNNI simulator. Chapter 2 provides a set of test cases that determine the correctness of the KU PNNI simulator for single peer group networks. The set of tests that are used to evaluate the performance of the KU PNNI simulator for single peer group are described in Chapter 3. Chapter 4 decribes the correctness and Chapter 5 the performance tests for multiple peer group scenarios and features of the simulator.

Chapter 2

Single Peer Group Correctness Tests

The KU PNNI simulator possesses many features that perform different tasks. Therefore, we created test cases for each function of the KU PNNI Simulator. Section 2.1 describes the test cases in the scenario where multiple hosts are connected to a switch. Routing test cases are given in Section 2.2. This section uses a variety of network topologies to verify the accuracy of the routing algorithm. Section 2.3 considers test cases used to determine the correctness of alternate routing when a crankback occurs in the network. This section also gives different network topologies to strengthen the correctness testing of the KU PNNI simulator. Advanced feature test cases for *Multiple QoS Routing Algorithm* of the KU PNNI simulator are given in Section 2.4. The directory structure of the test suite is shown in Figure 2.1.

2.1 Multiple Destination Tests

To test the feature of multiple hosts connected to a node. Calls can be made to multiple destinations with different call probabilities. As a special case of the above, providing the UNIFORM_ANY option call all the hosts with uniform probability, which means that all calls are uniformly distributed among the given destinations. A sample script is shown in Script 2.1.

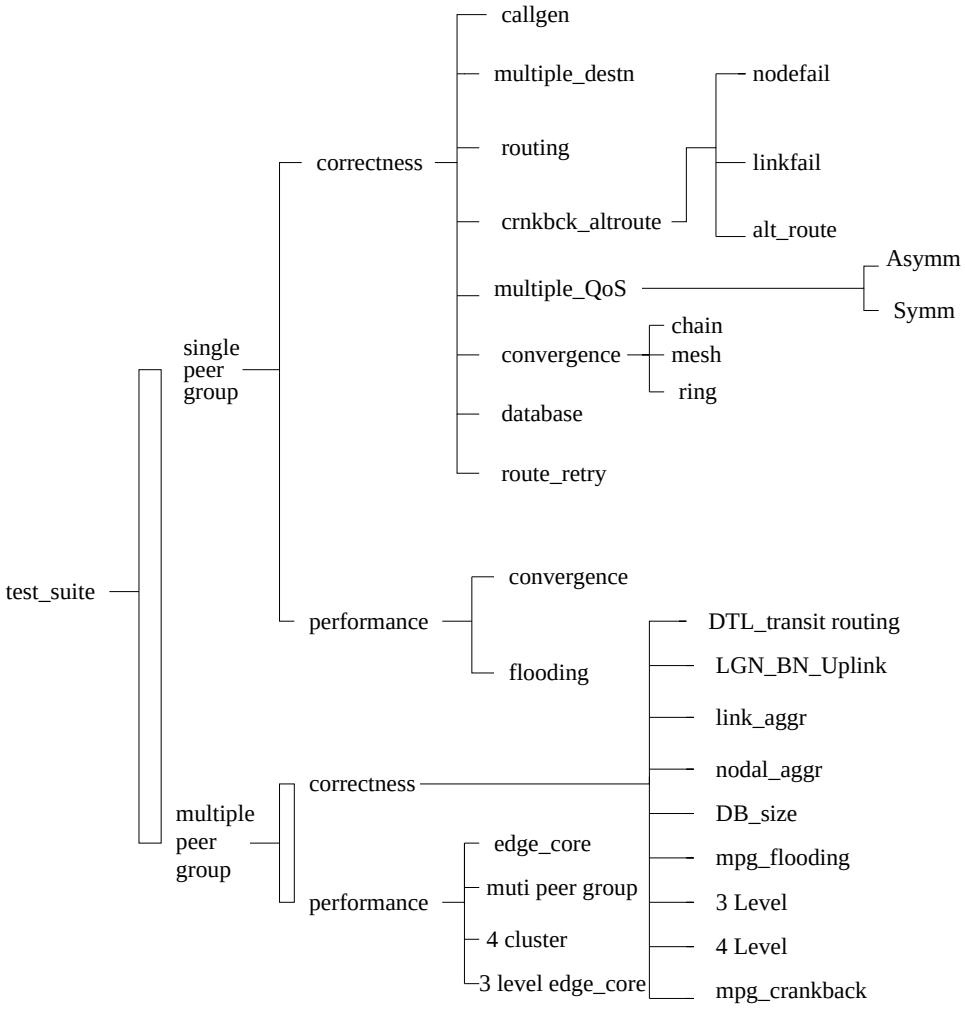


Figure 2.1: Test suite directory structure

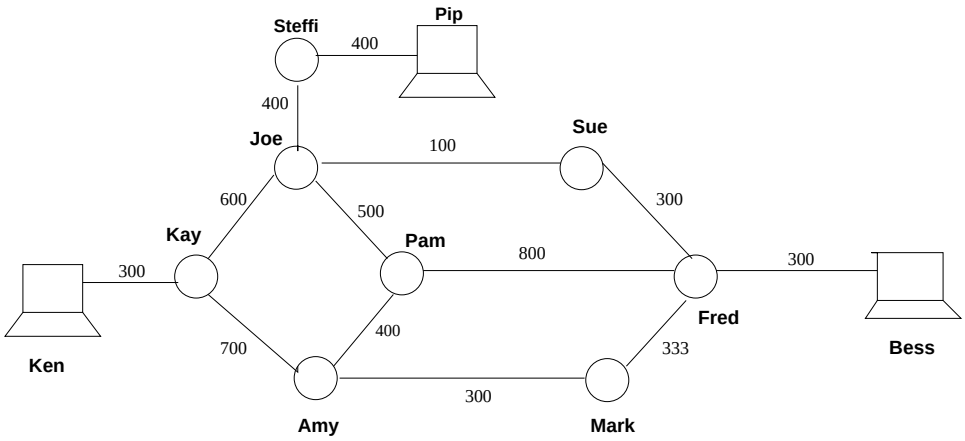
Script 2.1 Sample of a part of the script that shows how to call multiple destinations with different call probabilities

```
host Ken{
        parameter_block newton,
        address = 0x4705ffef56000000000000001100ec301121aa00
        };

host Bess{
        parameter_block newton,
        address = 0x4705ffef56000000000000002200ec301100bb00
        };

host Pip{
        parameter_block stephens,
        address = 0x4705ffef56000000000000008800ec301100cc00
        };

load Ken{
        calltype        = cbr,
        arrival_period  = 5,
        duration_period = 10,
        calls           = 10,
        numdestinations = 2,
        destinations    = [Bess Pip],
        destn_prob      = [0.75 0.25]
        };
```



Note: The numbers printed near each link represent the bandwidth of that link.

Figure 2.2: Multihost Topology

In Script 2.1, the host named 'Ken' makes ten CBR-type calls to hosts 'Bess' and 'Pip'. Host 'Bess' is called with a probability of 0.75 and host 'Pip' is called with a probability of 0.25. The calls are periodically generated. There is a 5 second interval between calls and the duration of each call is 10 seconds.

2.2 Single QoS Routing Algorithm Tests

In this directory, there are tests for various routing policies. The routing policies in our simulator are:

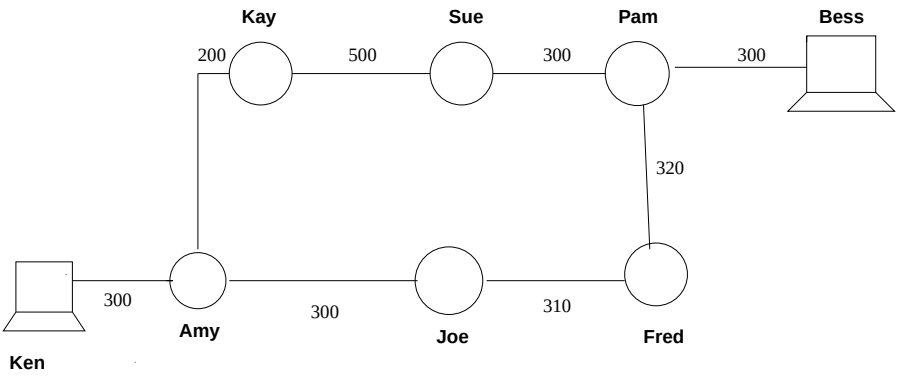
- minimum hop count (min_hop)
- maximum bandwidth (max_bw)
- minimum administrative weight (min_adw)
- minimum delay (min_delay)

The single QoS routing algorithms are tested in different network topologies as described in the following sections.

2.2.1 2-path Topology

The basic idea of this scenario is to have only two choices that can be returned from the routing algorithm. From this scenario, we can easily check that the routing model returns a path that fulfills the routing policy and call requirements. The 2-path topology is shown in Figure 2.3.

If the Routing Policy is chosen as maximum bandwidth and if calls are made from Ken to Bess, then the routing algorithm should return the path to be Ken - Amy - Joe - Fred - Pam - Bess.



Note: The numbers represent the bandwidth of that particular link

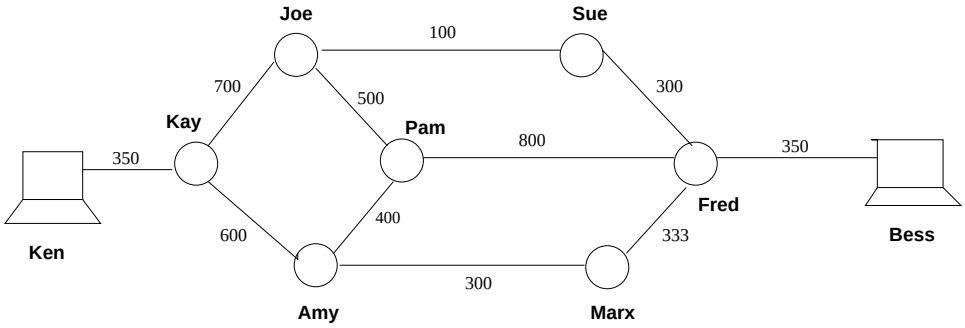
Figure 2.3: 2-Path Topology

2.2.2 Pyramid Topology

The Pyramid topology is created to verify the routing algorithm when we have the same number of sources and destinations in the topology as in the previous case but there are more links and switches between source and destination.

The Pyramid network topology is created from the 2-path network topology by adding more links between switches, and is given in Figure 2.4.

If the Routing Policy is chosen as maximum bandwidth and if calls are made from Ken to Bess, then the routing algorithm should return the path to be Ken - Kay - Joe - Pam - Fred - Bess



Note: The numbers printed near each link represent the bandwidth of that link.

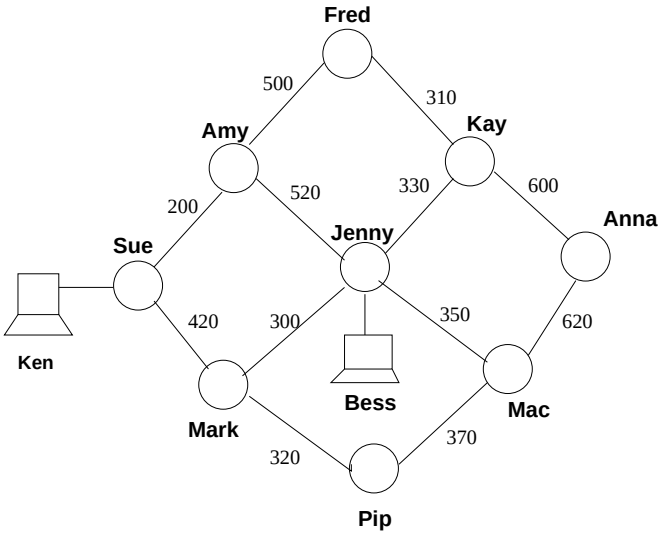
Figure 2.4: Pyramid Topology

2.2.3 Diamond Topology

The idea of the *Diamond* topology is to determine how well the routing algorithm performs when there are many paths inside a topology. This network topology has more links and more

switches which means that there are many routes from the source to the destination. The *Diamond* topology is given in Figure 2.5.

If the Routing Policy is chosen as maximum bandwidth and if calls are made from Ken to Bess, then the routing algorithm should return the path to be Ken - Sue - Mark - Pip - Mac - Anna - Kay - Jenny - Bess



Note: The numbers printed near each link represent the bandwidth of that link.

