Performance Evaluation of Multiple Criteria Routing Algorithms in Large PNNI ATM Networks

by

Phongsak Prasithsangaree

B.E. (Electrical Engineering),
King Mongkut's Institute of Technology, Ladkrabang campus,
Bangkok, Thailand, 1995

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science

_____

Professor in Charge

_____

_____

Committee Members

_____

Date Thesis Accepted

To Mom and Dad, for your encouragement

To Kanokwan, for your support, your patience, and your love

Acknowledgments

I would like to express my sincere gratitude to Dr. Douglas Niehaus, my advisor and committee chairman, for his guidance and advice throughout this research and all of my work with him and for helping to make this thesis possible. I would like to thank Dr. Victor Frost and Dr. Jerry James for serving as my committee members.

I would like to express my appreciation to Sprint Corp. for sponsoring this research project and to Dr. Nail Akar and Sohel Khan for their feedback and interest in my work.

I would like to thank my colleagues, Kamalesh Kalarickal, Gowri Dhandapani, and Bhavanis Shanmugam for their help during the thesis development and for being parts of the KU-PNNI project group. I would also like to take this opportunity to thank the other Team Niehaus members that I have had an opportunity to work with at one point or another: Pramodh Mallipatna, Alejandro Parra-Briones, Anitha Rajesh, and anybody else that I have inadvertently forgotten.

I would like to thank Sandeep Bhat. His preliminary work on the KU-PNNI Simulator/Emulator was a foundation on which I have based much of my work. I also would like to thank my volleyball team, and badminton club fellows for their friendship and entertainment. I would like to especially thank Lanny Maddux and Ann Meechai for helping me in writing my thesis, and Aparna Ramkumar, Arun Gautam Dugganapally, and Priyanka Parameswaran for helping me with the presentation.

Also I would like to thank my girlfriend, Kanokwan Wichiwaniwed, who gave me her encouragement, patience, understanding, and love during my thesis work and throughout 4 years of being apart on another side of the world. Finally I would like to thank my Mom, Dad and my family for their encouragement during my stay in the USA. Without their encouragement and their support I would have never been here at this point of my life.

Abstract

The ATM network is expected to become a backbone network for high-speed multimedia services because of its capability of supporting a large computer network with robustness, scalability, and Quality of Service (QoS), such as bandwidth, delay, and delay variation for a variety of service classes. Therefore, the performance issues relating to the Private Network to Network Interface (PNNI) protocol, which provides link state based dynamic and QoS guaranteed routing capability in an ATM network, have assumed significance. Therefore, the ATM forum has released PNNI specification version 1.0, but this specification does not define path selection algorithms that select appropriate paths with a guaranteed QoS satisfying several constraints. Thus, we introduce multiple criteria routing algorithms (MCRAs) to be used for routing in large PNNI ATM networks. In this thesis, we evaluate the performance of MCRAs using metrics such as the call failure rate, the call setup time, routing inaccuracy, and link utilization. The results are taken from the PNNI ATM simulator, which shares about 90% of the real ATM switch signaling software. Our MCRAs are tested in various kinds of network topologies, and the results of the performance evaluations are discussed.

Contents

List of Tables

List of Figures

List of Programs

Chapter 1

Introduction

In the area of communication networks, Asynchronous Transfer Mode (ATM) technology has been claimed to be a network of future communication. The ATM network is expected to become a backbone network for high-speed multimedia because it is able to support a large computer network with robustness, scalability, and Quality of Service (QoS), such as bandwidth, delay, and delay variation, for a variety of service classes. Because of its scalability, the ATM network can support various sizes from local area networks (LANs) to wide area networks (WANs). It also provides a multi-vendor system which can support a wide variety of networks with seamless connections and internetworking among a variety of computer networks.

To support a large scale network, a dynamically automatic network configuration mechanism which can automatically control a topology of switches and links is required. The ATM forum [6] therefore has introduced a dynamic configuration protocol for supporting private networks called the Private Network to Network Interface (PNNI) protocol.

The PNNI protocol consists of two parts: signaling and routing. The PNNI signaling protocol was designed for call connection setup using a message exchanging mechanism between switches. The message includes the resource requirements of calls, call setup information, and etc. The PNNI routing protocol

was designed for an exchange of topology state information including the ”image” of network topology, traffic attributes and metrics of each link between switches. The PNNI routing protocol also performs a Connection Admission Control (CAC). In order to standardize these protocols, the ATM forum has released PNNI specification version 1.0 [3]. However, this standard does not define path selection algorithms that select appropriate paths with a guaranteed QoS satisfying several constraints.

Path selection algorithms, sometimes called routing algorithms, affect both the connection setup delay and call blocking probability and influence the quality of service for users and network utilization which affects the network efficiency for providers. To select a path which is efficient for not only the user but also the provider, a routing scheme which supports both user and provider constraints is necessary. Since the user and provider constraints address different properties, a routing algorithm must select routes based on more than one criterion

However, as stated in the PNNI specification [3], the original work of the PNNI routing algorithm was developed using link-state routing. The commmon link-state routing algorithm is Dijkstra’s algorithm [21]. It provides a deterministic solution based on a *single* criterion. Thus, the original Dijkstra’s algorithm cannot be used for multiple criteria routing. In addition, a problem arises because it is also known that routing with more than one requirement is an NP-complete problem [24]. For this reason, we introduce a *multiple criteria routing algorithm* (MCRA) based on a heuristic approach to avoid the NP-complete problem. Section 1.1 states the problem which arises when multiple criteria routing is needed to support the user’s QoS requirements.

1.1 Problem Statement

QoS routing faces a basic problem of finding a path that satisfies the multiple constraints imposed by the QoS requirement contained in the user's call request. Switch Virtual Circuits (SVCs), which are created for transferring the user data across the network, cannot be set up until the routing algorithm has found paths with sufficient resources to meet the user's requirements. In traditional data networks, the network is usually characterized by a single metric such as hop count or delay, and the shortest-path algorithm is used for path computation.

ATM networks, however, need routing to support multiple metrics to cover a wide variety of QoS requirements. In order to evaluate a link's ability to support a call with a specific QoS constraint, the network must characterize a link by the highest QoS level that it is able to support, i.e., by QoS link metrics. Therefore, QoS link metrics are combined to provide the QoS characterization of a path. There are many QoS metrics that are relevant to ATM networks, but the following four QoS metrics are most commonly considered: bandwidth, delay, delay jitter, and loss rate. The ATM Forum [6] has defined another metric, Administrative Weight (AW), which was included in the PNNI standard. This metric is defined by a network operator. For example, it may be used as a hop counter or to prioritize links by the desirability of usage.

When a user's call request arrives, it contains a traffic descriptor and end-to-end QoS requirements. The network determines the five metrics needed for path computation as follows.

- The bandwidth requirement, BW, for the call is determined from the user's traffic descriptor.
- The end-to-end delay requirement, D, is determined from the user-specified maximum cell transfer delay (maxCTD) parameter.

- The end-to-end jitter requirement, *J*, is determined from the user-specified cell delay variation (CDV) parameter.
- The end-to-end loss rate, *L*, is determined from the user-specified cell loss ratio (CLR) parameter, which indicates a maximum tolerable loss rate.
- Administrative weight, *AW*, is determined from the network operator control parameter.

The user's call request is now converted to a quintuple, which we call the *QoS constraints*, (*bw, d, j, l, aw*), when the QoS-routing problem is reformulated as a typical optimization problem. Note that not all of these parameters may be provided in a user's call request. Furthermore, the subset of parameters used depends upon the category of service being requested.

Each link *i* in an ATM network is characterized by a quintuple $(bw_i, d_i, j_i, l_i, aw_i)$ that represents the current available bandwidth, delay, jitter, loss rate and administrative weight on the particular link. In Figure 1.1 , the sample network shows links characterized by this quintuple. The delay, jitter, and loss rate also determine the behavior of the output buffer of the switch transmitting on that link. Therefore, these characteristics also affect other elements such as traffic flows using the output buffer and the switch scheduling algorithm. It is assumed (as stated in the PNNI standard [3]) that these metrics are updated periodically by measurement methods and thus reflect the latest status of the link.

In PNNI terminology, additive metrics such as delay, jitter, administrative weight, and hop count correspond to *path constraints*. Non-additive metrics such as bandwidth and loss rate are viewed as *link and node constraints*, respectively. The QoS constraints provide both path and link constraints. The link constraints are one part of the input parameters to the routing algorithm as shown in Figure 1.2. In Figure 1.2, the topology "picture" and routing policy are also considered as input to the routing model.

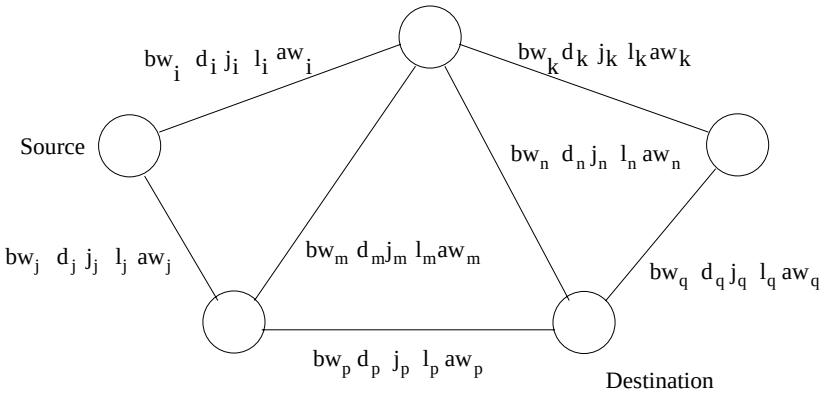

Figure 1.1: The Sample Network Showing Metrics for Problem Resolution

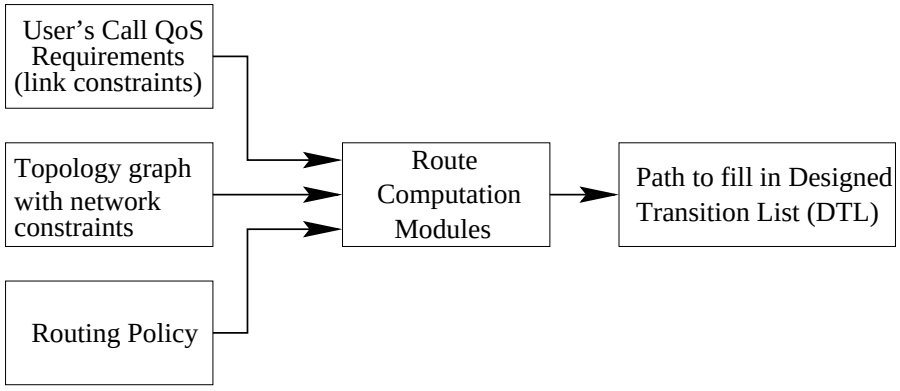

Figure 1.2: The Routing Model to Our Solution

The rule for combining the link metrics to form path metrics depends upon the semantics. For example, delay and jitter are additive metrics. Therefore, the sum of the delay, d, of the links along the path P, $\sum_{i \in P} d_i$, gives the total path delay, and the sum of the jitter, j, of the links along the path P, $\sum_{i \in P} j_i$, gives the path jitter. The AW metric is required by the PNNI specification to be an additive metric. The loss rate is a *multiplicative* metric because the loss rate on the whole path is computed according to $1 - \prod_{i \in P}(1 - l_i)$ when l is the loss rate of a link. The bandwidth metric is a *comparable* metric since the bandwidth of the path is computed from $min[b_i]$

Therefore, we can represent the general problem as follows: given a source and a destination, and the user end-to-end QoS requirement (*BW*, *D*, *J*, *L*, *AW*), find a path, P, such that all QoS requirements are fulfilled by the following constraints:

$$min[bw_i] \geq bw \quad when\ i \in P \quad for\ bandwidth \tag{1.1}$$

$$\sum_{i \in P} d_i \leq d \quad for\ delay \tag{1.2}$$

$$\sum_{i \in P} j_i \leq j \quad for\ delay\ jitter \tag{1.3}$$

$$1 - \prod_{i \in P}(1 - l_i) \leq l \quad for\ loss\ rate \tag{1.4}$$

$$\sum_{i \in P} aw_i \leq aw \quad for\ administrative\ weight \tag{1.5}$$

In order to convert this to a proper optimization problem, one or more of these criteria can be used in the routing function while the other criteria remain as hard constraints.

In this thesis, we evaluate the performance of multiple criteria routing algorithms in a PNNI ATM network, which support QoS constraints. Our evaluation experiments use a simulation tool which has been developed at the Information and Telecommunication Technology Center (ITTC) of the University of Kansas.

The simulation tool includes the UNI signaling messages, the data link QSAAL layer, the Q93B layer, the Switch Call Control Layer, and the PNNI layer. This simulation tool implementation shares about 90% of the real ATM switch signaling software, and the results obtained from the simulation are expected to closely match those obtained using real network experiments. This is the most advantageous feature of our simulation tool over other simulation tools. Therefore, the aim of this thesis is twofold.

- Showing our simulation tool can support simulations of large scale networks using our multiple criteria routing algorithms.
- Giving multiple criteria routing results for single peer group PNNI ATM networks.

The rest of this thesis is organized as follows: we discuss related work in Chapter 2, and then explain our solution to the multiple criteria routing algorithm in Chapter 3. Chapter 4 explains our test scenarios, and Chapter 5 presents our experimental results and evaluation of our solutions to the multiple criteria routing problems. Our conclusions and future work regarding these issues are discussed in Chapter 6.

Chapter 2

Background & Related Work

PNNI consists of two protocols, a signaling protocol and a routing protocol [3]. The signaling protocol is based on ATM Forum User Network Interface (UNI) signaling [4], which in turn is related to ITU-T Q.2931 [22]. The modification has been made to provide a network to network interface (NNI), rather than UNI to make use of the routing protocol, and to provide certain other extra functionalities. The PNNI routing protocol uses similar concepts to the Open Shortest Path First (OSPF) protocol used within Internet Protocol (IP) networks [16]. However, it provides QoS-based dynamic hierarchical source routing, and it is scalable. In this Chapter, Section 2.1 explains the PNNI signaling protocol. In addition, the PNNI routing protocol is described in Section 2.2.

2.1 PNNI Signaling

PNNI signaling is designed to be compatible with UNI version 3.1, but also provides functionality from UNI version 4.0 signaling, including [2]:

- point-to-point and point-to-multipoint calls
- parameterized quality of service
- anycast (unicast and multicast)

- signaling for ABR class
- switched virtual path connections
- negotiation of traffic parameters – peak cell rate (PCR), sustainable cell rate (SCR), and maximum burst size (MBS)

In addition, PNNI is created on the capabilities of UNI version 4.0 signaling to provide soft permanent virtual circuits (SPVC), at both virtual path (VP) and virtual channel (VC) levels. Moreover, due to the source-based routing of PNNI, the signaling capabilities support the use of designated transit list (DTL), crankback procedures, and alternate routing. In this thesis, the call setup procedure is summarized in Section 2.1.1. Section 2.1.2 describes the DTL. Crankback procedures and alternate routing are described in Section 2.1.3.

2.1.1 Call Setup Procedure in a PNNI Network

When a call from an end user arrives at a PNNI network, the node that connects to the end user, the *source node*, starts the setup procedure. First, the source node determines the call request and contacts its topology database to find the route that leads to the destination specified the call request. The route can be different depending on the routing policy specified at the source node. After the route is found, the source node pushes the list of nodes (route) into the information element, called a *Designated Transit List (DTL)*. The DTL is included in the signaling message that is passed to the next transit node. The DTL procedure is explained in Section 2.1.2. As the call goes along the PNNI network, it can fail. The reason is that the routing information given at the time of routing at the source node is out of date. This case can happen in a large network because of the propagation delay between nodes. Therefore, PNNI implements the *crankback* procedure to report the failure to the source node so that the source node can find an alternate route for

this call request. The crankback procedure and the alternate routing are described in Section 2.1.3.

2.1.2 Designated Transit List (DTL)

PNNI uses source routing to forward an SVC request across one or more groups in a PNNI routing hierarchy. The PNNI term for the source route vector is the designated transit list (DTL). A DTL 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 topology database, it computes a path to the destination that will satisfy the QoS objective of the request. Intermediate nodes obtain the next element (hop) in the DTL, perform the call admission control and forward the SVC request through the network.

A DTL is implemented as an information element which is sent in the PNNI signaling *SETUP* message. The source node computes the DTL for the entire path to the destination across the peer groups. One DTL is computed per request for every peer group. While the source node provides an explicit DTL for its peer group, it gives the names of the other peer groups it has to traverse. The DTL then contains the explicit addresses of switches within the same peer group of the source node and the "logical" addresses of switches which are in other peer groups. When the user's request reaches a border node in the new peer group, it removes the old DTL and computes the new DTL to traverse its peer group. When the request reaches the destination peer group, the border node of that peer group computes the route to the destination node.

2.1.3 Crankback and Alternate Routing

In a PNNI ATM network, when finding a route to the destination, the route is computed using the topology database, containing node information, at the time of the

connection request. In a big network, the topology database of each node may not be up to date due to long convergence time and propagation delays between the nodes. In such a case, it may not be possible to route the call request to the destination. At the intermediate node, the request might fail because of the unavailability of bandwidth on the connecting link due to inaccuracy of the bandwidth information, which is different at the time the DTL was created and the time the call was actually routed. The node where the DTL is blocked sends a *RELEASE* message to the preceding node and also includes an information element called *Crankback IE*, which contains all the information needed to make an alternate route. The information element determines the reason for failure of the connection setup and the blocked node or links. This information is used at the source node to find an alternate route. The source node eliminates the blocked node or links and tries to find another route to the same destination. If it finds a route, then a new *SETUP* message filled with the new DTL is sent to the destination node along the alternate path. If no alternate route is available, then the call is released and indicated as a failed call. Crankback and alternate routing give PNNI the advantage to increase the call success rate. The maximum number of crankback retries allowed for a connection attempt can be set as a parameter at the source nodes connected to the end user system.

2.2 PNNI Routing

As stated in the PNNI specification [3], the original work of the PNNI routing algorithm was developed from the original Dijkstra algorithm [21]. However, it provides a deterministic solution based on a routing requirement of a *single* QoS parameter. Therefore, the original Dijkstra algorithm cannot be used for multiple-QoS routing.