Figure 2.5: Diamond Topology

2.3 Crankback & Alternate Routing Tests

In this section, we introduce the test cases to show that the crankback mechanism and the alternate routing function of the KU PNNI simulator. In addition, we introduce user controlled options to force a node, port, or link to fail at the time of Call Admission Control (CAC) in order to emulate a realistic test scenario.

In Section 2.3.1, we test the user controlled option to force a *node* (switch) in the network to fail. This means the node will refuse all call connections through itself. This option is called *nodefail*. Section 2.3.2 introduces another user controlled option to force a particular *port* of the link at a node (switch) to have failed. This means that a call made to a particular destination will fail, but a call made to other nodes connected to this node will be successful. This option is called *portfail*. Furthermore, forcing a link to fail can be performed by forcing the ports at each end of the link to have failed. Note that the failure is discovered by the mechanism of Call Admission Control (CAC) performed at each node.

We also create more test cases with different network topologies to prove the robustness of our KU PNNI simulator handling the failure of node, port, or link in a network. More test cases are described in Section 2.3.3 and Section 2.3.4

2.3.1 Node Failure in the Network

This section determines the correctness of the routing algorithm when there is a *node* (switch) failure in the network. The scenario is that when there is a node failure in the network, a crankback will occur and the routing algorithm needs to re-route again using the failure information given by the crankback. The idea of this test scenario is to assert that the crankback mechanism functions properly and the route given by the routing algorithm (alternate routing) is valid when there is a failed node (or switch) in the network.

A test scenario is that there are two source nodes trying to make calls to the same destination node. The result is different depending on which node has failed. The sample topology is given in Figure 2.6.

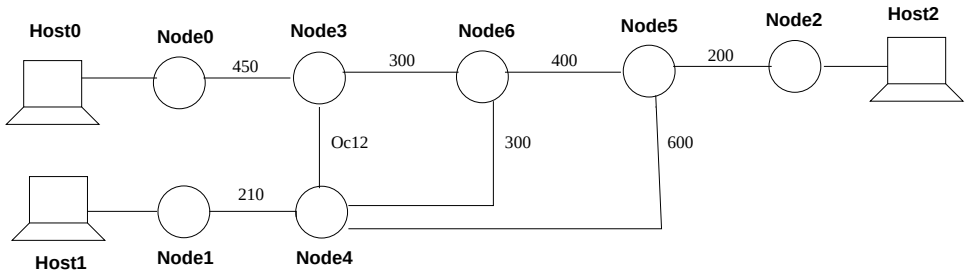


Figure 2.6: Node Failure Test

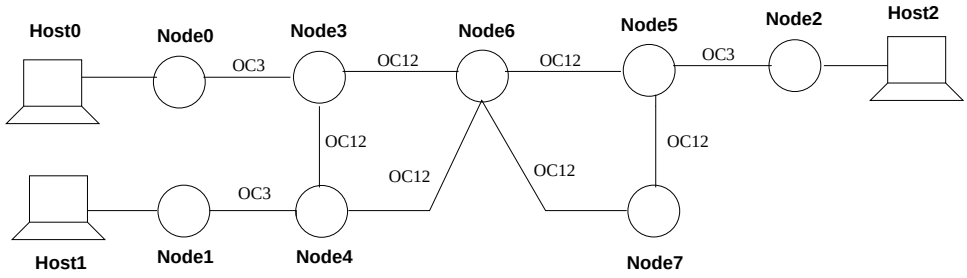
2.3.1.1 Example of Node Failure

When Host0 and Host1 want to call Host2

- When there is no failed node, calls from Host0 go though route Node0–Node3–Node6–Node5–Node2, and calls from Host1 go though route Node1–Node4–Node5–Node2. Note that the route depends on the routing policy.
- If node Node2 has failed, all calls will fail.
- If node Node3 has failed, all calls from Host0 will fail because there is no route to the destination, and all calls from Node1 will be successful.
- If node Node6 has failed, calls from Host1 will be successful, and calls from Host0 will go though an alternate route since node Node6 has failed. When Node6 has failed, Node0 receives a crankback element saying that node Node6 has failed. Then, Node0 prunes all links connected to node Node6 in its representation of the network used for routing and then performs routing to get an alternate route.

2.3.2 Port/Link Failure in the Network

This section determines the correctness of a routing algorithm when there is a *port* failure in a link in the network. The scenario is that when there is a port failure, a crankback will happen and the routing algorithm needs to re-route again using the failure information given by the crankback. The idea of this test scenario is to assert that the crankback mechanism functions properly and the route given by the routing algorithm (alternate routing) is valid when there is a failed port of a link in the network. Note that a failed link occurs when the ports at either end of the link have failed. The test case is that there are two source nodes trying to make calls to the same destination node. The result is varied depending on which port in the link has failed. The sample topology is given in Figure 2.7.



Note: The figures printed on each link represent the bandwidth of that link.

Figure 2.7: Port/Link Failure Test

2.3.2.1 Example of Port/Link Failure

When there is no failed link, calls from Host0 go though route Node0–Node3–Node6–Node5–Node2, and calls from Host1 go though route Node1–Node4–Node6–Node5–Node2. Note that the route depends on the routing policy.

- If the link between Node2 and Node5 fails, all calls will fail.

- If the link between Node6 and Node5 fails, all calls from Host0 and Host1 will go through another route. Calls from both Node0 and Node1 go through the link Node6-Node5. When this link fails, the crankback procedure occurs saying that link Node6-Node5 has failed. Then Node0 finds a new route for calls from Host0 and Node1 also finds a new route for calls from Host1. The 'detour' link from node Node6 to Node5 is link Node6-Node7 and Node7-Node5.

- If link Node3-Node6 fails, calls from Node0 will get crankback element back to Node0, and Node0 finds the new route to be Node0–Node3–Node4–Node6–Node5–Node2. However, calls from Node1 will be successful.

2.3.3 Alternate Routing Test I

This section introduces more test cases with a different topology to strengthen our tests. This test case is the simplest case to perform crankback mechanism and alternate routing. The case is that there are only two paths available in the network and one path has failed.

A test scenario is that there is one source node trying to make calls to one destination node. A different result is got depending on which node has failed. The sample topology is given in Figure 2.8.

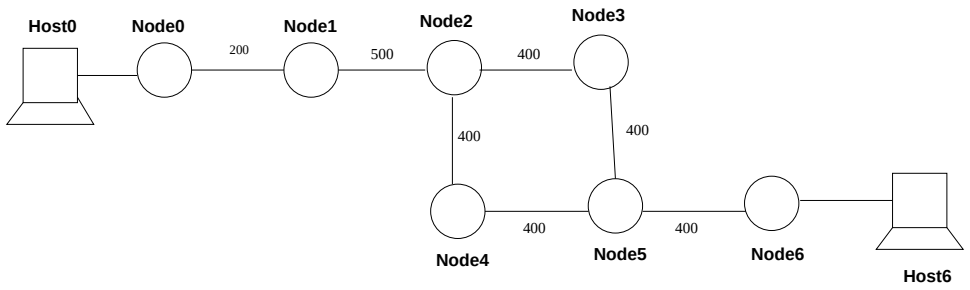


Figure 2.8: Alternate Routing Test I

2.3.3.1 Example of Alternate Routing Test I

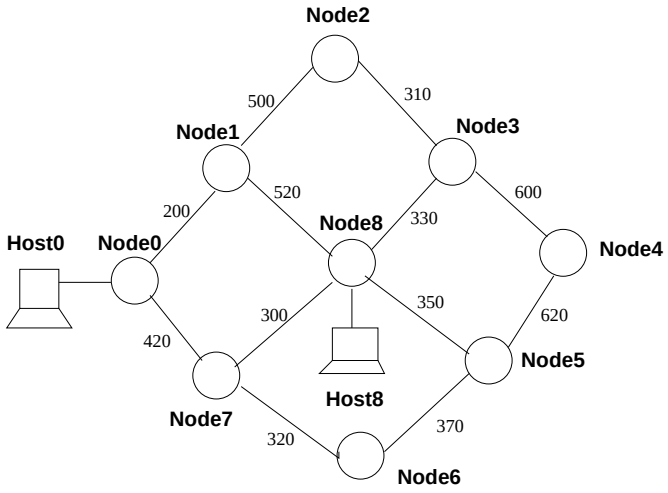
When there is no failed node, calls from Host0 go though route Node0–Node1–Node2–Node3–Node5–Node6. Note that the route depends on the routing policy.

- If the link between Node2 and Node3 fails, calls will go through an alternate route given by Node0–Node1–Node2–Node4–Node5–Node6.

- If node Node3 has failed, all calls will go through an alternate route such as Node0–Node1–Node2–Node4–Node5–Node6.

2.3.4 Alternate Routing Test II

This section introduces more test cases with a different topology to strengthen our testing. This test case is more complicated in performing crankback mechanism and alternate routing. The case is that there are many possible routes from a source to a destination in the network and



Note: The numbers printed on each link represent the bandwidth of that link.

Figure 2.9: Alternate Routing Test II

one of them has failed. The purpose of this test case is to show that the routing algorithm still returns the best route to a destination node even though there is a failed route in the network.

A test scenario is that there is one source node trying to make calls to one destination node. The result is different depending on which node has failed. The sample topology is given in Figure 2.9.

2.3.4.1 Example of Alternate Routing Test II

When there is no failed node, calls from Host0 go, for example, through route Node0–Node7–Node6–Node5–Node8. Note that the route depends on a routing policy.

- If the link between Node0 and Node7 fails, Calls will go through an alternate route such as Node0–Node1–Node8.
- If node Node7 has failed, all calls will go through an alternate route such as Node0–Node1–Node8.

2.4 Multiple QoS Routing

To enhance the KU PNNI Simulator features, we have implemented the Multiple QoS Routing which will be able to find a route using more than one routing criteria. Multiple QoS Routing is often more efficient than routing based on one routing criterion. The routing metrics considered when performing Multiple QoS Routing are bandwidth, delay, number of hops and administrative weight. The routing criteria can be as follows:

- double criterion policies which minimum hop count is the primary criterion
  - Minimum number of hops as the primary and maximum bandwidth as the secondary criterion (widest_min_hop)
  - Minimum number of hops as the primary and minimum delay as the secondary criterion (shortest_min_hop)
- double criterion policies which maximum bandwidth is the primary criterion:
  - maximum bandwidth as the primary and minimum delay as the secondary criterion (shortest_widest)
  - maximum bandwidth as the primary and minimum number of hops as the secondary criterion (min_hop_widest)
  - maximum bandwidth as the primary and minimum administrative as the secondary criterion (min_adw_widest)
- double criterion policies which minimum delay is the primary criterion:
  - Minimum delay as the primary and maximum bandwidth as the secondary criterion (widest_shortest)
  - minimum delay as the primary and minimum administrative as the secondary criterion (min_adw_shortest)
  - minimum delay as the primary and minimum hop count as the secondary criterion
  - (min_hop_shortest)
- double criterion policies which minimum administrative weight is the primary criterion:
  - minimum administrative as the primary and minimum delay as the secondary criterion (shortest_min_adw)
- triple criterion policies:
  - minimum delay as the primary, maximum bandwidth as the secondary criterion, and minimum administrative weight as the tertiary criterion (min_adw_widest_shortest)
  - minimum number of hops as the primary, minimum delay as the secondary criterion, and maximum bandwidth as the tertiary criterion (widest_shortest_min_hop)
  - minimum number of hops as the primary, maximum bandwidth as the secondary criterion, and minimum delay as the tertiary criterion (shortest_widest_min_hop)

In case of testing multiple QoS routing with two routing criteria, each link in the network has two metrics which are different from other link metrics. The routing algorithm will route based on the primary metric first. If there are several possible routes which have the same primary metric, the routing algorithm will route based on the secondary metrics. The primary metric and secondary metrics are based on the routing policy selected.