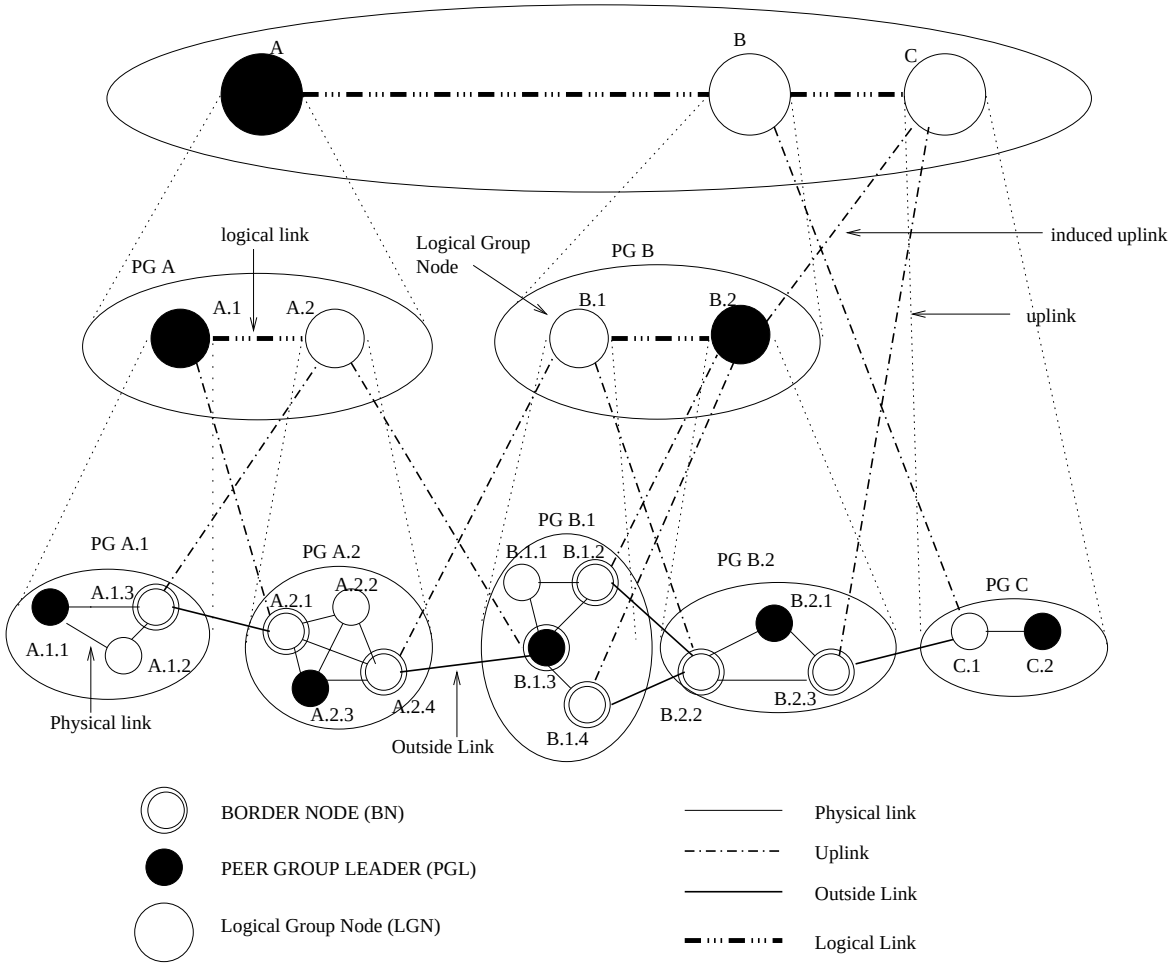

Figure 2.1: A PNNI Hierarchical Topology

2.2.1 PNNI Topology

A PNNI topology creates the PNNI routing hierarchy. Figure 2.1 shows an example of the PNNI hierarchical architecture. All elements in this figure are explained below.

Peer Group (PG)

A collection of nodes that shares the topology information generated by each node through topology information flooding is called a *peer group*. Members of a peer group discover their neighbors using a *HELLO* protocol. Each node sends a

HELLO packet through the port that is connected to other nodes to obtain information about other nodes. Physical peer groups consist of physical nodes. Logical peer groups are peer groups composed of logical nodes, which is a node that represents a lower level peer group at the next higher level of the hierarchy. The logical node function is explained below.

Peer Group Identifier

The peer group identifier is used to indicate the nodes that are within the same peer group. It is the first 14 bytes of the ATM address of the nodes. This means the nodes within the same peer group have the same first 14 bytes of their addresses.

Peer Group Leader (PGL)

Within a peer group, after the nodes exchange the HELLO protocol, the election to select any node in the peer group to be a Peer Group Leader (PGL) begins. The PGL is the representative of its peer group at the next higher level. The function of the PGL is to summarize peer group information and send it to the logical node that represents its peer group in the next level. Also, it passes the higher level peer group information obtained from the parent peer group to its peer nodes. This information is used to route the user request across the peer group.

Logical Group Node (LGN)

The Logical Group Node (LGN) is a node which represents the peer group of the nodes in the next higher level. The LGN contains the topology information that is aggregated at the lower level by the PGL. This information is flooded to the node (which can be a physical node or a logical node) that resides in the same peer group.

Parent and Child Peer Group

A parent peer group is a group which is composed of a LGN of a lower level peer group in the next higher level. However, the child peer group is the group of nodes in which the topology information is exchanged between itself and the logical node representing this group in the parent peer group.

Hello Protocol

HELLO protocol is a standard link state procedure used by neighbor nodes to discover the existence and identity of each other. After exchanging the HELLO protocol, the node generates the PNNI Topology State Element (PTSE) and floods to its neighbor nodes. The PTSE is described in the next section.

PNNI Topology State Element (PTSE)

PTSE is a unit of information used by nodes to build and synchronize a topology database within their peer group. PTSEs are reliably flooded between nodes in a peer group and downward from an LGN to the child peer group to inform neighbor nodes about its resource information. PTSEs contain topology information about the links and nodes in the peer group. A group of PTSEs are carried in PNNI Topology State Packet (PTSP). The new PTSPs with the updated topology information are sent out if a significant change in the topology occurs.

Border Node

A border node is a node in a peer group which is connected to the nodes which are not in the same peer group. This is found during the Hello protocol packet exchange by matching different peer group identifiers. The link connecting to border nodes is called the *outside link*.

Uplink

An uplink is topology information advertised from a border node to a higher level LGN. The existence of the uplink is derived from an exchange of Hello packets between the border nodes. These exchanges determine the higher hierarchical level where the two peer groups have logical nodes represented in a common parent peer group. They advertise the common level along with the address of the peer nodes in the common level in the uplink information. The uplink information is flooded all the way up the hierarchy until the information reaches the LGNs in the common higher level peer group. The LGNs which are its neighbors try to establish a logical link by using the addressing of the peer node specified in the uplink information.

Logical Link

A logical link is a connection between two logical group nodes (LGNs). A logical link is built from the aggregation done by the PGL of the peer group at the lower level. A Logical link instance is created to represent one or more physical links, and its behavior is similar to that of the physical link.

Routing Control Channel

The Virtual Path Identifier number 0 (VPI = 0) and Virtual Circuit Identifier number 18 (VCI = 18) are reserved as the virtual channels used to exchange PNNI topology information between physical nodes. Examples of PNNI information include the PTSE Packet (PTSP) and the Hello Packet.

Topology Aggregation

To represent the PNNI topology information at the child level to the logical node at the parent level, aggregation of node and link information is necessary. This pro-

cess summarizes information at one peer group level to be advertised into the next higher level peer group. Topology aggregation is performed by PGLs. Multiple links at the child level are aggregated into one link at the parent level and a peer group of nodes is aggregated into one LGN at the next higher level.

2.2.2 PNNI Topology Metrics and Attributes

Basically, PNNI is a topology-state protocol which has topology-state parameters. These parameters are exchanged among network nodes, and they are classified as *metrics* and *attributes*. A metric is a parameter whose value must be combined for all links and nodes in the SVC request path to determine if the path is acceptable. An *attribute* is a parameter that is considered individually at a switch to determine if a path is an acceptable candidate for an SVC request. The metrics and attributes that are supported by PNNI are shown in Table 2.1.

Topology State Parameters		
Topology Metrics	Topology Attributes	
	Performance Resource Related	Policy Related
Cell Delay variation Maximum Cell Transfer Delay Administrative Weight	Cell Loss Ratio for CLP=0 Cell Loss Ratio for CLP=0+1 Maximum Cell Rate Available Cell Rate Cell Rate Margin Variance Factor Restricted Branching Flag	Restricted Transit Flag

Table 2.1: Topology State Parameters [3]

Maximum Cell Transfer Delay (MaxCTD)

MaxCTD is the maximum delay for a cell transfer through all the links in a path. As shown in Figure 2.2, MaxCTD is the sum of the fixed delay component across

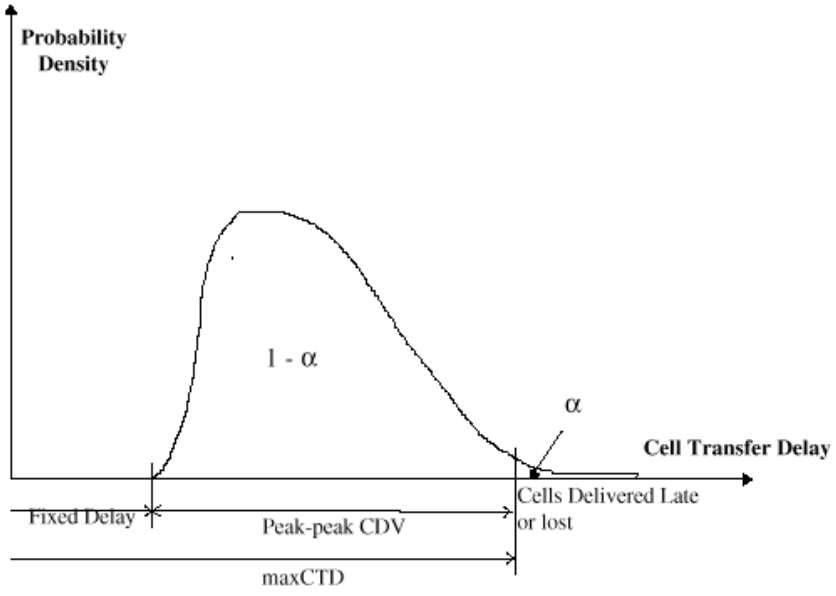

Figure 2.2: The probability density model [1]

the link or node and the peak-to-peak cell delay variation. It must be less than or equal to the delay that is requested by a user so that the user cell is accepted.

Cell Delay Variation (CDV)

From Figure 2.2, CDV is the peak-to-peak cell delay variation that determines the delay of the cell that can be accepted. Cells arriving after the peak-to-peak CDV interval are considered late. Standards currently define CDV as a measure of cell clumping. Standards define CDV at either a single point against the nominal entry point or an exit point. The ATM Forum UNI specification versions 3.1 [4] and 4.0 [2] cover details on computing CDV and its interpretation.

Administrative Weight (AW)

Administrative Weight is the link or nodal-state parameter set by the network administrator to indicate the desirability of using a link or node for whatever reason significant to the network administrator.

Cell Loss Ratio (CLR)

CLR is the ratio of the dropped cells to the transmitted cells. It describes the expected CLR at a node or link for Cell Loss Priority (CLP). A QoS class defines the cell loss ratio for the CLP=0 flow and the CLP=1 flow [4]. The CLP=0 flow refers to only those cells which have the CLP header field set to 0, while the CLP=1 flow refers to only those cells which have the CLP header field set to 1. The aggregate CLP=0+1 flow refers to all cells in the virtual path or channel connection.

Maximum Cell Rate (MaxCR)

Maximum Cell Rate indicates the maximum capacity used by connections. It can be a link or node capacity.

Available Cell Rate (AVCR)

Available Cell Rate is a measure of the effective available bandwidth on the link.

Cell Rate Margin (CRM)

Cell Rate Margin is a measure of the difference between the available bandwidth allocation and the allocation for the sustainable cell rate. The CRM is the bandwidth margin allocated above the aggregate sustainable cell rate. The CRM is an optional attribute.

Variance Factor (VF)

Variance Factor is a relative measure of the square of the cell rate margin normalized by the variance of the sum of the cell rates of all existing connections. The VF is an optional topology attribute.

Restricted Branching Flag

It is used to indicate if a node can branch point-to-multipoint traffic.

Restricted Transit Flag

This is the nodal state parameter that indicates whether a node supports transit traffic. The transit traffic is the traffic which passes through an intermediate node in a connection. If a node does not want to act as an intermediate node for the SVC connection, it will set this flag. In this case, it will accept only the connections which terminate at a called host connected to it.

2.3 Routing with Multiple QoS Metrics

In multiple-QoS routing, the number of QoS parameters of an ATM network considered by the routing algorithm could be as high as five, including Peak Cell Rate (PCR), Sustainable Cell Rate (SCR), Cell Loss Ratio (CLR), Cell Transfer Delay (CTD), and Cell Delay Variation (CDV). The problem of routing with multiple criteria arises. The problem of multiple criteria routing with more than one additive QoS metric is known as an NP-complete problem [24]. Therefore, since an optimal solution method is not computationally feasible, the challenge is how to develop a *heuristic* method providing an adequate solution to the NP-complete problem in an acceptable computational time. Therefore, a number of heuristic algorithms for multiple-QoS routing have recently been proposed.

Wang and Crowcroft studied complexity of QoS routing with multiple constraints [24]. They propose the widest and the shortest-widest path algorithm as a way of minimizing the call blocking rate. However, there is no performance evaluation of an implementation of a routing protocol based on these algorithms.

Ma and Steenkiste propose four routing algorithms: widest-shortest, shortest-widest, shortest-distance, and dynamic-alternative path algorithms [13].

Their widest-shortest path algorithm is based on the Bellman-Ford algorithm, and their shortest-widest path algorithm simply applies Dijkstra's algorithm twice. This algorithm can significantly increase the routing time if the network topology is large. A shortest-distance path algorithm selects a path which has the minimum "distance" which is derived from any distance function. A dynamic-alternative path algorithm selects a path using the widest-shortest path algorithm while imposing hop count restrictions on the nodes being selected.

Iwata and et al. propose a new QoS routing algorithm for the PNNI protocol that can find a path guaranteeing several QoS parameters requested by users [11]. They proposed a pre-calculation of the path. The pre-calculation is designed to find a path which uses as few network resources as possible with a sufficiently short connection setup delay. This is done by pre-calculating paths using no knowledge of the user's request. The paths are calculated beforehand, and when there is a user's request, the pre-calculated path is returned in response to the user's request immediately. However, from their experiments, it has been shown that the pre-calculated path scheme did not improve much of the call blocking probability of the PNNI routing request. In addition, they also proposed a combination of an on-demand path selection with a single criterion, such as the administrative weight and available bandwidth, with the pre-calculated path selection. First, the pre-calculated path is returned to the user's routing request. If the routing with all the pre-calculated paths fails, the path which is calculated by using the on-demand path selection algorithm based on a single criterion is returned. If the routing still fails, there is an option whether this user's request is rejected, or another path which is calculated by using the on-demand path selection based on another single criterion is returned. At this point, if the routing still fails, the user's request is rejected. From their experiments, it has been shown that using a three-path routing scheme, as described above, significantly increases the call setup time and hardly improves the call blocking probability of the routing request.

Fang et al. investigate the performance of the PNNI routing protocol in a large ATM network [10]. Their experiments are focused on the inaccuracy of routing information due to two factors: the topology aggregation and delayed PTSE updates. Their experiments are done on a virtual PNNI testbed written in the new network description language *TeD* [18, 20]. They use the C++ version of the *Ted* software system that deploys the Georgia Tech Time Warp (GTW) system as the underlying simulation engine [7, 19]. They found that the routing information inaccurancy in large PNNI networks is affected by the PTSE update interval and topology aggregation. In addition, the effect of crankback tends to be more important when the routing at each switch is less accurate.

Neve and Mieghen proposed a multiple QoS routing algorithm, called TAMCRA, which stands for Tunable Accuracy Multiple Constraints Routing Algorithm [17]. TAMCRA has one integer parameter k which can increase the accuracy of a returned shortest path at the expense of calculation time. The value k is defined to reflect the number of shortest paths. Their work was developed from Jeffe's algorithm [12]. The principle of TAMCRA is to find the shortest path with two constraints. However, the constraints have to be additive. Therefore, TAMCRA is not suitable to find a path with a constraint that is not additive, such as the available bandwidth of the link.

Sun and Langendőrfer proposed a new distributed unicast routing algorithm which can find a loop-free delay-constrained path with a small message complexity [23]. They have chosen cost and delay as the routing metrics, and both are additive. Even though the link cost can be chosen to be a function of residual bandwidth, residual buffer space, and estimated delay bound of the link [25], their algorithm is unable to *directly* find a path whose constraints are not additive, such as a maximum bandwidth.

There are many research papers that show the performance of QoS routing using the different approaches. However, our performance evaluation exper-

iments use the simulation tool which has been developed at the Information and Telecommunication Technology Center (ITTC) of the University of Kansas. This simulation tool is part of a comprehensive architecture which supports a common interface for simulations, emulation, and real-time ATM network experiments. It is supported on *Bellcore*'s *Q.Port* signaling software. This software includes UNI signaling messages, the data link Qsaal layer, and the Q93B layer. Our simulation tool is built on this software with all the necessary protocol stacks in a real ATM switch. Since the simulation tool implementation shares about 90% of the real ATM switch signaling software, the results obtained from the simulation are expected to closely match those obtained using real network experiments. This is the most advantageous feature of our simulation tool over other simulation tools.

Therefore, the aim of this thesis is twofold.

- Showing our simulation tool can support simulations of large scale networks using our multiple criteria routing algorithms.
- Giving multiple criteria routing results for single peer group PNNI ATM networks.

Chapter 3

Implementation

As stated in the PNNI specification, the PNNI routing algorithm was developed from Dijkstra's original algorithm [3]. However, it provides a routing method based on a single routing criterion. Thus, the "best" route found by this method might not be the best because in the ATM network there are many criteria for the route that can be considered, such as link bandwidth, link delay, and number of hops. For example, using the original routing algorithm specified in the PNNI specification, the route returned has a maximum bandwidth, but it might have a very high delay. Therefore, our solution is to compromise among two or more criteria of the routing policy.

Since it is known that routing with more than one criterion an NP-complete problem [24], we introduce a heuristic to compromise among two or more criteria.

In this chapter, we discuss the routing criteria we used for our solution in Section 3.1. Our approach to the solution is described in Section 3.2. Section 3.3 explains our implementation of the multiple criteria routing algorithm (MCRA) for on-demand routing.

3.1 Routing Criteria

In order to find a route to fulfill both a call requirement and reasonable use of network resources, we need to carefully specify multiple routing criteria. Conventionally, only one routing criterion is used to find a route, examples of which include a route with: minimum hop, maximum bandwidth, or minimum delay, as a criterion. The routing algorithm using a single routing criterion has a disadvantage. For example, if there are many calls that have to be routed through the same destination, using the minimum hop as the routing criteria, some calls will go through the same route, *Route A*, and allocate network resources, such as the bandwidth of each link along the route. Many calls are routed through *Route A* until any link within *Route A* cannot support the call request. However, there might be another route with the same number of hops as *Route A* which has a more available bandwidth. This example shows that a single criterion routing algorithm often does not balance the use of network elements (such as links). In addition, in a link there are many kinds of resource information, i.e., bandwidth, hop count, and delay to be used to make a routing decision. However the single criterion routing algorithm does not compromise those available resources. Therefore, the route returned by this routing algorithm might not give the best route to the user request. For this reason, *multiple* routing criteria are necessary to solve the problem above.

In this section, we propose routing criteria for our routing algorithms. The multiple criteria routing algorithms using *two* routing criteria are shown in Figure 3.1. The row of the table shows the primary criterion, and the column of the table shows the secondary criterion. Marks in the table show the routing algorithms we propose. For example, the algorithm in the first row and third column is the minhop_widest routing algorithm. This algorithm finds a route based on the maximum bandwidth as the primary criterion and the minimum delay as the secondary criterion.

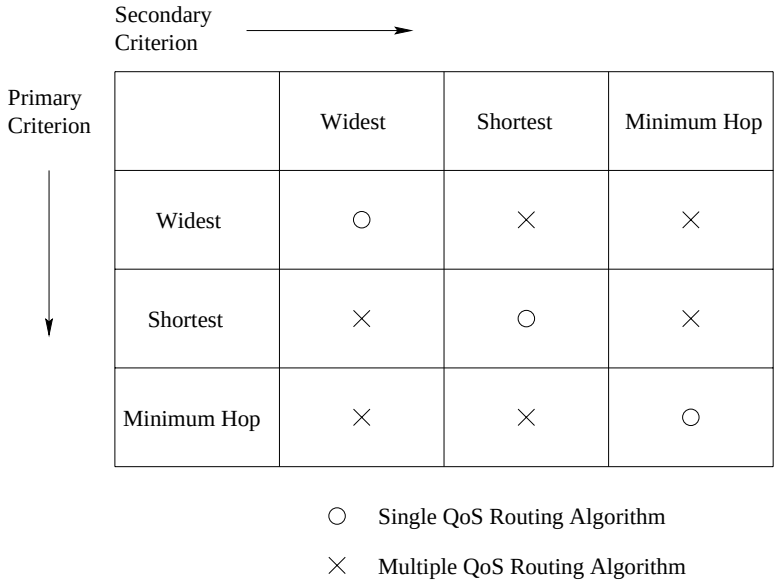

Figure 3.1: The Multiple Criteria Routing Algorithm Using Two Routing Criteria

In addition, the multiple criteria routing algorithms using *three* routing criteria are:

- a path with a minimum hop count as a primary objective, a minimum delay as a secondary objective, and the maximum bandwidth as a tertiary objective (called ”widest-shortest-minhop”)
- a path with a minimum hop count as a primary objective, the maximum bandwidth as a secondary objective, and a minimum delay as a tertiary objective (called ”shortest-widest-minhop”)

We have decided to focus on the maximum bandwidth, minimum delay, and minimum hop count to be the criteria for our routing algorithms. We did not yet consider a loss rate in the routing algorithms implemented in our simulation tool and in our simulation experiments. However, we could easily use a multiplicative metric easily as well. The routing algorithm using the loss rate as a routing criterion can be implemented by converting the loss rate metric from a multiplicative object to an additive object by using a logarithmic function. For ex-

ample, the loss rate of path P, L_P, is the product of the link loss rates (l_n, n = link number) along the path.

$$\text{Loss Rate } (L_P) = 1 - [(1 - l_1)(1 - l_2)(1 - l_3)] \tag{3.1}$$

$$L_P = 1 - \exp \ln[(1 - l_1)(1 - l_2)(1 - l_3)] \tag{3.2}$$

$$L_P = 1 - \exp[\ln(1 - l_1) + \ln(1 - l_2) + \ln(1 - l - 3)] \tag{3.3}$$

After we change the loss rates into a logarithmic function, we can simply add them together. Thus, we can use our routing algorithm which is designed for any additive parameter to be used with a loss rate metric also. However, we consider the loss rate to be a hard constraint. In addition, we did not consider the jitter to be a criterion for our algorithm, but we have kept it as a hard constraint.

3.2 Our Approach to Route Computation

In this section, we describe our solution to the problems described in Section 3.1, and explain the routing mechanism when a user's call arrives at the first node. Figure 3.2 shows the route computation flow chart specifying the network's reaction to a call request. The routing algorithm used to construct the Switch Virtual Circuit (SVC) path is based on user demand, and it is called an "*on-demand*" path routing algorithm. The algorithm finds a path based on the link-state information of the network including the recent status of each network link, such as the quintuple (*bw, d, j, l, aw*) and other nodal topology information. The link-state information is maintained in a topology database at each node. The topology database is updated periodically by a flooding mechanism. A node floods the network with a PNNI status message either when significant changes in the status occur or after a pre-specified time interval. The significant change is determined by the differ-

ence of the network resource, for example, the link available bandwidth of the last call made from the available bandwidth for the next call to be made. The range of difference which indicates whether the change is significant is a part of the routing parameter set network which is specified by a network operator.

In Figure 3.2, when a call request arrives, we use the on-demand path routing algorithm to find a path to a single specific destination. There are two parameters related to the on-demand path routing algorithm, the routing criterion, and the routing policy. The routing criterion specifies the maximum or the minimum of the network parameters, e.g., maximum bandwidth and minimum delay. The routing policy is composed of one or more routing criteria, e.g., the minimum-delay, maximum-bandwidth policy. In an on-demand routing algorithm, we first conduct a *pruning procedure* to remove the links which are unable to support the user's request, and also links within a failed path. The failure of the path can be either because it is unable to satisfy the user's QoS or because it is returned from the *Crankback procedure*.

After the pruning procedure, we have the topology in which all links tend to support the user's request. We then use an on-demand path routing algorithm to find a possible path that fulfills both the user's call request and the routing policy. At this step, a route may not be found because some links are pruned, and there is no path from the source node to the destination node. The call is rejected because no path is found. If there is a possible path found, the path is checked to see whether the path can satisfy the user's QoS needs. If the path is unable to support the QoS needs, then all the links of the path are pruned in the pruning procedure, and the routing procedure is started over again. Otherwise, the path is selected and the Call Setup procedure is called. Then, the Call Admission Control (CAC) procedure is performed to reserve switch (or node) resources along the path for the user's call request. If a switch is unable to support the user's call request, the *Crankback* occurs, and the call request is returned to the source switch for

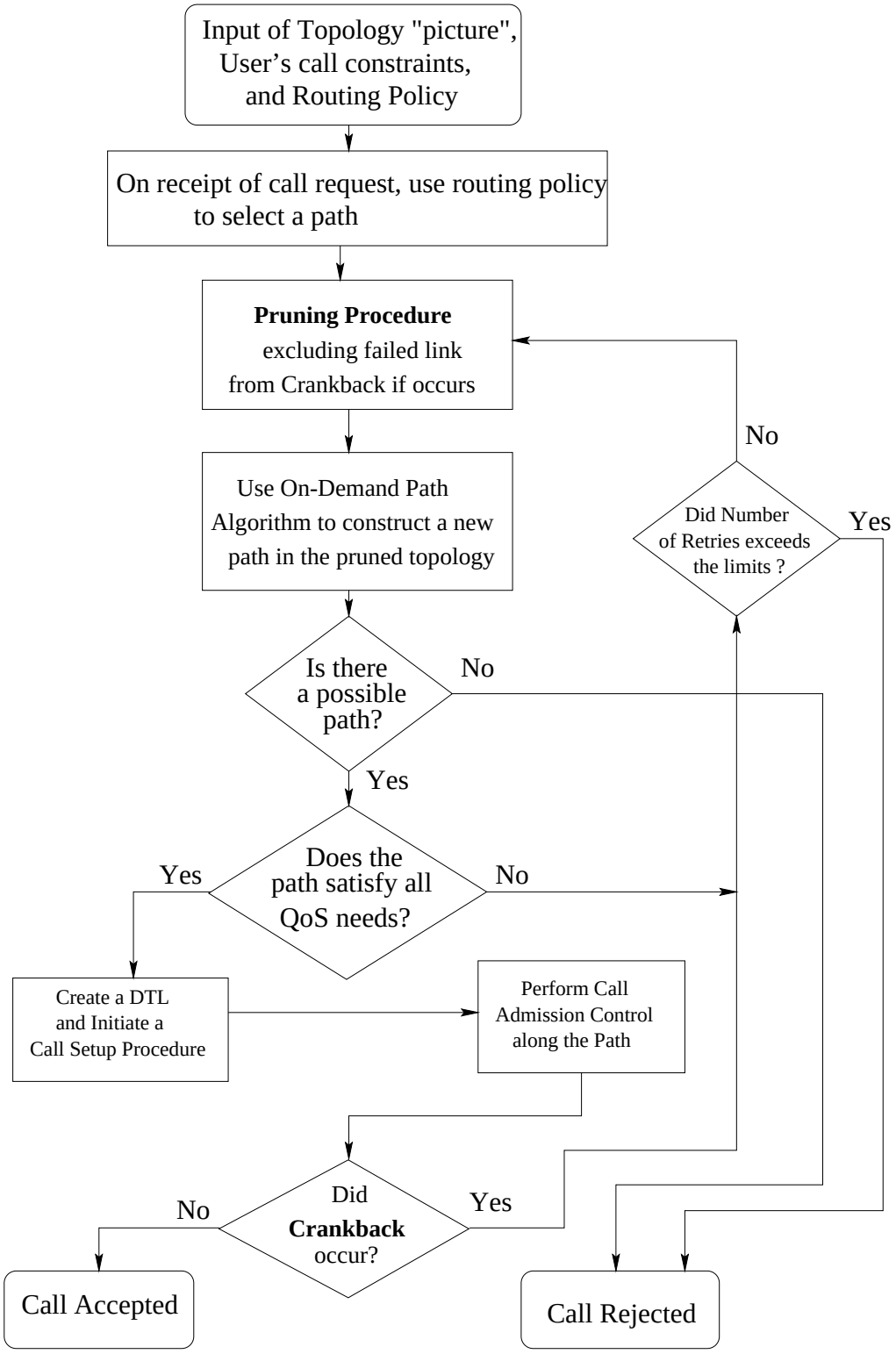

Figure 3.2: Route Computation Flow Chart

re-routing. The routing procedure is started over again.

The call is routed over the feasible path found. Calls may be rejected for two reasons: either because there is no feasible path after the algorithm tries to find a path which can support all the QoS requirements or the number of the retries exceeds the limit which is specified by a network operator or an end user. Recall that our routing algorithms use the local topology database maintained at each node to find paths. However, since the information in this database is updated periodically, it may be out-of-date. In order to validate the most recent status information about available resources, the PNNI call setup procedure sends a *SETUP* message to each node along the path selected by the routing algorithm. A nodal Connection Admission Control (CAC) procedure is used at each node to determine if the node currently has enough available resources to accept the call. If there are sufficient resources at this node, then the call setup message will be forwarded to the next node along the selected path. This procedure is repeated until all the nodes along the path have been checked. If any of these nodes cannot support the call request, the condition called *Crankback* occurs. Crankback is the mechanism that returns the call setup message to its source with the cause of setup failure. The call return with the cause of failure message is used to alter the search for an SVC route for this call if the call setup time has not expired or the number of retries does not exceed the limit. The search for a route avoids the failed link which is indicated in the returned message of the call failure.

3.3 Dynamic On-Demand Routing

The procedure for on-demand routing is divided into two steps. The first is the construction of a network representation from the topology database, and the pruning of links in the representation is to eliminate the links which are unable to support the user's request. Links are pruned following the mechanism as speci-

fied in the PNNI specification [3]. The mechanism to prune all links, which cannot support a user's call requirements, is called Generic Connection Admission Control (GCAC). The GCAC mechanism is explained in the Section 3.3.1.

In case of a call failure, the Crankback mechanism is used to find an alternate path. A failed link or a failed node is indicated in the *RELEASE* message which is returned from the node, where all of its links cannot support the user's call requirements to the source node. The failed link is pruned before finding another possible path.

After all of the links that cannot support the user's request are pruned in the network representation, an on-demand routing algorithm is used to find a feasible path within our pruned representative of database topology. The algorithm for finding a feasible path for on-demand routing is explained in Section 3.3.3.

3.3.1 Path Pruning with Generic Connection Admission Control (GCAC)

PNNI routing shall determine paths that satisfy performance constraints but are not necessarily optimized with respect to any predetermined performance criteria. Performance constraints may be implemented in one of three ways: link constraints, node constraints, and path constraints. For link constraints, non-additive link-state parameters are used. For node constraints, non-additive nodal state parameters are used. For path constraints, additive link-state parameters are needed.

Link constraints and node constraints are used to prune the network graph during the path selection. PNNI routing supports link constraints and node constraints implementation of (1) Cell Loss Ratio (CLR) and (2) generic CAC parameters (i.e. Available Cell Rate (AvCR), Cell Rate Margin (CRM) and Variance Factor (VF)).

As part of providing quality of service (QoS) and throughput guarantees, a switching system performs the connection admission control (CAC) during a con-

nection setup phase to determine if a connection request can be accepted without violating the existing connections' QoS and throughput requirements. To enable the routing to produce paths that are likely to be accepted, it is necessary for the switching systems to advertise some information about their internal CAC states. However, requiring switching systems to expose detailed and up-to-date CAC information may result in a high volume of unacceptable traffic. Furthermore, the CAC is not subject to standardization in the PNNI specification; and thus, it is not practical for the switching systems to advertise their potentially different detailed CAC states [3].

The Generic Connection Admission Control (GCAC) solves this problem by allowing switching systems to advertise CAC information that is generic (i.e., independent of the actual CAC used in the switching systems) and compact, but yet rich enough to support any CAC. GCAC defines a set of parameters to be advertised and a common admission interpretation of these parameters. This common interpretation is in the form of a generic CAC algorithm to be performed during path selection to determine if a link or node can or cannot be included for consideration. The algorithm uses the advertised GCAC parameters (available from the topology database) and the characteristics of the connection being requested (available from signalling) to determine if a link/node will be likely to accept or reject the connection. A link/node is included if the GCAC algorithm determines that it will be likely to accept the connection and excluded otherwise.

PNNI routing supports path constraints implementation of Administrative Weight (AW), Maximum Cell Transfer Delay (MaxCTD) and Cell Delay Variation (CDV). In the 1.0 version of the PNNI specification, the GCAC algorithm supports only CBR and VBR services [3].

Traffic Combination	*PCR*	*SCR*
1 and 3	PCR(CLP=0)	PCR(CLP=0)
2 and 4	PCR(CLP=0+1)	SCR(CLP=0)
5	PCR(CLP=0+1)	PCR(CLP=0+1)
6	PCR(CLP=0+1)	SCR(CLP=0+1)

Table 3.1: PCR and SCR values used in GCAC for CLP=0 traffic [3]

3.3.2 GCAC Algorithm for CBR and VBR Services

In the GCAC, a connection is characterized by two traffic parameters: Peak Cell Rate (PCR) and Sustainable Cell Rate (SCR). These parameters are used to verify whether a link in the network can support the user's call request. PCR and SCR values are different when the Cell Loss Priority (CLP) is set to "zero" or "one". Note that the CLP is a bit in the ATM cell header that indicates two levels of priority for ATM cells. CLP=0 cells are higher priority than CLP=1 cells. CLP=1 cells may be discarded during periods of congestion to preserve the CLR of CLP=0 cells. Therefore, the selection of PCR and SCR for the GCAC algorithm is determined by the one of the six traffic combinations that is described in the UNI standard version 3.1 [4]. The PCR and SCR parameter selection is described in Section 3.3.2.1, and Section 3.3.2.2 describes the algorithm for GCAC mechanism.

3.3.2.1 PCR and SCR Parameter Selection for GCAC

Due to the traffic in the network, the PCR or SCR for the GCAC can be different. For any topology element along a candidate path, if the Resource Availability Information Group (RAIG) Cell Loss Priority (CLP) bit is set to 0, this indicates that it allocates resources based on CLP=0 traffic. In this case, PCR and SCR values of the connection used in GCAC are set according to Table 3.1.

If the RAIG CLP bit is set to 1, this indicates that it allocates resources based on the CLP=0+1 traffic. In this case, the PCR and SCR values of the connection

Traffic Combination	*PCR*	*SCR*
1, 2, 3 and 4	PCR(CLP=0+1)	PCR(CLP=0+1)
5	PCR(CLP=0+1)	PCR(CLP=0+1)
6	PCR(CLP=0+1)	SCR(CLP=0+1)

Table 3.2: PCR and SCR values used in GCAC for CLP=1 traffic [3]

used in the GCAC are set according to Table 3.2.

3.3.2.2 Algorithm for GCAC mechanism

Generally, PCR and SCR are retrieved from a connection request, and these two parameters are used in the GCAC mechanism. In PNNI 1.0, there are two choices of GCAC mechanism: complex GCAC and simple GCAC [3]. The use of either a complex GCAC or a simple GCAC is based on the GCAC parameters, such as AvCR, CRM and VF. When AvCR, CRM and VF are advertised for a given link, the complex GCAC algorithm is recommended for use. Otherwise, a simple GCAC is used when only the AvCR is advertised. Note that AvCR (Available Cell Rate) is the current link rate in cells/sec at which a source is allowed to send cells. CRM (Cell Rate Margin) is a measure of the differences between the effective bandwidth allocation and the allocation for a sustainable rate in cells per second. In addition, VF (Variance Factor) is a relative measure of the cell rate margin normalized by the variance of the aggregate cell rate on the link.

In a complex GCAC, the steps used to include or exclude links are shown in Program 3.1.

Note that if a SCR is not specified in the Traffic Descriptor IE, then PCR = SCR and only step 1 and 2 need to be performed. Also, note that when CRM and VF are zero, step 3 will always result in ”include.” On the other hand, when VF is infinity, step 3 will always result in ”exclude.”

If only AvCR is advertised, the use of a simple GCAC is recommended. In

Program 3.1 Complex GCAC Algorithm [3]

```
Step 1:  If AvCR(i) >= PCR, include the link i; end;
Step 2:  If AVCR(i) < SCR, exclude the link i; end;
Step 3:
  If [AvCR(i) - SCR] x [AvCR(i) - SCR + 2CRM(i)]
                                        >= VF(i) x SCR(PCR - SCR)
     include the link i;
  Else
     exclude the link i;

Step 4:  End;
```

a simple GCAC, the following step is used to include or exclude links.

Program 3.2 Simple GCAC Algorithm [8]

```
Step 1. If AvCR >= C
           include the link;
        Else
           exclude the link;

Where C is given by
if      (PCR <= 4 x SCR),  C = (PCR + SCR) / 2
else if (PCR <= 16 x SCR), C =  PCR / 8 + 2 x SCR
else if (PCR <= 64 x SCR), C = (3 x PCR + 465 x SCR) / 128
else    C = (13 x SCR + 4413 x SCR) / 1024
```

3.3.3 Algorithm for On-demand Path Computing

We classify the on-demand routing into two cases: routing with a single criteria requirement and routing with multiple criteria requirements. The single criteria routing algorithm takes one of the criteria mentioned in Section 3.1. The returned path satisfies the single criterion, such as the maximum bandwidth, a minimum delay, or a minimum number of hops. The multiple criteria routing algorithm considers two or more of the criteria. The returned path satisfies the multiple criteria, such as the maximum bandwidth and a minimum number of hops, the maximum

bandwidth and a minimum delay, and a minimum delay and a minimum number of hops.

3.3.3.1 Single-Criteria Routing Algorithm

For single-criteria routing, we can use the original version of Dijkstra's algorithm to find a path whose single criteria requirement is minimized [21]. Dijkstra's algorithm solves the problem of finding the shortest path by looking at a *single* cost from a point in a graph (the source) to a destination. Dijkstra's algorithm is shown in Program 3.3 [5].