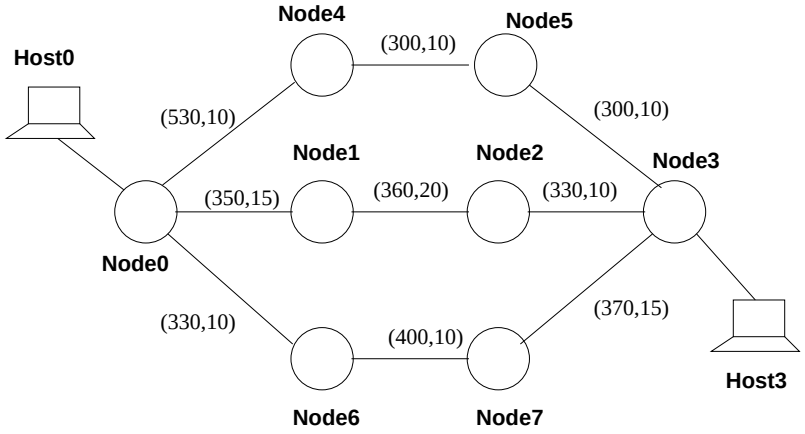
We run experiments based on different routing policies such as widest_shortest, shortest_min_adw, or min_adw_shortest. The details of all the multiple QoS Routing algorithms are in the KU PNNI User Manual.

2.4.1 Sample Scenario

We have tested our multiple QoS routing on two network topologies, Symmetric and Asymmetric networks. These two scenarios are used to test the correctness of the route returned from KU PNNI routing model.

2.4.1.1 Symmetric Network

In Figure 2.10, the test scenario has three disjoint routes. No link in one route participates in other routes. When one link is a bottleneck, the other routes are not affected by the congested link.



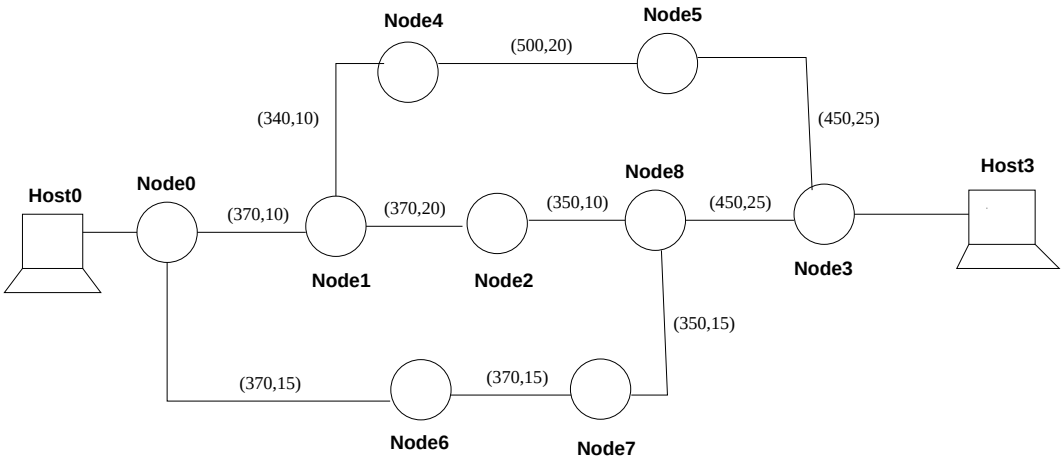
Note: The figures in parentheses represent (bandwidth, delay) of that particular link.

Figure 2.10: Symmetric Network

2.4.1.2 Asymmetric Network

In Figure 2.11, the scenario has three different routes. Some of the routes have a common link such as the link between Node0 and Node1. When the associated link is a bottleneck, two routes fail. For example, when link between Node0 and Node1 is a bottleneck, there is only one route from Host0 to Host3 using link

Node0 –> Node6 –> Node7 –> Node8 –> Node3



Note: The numbers in parentheses represent (bandwith,delay) of that particular link

Figure 2.11: Asymmetric Network

2.5 Call Bandwidth Distribution Tests

To enhance our Call Generator, we implement the bandwidth distribution of a call. The former Call Generator is able to generate calls with fixed bandwidth. It means every call has the same bandwidth requirements. The present Call Generator has a new feature to make a call with a random bandwidth requirement. The bandwidth distribution can be Poisson or Uniform distribution. Also we still have an option of fixed bandwidth requirement for every calls.

To make calls with any bandwidth distribution, you need to specify it in the user script as

```
call_bw = [ distribution_type  distribution_param ]
```

For example, the option to make calls with Poisson bandwidth distribution with bandwidth mean of 128 kbps is shown below.

```
call_bw = [ poisson 128 ]
```

For making call with Uniform bandwidth distribution with the lower bound of 64 kbps and upper bound of 128 kbps, the option is shown below.

```
call_bw = [ uniform 64 128 ]
```

For making call with a fixed bandwidth for every call with the fixed bandwidth of 200 kbps, the option is shown below.

```
call_bw = [ fixed 200 ]
```

In conclusion, we enabled the Call Generator to make calls with the random bandwidth requirements. The bandwidth distribution can be Poisson or Uniform distribution as well as the fixed bandwidth.

2.6 Limitation of Alternate Routing Retries Tests

To enhance our PNNI simulator, we enable the alternate routing mechanism to be able to limit number of route retries, and to handle more cases of errors. This enhancement makes the simulator to be more robust and stable while testing in the high-rate traffic network.

First, we introduce the test case for the alternate routing to handle the error when there is a bottle neck in the network. In Figure 2.12, there are two possible routes from the source (Host0) to destination (Host6). We make the call which requests 200 Mbytes. So the link between Node2 and Node3 can handle only one call, and so can the link between Node4 and Node5. After the simulation starts, call 1 will go through PATH-1.

```
PATH-1 : Node0 --> Node1 --> Node2 --> Node3 --> Node5 --> Node6
```

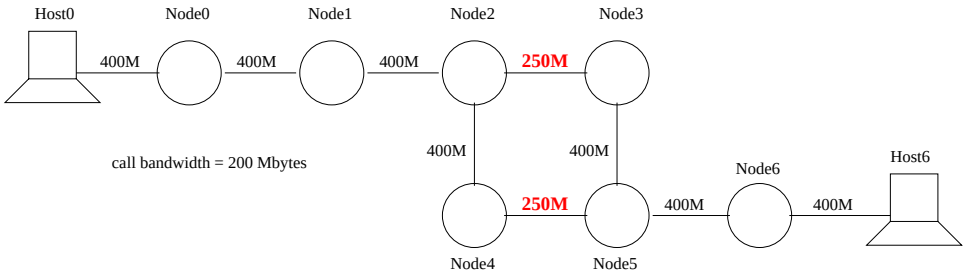


Figure 2.12: Two Route Testing for Alternate Routing

As the updating of the PNNI database is not quick enough, the second call will also go through the PATH-1. However, the call will be failed since the link between Node2 and Node3 will be blocked. The crankback mechanism is invoked. The alternate routing mechanism will be invoked at the source node. At the source node, the bandwidth between Host0 and Node0 as well as Node0 and Node1 will be *deallocated*. The call is sent up to PNNI module to request for an alternate route. Then, PATH-2 is given as an alternate route.

```
PATH-2 : Node0 --> Node1 --> Node2 --> Node4 --> Node5 --> Node6
```

Before sending out the request, the bandwidth between Host0 and Node0 as well as Node0 and Node1 will be *allocated* again as the second times for this call. So the second call will be successful. The later call will fail until the first or second call is finished. To prove the robustness of our simulator, we create more tests. First, we propose the conventional edge-core network topology with 24 nodes and 24 hosts as shown in Figure 2.13.

In the first test, we run the experiment with high bandwidth calls. The call bandwidth is uniformly distributed between 20 Mbytes and 50 Mbytes. Every hosts makes 100 calls, and each call can be retried 10 times if the call fails. Therefore, there will be a total of 2400 high-bandwidth calls. In another test, we increase the number of calls that each hosts makes, but we reduce the bandwidth of the call. The call bandwidth is uniformly distributed between 2 Mbytes and 5 Mbytes. Every hosts makes 500 calls. Two hosts are connected to one edge node. Therefore, one edge node will handle 1000 calls totally, and there will be 12,000 calls in the network. The simulation execution time for this test is quite long, and it also proves the fault-tolerance and robustness of our simulator.

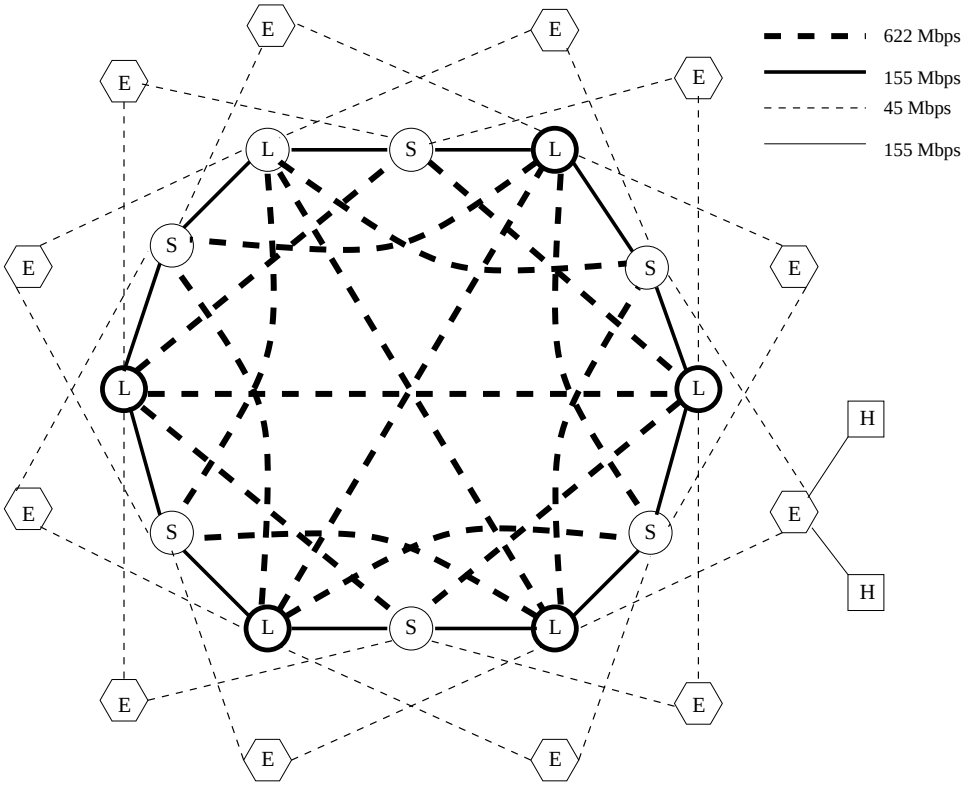


Figure 2.13: Typical Edge-Core Network Topology

Chapter 3

Single Peer Group Performance Tests

This chapter introduces the set of test cases to evaluate the performance of the single peer group KU PNNI Simulator. Section 3.1 describes test cases to evaluate the convergence time which is the time that it takes for all the nodes in the PNNI network to synchronize with initial topology information. Section 3.2 describes test cases for evaluating the influence of the percentage of the change of available cell rate to the flooding of PNNI topology update messages. Section **??** describes the test case for evaluating the effect of the size of the PNNI network to performance metrics.

3.1 Single Peer Group Topology Convergence Tests

Topology convergence time is the time it takes for all the switches in the network to synchronize with the initial topology information generated by each of the participating nodes. The initial topology information generated by each node includes:

- *Nodal Information*: which gives information about the node identifier and peer group of the node along with some node specific characteristics such as leadership priority and others.
- *Horizontal Link Information*: This contains link information regarding bandwidth, delay, and others. Each node advertises all the links to neighboring nodes.

These primary topology information elements are sufficient for routing a call connection request to any other node in the peer group.

Convergence time is an important performance metric as it lets us know how quickly topology information which is flooded could be distributed among the different nodes in the network and help all the nodes in the network have a common view of the network. The convergence time depends upon the following factors:

- The *size* of the network.
- The *the number of the links* in the network.
- The *the number of the nodes* in the network.
- The *processing* capabilities of the nodes in the network.
- The *link delay* between nodes in the network.

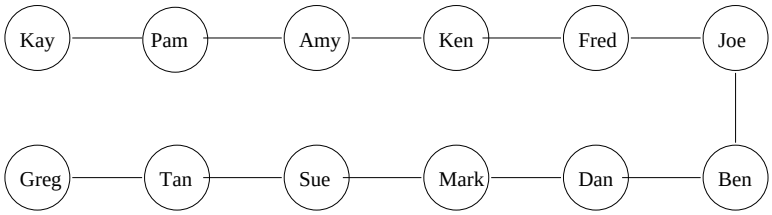


Figure 3.1: Chain Network with 12 nodes and 22 connections

3.1.1 Topologies

This section describes the test cases that we created for testing the convergence time of different topologies.

Different topologies can be constructed and the parameters, "number of nodes", "number of connections" and "link delay" can be changed. In addition, the topologies include the standard chain, ring and mesh topologies. Examples of topologies are described as follows:

- **12node_22conn_delay8_chain**: this topology has 12 nodes, 22 connections among the nodes. Every link has 8 millisecond link delay. The network is a traditional chain topology, and is shown in Figure 3.1.
- **32node_32conn_delay8_ring**: this topology has 32 nodes, 32 connection among the nodes. Every link has 8 millisecond delay. The network is a traditional ring topology, and is shown in Figure 3.2.
- **36node_60conn_delay8_mesh**: this topology has 36 nodes, 60 connection among the nodes. Every link has 8 millisecond delay. The network is a traditional mesh topology, and is shown in Figure 3.3.

These networks do not have hosts and there are no calls made, in other words, it just tests the convergence of the PNNI code of the simulator.

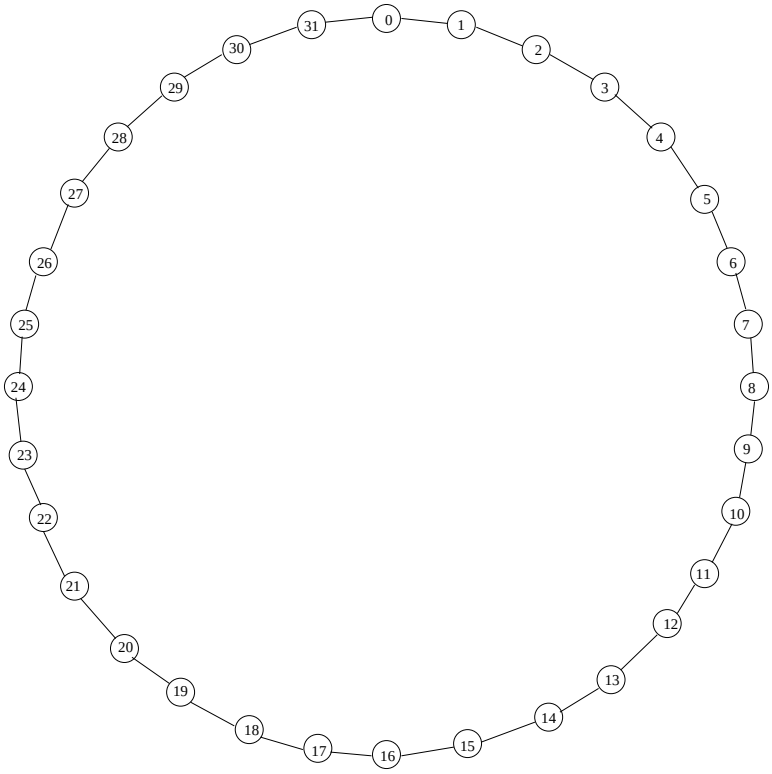


Figure 3.2: Ring Network with 32 nodes and 32 connections

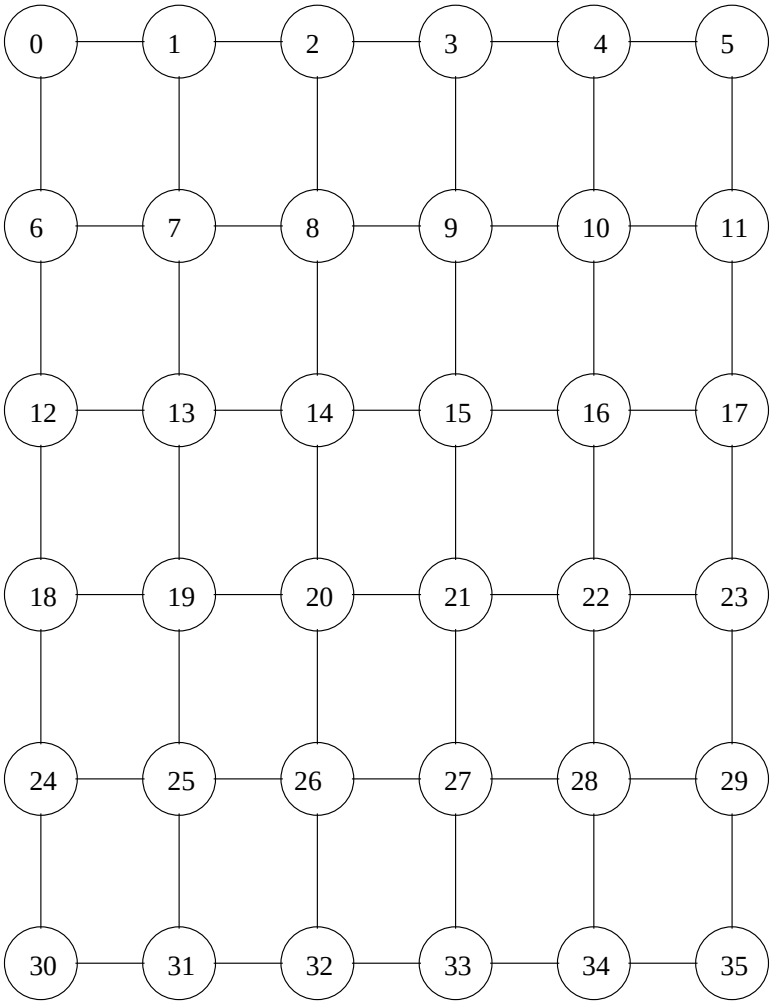


Figure 3.3: Semi-Mesh Network with 36 nodes and 60 connections

3.2 Proportional Multiplier Tests

The proportional multiplier (PM) is the percentage of the last advertised Available Cell Rate (AvCR). The importance of the proportional multiplier is that it indicates the significant change of AvCR of a link in the PNNI topology information. The lower the proportional multiplier, the more significant change of the AvCR in the topology information. If the change in the PNNI topology information is considered to be significant, the PNNI topology information flooding mechanism will start sending a new update information to every node. Note that a lot of update messages sent among nodes can increase the call setup time. In addition, the low proportional multiplier can cause a higher rate of failed calls.

These scripts are for testing the network flooding. The proportional constant can be varied from 25% to 75% and the routing policy is minimum hop, this however could be changed if required. Example of test scripts:

- 45node_prop25_minhop: this topology has 45 nodes. The routing criterion is minimum hop count. The proportional multiplier here is 25.
- 45node_prop50_minhop: this topology has 45 nodes. The routing criterion is minimum hop count. The proportional multiplier here is 50.
- 45node_prop75_minhop: this topology has 45 nodes. The routing criterion is minimum hop count. The proportional multiplier here is 75.

Chapter 4

Multiple Peer Group Correctness Tests

In this chapter, we explain tests of our simulator in PNNI hierarchical networks or multiple peer group (MPG) networks.

4.1 Transit Routing and DTL

This test is used to determine the correctness of transit routing and the DTL in different peer groups.

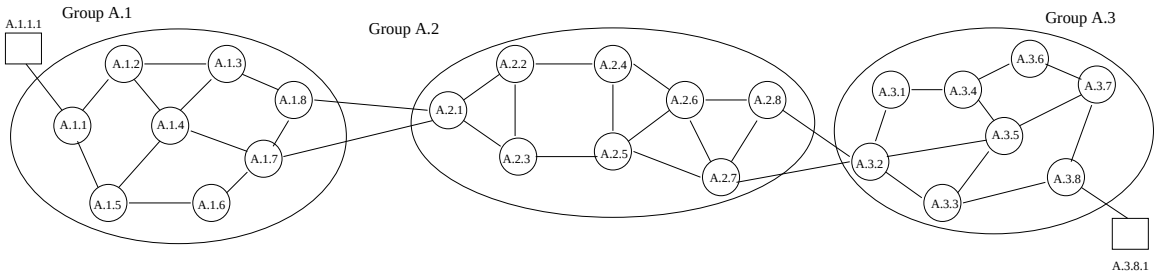


Figure 4.1: DTL and Transit Routing

From Figure 4.1, a call is made from the source host which is connected to Node A.1.1 to the destination host which is connected to Node A.3.8. From this topology, Node A.1.1 does source-routing to find a path from itself to the destination node. The first DTL for the peer group A.1 is created for signaling the call to the second peer group – peer group A.2. After the call arrives in the peer group A.2, a border node in the peer group A.2 (Node A.2.1 from Figure 4.1) will do a transit routing to find the second DTL for signaling the call through its peer group to the destination node connected to the peer group A.3. Then the call will be routed to the peer group A.3. Similarly, a border node in the peer group A.3 (Node A.3.2 from Figure 4.1) will do a transit routing within its peer group to the destination node. Thus, for routing a call in the topology in Figure 4.1, there is one source routing and two transit routing for this topology.

The source routing gives a whole route from a source node to a destination node. However, the route does not have full detail of route inside the other peer groups. When a call with a DTL arrives in the next peer group, the DTL will be expanded by transit routing to signal the call across the peer group. From Figure 4.1, the *possible* DTL from the source routing is [A.1.1 A.1.2 A.1.3 A.1.8 A.2 A.3]

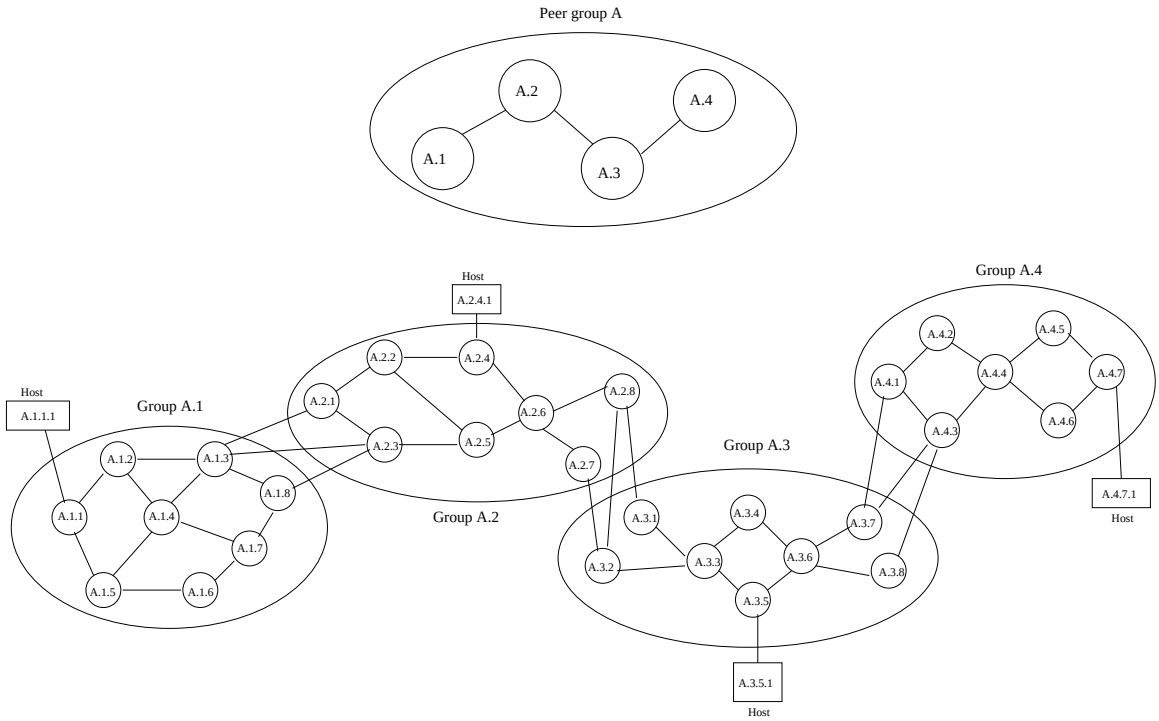


Figure 4.2: Hierarchical Routing

The route across the peer group A.2 and A.3 will be expanded by transit routing in their peer groups when the call arrives in those peer groups.