In Program 3.3, in which the *Relaxation* function performs for each vertex $v \in V$, we maintain an attribute $d[u]$, which is an upper bound on the weight of a shortest path from source s to v. We call $d[v]$ a shortest-path estimate. We initialize the shortest-path estimates and predecessors by the procedure on Lines 1-6. After initialization, we get $\pi[v] = \text{NIL}$ for all $v \in V$, $d[v] = 0$ for $v = s$, and $d[v] = \infty$ for $v \in V - s$ The process of *relaxing* an edge (u, v), as shown on Lines 7-11, consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating $d[v]$ and $\pi[v]$. A relaxation step may decrease the value of the shortest-path estimate $d[v]$ and update v's predecessor field $\pi[v]$. The code on Lines 7-11 performs a relaxation step on edge (u, v).

In general, Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are non-negative. Therefore, we assume that $w(u, v) \geq 0$ for each edge $(u, v) \in E$.

Dijkstra 's algorithm maintains a set S of vertices whose final shortest path weights from the source s have already been determined. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$ where $\delta(s, v)$ is the shortest path weight between source node s and node v. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S, and relaxes all edges leaving u. In the Program 3.3, we maintain a priority queue Q that contains all the

Program 3.3 Dijkstra's Algorithm [5]

```
1   Initialize_single_source (G, s) {
2       for each vertex v ∈ V[G]
3           d[v] ← ∞
4           π[v] ← NIL
5       d[s] ← 0
6   }
7   Relaxation (u, v, w) {
8       if d[v] > d[u] + w(u,v)
9           then d[v] ← d[u] + w(u,v)
10              π[v] ← u
11  }
12  Extract-Min (Q) {
13      for each vertex v ∈ Q
14          if d[v] is minimum
15              return v
16  }
17  Dijkstra (G, w, s) {
18      Initialize_single_source(G,s)
19      S ← ∅
20      Q ← V[G]
21      while Q ≠ ∅
22          u ← Extract-Min(Q)
23          S ← S ∪ {u}
24          for each vertex v ∈ Adjacent[u]
25              Relaxation(u,v,w)
26  }
```

vertices in $V - S$, keyed by their d values. The Program assumes that graph G is represented by adjacency lists.

In Program 3.3, Line 18 performs the usual initialization of d and π values using the function shown on Lines 1-6. Line 19 initializes the set S to the empty set. Line 20 then initializes the priority queue Q to contain all the vertices in $V - S = V - \emptyset = V$. Each time through, the **while** loop of line 21-26 iterates exactly $|V|$ times.

Note that (on Lines 7-11 in the *Relaxation* function) Dijkstra's algorithm uses the *addition* operation to update the distance from a source to any node. For example, we can use the Dijkstra's algorithm to find a path with minimum delay since it is an additive metric. However, finding a maximum bandwidth path would be problematic using the Dijkstra algorithm since the bandwidth is not additive.

Therefore, we modified *Dijkstra algorithm* to solve the problem of using a non-additive cost of the link [21], and we renamed it as *D_Widest path* algorithm. The *D_Widest path algorithm* finds a path between two nodes with maximum bandwidth. The D_Widest path algorithm is shown in Program 3.4.

Note that in Program 3.4, the *Relaxation* function is performed for each vertex, $v \in V$. We maintain an attribute $d[u]$, which is the *lower* bound on the bandwidth of the widest path for source s to v. We call $d[v]$ here a widest path estimate.

Here are the details of the widest path algorithm as shown in Program 3.4. The widest path algorithm is very similar to the Dijkstra's algorithm. First, we initialize the widest path estimates, $d[v]$, and predecessors, $\pi[v]$, shown on line 1-4. In addition, we initialize the widest estimate of source to be infinity as shown on line 5. After the initialization, we get $\pi[v] = \text{NIL}$ for all $v \in V$, $d[v] = \infty$ for $v = s$, and $d[v] = \infty$ for $v \in V - \{s\}$.

Lines 7-11 show the *Relaxation* process on edge (u, v). It consists of testing whether we can improve the widest path to v found so far to be going through u and, if so, updating $d[v]$ and $\pi[v]$.

Program 3.4 D_Widest Path Algorithm

```
1   Initialize (G, s) {
2       for each vertex v ∈ V[G]
3           d[v] ← NIL
4           π[v] ← NIL
5       d[s] ← ∞
6   }
7   Relaxation (u, v, bw) {
8       if d[v] < min{d[u], bw(u,v)}
9           then d[v] ← min{d[u], bw(u,v)}
10                  π[v] ← u
11                  Q ← {v}
11   }
12   Extract-Max (Q) {
13       for each vertex v ∈ Q
14           if d[v] is maximum AND v ∉ S
15               return v
16   }
17   D_Widest (G, bw, s, d) {
18       Initialize(G,s)
19       S ← ∅
20       Q ← {s}
21       for each vertex v ∈ Adjacent[s]
22           Q ← {v}
23           d[v] ← bw(s,v)
24       while Q ≠ ∅
25           u ← Extract-Max(Q)
26           S ← S ∪ {u}
27           for each vertex v ∈ Adjacent[u]
28               Relaxation(u,v,bw)
29   }
```

The *Extract-Max* process shown on Lines 12-16 finds the vertex, v, which has the maximum widest-path estimate. The vertex is selected from the priority queue, Q which contains the vertices that are potentially to be determined and keyed by their d values.

In the *D_Widest* function, first, we initialize all the parameters using the *Initialize* function shown on Line 18. Then, on Lines 19-20, we insert the source into a priority queue, Q, and we also add the adjacent vertices to the source node into Q, and initialize the bandwidth between the source node and adjacent node into their widest estimates as shown on line 19-23.

In the **while** loop, first, we extract the vertex with the maximum widest-path estimate, and add it to a set S of vertices whose final widest path from the source s have already been determined. That is, for all vertices $v \in S$, we have $d[v] = bw(s, v)$ as the widest path weight between source node S and vertex v. The algorithm repeatedly selects the vertex $u \in V - S$ with the maximum widest-path estimate, inserts u into S, and relaxes all edges leaving u. Each time through the **while** loop of line 23-28 iterates exactly $|V|$ times.

The result of the widest path algorithm is the *widest* path from a source to any destination which can be constructed from $\pi[v]$.

In conclusion, we use Dijkstra's algorithm to find a path whose criterion is additive, such as delay and number of hops. In addition, we modified Dijkstra's algorithm for finding a path whose criterion is non-additive (maximum) bandwidth and called the *D_Widest Path* algorithm.

3.3.3.2 Multiple Criteria Routing Algorithms

In this section, we describe algorithms for multiple criteria routing, which we developed and which we evaluated. We described our criteria for multiple criteria routing in Section 3.1.

We divided our algorithms into three groups. The first group has algorithm-

s for which a minimum additive metric such as a delay and number of hops is the primary criterion and the maximum bandwidth is the secondary criterion. The second group has algorithms with the maximum bandwidth as the primary criterion, and any minimum additive metric as the secondary criterion. The last group has algorithms using three criteria for selecting a path.

The first group consists of an algorithm for a path routing in which a *minimum additive metric* such as a delay or hop count is the *primary criterion*, and the *maximum bandwidth* is the *secondary criterion*. Examples of such algorithms include the widest-shortest algorithm and the widest-min_hop algorithm. First of all, we define two cost functions: the minimum "distance", such as a hop count or a delay, as the primary cost and the maximum bandwidth is the secondary cost. In addition, we modify Dijkstra's algorithm to consider the maximum of the secondary criterion (bandwidth) when there is more than one node having an equal amount of the first criterion (any additive metric). An example of the modified version of Dijkstra's algorithm for widest-shortest algorithm is shown in Program 3.5. The link delay is given as the primary cost indicated as d_1, and the link bandwidth is given as the secondary cost indicated as d_2.

The second group consists of an algorithm for a path routing in which *maximum bandwidth* is a *primary criterion*, and any *minimum additive metric* is a *secondary criterion* such as, shortest-widest and minhop-widest. First of all, we defined two functions of costs. The bandwidth is the primary cost, and any additive metric, e.g. delay, is the secondary cost. Then, we used the *D_Widest* path algorithm, the modified version of Dijkstra's algorithm, to find the maximum bandwidth path as described in Section 3.3.3.1. After we got the maximum bandwidth of the possible path from the network, we extensively used Dijstra's algorithm to find a minimum cost route with a bandwidth equal to or more than the maximum bandwidth from the first pass. This algorithm needs two passes to find the minimum additive cost and maximum bandwidth path.

Program 3.5 Widest-Shortest Path Algorithm

```
1    Initialize-wide-short (G, s) {
2        for each vertex v ∈ V[G]
3            d1[v] ← ∞, d2[v] ← NIL
4            π[v] ← NIL
5        d1[s] ← NIL, d2[s] ← ∞
6    }
7    Relaxation-wide-short (u, v, w, bw) {
8        if d1[v] > d1[u] + w(u,v)
9            then d1[v] ← d1[u] + w(u,v)
10               d2[v] ← min{d2[u], bw(u,v)}
11               π[v] ← u
12       else if d1[v] == d1[u] + w(u,v)
13           if d2[v] < min{d2[u], bw(u,v)}
14               then d1[v] ← d1[u] + w(u,v)
15                   d2[v] ← min{d2[u], bw(u,v)}
16                   π[v] ← u
19       else do nothing;
20   }
21   Widest-Shortest (G, w, bw, s) {
22       Initialize-wide-short(G,s)
23       S ← ∅
24       Q ← V[G]
28       while Q ≠ ∅
29           u ← Extract-Min(Q)
30           S ← S ∪ {u}
31           for each vertex v ∈ Adjacent[u]
32               Relaxation-wide-short(u, v, w, bw)
33   }
```

For example, for the shorest-widest path algorithm, we first used the D_Widest path algorithm to find a maximum bandwidth path in the network, assuming that its bandwidth is bw. After we knew the maximum bandwidth, we used Dijkstra's algorithm to find a minimum delay (shortest) route with bandwidth equal to bw or higher. The shortest-widest path algorithm is given in Program 3.6.

Program 3.6 Shortest-Widest Path Algorithm

```
1   Initialize-short-wide (G, s) {
2       for each vertex v ∈ V[G]
3           d2[v] ← ∞
4           π[v] ← NIL
5   }
6   Relaxation-short-wide (u, v, BW, w) {
7       if d1[v] > BW {
8           if d2[v] > d2[u] + w(u,v) {
9               then d2[v] ← d2[u] + w(u,v)
10                  π[v] ← u
11                  Q ← {v}
12          }
13      }
14  }
15  Shortest-Widest (G, bw, w, src, dest) {
16      D_Widest(G, bw, src, d1)
17      BW ← d1[dest]
18      Initialize-short-wide(G, src)
19      S ← ∅
20      Q ← V[G]
21      while Q ≠ ∅
22          u ← Extract-Min(Q)
23          S ← S ∪ {u}
24          for each vertex v ∈ Adjacent[u]
25              Relaxation-short-wide(u, v, BW, w)
26  }
```

Note that the single-pass link-state shortest-widest path algorithm given in Wang's paper does not always find the shortest-widest path [24]. For example, for the topology in Figure 3.3, the algorithm will select the upper path walking

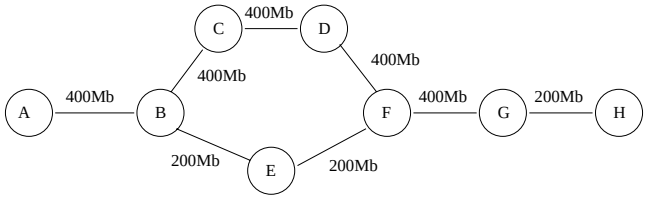

Figure 3.3: The Sample Network with Two Paths

through the links with bandwidth 400Mb. This is because when a link (e.g., the link connection to node H) with the lower bandwidth (e.g., 200Mb) has to be added to the path, the earlier shortest-widest segment may no longer be the shortest-widest one.

The third group consists of an algorithm for a path routing with three cost functions such as minhop-widest-shortest path, widest-shortest-minhop, and shortest-widest-minhop. First, we defined three cost functions, for instance, for shortest-widest-minhop routing algorithm, the number of hops as the primary cost, maximum bandwidth as the secondary cost, and delay as the tertiary cost. At the step of finding a node with the minimum number of hops, if more than one node has the same minimum number of hops, the algorithm will consider the node whose bandwidth (the secondary cost) is maximum. If more than one node has the same minimum number of hops *and* the maximum bandwidth, the algorithm will consider the node whose delay is minimum. The algorithm is finished when all the nodes are considered. The algorithm for routing with three criteria for this group is similar to, for example, the Widest-Shortest algorithm. However, we add one more criterion into the Widest-Shortest algorithm. An example of the shortest-widest-minhop is shown in Program 3.7. The hop count is given as the primary cost indicated as d_1, the link bandwidth is given as the secondary cost indicated as d_2, and the link delay is given as the tertiary cost indicated as d_3.

In conclusion, we divided our multiple criteria routing algorithms into three groups.

- The routing algorithm with any additive metric as the primary criterion and

Program 3.7 Shortest-Widest-Min_Hop Path Algorithm

```
1   Initialize-short-wide-hop (G, s) {
2       for each vertex v ∈ V[G]
3           d1[v] ← ∞, d2[v] ← NIL, d3[v] ← ∞
4           π[v] ← NIL
5       d1[s] ← NIL, d2[s] ← ∞ d2[s] ← NIL
6   }
7   Relaxation-short-wide-hop (u, v, w, bw, h) {
8       if d1[v] > d1[u] + w(u,v)
9           then d1[v] ← d1[u] + w(u,v)
10                   d2[v] ← min{d2[u], bw(u,v)}
11                   d3[v] ← d3[u] + h(u,v)
12                   π[v] ← u
13      else if d1[v] == d1[u] + w(u,v)
14          if d2[v] < min{d2[u], bw(u,v)}
15              then d1[v] ← d1[u] + w(u,v)
16                   d2[v] ← min{d2[u], bw(u,v)}
17                   d3[v] ← d3[u] + h(u,v)
18                   π[v] ← u
19          else if d2[v] == min{d2[u], bw(u,v)}
20              if d3[v] > d3[u] + h(u,v)
21                  then d1[v] ← d1[u] + w(u,v)
22                       d2[v] ← min{d2[u], bw(u,v)}
23                       d3[v] ← d3[u] + h(u,v)
24                       π[v] ← u
25      else do nothing;
26  }
27  Shortest-Widest-Min_Hop (G, w, bw, h, s) {
28      Initialize-short-wide-hop(G, s)
29      S ← ∅
30      Q ← V[G]
31      while Q ≠ ∅
32          u ← Extract-Min(Q)
33          S ← S ∪ {u}
34          for each vertex v ∈ Adjacent[u]
35              Relaxation-short-wide-hop (u, v, w, bw, h)
36  }
```

bandwidth as the secondary criterion.

- The routing algorithm with bandwidth as the primary criterion and any additive metric as the secondary criterion.
- The routing algorithm with three criteria.

For the first group, we modified Dijkstra's algorithm to consider the second criterion (maximum bandwidth) when more than one node has the same primary criterion (any minimum additive metric). For the second group, we implemented the *D_Widest Path* algorithm to find a maximum bandwidth in the network. We then used the modified Dijkstra's algorithm to find the path whose bandwidth is not less than the bandwidth calculated by the D_widest algorithm and whose additive cost is minimum. For the third group, we modified Dijkstra's algorithm to consider three cost functions. The algorithm considers the tertiary criteria when more than one node has the same primary and / or secondary criteria.

Chapter 4

Experiment Scenarios

In this chapter, we explain how we tested our routing algorithms. Section 4.1 explains the topologies used in our tests, and the performance metrics used in our experiments are described in Section 4.2.

4.1 Topologies

It is known that routing algorithms perform differently in different types of network topologies. It is crucial to select appropriate network topologies in a simulation-based evaluation of routing algorithms. The most common factors that are important to consider when selecting topologies are: size, heterogeneity of link capacity, symmetry, and connectivity. Our focus is to understand how well a routing algorithm achieves a high network throughput for a variety of network topologies. Therefore, we studied the performance of our algorithms in four different types of topologies:

- Multiple Cluster Topology
  - 3-cluster topology
  - 8-cluster topology

- Conventional Edge-Core topology
  - Dense topology
  - Light topology

4.1.1 Multiple Cluster Topology

In a multiple cluster topology, nodes are clustered in a small strongly-connected group, and one node is connected to one or many nodes with a link such as the ATM OC-3 link. Each of the small groups is connected with high capacity links to another group. We propose two multiple cluster topologies: 3-cluster topology and 8-cluster topology.

The 3-cluster topology has 8 nodes in one cluster and 3 clusters in the topology. The links between two clusters (outside links) are OC-12 links, and there are three outside links between two clusters. The links in the cluster (inside links) are OC-3 links, and there are 10 inside links in each cluster. Each node is connected to one host which provides traffic to the network.

In the 8-cluster network, there are 3 nodes in one cluster and 8 clusters in the topology. Similar to the 3-cluster network, the links between the clusters are OC-12 links, and the links between nodes in the cluster are OC-3 links. Each cluster has three inside links, and each cluster has one OC-12 link connecting to the neighbor cluster. Each node is connected to one host system. The multiple cluster topologies and link characteristics are summarized in Table 4.1. The 3-cluster topology is shown in Figure 4.1, and the 8-cluster topology is shown in Figure 4.2. In Table 4.1, the connectivity is the total number of links divided by the total number of switches in the network.

Regarding the traffic of these two networks, the destination host requested by the source host is uniformly selected. The traffic is CBR-typed, and the call arrival time is selected from Poisson distribution with a mean of 5 seconds. The call

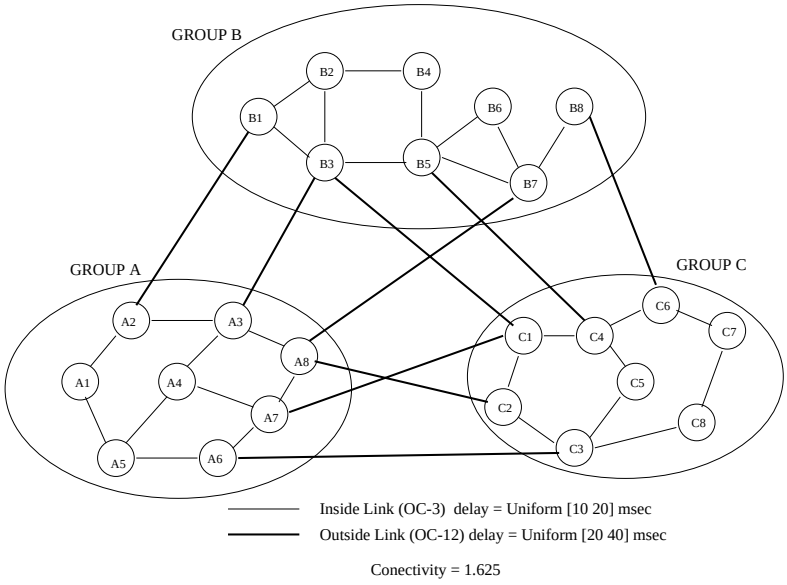

Figure 4.1: 3 Cluster Topologies

holding time is selected from a Poisson distribution with a mean ranging from 60 seconds to 100 seconds. The call bandwidth is selected from the uniform distribution with a mean ranging from 5 Mb to 50 Mb. There is one host connected to each node, and every host makes 100 calls. Therefore, there will be 2400 call connection requests in the network.

Topology	nodes	Link Type	links	Bandwidth	Delay	Connectivity
3-cluster	24	Outside Inside	9 30	OC-12 OC-3	Uniform[20 40] Uniform[10 20]	1.625
8-cluster	24	Outside Inside	13 24	OC-12 OC-3	Uniform[20 40] Uniform[10 20]	1.541

Table 4.1: Metrics for Multiple Cluster Topologies

4.1.2 Edge-Core Topology

The conventional edge-core topology (ECT) is commonly used in setting up a private ATM network. An edge switch is connected to an end-user system. It is analo-

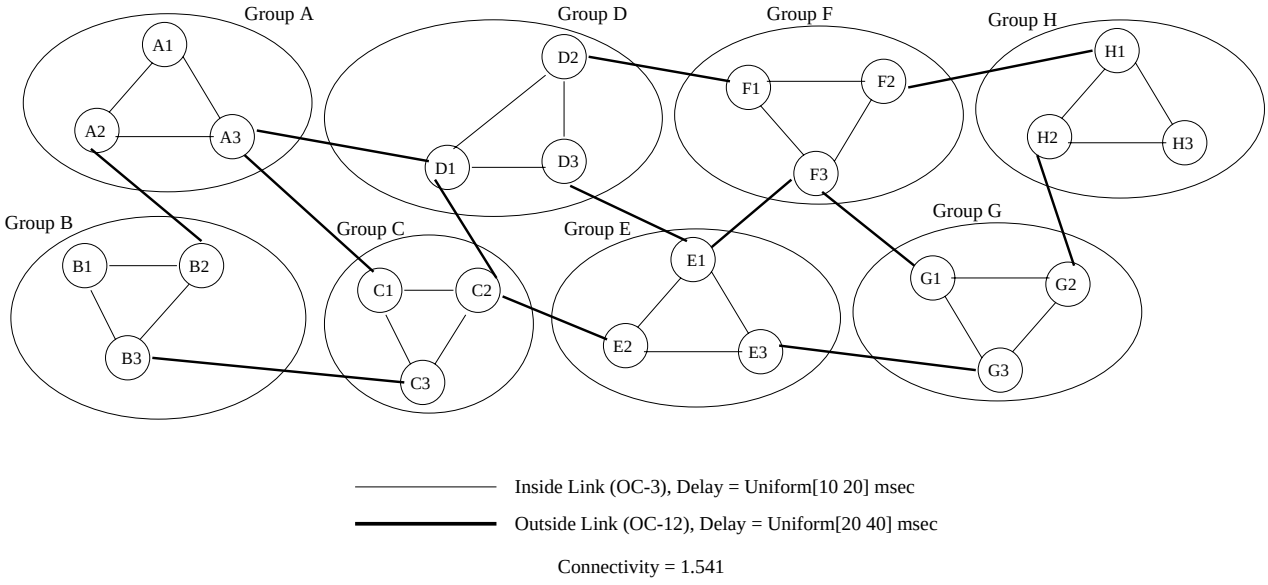

Figure 4.2: 8 Cluster Topologies

gous to a gateway router connecting the user system to the backbone network. The edge switch is connected to a core switch with high-speed links. The core switch is analogous to a high performance router. It connects to other core switches with high capacity links, and it also connects to edge switches with low capacity links. There can be many links connected from edge switches to a core switch.

In our experiments, we considered two types of edge-core topologies, ”dense” and ”light” edge-core topologies. The light edge-core topology has a lower number of links than the dense edge-core topology. The light and dense edge-core topologies are shown in Figure 4.3 and Figure 4.4, respectively.

In each edge-core topology, core nodes are connected to some of the core and edge nodes. Each edge node is connected to two core nodes and also connected to two hosts which generate traffic. No connection is allowed between two edge nodes. Each core node is classified into one of these following two categories:

- A large-scale node which is connected to other large-scale core nodes with a high capacity link with high link delay to support long distance traffic.
- A small-scale node which is connected to other core nodes with a small ca-

pacity link with small link delay.

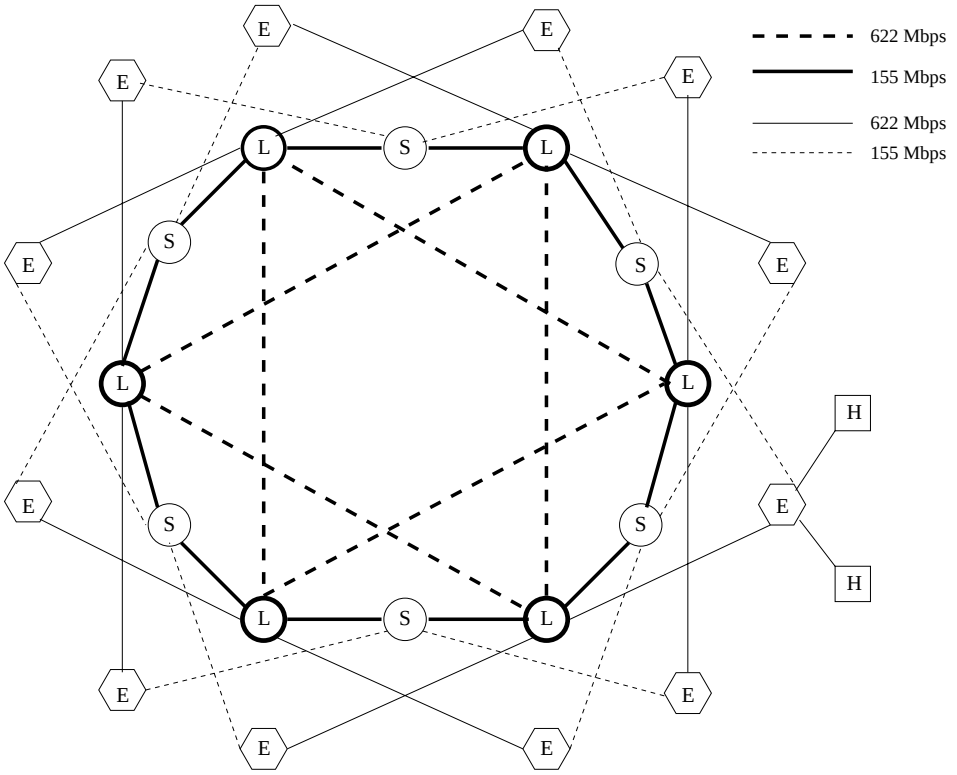

Figure 4.3: Light Edge-core Topologies

In Figure 4.3 and Figure 4.4, nodes in these two categories are labeled by the letter **S** for a small-scale node and **L** for a large-scale node. An edge node and host are labeled as **E** and **H**, respectively. The link metrics of the topologies in Figure 4.3 and Figure 4.4 are summarized in Table 4.2.

In summary, the two edge-core topologies are summarized in Table 4.3. The connectivity in Table 4.3 shows the total number of links divided by the total number of switches in the topology. Note that we select the link between Host (H) and Edge (E) node to have a higher bandwidth than that between the edge node to core node to avoid creating a bottleneck at the host. In addition, the sample script of the dense edge-core network that we used in our experiment to run our simulator is shown in Appendix A.1.

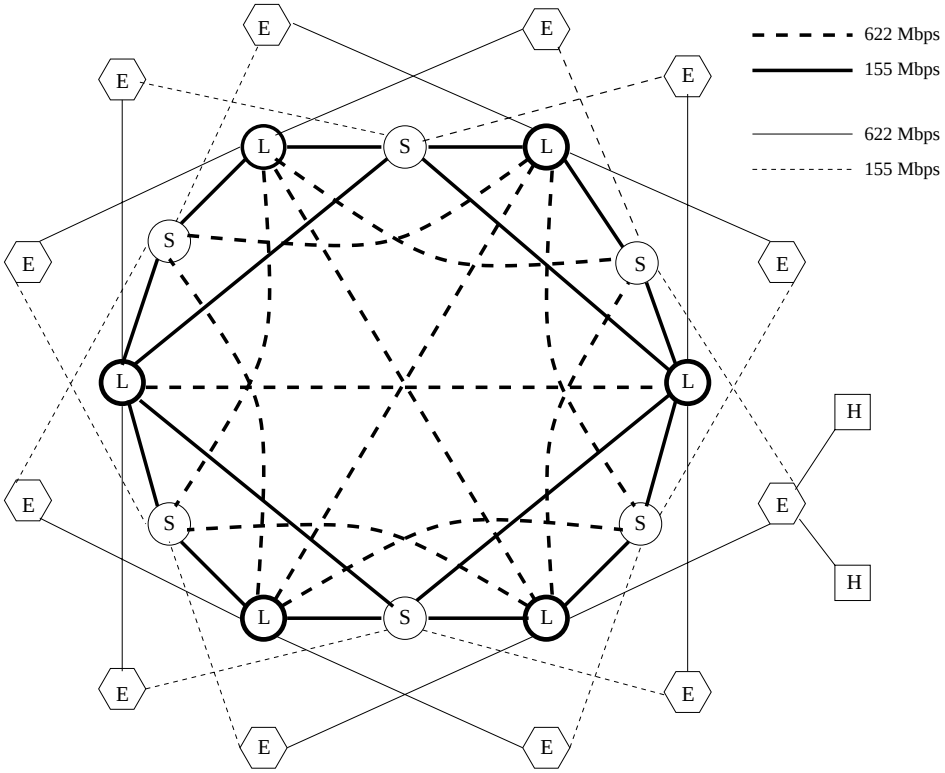

Figure 4.4: Dense Edge-core Topologies

In summary, we used two types of topologies in our experiments, multiple cluster and conventional edge-core topologies. We summarized all of the metrics of the topologies for our simulation experiments in Table 4.4.

Link	Capacity	Delay (ms)
L node to L node	OC-12	Uniform[25 40]
S node to L node	OC-3 or OC-12	Uniform[10 25]
E node to L node	OC-12	Uniform[5 10]
E node to S node	OC-3	Uniform[5 10]

Table 4.2: Link Metrics for Conventional Edge-core Topologies

Topology	Nodes	Link Type	Links	Connectivity
Light	12 core 12 edge	S-L L-L E-L E-S	12 6 12 12	1.75
Dense	12 core 12 edge	S-L L-L E-L E-S	24 3 12 12	2.125

Table 4.3: Summary of Edge-Core Topologies

Type	Total Number of Nodes	Number of Links	Number of Core Nodes	Link Bandwidth	Connect-ivity
3 Cluster	24	39	n/a	Within PG: OC-3, Among PG: OC-12	1.625
8 Cluster	24	24	n/a	Within PG: OC-3, Among PG: OC-12	1.541
Light Switch	24	42	12	See Figure 4.3	1.75
Dense Switch	24	51	12	See Figure 4.4	2.125

Table 4.4: Topologies Used in Our Simulation Experiments
Note: PG = Peer Group

4.2 Performance Metrics

We use the following performance metrics to compare the behaviors of our path selection algorithms in the different topologies:

- Average Call Blocking Rate
- Average Call Setup Time
- Routing Inaccuracy
- Link Utilization

The average call blocking rate is the common performance metric used to evaluate how well the routing protocol finds a route from a source to a destination. Depending on the routing criteria, the call blocking rate can be different because each routing algorithm allocates a path differently. Allocating the ”right” path tends to reduce the call blocking probability. The average call blocking rate metric is described in Section 4.2.1.

Another important metric is the call setup time. The routing protocol uses a routing algorithm to find a feasible route from a source to a destination. Any routing algorithm can find the best route yet using an exhaustive technique. However, if it takes too much time, the total performance of the network will be worse rather than better. Thus, the average call setup time is another important issue here, and it is described in Section 4.2.2.

In a large network, the information available for making routing decisions can be inaccurate because of a network delay. Therefore, the routing protocol which uses the information can make a mistake by giving an ”incorrect” route to the user call. Thus, the routing inaccuracy is another important performance metric, and it is discussed in Section 4.2.3.

Lastly, another important metric from the network planning point of view is the link utilization. When network engineers are designing the network topology,

their goal is to make sure the capacity of any link is enough for user traffic, which makes sure they will not have a traffic congestion problem in the near future. On the other hand, they also want to make sure that the capacity of any link will not be too high, in order to avoid spending too much money to achieve the goal. The link utilization which reveals the usage of the link is described in Section 4.2.4.

4.2.1 Average Call Blocking Rate

A call can be rejected in two cases. First, it is rejected because a feasible path with sufficient resources cannot be found by a routing algorithm at the source node. Secondly, the call is refused at an intermediate node because during the call connection period the resource availability on the selected path has changed since the time when the routing decision was made. In other words, the call is rejected because the source node network state information is out-of-date when the routing decision is made. The call is *crankbacked* to the source node to find an alternate route. The number of routing retries is limited by the network operator. Therefore, the call in this case can be rejected when the number of routing retries exceeds the limit. The call blocking (or failure) rate, therefore, is a good performance metric for studies of PNNI routing in connection-oriented networks like ATM Networks. We defined *call blocking rate* as:

$$\text{Call Blocking Rate} = \frac{\text{Total Number of Rejected Calls}}{\text{Total Number of Calls}}$$

4.2.2 Average Call Setup Time

Besides a call connection guarantee, a call connection time is also crucial. The call setup time is the duration from the time when the call request (*setup_req*) message is sent out by the host system to the time when either the setup confirm (setup_conf) message or the release indicate (*release_ind*) message is received at the host system.

The latter case happens when the call is rejected. Note that the call setup time of the failed call is not used to calculate the average call setup time.

Generally, most of the time spent in a call setup period is used not only to find a feasible path which can fulfill the call requirements but also to perform the call admission control at an intermediate node. If the call fails at the intermediate node, crankback will occur. The crankback procedure returns the call to the source node, and a new route will be given for another routing retry. If the routing retried call setup is successful, the total call setup time is the time spent for the first time routing and alternate routing retries if crankback occurs. In addition, the call setup time depends on what type of routing algorithms are used for routing. Therefore, to evaluate the performance of our routing algorithms, we defined the *call setup time* as:

$$\text{Average Call Setup Time} = \frac{\text{Total Call Setup Time}}{\text{Total Number of Successful Calls}}$$

4.2.3 Routing Inaccuracy

Since the information used for the routing decision at the source node can be out-of-date by the connection setup and routing information distribution delay, the routing algorithm can generate an incorrect path. The inaccuracy of the routing decision can thus cause a call connection to be rejected or re-routed. To evaluate the performance of routing algorithms, we defined *routing inaccuracy* as:

$$\text{Routing Inaccuracy} = \frac{\text{Number of Crankback Events}}{\text{Total Number of Call Requests}}$$

4.2.4 Link Utilization

Since the financial cost of the network link depends on its capacity, network engineers do not want to spend more money than required for the link that will have a low utilization. In addition, we do not want to spend too little money for a low capacity link that will have a high utilization, which means in the near future the link needs to be upgraded, and the additional cost will be added. Therefore, the link utilization is another important metric that reveals the usage of the link and the quality of the network. We defined the *link utilization* metric as:

$$\text{Link Utilization} = \frac{\text{Used Link Bandwidth}}{\text{Total Link Bandwidth}}$$

The used link bandwidth will be sampled periodically. The total link bandwidth is the maximum cell rate of the link.

Chapter 5

Experimental Results

In this chapter, we discuss the results from our experiments. In Section 5.1, we discuss the results of multiple criteria routing with bandwidth guarantees, where the most important criterion of these routing algorithms is the maximum bandwidth. Section 5.2 discusses the results of multiple criteria routing algorithms for the minimum delay services. These routing algorithms find a feasible route with the minimum delay as their most important criterion. Section 5.3 discusses link utilizations of a network. The results of link utilization using a single criterion routing are compared to those using a multiple criterion routing. Results of the experiments of alternate routing are shown in Section 5.4. This section shows the performance of each multiple criteria routing algorithm while the number of alternate routing retries increases. Section 5.5 shows the effects of the network density on the network performance. In this section, three networks with the different network density are tested to examine the effect of changing the network core density using different routing algorithms.

5.1 Multiple Criteria Routing for Bandwidth Guarantees

In this section, we study how to route traffic requiring bandwidth guarantees using multiple routing techniques to find a feasible path. In general, the common routing algorithm that has been used to find a feasible route for a user request is the *single source shortest path* (SSSP) algorithm which is based on Dijkstra's algorithm. The SSSP algorithm can find a route based on only one additive QoS cost such as a hop number or delay. In addition, the SSSP algorithm cannot be used to find a route based on a non-additive cost such as the bandwidth. Many path selection algorithms have been proposed, for example, shortest-widest [24], widest-shortest [9], and utilization-based path selection algorithm [14]. However, performance evaluations of these algorithms are not provided here.

In this section, we present the performance evaluation of three multiple criteria: minimum hop count, maximum available bandwidth, and minimum delay. Our evaluations consider the call blocking rate, the call setup time, and the routing inaccuracy. The rest of this section is organized as follows. The routing criteria and our algorithms are described in Section 5.1.1. Section 5.1.2 explains our experiment sets, and Section 5.1.3, Section 5.1.4, Section 5.1.5, and Section 5.1.6 discuss the performance of different routing algorithms in different networks.

5.1.1 Routing Criteria and Algorithms

In this section, we explain our four multiple criteria routing algorithms (MCRAs), whose routing routing criteria include the maximum bandwidth criterion (widest). First, the four MCRAs are shown below:

- *minhop-widest* path algorithm: a path with the maximum bandwidth among all feasible paths. If there are several such paths available, the one with the

minimum hop count is selected. If there are many such paths with the same hop count, one is randomly selected.

- *widest-minhop* path algorithm: a path with the minimum hop count among feasible paths. If there are several such paths available, the one with the maximum bandwidth is chosen. If there are several such paths with the same bandwidth, one is randomly selected.

- *shortest-widest* path algorithm: a path with the maximum bandwidth among all of the feasible paths. If there are several such paths available, the one with the minimum delay is selected. If there are many such paths with the same delay, one is randomly selected.

- *shortest-widest-minhop* path algorithm: a path with the minimum hop count among all feasible paths. If there are several such paths available, the one with the maximum bandwidth is selected. If there are several such paths with the same bandwidth, the one with the minimum delay is selected. If there are several such paths with the same delay, one is randomly selected.

The widest-minhop routing gives high priority to limiting the hop count number, while the minhop-widest routing gives high priority to balancing the network load by selecting the one with the maximum bandwidth. The algorithm to find the widest-minhop path is described in Section 3.3.3.2. The algorithm is shown in Program 3.5; but instead of using the delay as the primary objective, we used the hop count as the primary objective.

In addition, the shortest-widest algorithm not only balances the network load but also considers network delay. It ensures a user request will be given the lowest delay path among maximum bandwidth paths. This implies that the switch will provide the minimum delay service to the user request as well as balance the network load. The algorithm is shown in Program 3.6.