4.2 Logical Nodes, Border Nodes, Logical Links, and Uplinks

In this section, we test routing in multiple peer group PNNI network with new features for hierarchical networks. The new features involve peer group leader nodes, border nodes, logical links and uplinks. In addition, the peer group leader node creates its representative known as logical group node (LGN) to represent information in the next higher level (logical level).

In hierarchical PNNI routing, the routing information is composed of *physical* and *logical* topology information. Physical topology information consists of nodal PTSEs and horizontal link PTSEs. Logical topology information consists of uplink PTSEs, nodal state PTSEs for Logical nodes, nodal PTSEs, and horizontal link PTSES for logical links. To route a call across the hierarchical network, all this information has to be represented correctly. Therefore, we propose the hierarchical routing tests to prove the correctness of multiple peer group routing with physical and logical information.

We use the topology in Figure 4.2 for our hierarchical routing. There are four peer groups. Calls are routed from peer group A.1 to peer group A.4. The source A.1.1.1 makes calls to the destination A.4.7.1. The source node A.1.1 needs to get enough information from peer group A.2, A.3, and A.4 in order to route calls from A.1.1 to the destination node A.4.7.

4.3 Link Aggregation

This section explains our tests of link aggregation with different aggregation policies. The link aggregation is used to summarize the information of links between two peer groups (outside links). Based on that information, the logical link in the higher level is created. The method of summarizing that outside link information could be different. We have three different aggregation policies, the optimistic, pessimistic, and average, in our simulation tool. In our tests for link aggregation, we use the hierarchical network from Section 4.2.

Optimistic Aggregation

The optimistic aggregation policy summarizes the link information by finding the "best" of all link information among the outside links. Some of the link informations are bandwidth, delay, and administrative weight. We also call link informations as costs of a link. The "best" of costs of links can be interpreted in two different ways, maximum and minimum. For a *metric* parameter, the best of the link cost is the *minimum* cost of all outside links for aggregation. For example, the "best" of the cost such as link delay is the *minimum* delay of all outside links between two peer groups. In contrast, for an *attribute* parameter, the best of the cost of the link is the *maximum* cost of all links between two peer groups. For example, the "best" of the link bandwidth is the *maximum* bandwidth of all links between two peer groups. The example of optimistic link aggregation for bandwidth values is shown in Figure 4.3.

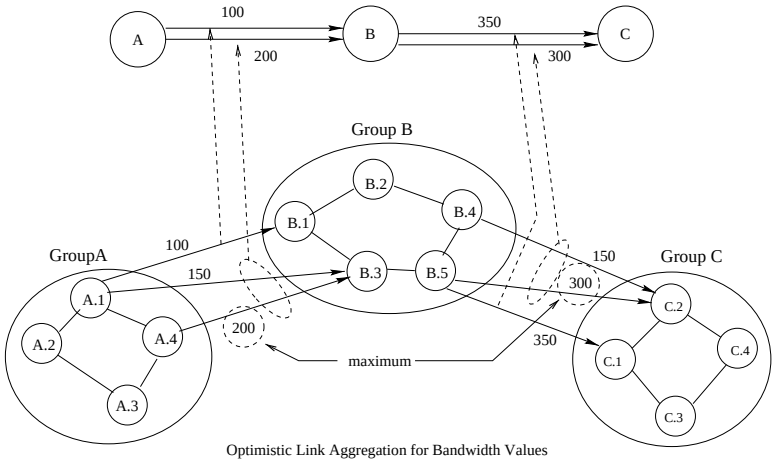


Figure 4.3: The Optimistic Link Aggregation for Bandwidth Values

This is an aggressive algorithm because it probably selects all the best QoS values from different links. This means there may be no physical links which can simultaneously support all the advertised QoS values. This algorithm tends to give the better network utilization, but crankback probably occurs frequently in the network.

Pessimistic Aggregation

The pessimistic aggregation policy summarizes the link information (or link QoS parameters) by finding the "worst" of all link QoS parameters of all links between two peer groups. This aggregation policy is totally opposite to the optimistic aggregation policy. For example, the

"worst" of the link delay is the *maximum* delay of all links between two peer groups. In addition, the "worst" of the link bandwidth is the *minimum* of all links between two peer groups. The example of pessimistic link aggregation for bandwidth values is shown in Figure 4.4.

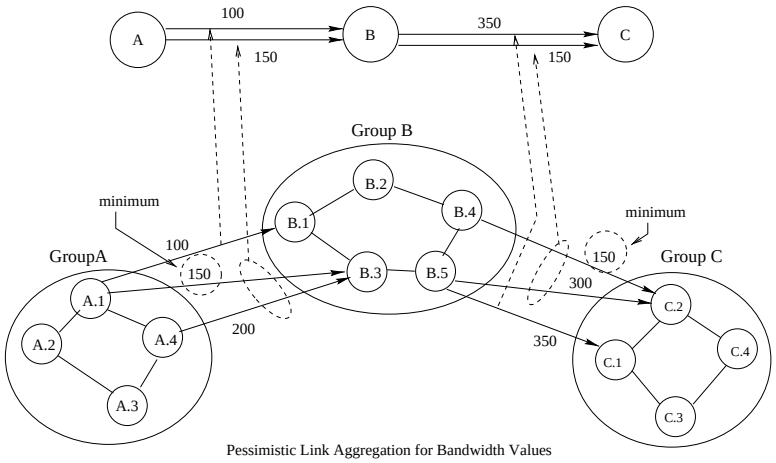


Figure 4.4: The Pessimistic Link Aggregation for Bandwidth Values

This algorithm is the opposite of the optimistic aggregation algorithm. In this algorithm, the network utilization tends to be very low because of unnecessary rejection of call requests. However, using this approach the logical link will be less sensitive to dynamic changes due to the network state change.

Average Aggregation

The average aggregation policy summarizes the link QoS parameters by calculating the average of each link QoS parameters of all links between two peer groups. For example, we consider delay, bandwidth and administrative weight as link QoS parameters of the links between two peer groups. Then the logical link between two peer groups will have the average bandwidth, the average delay, and the average administrative weight of all links between two peer groups. This aggregation policy tends to optimize a link QoS parameter of all links between two peer groups. The example of average link aggregation for bandwidth values is shown in Figure 4.5.

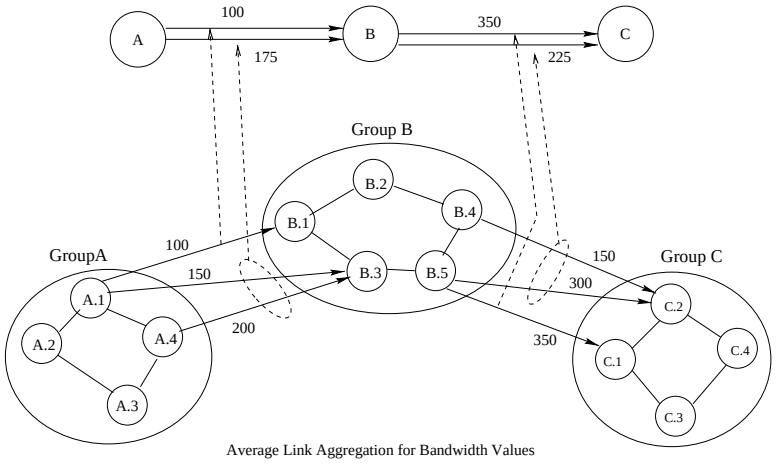


Figure 4.5: The Average Link Aggregation for Bandwidth Values

4.4 Nodal Aggregation

This section explains our tests of nodal aggregation with different policies. The nodal aggregation is used to summarize information of a peer group. The summarized information will be represented as a logical node in the next higher level. The form of summarized information to be represented as a logical node is called "complex node." The complex node is a form of star topology which has a nucleus and spokes from the nucleus to logical ports. One logical port of the complex node represents one border node of the peer group. The nodal aggregation is used to form a complex node by calculating values of spokes from the nucleus to all the logical ports (or border nodes). For more details of the complex node, please refer to the PNNI specification version 1.0 of ATM Forum.

In the nodal aggregation, one fundamental step is to assign weights (or QoS value) to spokes. In the first step, we construct a full-mesh representation of connectivity among border nodes within the same peer group. Then we summarize the representation to reduce some amount of peer group information. This summarized representation will be used to create a complex node representation to represent the logical group node (LGN) at the upper level of the hierarchical network. The full-mesh representation and the complex node after the aggregation are shown in Figure 4.6.

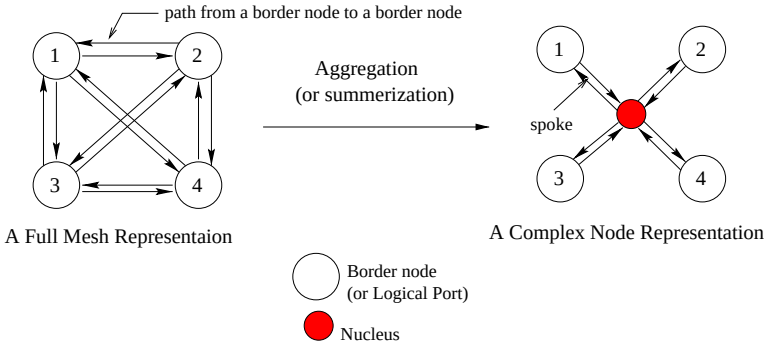


Figure 4.6: Nodal Aggregation

Note that the complex node representation is a star topology. The nucleus is in the center. Spokes are connected from the nucleus to all logical ports (or border nodes). Therefore, the path from one logical port to another logical port in the complex node is alway composed of two spokes. Therefore, for every aggregation policy, after we calculate values for a spoke in the complex node, we will divide those values by two.

The method of aggregation can be different. In our simulation tool, we have three different summarization policy for nodal aggregation, optimistic, pessimistic, and average. For our testing, we use the hierarchical network shown in Figure 4.1 to test our different nodal aggregation policies.

The Optimistic Policy

This algorithm will find the *best* values for each QoS parameter and assign these values to a spoke. As described in Section 4.3, the "best" values of paths between two border nodes can be interpreted to two different ways, maximum and minimum. For example, the best of path

bandwidth is the maximum bandwidth of all paths between two border nodes. However, the best of path delay is the minimum delay of all paths between two border nodes. The example of the optimistic aggregation for bandwidth values is shown in Figure 4.7.

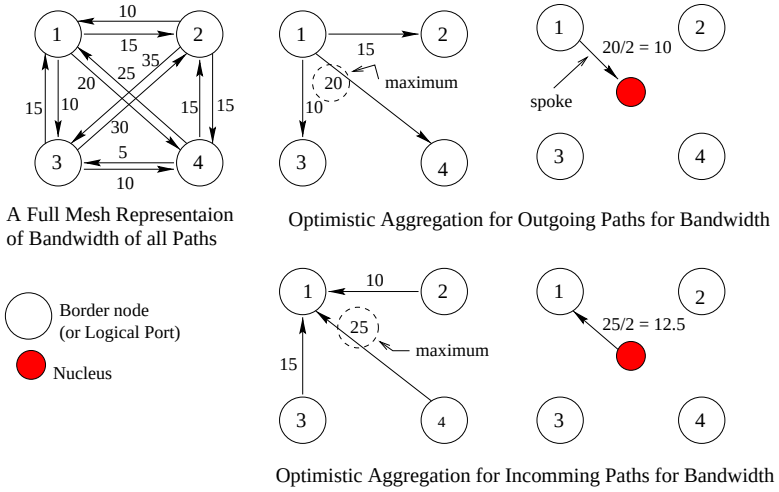


Figure 4.7: The Optimistic Aggregation for Bandwidth Values

Note that each QoS best values used to form a spoke may be obtained from different paths between two border nodes. For example, the maximum bandwidth value is obtained from the path that is different from the path we obtain the minimum delay. This is an aggressive algorithm because it probably selects all the best QoS values from different paths. This means there may be no physical paths which can simultaneously support all the advertised QoS values. This algorithm tends to give the better network utilization, but crankback probably occurs frequently in the network.

The Pessimistic Policy

This algorithm will find the *worst* values for each QoS parameter and assign them to a spoke in the complex node. Also note that all QoS worst values may not be obtained from the same path. This algorithm is the opposite of the optimistic aggregation algorithm. In this algorithm, the network utilization tends to be very low because of unnecessary rejection of the call request. However, using this approach the logical link will be less sensitive to dynamic changes due to the network state change. The example of the pessimistic aggregation for bandwidth values is shown in Figure 4.8.