Furthermore, the shortest-widest-minhop algorithm combines three criteria together to provide the most appropriate path to the user request. The idea of this algorithm is to find out whether we can find an even better path than using double criteria routing algorithms. The algorithm is shown in Program 3.7.

5.1.2 Experiments with MCRAs with Bandwidth Guarantees

In this section, we describe the topologies we used in our experiments to evaluate the performance of the MCRAs with the bandwidth guarantees. To evaluate the MCRAs, we used the following four different topologies:

- Dense Edge-Core Network
- Light Edge-Core Network
- 3-Cluster Network and
- 8-cluster Network

The first two networks are built on the same concept (edge-core topology), and they have the same number of edge and core nodes. These two networks are explained in Section 4.1.2. However, from Table 4.3, they have the different number of links in their networks. The dense network has a 2.125 connectivity, but the light network has a 1.75 connectivity. The difference in the connectivity can make these two networks exhibit different performance.

The last two networks are created according to the clustering scheme, but are grouped differently. The 3-cluster network has 3 clusters, and each cluster has 8 nodes grouped together. On the other hand, the 8 cluster network has 8 clusters, and each cluster has 3 nodes. For both networks, each node in the cluster is connected via the OC-3 link, and each group is connected via the OC-12 links. The forming of the network of these two algorithm can make a difference in the network performance.

5.1.3 Call Blocking Rate as a Function of Requested Bandwidth

In this section, we evaluated our four routing algorithms containing a *maximum bandwidth* criterion, which are the widest-minhop, minhop-widest, shortest-widest and shortest-widest-minhop routing algorithms as described in Section 5.1.1. We examined the call blocking rate of these four routing algorithms using different requested bandwidths and topologies. In the experiments below, the call bandwidth is uniformly distributed. Traffic is also uniformly distributed. The destination of the request is uniformly selected among other nodes. The total number of calls is 2400 calls in any network.

5.1.3.1 Performance of Edge-Core Networks

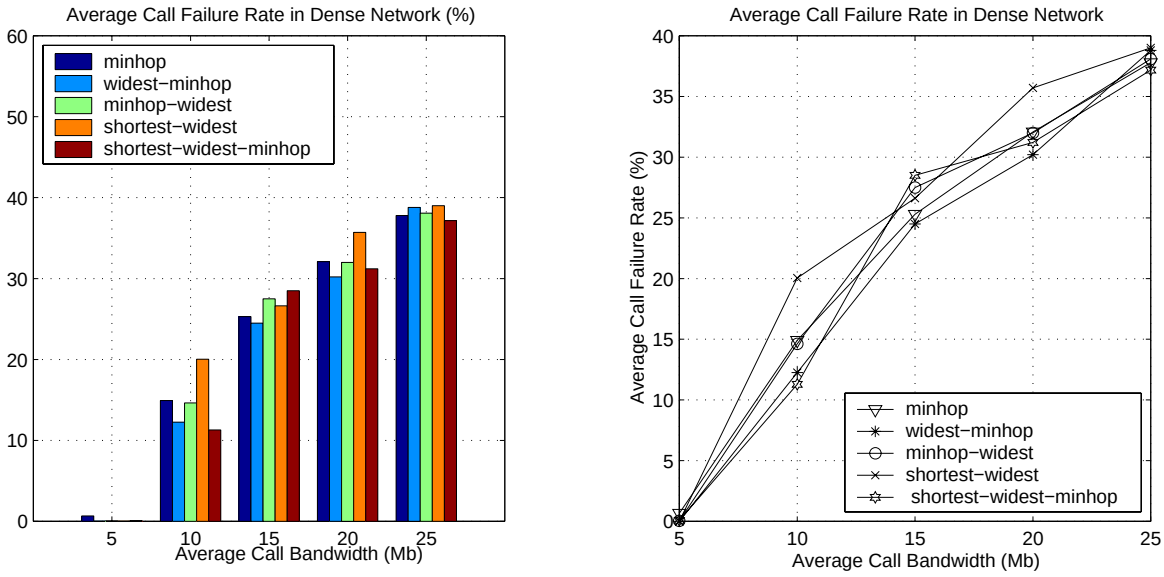

Figure 5.1: Average call blocking rate as a function of average requested bandwidth: Dense Edge-Core Network

Figure 5.1 shows the call failure rate as a function of the average call bandwidth in the dense edge-core topology. Overall, we see that the call failure rate is linearly proportional to the call bandwidth. On the left-hand side, when the average call bandwidth is as low as 5 Mb, almost no calls failed except for the minhop

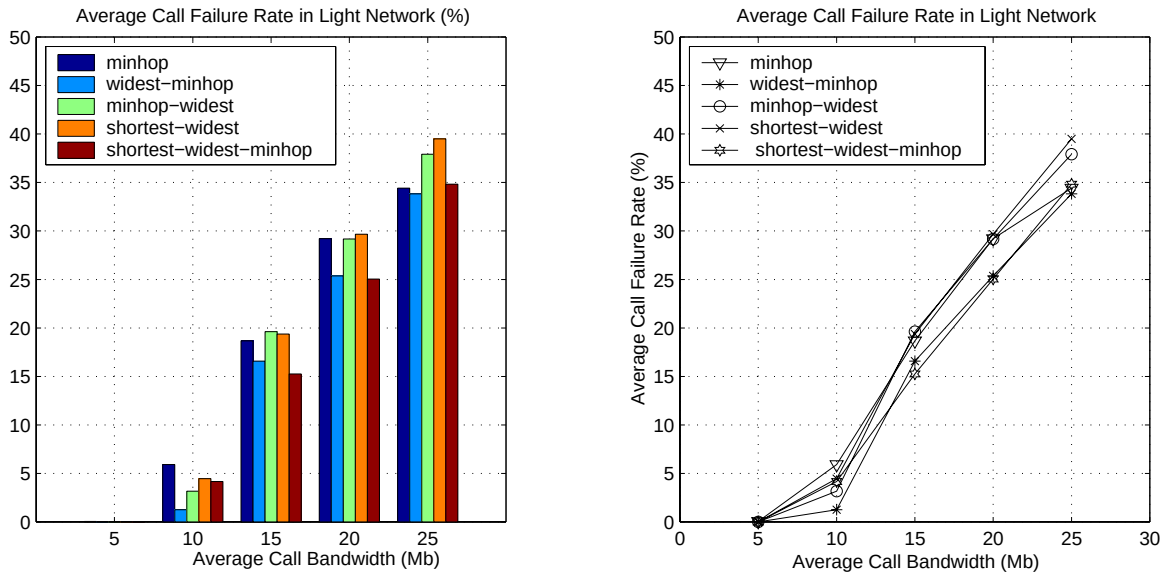

Figure 5.2: Average call blocking rate as a function of average requested bandwidth: Light Edge-Core Network

routing whose call failure rate is about 2%. On the right-hand side, when the call bandwidth increases, the shortest-widest routing performs poorly because it tends to allocate long ”expensive” paths, which penalize calls which later arrive. On the average, the widest-minhop routing tended to perform better than others. Note that the edge-core topology is a symmetrical network which probably contains many feasible paths from one source to one destination which have the same hop count. Therefore, selecting the path which has the maximum available bandwidth among those feasible paths would reduce the call failure rate because it balances the bandwidth resource. For the low call bandwidth, the shortest-widest-minhop tended to perform best, but when the call bandwidth was increased, it performed as well as the widest-minhop routing.

In Figure 5.2, we show the call blocking rate as a function of the average call bandwidth in a light edge-core network. When the call bandwidth was increased, the call failure rate tended to rapidly increase when the call bandwidth was more than 5 Mb. This is because the network tended to have more congested links across the network (the link between L-core nodes) unlike the dense edge-core network

which has more available links across the network. Overall, the widest-minhop routing tended to perform better than the others, and the call failure rate of the shortest-widest-minhop routing was close to that of the widest-minhop routing.

5.1.3.2 Performance of Multiple Cluster Network

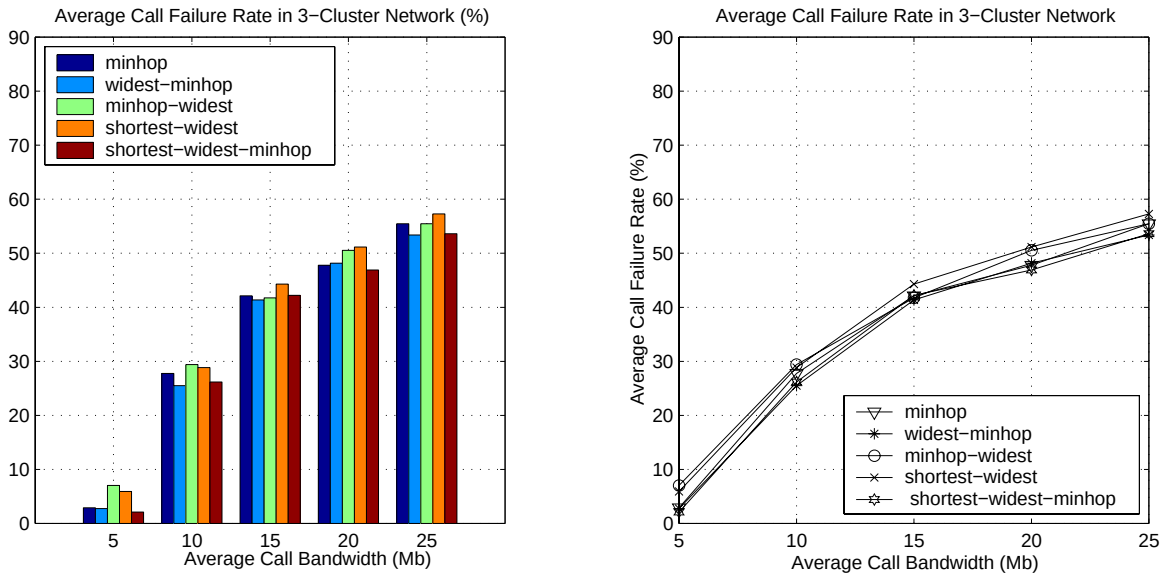

Figure 5.3: Average call blocking rate as a function of average requested bandwidth: 3-cluster Network

Figure 5.3 shows the call failure rate as a function of average call bandwidth of the 3-cluster network. Overall, when the call bandwidth was increased and some links became congested, the shortest-widest routing tended to perform worse than the others because it tended to allocate the long paths which have a maximum bandwidth. This increased the blocking probability of the later arrivals of some links in the network. This is similar to the performance of the minhop-widest routing. Even though the minhop-widest path has the minimum hop count among the maximum bandwidth, the hop count number is still high because the maximum bandwidth routing has a higher priority than minimum hop count routing. On the other hand, the widest-minhop routing performed better

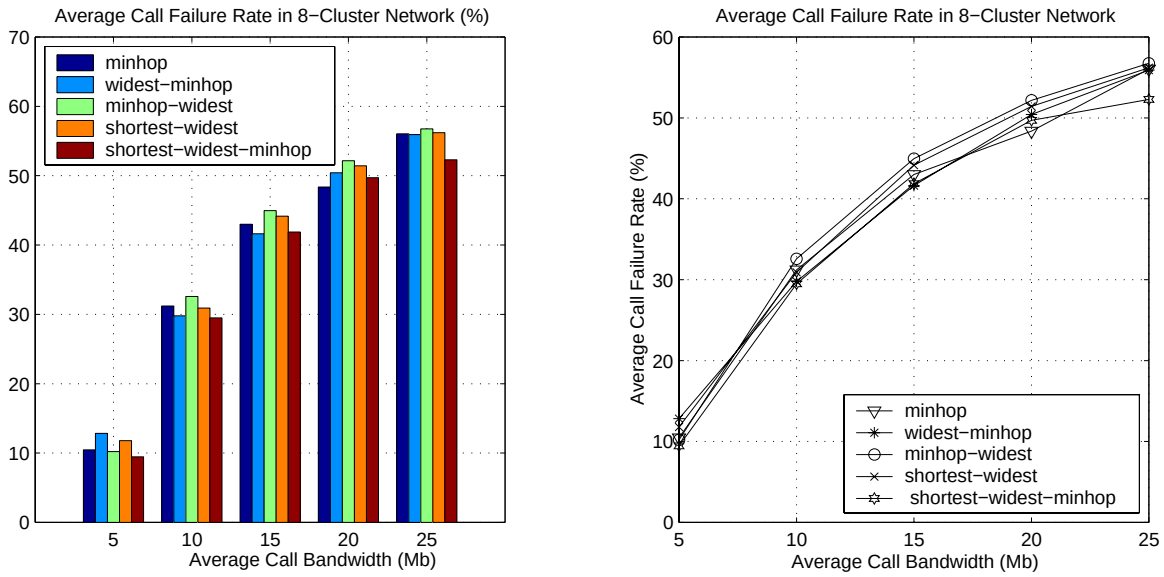

Figure 5.4: Average call blocking rate as a function of average requested bandwidth: 8-cluster Network

than the minhop routing and others. This shows that the widest-minhop routing probably reduced the congestion by using the minimum resource and balancing the bandwidth resource. Note that the effect of congestion can penalize the later arrivals especially when the traffic is high. This is similar to the shortest-widest-minhop routing, and also its performance is not much different because the chance that there will be two or more feasible paths with the minimum hop count and the maximum bandwidth is very low.

Figure 5.4 shows the call failure rate as a function of the average call bandwidth of the 8-cluster network. The performance of routing algorithms in this network is quite similar to that of the routing algorithms in the 3-cluster network. The widest-minhop and the shortest-widest-minhop routing algorithms tended to perform better than the others. In addition, the call failure rate in the 8-cluster network is slightly higher than that in the 3-cluster network. This is because the diameter of the 8-cluster network is wider, so the probability that the later arrivals will be blocked is increased.

5.1.4 Call Blocking Rate as a Function of Call Holding Time

In this section, we examine the call blocking rate of four routing algorithms mentioned in Section 5.1.1 as the function of average call holding time. We test our algorithms using four topologies: dense edge-core, light edge-core, 3-cluster, and 8-cluster topologies. Each experiment generated a total of 2400 calls with an average call bandwidth of 10 Mb. The destination of the call was uniformly distributed. The distribution of the call arrival is the Poisson with an average of 5 seconds between call. The distribution of the call holding time is also the Poisson.

5.1.4.1 Performance of Edge-Core Networks

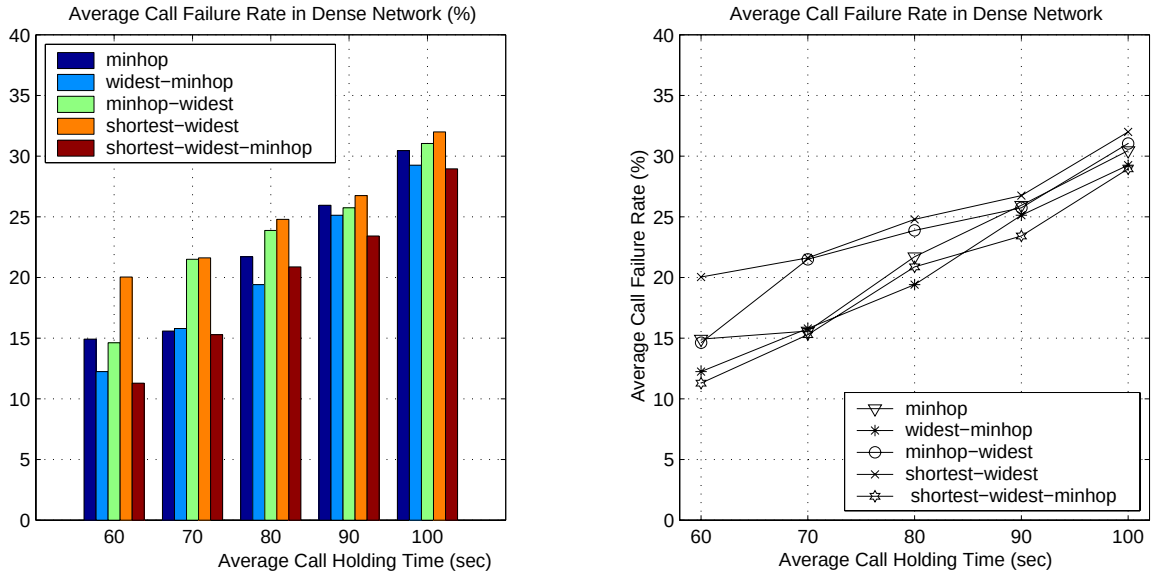

Figure 5.5: Average call blocking rate as a function of average call holding time: Dense Network

Figure 5.5 shows the average call failure rate as a function of average call holding time of the dense edge-core topology. Overall, the average call failure rate increases linearly as the average call holding time increases. The widest-minhop and the shortest-widest-minhop routing algorithms tend to perform better than the minhop routing and others. This is because those two algorithms balance the net-

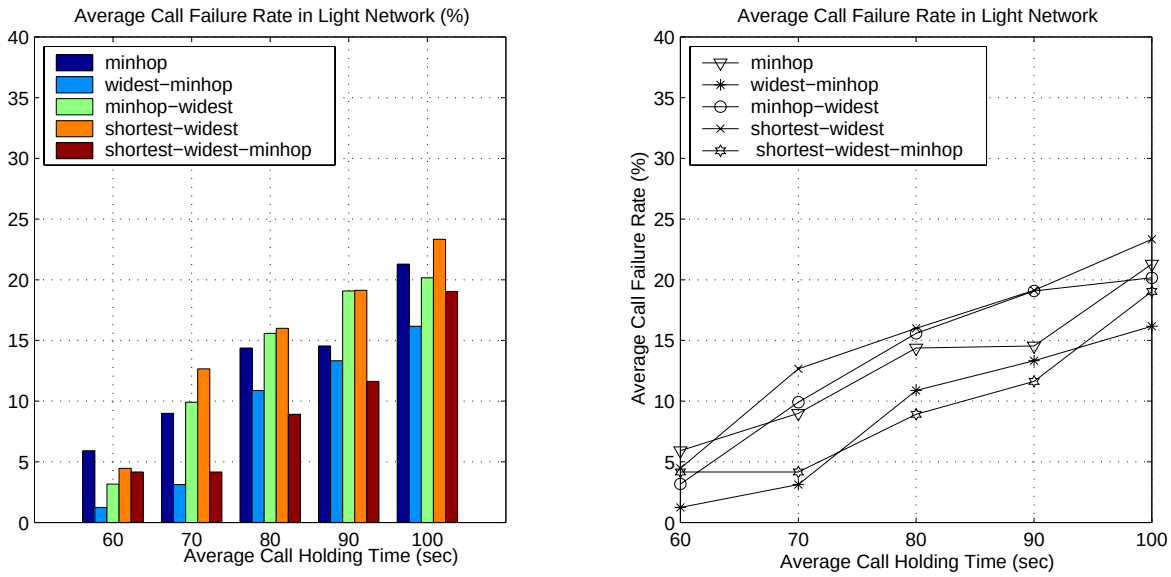

Figure 5.6: Average call blocking rate as a function of average call holding time: Light Network

work resource by selecting the less congested path when there are many feasible paths which have the same hop count number. The widest-minhop and shortest-widest-minhop routing algorithms have a close performance because there are fewer paths which have different path delays, but have the same path hop count and the same path bandwidth.

In Figure 5.6, we show the average call failure rate as a function of the average call holding time of the light edge-core topology. Compared to those in the dense edge-core network, the widest-minhop and the shortest-widest-minhop routing algorithms seem to perform much better than the minhop routing algorithm in the light edge-core network. This is because generally there are more congested links in the light network than those in the dense network. Therefore, it is crucial to select a path which is not congested, and the widest-minhop or shortest-widest-minhop routing tended to give a route which was not congested. In addition, the failure rate using the minhop-widest or the shortest-widest routing algorithm was higher than the other routing algorithms because the algorithm tended to give a longer feasible path than others. Note that the longer path con-

sumes the larger total amount of network resource. Therefore, the later arrivals tend to be rejected more often.

5.1.4.2 Performance of Multiple Cluster Network

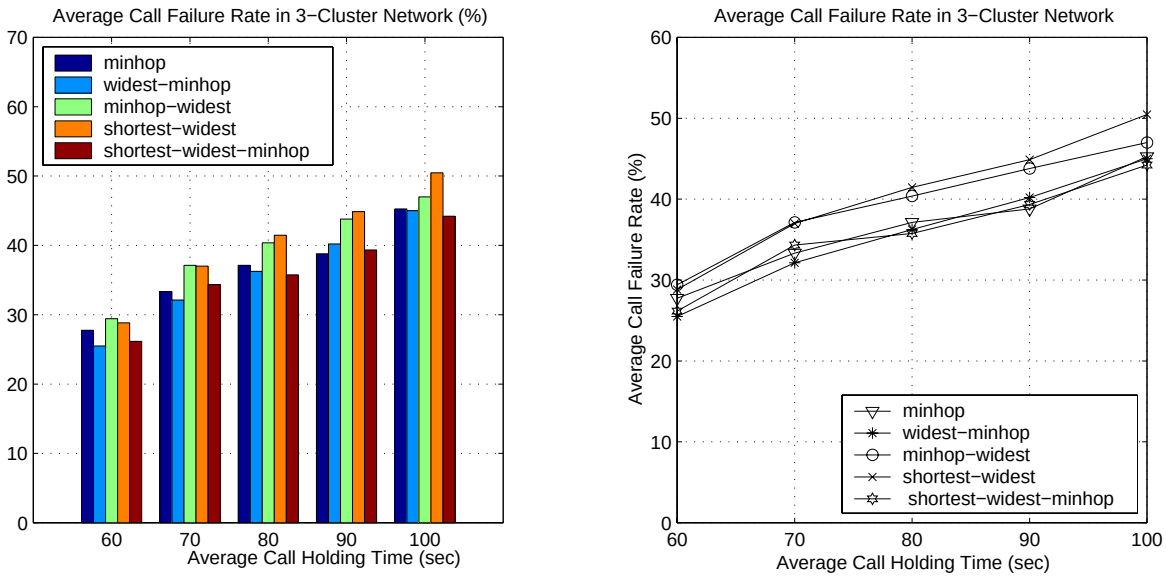

Figure 5.7: Average call blocking rate as a function of average call holding time: 3-cluster Network

Figure 5.7 shows the average call failure rate as a function of average call holding time of the 3-cluster topology. Overall, it shows that the call blocking rate was linearly proportional to the average call holding time. The shortest-widest routing tended to perform poorly because it probably allocated a long path which has a maximum bandwidth. Allocating a long path could increase the blocking probability of the later arrivals. On the other hand, the widest-minhop tended to perform well and so did the shortest-widest-minhop routing. However, their performance was not much improved compared to the minhop routing algorithm.

In Figure 5.8, we show the average call failure rate as a function of average call holding time of the 8-cluster topology. Overall, the call blocking rate of any routing algorithm is closely similar to that of the 3-cluster topology. However, the

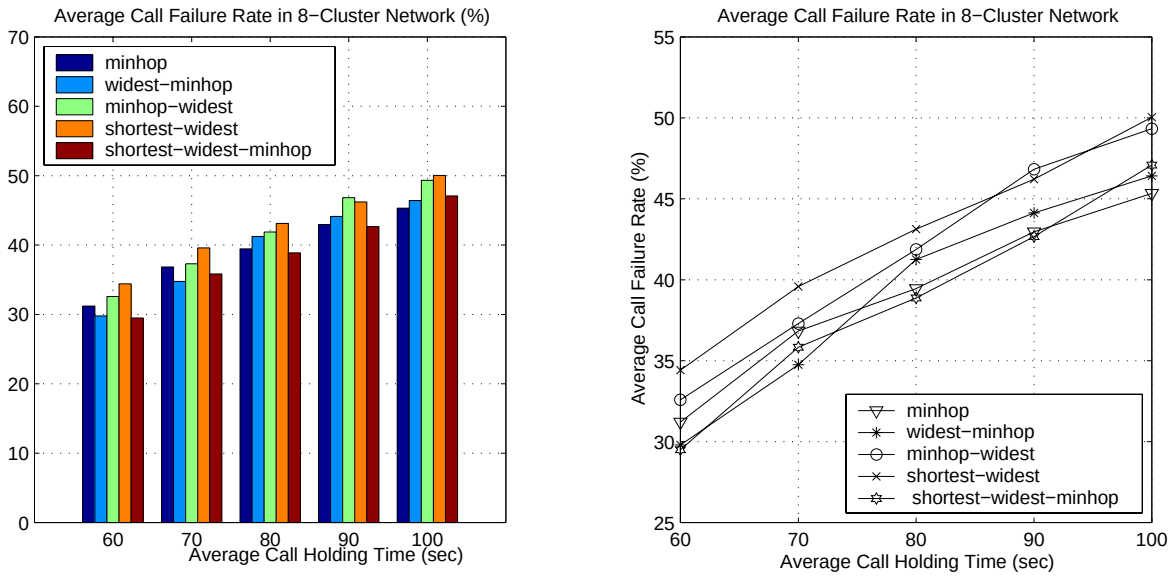

Figure 5.8: Average call blocking rate as a function of average call holding time: 8-cluster Network

increase in the call blocking rate is likely to be higher than that of the 3-cluster topology. This is because the 8-cluster topology has a larger diameter, which tends to increase the congestion in the network.

5.1.5 Evaluation of Call Setup Time

In this section, we examine the call setup time of four routing algorithms as mentioned in Section 5.1.1 for different requested bandwidths and topologies. In our experiments below, the call bandwidth was uniformly distributed. Traffic was also uniformly distributed. The destination of the requested call was uniformly selected among all other nodes. The total number of calls was 2400 calls in all the experiments. The distribution of the call holding time is the Poisson with a mean of 60 seconds, and the distribution of the call arrival is the Poisson with an average of 5 seconds between calls.

5.1.5.1 Performance of Edge-Core Networks

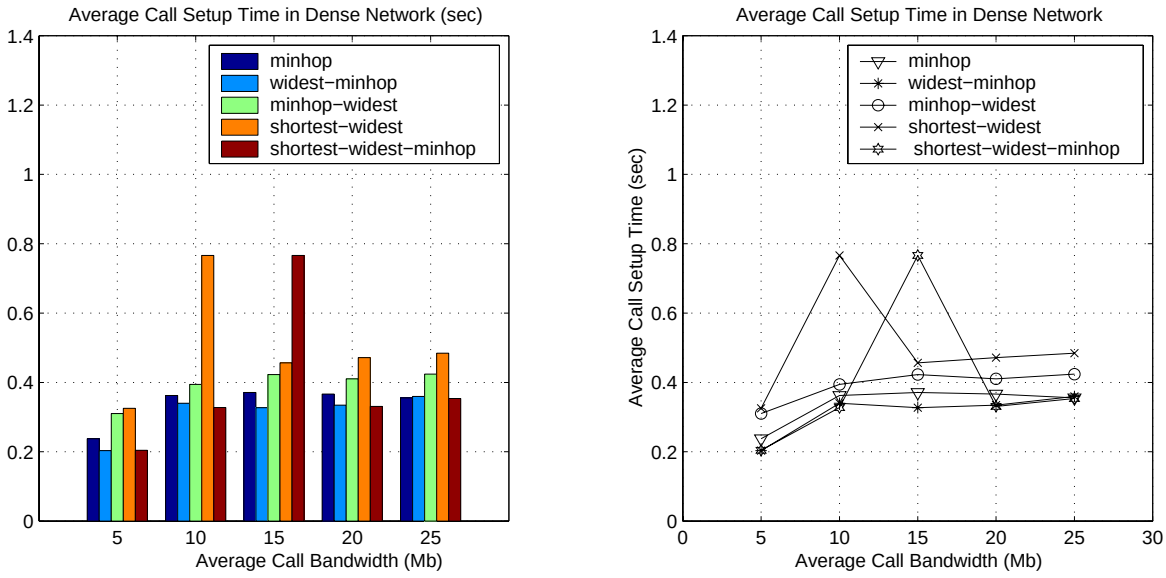

Figure 5.9: Average call setuptime as a function of average requested bandwidth: Dense Network

In Figure 5.9, we show the average call setup time as a function of the average call bandwidth of the dense edge-core network. First of all, when the call bandwidth changes from 5Mb to 10Mb, the setup time tends to proportionally increase, but after this point the call setup time tends to be steady. This is because

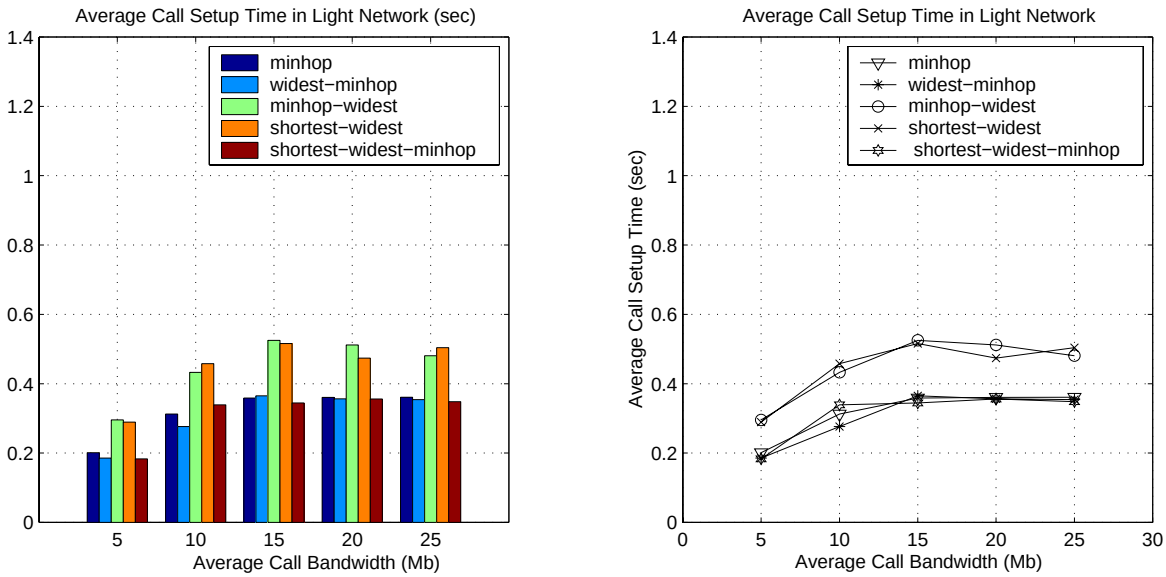

Figure 5.10: Average call setup time as a function of average requested bandwidth: Light Network

at the point where the call bandwidth is 5Mb, the traffic is sparse so most of the calls are successfully routed. However, when the call bandwidth increases, and the network becomes more congested, the calls tend to not be successfully routed. Thus, the crankback occurs, and an alternate routing is used. The average call routing time seems to be increased when there are alternate routings. Overall, the widest-minhop routing probably performs best. However, the shortest-widest routing tended to perform poorly because it probably allocated a long path which requires a long setup delay which mainly includes the routing time and the call admission control time. It also increases the chance of crankback.

Note that in Figure 5.9, you can see the ”jump” in the call setup time graph. This is because there are several crankbacks and alternate routing retries. When the call is crankbacked, the amount of time of alternate routing retries is not averaged but added to the total setup time for that call. Therefore, the tremendous number of alternate routing retries can significantly increase the average call setup time. Note that in our experiments, we set the number of the alternate routing to 10 retries. Further discussion about the alternate routing performance is in Section 5.4.

Figure 5.10 shows the average call setup time as a function of average call bandwidth of the light edge-core network. The widest-minhop routing tended to perform well, but the average setup time did not improve much compared to the minhop routing. The shortest-widest-minhop routing performed closely to the widest-minhop routing. This could be because there are not many feasible paths which have the same minimum hop count and the same maximum bandwidth. On the other hand, the shortest-widest and the minhop-widest routing algorithms tended to have a higher call setup time. This is because those routing algorithms need to run the D_Widest algorithm to find the possible maximum bandwidth of a path, and then run the modified Dijkstra algorithm to find the shortest path or minhop path with the maximum bandwidth obtained from D_Widest algorithm. Therefore, the execution time of these two routing algorithms was higher than the other routing algorithms. The routing algorithms are described in Section 3.3.3.2.

5.1.5.2 Performance of Multiple Cluster Network

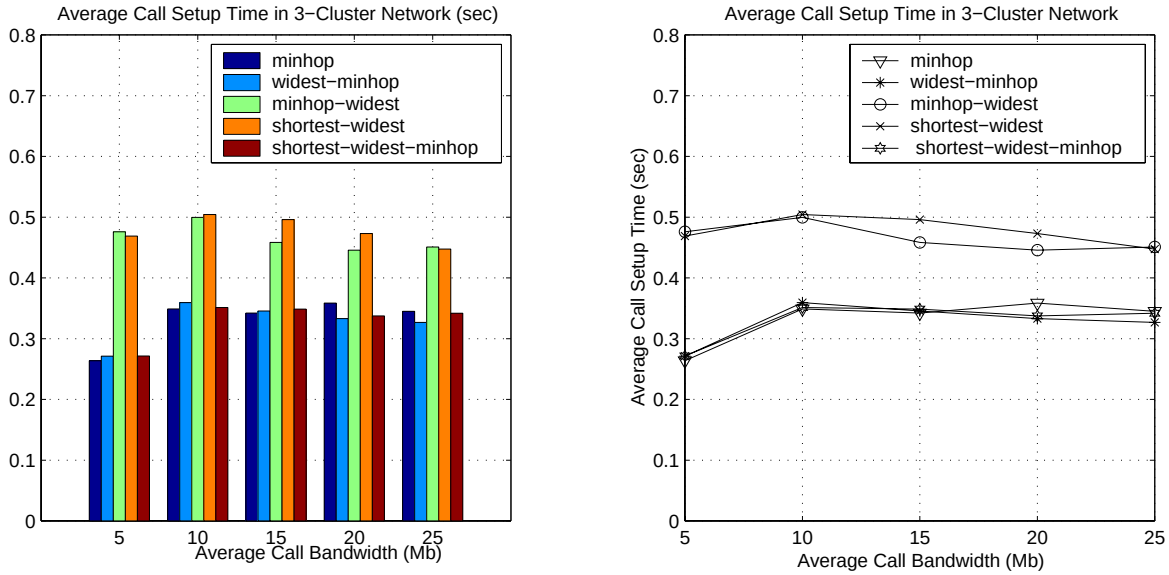

Figure 5.11: Average call setup time as a function of average requested bandwidth: 3-cluster Network

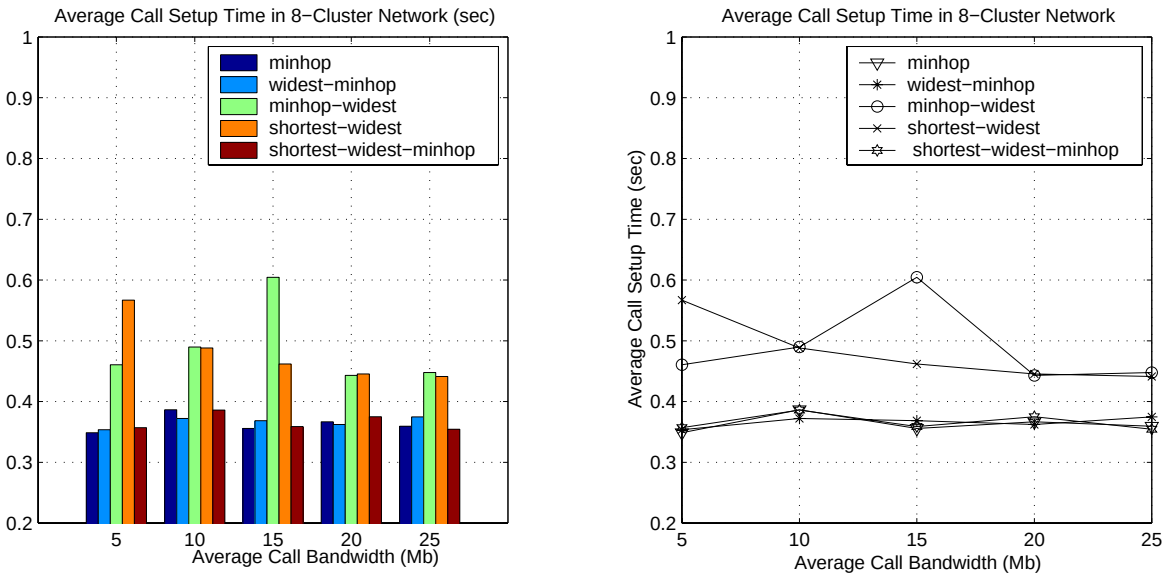

Figure 5.12: Average call setup time as a function of average requested bandwidth: 8-cluster Network

In Figure 5.11, we show the average call setup time as a function of average call bandwidth in the 3-cluster network. Overall, at the low call bandwidth, the setup time tended to be short because the routing has a high success rate. This is because when calls are successfully routed without an alternate routing, the total call setup time for calls will mostly result from one-time routing and call admission control. However, when the call bandwidth is more than 10 Mb, the call setup time is higher because there are more crankbacks and alternate routing. The shortest-widest routing and the minhop-widest routing seem to take more setup time than the others. This is because these algorithms need to run the D_Widest algorithm and the modified Dijkstra algorithm to find a feasible route, but other algorithms use only the modified Dijkstra algorithm. Note that the computation in the worst case of the D_Widest algorithm or the modified Dijkstra algorithm is $O(V^2)$, where V is the number of nodes. However, the call setup time of the shortest-widest or the minhop-widest routing, which uses both the D_Widest algorithm and the modified Dijkstra algorithm, is not twice as much as that of others as we had expected.

Figure 5.12 shows the average call setup time as a function of average call

bandwidth in the 8-cluster network. Compared to that of the 3-cluster network, the average call setup time of the 8-cluster network seems to be higher when the call bandwidth is low. However, when the call bandwidth is high, the call setup time tended to be the same as that in the 3-cluster network. In addition, there are two jumps from the minhop-widest and shortest-widest. This is because the 8-cluster network tends to be more congested than the 3-cluster network. This shows that when the network is congested, the minhop-widest and shortest-widest algorithms are probably not suitable since they provide a long-period setup. In addition, a given route using these two algorithms tended to be more out-of-date.