The Average Policy

From the full-mesh representation, this policy aggregates information by calculating the average of each QoS parameters of all paths between two border nodes. For example, the QoS value of bandwidth of a spoke from a border node to a nucleus is the average of all bandwidth of the *outgoing* paths of the border node. Since the spoke is bi-directional, the QoS value of bandwidth of a spoke from the nucleus to a border node is the average of all bandwidth of the *incoming* paths to the border node. The average aggregation is shown in Figure 4.9

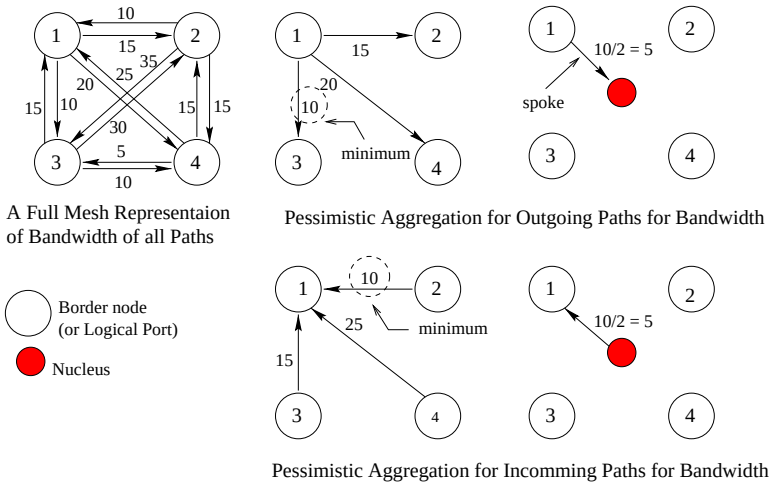


Figure 4.8: The Pessimistic Aggregation for Bandwidth Values

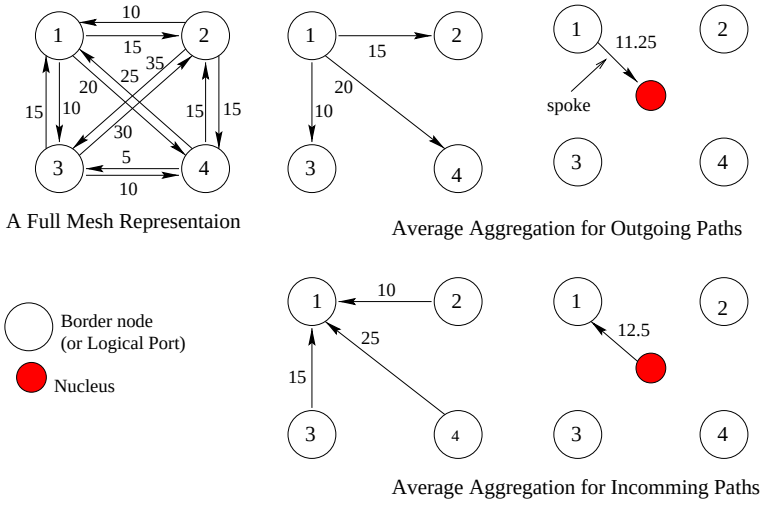


Figure 4.9: Average Aggregation Policy

4.5 Size of Database

This section discusses about the database size of a node in the hierarchical network. In the PNNI hierarchical network, the size of database of each node in the network is different. The difference depends on how to group nodes in the network. The number of peer groups in the network has a major effect on the database size kept at each node.

For our testing, we test our simulator using two networks, two-peer-group and four-peer-group networks. Each network has 40 nodes, and 50 links. The only different between these two network is how to group nodes together. The network with two peer groups is shown in Figure 4.15, and the network with 4 peer groups is shown Figure 4.11. From these two networks, the amount of topology information kept in each node is different. The size of database of topology information of the two peer group network will be more than that of the four peer group network.

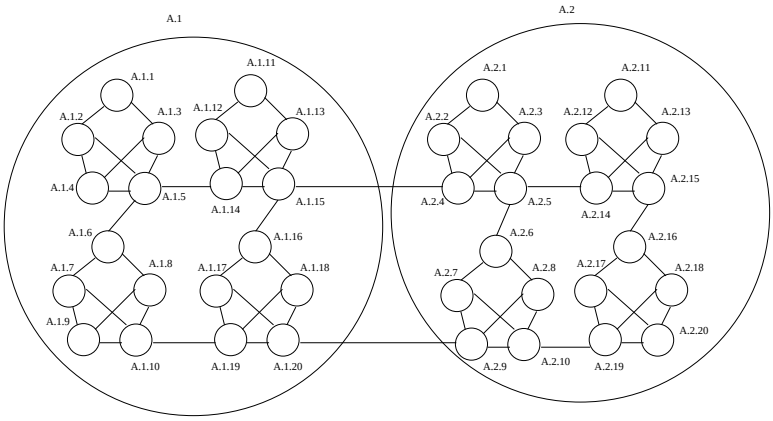


Figure 4.10: The Two Peer Group Network

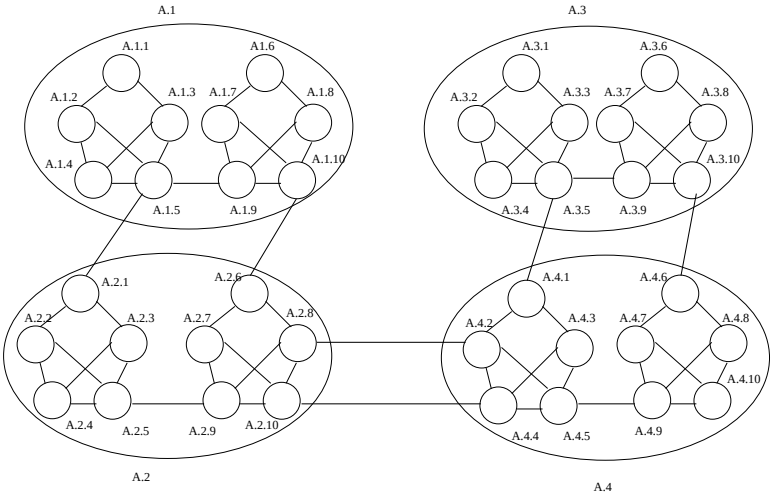


Figure 4.11: The Four Peer Group Network

4.6 Flooding in A Logical Level

In this section, we test the flooding feature of our simulation tool for a hierarchical network. At each hierarchical level, nodes in the same peer group start flooding topology information to their neighbors. The information is flooded through the PNNI routing connection control channel (RCC) which is VPI=0 and VCI=18 of the connection between two nodes.

After flooding for a while, all nodes within the same peer group will have the same topology information, and we call this time as the "convergence" time. After a peer group is convergent (all nodes have the same topology database), the aggregated information at a parent level is flooded down a child peer group. For testing the flooding mechanism, we have proposed different topologies.

The network in Figure 4.12 is the simplest network for testing flooding mechanism. It is used to prove that the logical connection between two LGNs is properly created, and also the topology information at the logical level is properly flooded.

In a peer group at the physical level, topology information gets flooded through the RCC channel of the physical connection between A.1.1 and A.1.2. In peer group A, at the logical level, A.1 is the LGN of the peer group A.1, and A.2 is the LGN of the peer group A.2. LGN A.1 and A.2 will create a logical connection (and RCC channel) between each other for information exchange or flooding at the logical level. After the topology database of node A.1 (or A.2) converges, A.1 will pass the topology information at its level to the peer group leader (A.1.1) of the child peer group. Then the PGL A.1.1 floods information from the parent peer group to its neighboring nodes.

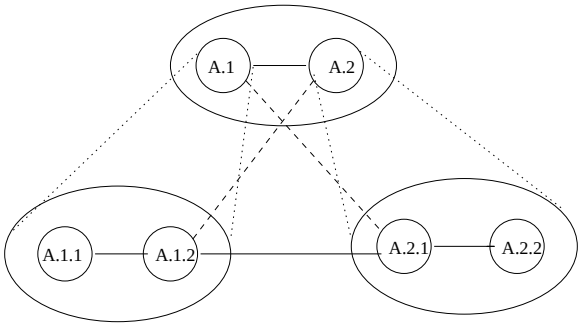


Figure 4.12: The network of two peer groups, each of which has two nodes

We also test our flooding mechanism in a larger network. The 3-cluster network has 3 peer groups, each of which has 8 nodes and 10 links. There are three outside links connected between two peer groups. Totally, we have 9 outside links. The 3-cluster network is shown in Figure 4.13. The 8-cluster network has 24 nodes similar to the 3-cluster network. It has 8 peer groups, each of which has 3 nodes and 3 links. The 8-cluster network has 13 outside links. Figure 4.14 shows the 8-cluster network.

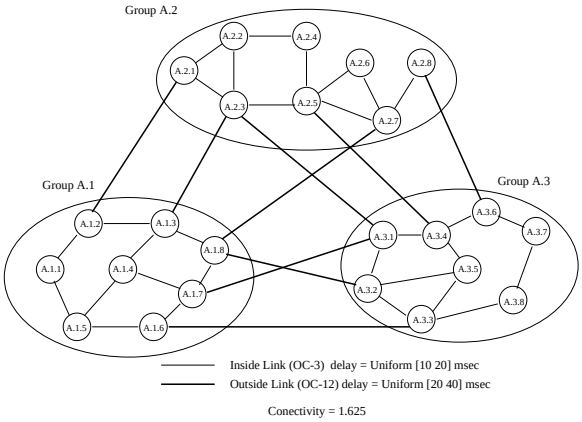


Figure 4.13: The network of three peer groups, each of which has eight nodes

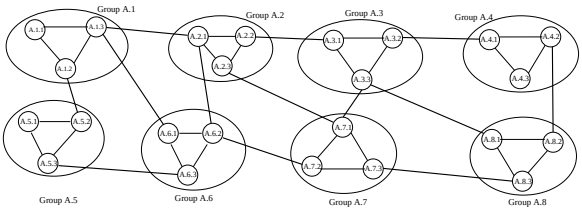


Figure 4.14: The network of eight peer groups, each of which has three nodes

4.7 Three-level Network

This section shows the testing of a 3-level PNNI hierarchical network.

These test scenarios have been designed with a view to test the scalability that can be achieved using hierarchy in PNNI. They also demonstrate the ability of our simulator to run multi level hierarchical PNNI experiments.

The first test scenario is the most basic and simple three level topology. In Figure 4.15 we can see that Level 0 has four peer groups with three nodes each. Each of these four peer groups are represented by a leader in Level 1 and hence we have four logical nodes in Level 1. By grouping two of these four nodes into a peer group, we end up with 2 logical nodes in Level 3. The Level 3 logical node thus has summarized or aggregated information about all the nodes beneath it. From this we can understand the importance that hierarchy plays in achieving scalable networks.

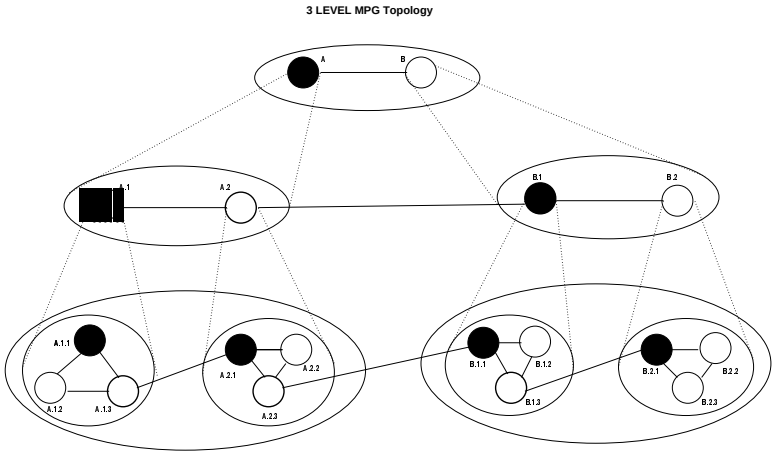


Figure 4.15: The Basic Three Level Network

This topology is an extension of the basic three level topology that we saw above. This topology has 6 peer groups with 3 nodes each in Level 0. Figure 4.16 shows how the grouping is done at the next two levels and we end up with 3 logical nodes in the topmost level which hold the aggregated information of all their child nodes.

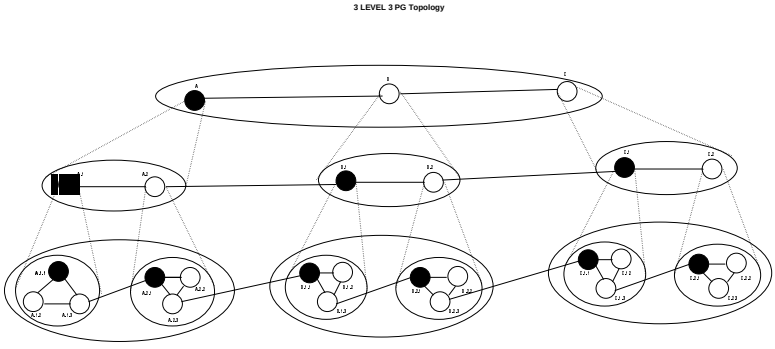


Figure 4.16: Three Level Network with Three Peer Groups

This topology, shown in Figure 4.17 has 8 peer groups with 3 nodes each at the physical level. As a result we have 8 logical nodes in Level 1 and by grouping 2 of these together at the topmost level we get 4 logical nodes in Level 3.

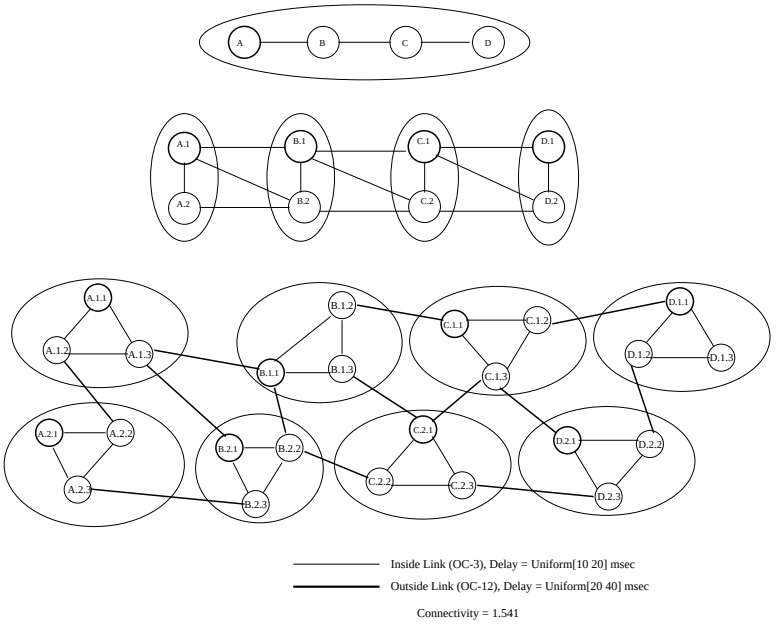


Figure 4.17: The 8cluster minhop Network

This test topology shown in Figure 4.18 indeed tests the scalability performance of our simulator. There are four more peer groups in the phisical level, 14 to be exact. The topology is essentially an edge core one. This test serves to establish the scalability as well as the robustness of the simulator.

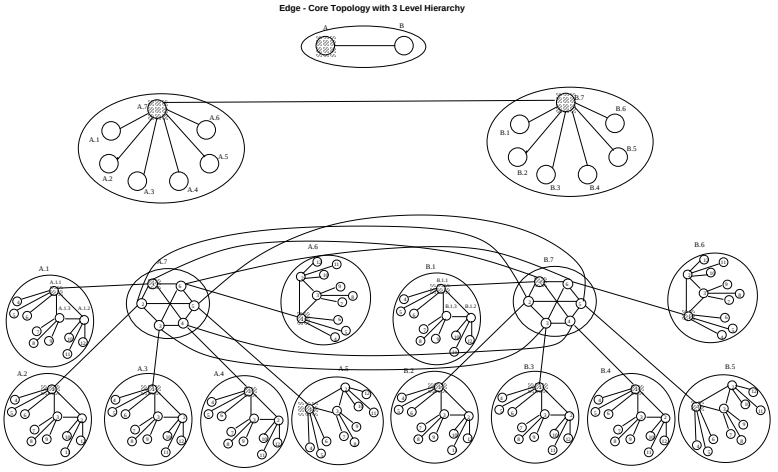


Figure 4.18: The Star EC 14 PG 156 Node Network

4.8 Four Level Network

This section describes a four level test that we ran in order to establish the performance and correctness of the simulator when computing routes for very large scalable networks. As obvious, this test also re-confirms the correctness of the simulator when running multi-level hierarchical experiments. The peering is shown in Figure 4.19

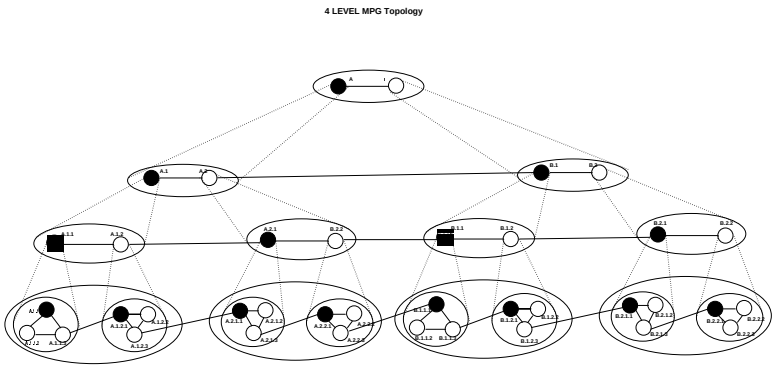


Figure 4.19: The Basic Four Level Network

4.9 Crankback and Alternate Routing Tests

When routing is performed, the currently available view of the network is used. It is possible that after routing has been done, and as the call is forwarded through the network according to the DTL that has been formed earlier, some node (or link) goes down or some link is not able to meet the required criteria (bandwidth, delay etc) for that call. The call would thus fail at that node or link. This could be because the network has changed since the initial source routing was done. As a solution to this, we have the crankback and alternate routing feature. When the call request fails to reach upto the destination host, the call is *crankbacked*. A crankback element is formed and the call backtracks through the previous nodes that it traveled through in the forward direction until it reaches the ingress border node of the current peer group (that is, the peer group where the call failed) and from here, alternate routing is done to find another alternate route. This is done by pruning out the failed nodes/links from the topology graph and routing performed once again.

We test this feature of our simulator using the topology shown in Figure 4.20. We conducted tests for three different scenarios. Using the *nodefail* or *portfail* option in the input script, we can force a node or link respectively to fail. In the first test scenario, we force a node in the first peer group (source peer group) to fail, say node A.1.4. A call is made from Host A.1.1.1 which is connected to the first peer group to Host A.3.8.1 which is connected to the third peer group A3. At the source node, a DTL is formed A.1.1 A.1.5 A.1.4 A.1.7 A.2 A.3.

When the call request reaches node A.1.4, it fails there and is crankbacked to the ingress node of its peer group (which in this case is the source node, A.1.1) and from here alternate routing is done and a new DTL of A.1.1 A.1.2 A.1.3 A.1.8 A.2 A.3 is generated and the call is forwarded through this new route.

In the second scenario, we force a node to fail in the second or intermediate peer group. The initial DTL given by the source node A.1.1 is A.1.1 A.1.5 A.1.4 A.1.7 A.2 A.3. Since the are no failed nodes/links in peer group A1, the call request succeeds until it reaches peer group

A2. In Peer group A2, we have forced node A.2.5 to fail. The ingress node of Peer group A2, that is, node A.2.1 does transit routing and gives the route as A.2.1 A.2.3 A.2.5 A.2.7 A.3. When the request reaches node A.2.5, it finds that that node has failed and hence it crankbacks to the ingress node of this peer group which is node A.2.1 which tries to find an alternate route to reach the destination peer group A3. After pruning out the node A.2.5 from its view of the topology, A.2.1 comes up with the new DTL of A.2.1 A.2.2 A.2.4 A.2.6 A.2.8 A.3 through which the call finally gets routed. In our third scenario, we force a node in the third (destination) peer group to fail and we tested the correctness of the crankback and alternate routing mechanism.

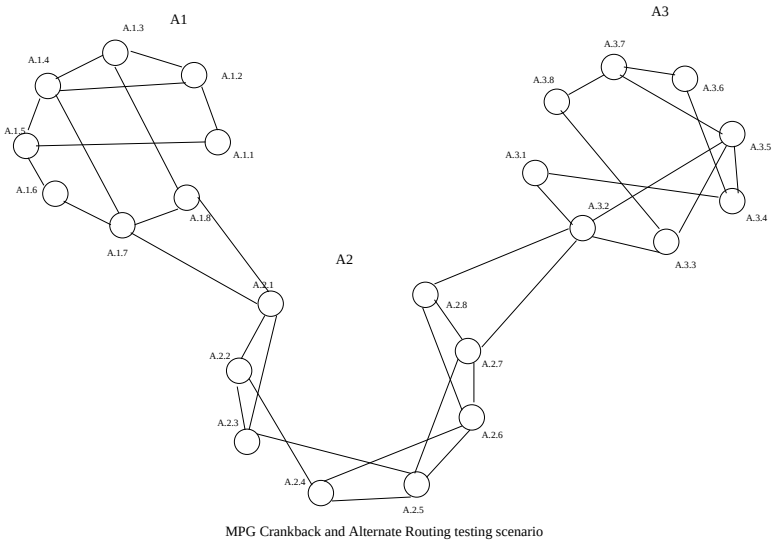


Figure 4.20: Scenario for testing Crankback and Alternate Routing

Chapter 5

Multiple Peer Group Performance Tests

The chapter deals with test cases that have been designed to test the performance of the KUPNNI simulator in running multiple peer group tests. Section 5.1 describes the edge core topology with 7 peer groups. Section 5.2 describes the test case for evaluating the performance of a multi peer group topology that has 40 nodes in 10 peer groups. Section 5.3 is about a topology that has 4 clusters with 15 nodes each. Finally, we have a three level topology to test the performance of the simulator in scalable networks.

5.1 Edge Core Topology

The common edge-core topology consists of core nodes and edge nodes. Core nodes are connected to other core nodes with high capacity links and to edge nodes with low capacity links. Each edge node connects to a host which generates calls. Each core node is classified into one of the following two categories.

- A large scale node which is connected to other large scale nodes with a link providing high capacity that also exhibits high delay.
- A small scale node which is connected to large scale nodes with high capacity links and to edge nodes with low capacity links.

Figure 5.1 shows an edge-core topology with 7 peer groups, the core peer group *A.7* provides the connectivity among the different peer groups. Each of the other peer groups consists of 3 small scale nodes and 9 edge nodes, each edge node is connected to a host. So, there are totally 54 hosts in the network trying to make calls. The small scale nodes are connected to the central core by high capacity OC12 links. In Figure 5.1 **L**, **S** and **E** stand for a large scale, small scale and an edge node respectively.

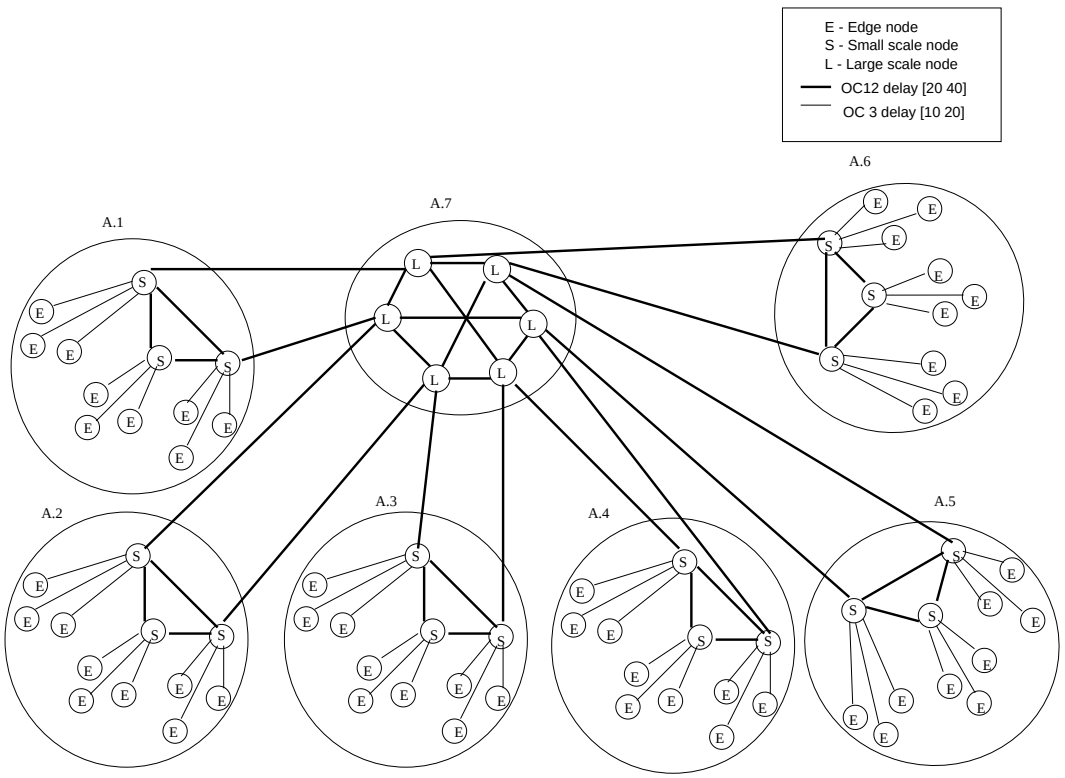


Figure 5.1: Edge-Core Topology

5.2 Multi Peer Group Topology

The topology shown in Figure 5.2 consists of 40 nodes grouped into 10 peer groups. However, the grouping can be varied to form 1, 2 and 5 peer groups with 40, 20 and 8 nodes in each peer group respectively. The links between peer groups are high capacity OC12 links. The link connectivity remains the same for all the groupings. For the same number of nodes and links, this helps us evaluate the effects of grouping on the network performance. Every node has a host connected to it and all the hosts generate calls. In all, the network has 40 hosts.

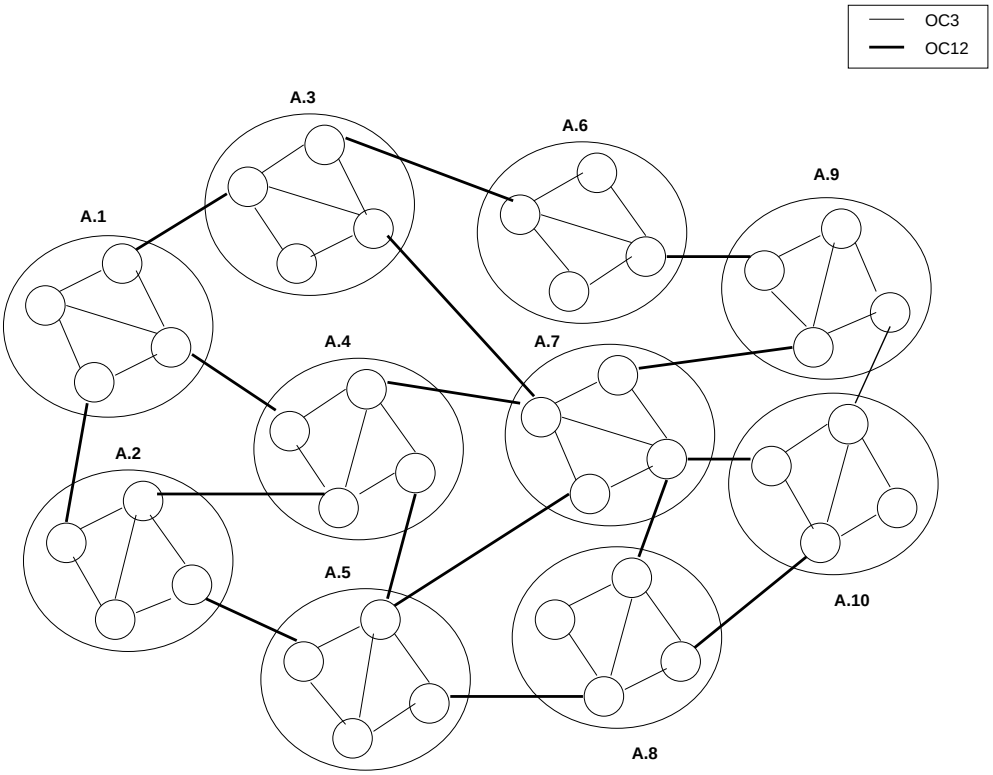


Figure 5.2: Multi Peer Group Topology

5.3 4-Cluster Topology

Figure 5.3 shows the 4-Cluster topology, this consists of four peer groups with 15 nodes each. The outside links are long haul links and have high delays. This topology is designed to study the effects of the aggregation policy chosen. A lot of internal traffic is generated in peer group *A.3* to congest the peer group, so this peer group advertises a higher cost for traversal. If the aggregation policy is effective, this is reflected properly in the other peer groups and they will be able to make informed routing decisions to bypass the peer group and take an alternate path.

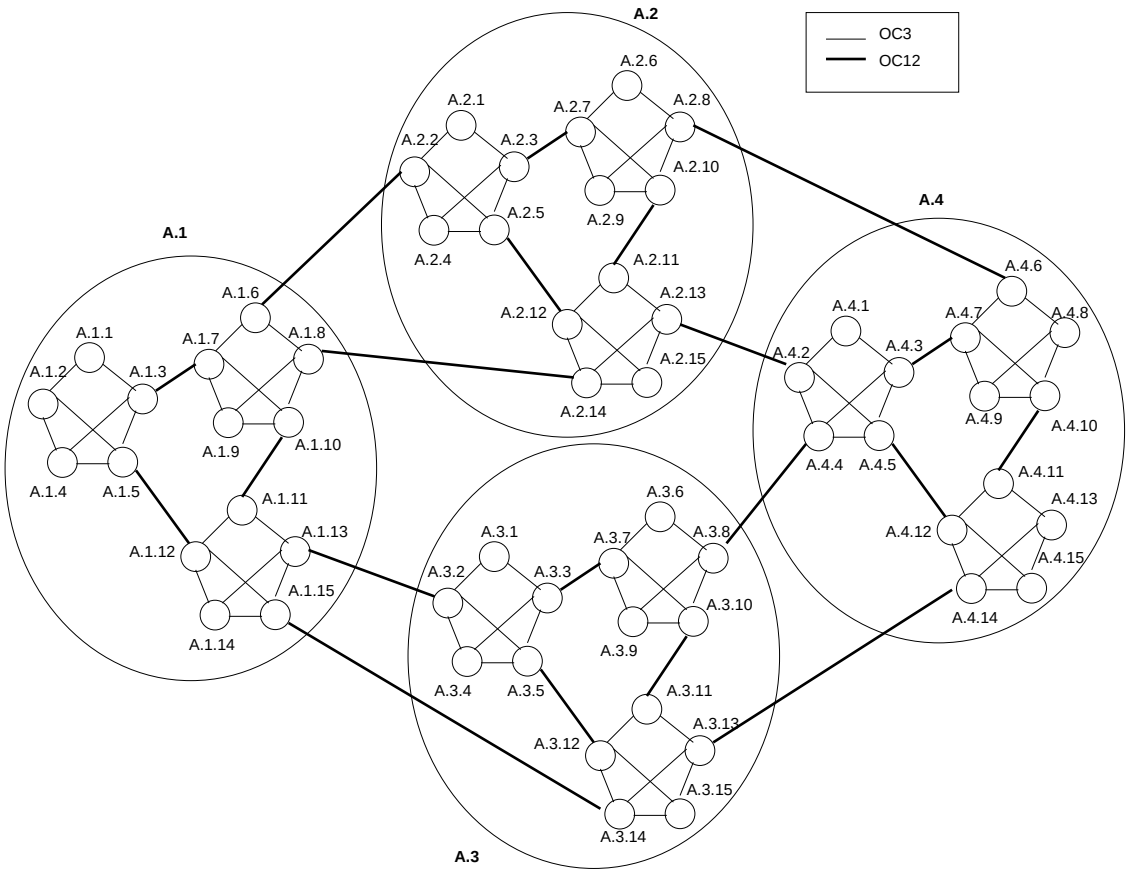


Figure 5.3: 4-Cluster Topology

5.4 3-Level Edge-Core Topology

Figure 5.4 shows the 3-Level Topology, this test is designed to mainly test the scalability achieved using hierarchy in PNNI. Now, the topology itself is an edge-core topology with *eight* peer groups at the lowest level, *A.1 - A.4* and *B.1 - B.4*, *two* peer groups *A and B* at the second level, and finally *one* peer group at the third level. There are core nodes and edge nodes, edge nodes are connected to hosts. Core nodes are connected to each other by high capacity OC12 links and edge to core connectivity is provided by OC3 links. The hierarchy is also flattened to form a single peer group and tests are also run with the flattened single peer group form for comparison purposes.

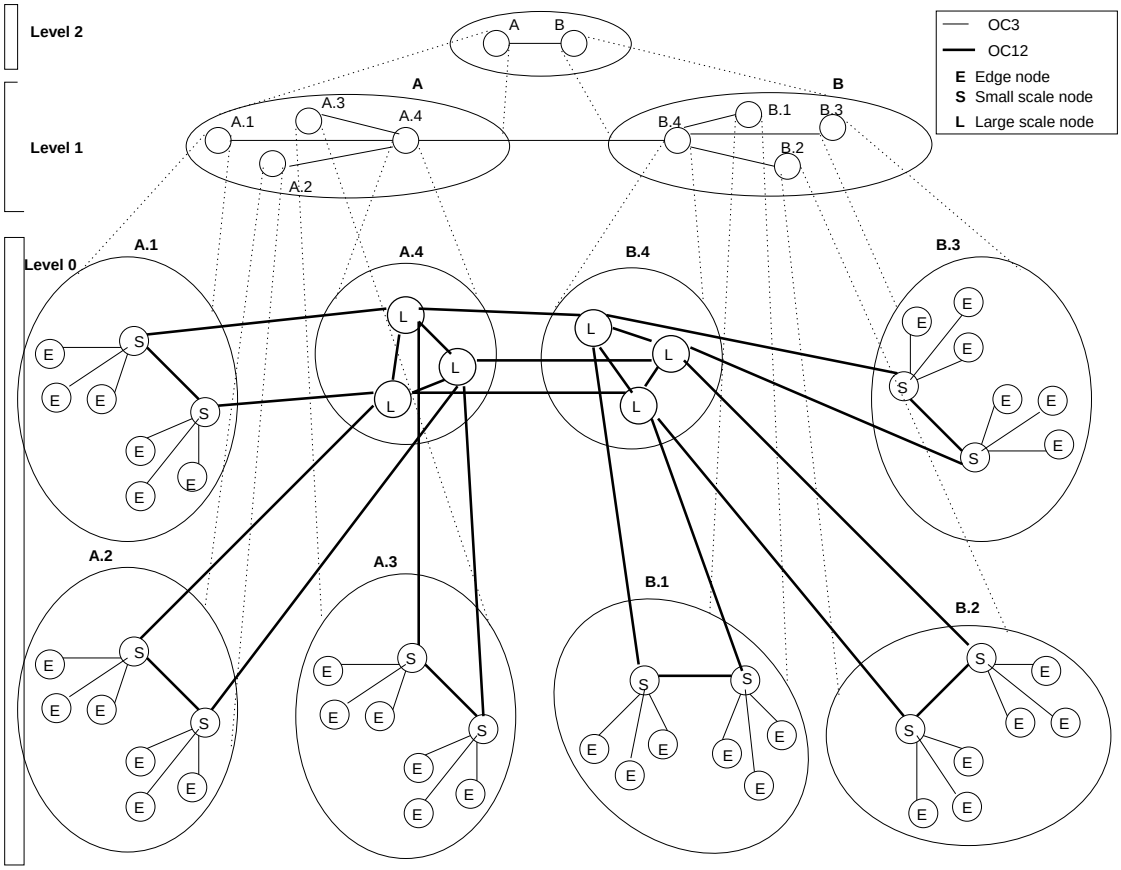


Figure 5.4: 3-Level Edge Core Topology