5.1.6 Evaluation of Routing Inaccuracy

In this section, we compared the efficiency of our routing algorithms using the routing inaccuracy metric described in Section 4.2.3. We show the routing inaccuracy of dense and light edge-core networks as well as 3-cluster and 8-cluster networks. In our experiments below, the call bandwidth was uniformly distributed. Traffic was also uniformly distributed. The destination of the call requested was uniformly selected among other nodes. The total number of calls was 2400 calls in any network. The distribution of the call holding time is the Poisson with a mean of 60 seconds, and also the distribution of the call arrival is the Poisson with an average of 5 seconds per call.

5.1.6.1 Performance of Edge-Core Networks

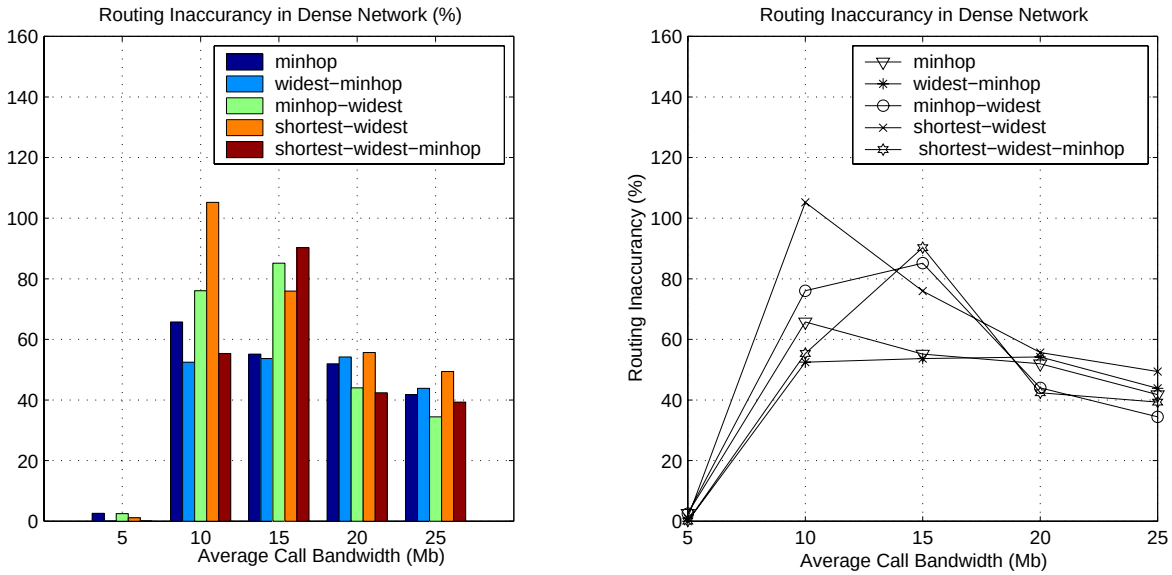

Figure 5.13: Routing Inaccuracy as a function of average requested bandwidth: Dense Network

In Figure 5.13, we show the routing inaccuracy of our routing algorithms as a function of the average call bandwidth in a dense edge-core network. Overall, all the routing algorithms performed well, but when the average call bandwidth was

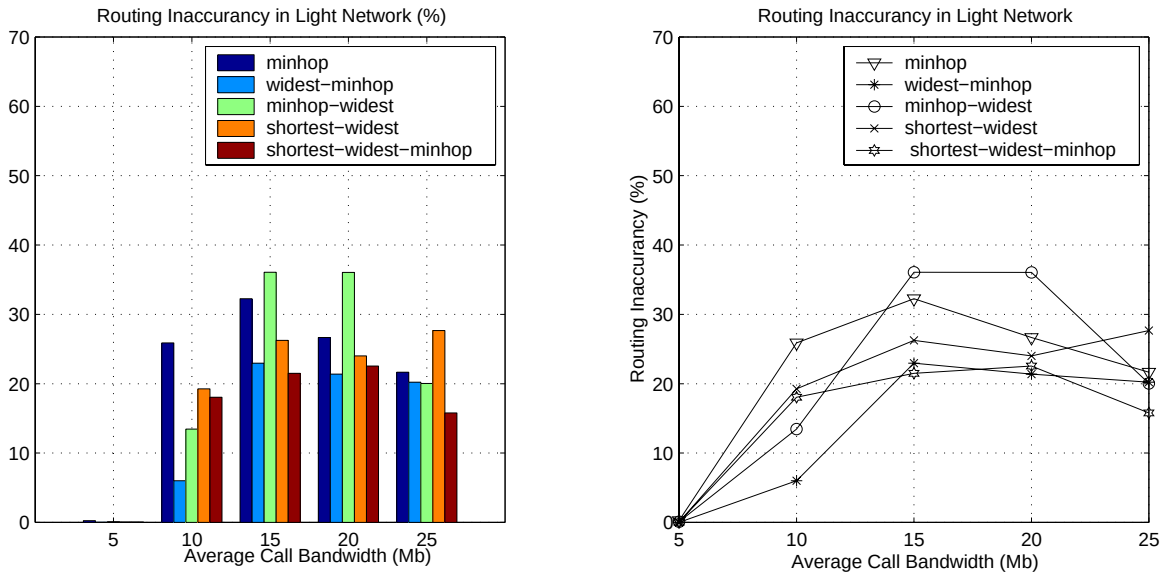

Figure 5.14: Routing Inaccuracy as a function of average requested bandwidth: Light Network

more than 5 Mb, the routing algorithms tended not to give the correct path to the user request. On the average, the widest-minhop routing tended to perform well on any value of average call bandwidth. Note that the routing inaccuracy can be more than 100% as shown in Figure 5.13. This is because every call has 10 alternate routing retries. This means each call can have the maximum of 10 crankbacks. Therefore, if there are several crankbacks in many calls, the routing inaccuracy can be more than 100%.

Figure 5.14 shows the routing inaccuracy of our routing algorithms as a function of the average call bandwidth in the light edge-core network. Overall, the routing inaccuracy seems to increase up to a certain point, and it is slightly increased when the average call bandwidth increases. On average, the minhop routing tended to perform worse than other multiple-criterion routing algorithms except the minhop-widest one. It has been shown that the minhop routing would probably give an ”incorrect” path to the user request more than other routing algorithms.

From Figure 5.13 and Figure 5.14, the shortest-widest-minhop routing tend-

ed to perform better than the minhop routing when the call bandwidth was high. This is because it used more information to find a feasible path than the minhop routing. Therefore, it has been shown that when the traffic load in edge-core network is high, and the routing information is not quite accurate, the multiple criteria routing tends to perform better than a single criterion routing algorithm.

5.1.6.2 Performance of Multiple Cluster Network

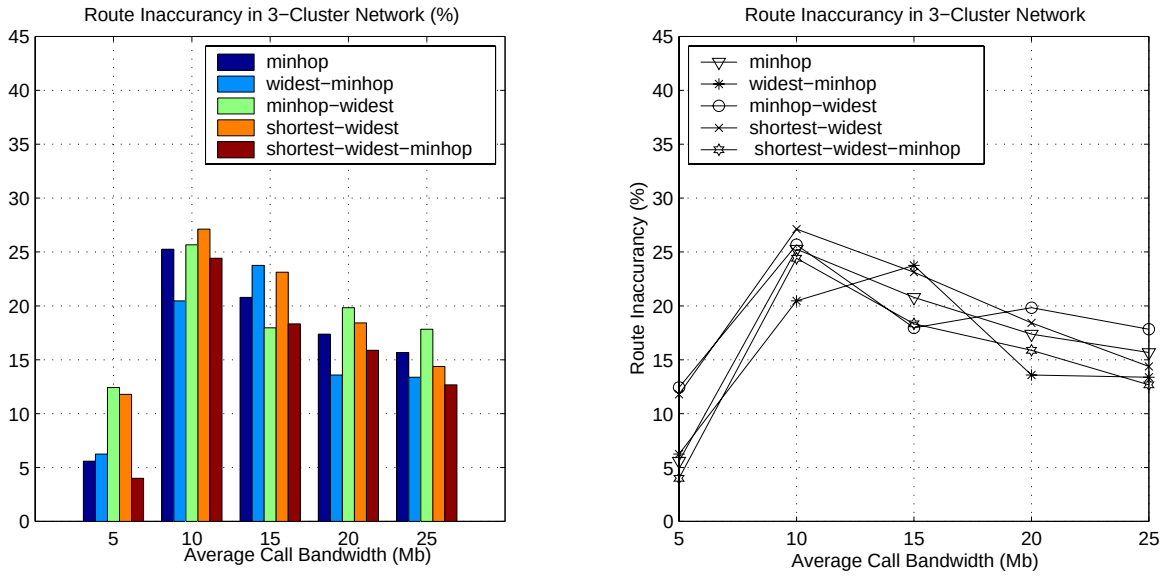

Figure 5.15: Routing Inaccuracy as a function of average requested bandwidth: 3-cluster Network

Figure 5.15 shows the routing inaccuracy of our routing algorithms as a function of the average call bandwidth in the 3-cluster network. Overall the routing inaccuracy tended to be increased up to a certain point, and it was slightly decreased when the average call bandwidth increased. The reason is that when the call bandwidth increases, the call failure rate is increased because the router cannot find a feasible route. Therefore, the user request tends to be rejected because there is no feasible route rather than given an incorrect path which introduces the crankback in an intermediate node. Note that the routing inaccuracy measures

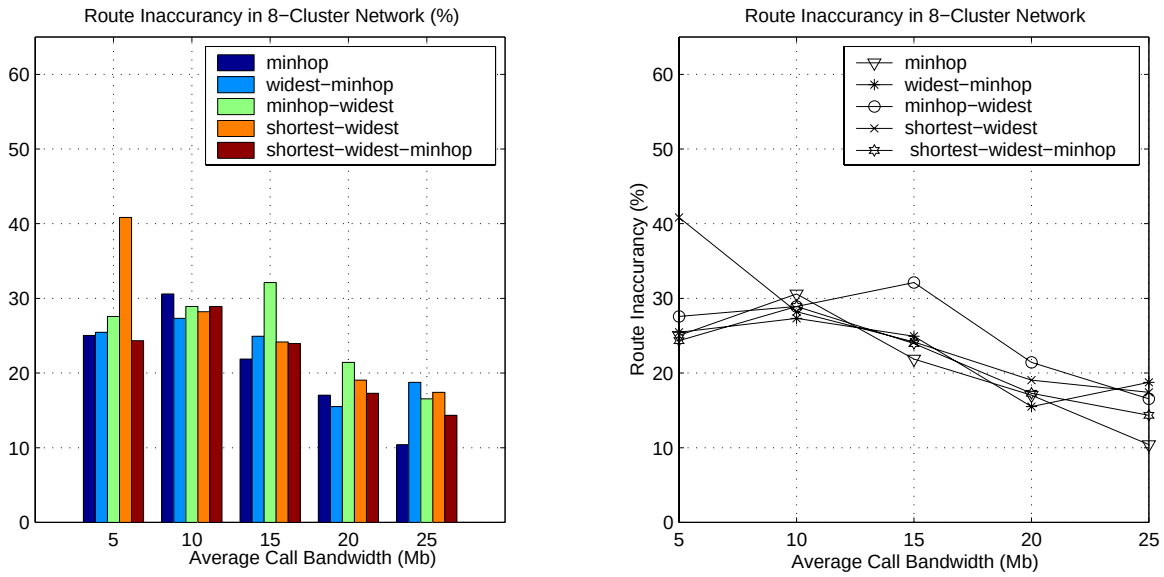

Figure 5.16: Routing Inaccuracy as a function of average requested bandwidth: 8-cluster Network

only the number of crankbacks rather than the number of failed calls because there is no route to the destination. On average, the minhop-widest routing tended to perform better than others.

In Figure 5.16, we show the routing inaccuracy of routing algorithms as a function of the average call bandwidth in the 8-cluster network. Overall, the routing inaccuracy tended to be high even though the call bandwidth was low compared to that of the 3-cluster network in Figure 5.15. This is because the diameter of the 8-cluster network is larger than that of the 3-cluter network. Thus, the updated information which is triggered by a significant change of network resources seems to be slowly converged. The 8-cluster network also seems to be congested even at the low call bandwidth. Therefore, the routing has a tendency to use out-of-date information to find a feasible path for the user request. Consequently, the request tends to be crankbacked at an intermediate node.

From Figure 5.15 and Figure 5.16, the widest-minhop routing tended to perform better than others. This is because the hop count is an important parameter for routing in the cluster network because the diameter of the network is long and

the network is not symmetric. Therefore, the call with the maximum bandwidth but the large hop count number route tended to not be successfully routed across the cluster network.

5.2 Multiple Criteria Routing for the Minimum Delay Services

Multimedia applications, such as Internet telephony and video conferencing, require strict QoS constraints on delay, delay jitter, and packet loss. In this section, we experiment with routing algorithms for traffic that require delay guarantees. The QoS constraints are composed of delay, delay jitter and bandwidth. Similar to routing traffic requiring bandwidth guarantees, the goal of routing traffic requiring delay guarantees is to find a feasible route which achieves at least the minimum link delay if one exists. More than one criterion for choosing those feasible paths is involved in finding an ”optimal” route.

The rest of this section is described as follows. Section 5.2.1 introduces the routing algorithms used in our experiments. The experiment description and test scenarios are explained in Section 5.2.2. Section 5.2.3 shows the results of call blocking rate as a function of requested bandwidths, and Section 5.2.4 shows the results of call blocking rate as a function of call holding time. The performance of routing in terms of call setup time is shown in Section 5.2.5. Lastly, the performance of the routing in terms of its accuracy is shown in Section 5.2.6.

5.2.1 Routing Criteria and Algorithms

In this section, we present the performance evaluation of four multiple criteria routing algorithms (MCRAs), whose routing criteria include the minimum delay criterion (shortest). The four MCRAs are shown below:

- *minhop-shortest* path algorithm: select a path with the minimum delay among all feasible paths. If there are several such paths available, the one with the minimum hop count is selected. If there are many such paths with the same hop count, one is randomly selected.

- *shortest-minhop* path algorithm: select a path with the minimum hop count among all feasible paths. If there are many such paths, the one with the minimum delay is selected. If there are many such paths, one is randomly chosen.

- *widest-shortest* path algorithm: select a path with the minimum delay among all feasible paths. The one with the maximum bandwidth is selected if there are many such paths. If there are still many such paths, one is randomly selected.

- *widest-shortest-minhop* path algorithm: select a path with minimum hop count among all feasible paths. If there are many such paths, the one with the minimum delay is chosen. If there are still many such paths, the one with the maximum bandwidth is chosen. Finally, if there are still many such paths, one is randomly selected.

5.2.2 Experiments with MCRAs with Minimum Delay

This section describes the network topologies used in our experiments, and some of the network parameters are described. We used four topologies to evaluate our MCRAs algorithms as follows:

- Dense edge-core topology

- Light edge-core topology

- 3-cluster Network and

- 8-cluster Network

The goal of our experiments is to find the performance of our MCRA algorithms in two kinds of networks with different connectivities and diameters. The edge-core topologies described in Section 4.1.2 have different connectivities. The

connectivities of the dense edge-core topology and the light edge-core topology are 2.125 and 1.75, respectively.

For our experiments, we have several types of links, and we set the delay of each link in the network from a selected distribution. In the edge-core topology, we have four kinds of links, large-large, small-large, edge-large and edge-small, each of which has a different link delay. The link delay of each type of link is uniformly distributed. The summary of links in edge-core topologies is described in Table 4.2.

The cluster networks used in our experiments are described in Section 4.1.1. Each of the cluster networks has different network diameters. The maximum diameter of the 8-cluster network is larger than that of the 3-cluster network. In the cluster networks, there are two kinds of links: the outside link and the inside link. The delay of the outside link is longer than that of the inside links. In addition, the delay of each link is uniformly distributed. The summary of links metrics of the cluster network is shown in Table 4.1.

5.2.3 Call Blocking Rate as a Function of Requested Bandwidths

In this section, we examine the call blocking rate using our four routing algorithms, each of which is composed of a minimum delay criterion, for example, the minhop-shortest, shortest-minhop, widest-shortest, and widest-shortest-minhop routing algorithms. The traffic in each experiment has a uniform distribution. The destination of a call is uniformly selected among all the other nodes. The call bandwidth generated by an end-user system is uniformly distributed. The total number of calls in each network is 2400 calls.

5.2.3.1 Performance of Edge-Core Networks

In Figure 5.17, we show the call failure rate as a function of average call bandwidth in the dense edge-core topology. Overall, the call failure is linearly increased as the call bandwidth increases. Performance of all routing algorithms are closed, but

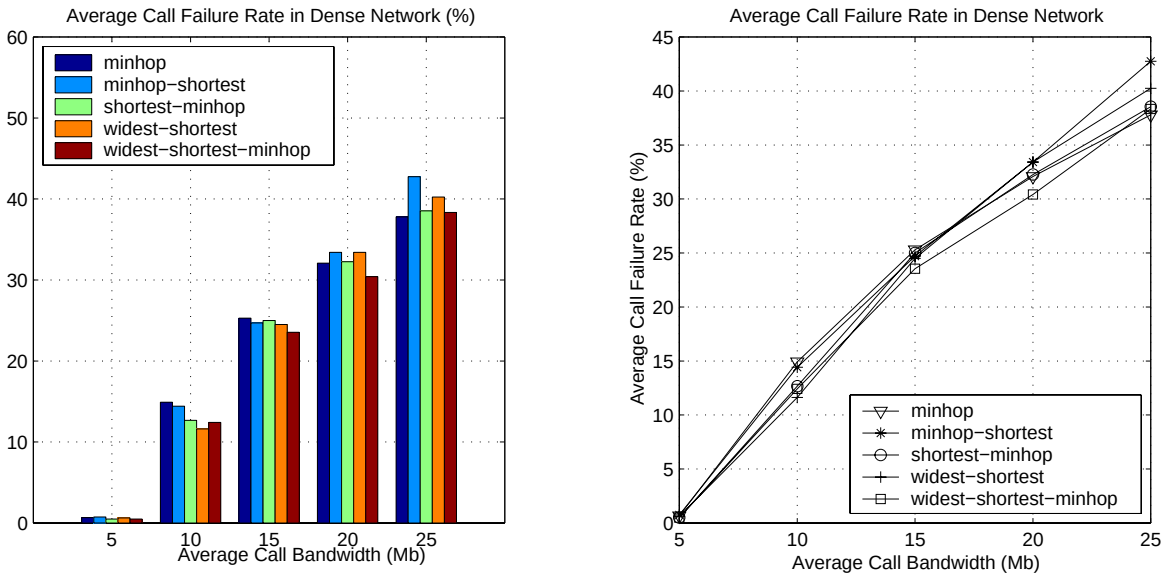

Figure 5.17: Average call blocking rate as a function of average requested bandwidth: Dense Edge-Core Network

on average, the widest-shortest-minhop tended to perform slightly better than the others. This is because the route with minhop seems to preserve network resources for later calls. With widest and shortest criteria, this algorithm can more efficiently use the network resource. Therefore, the later arrivals can be more successfully routed. In addition, the performance of minhop-shortest routing seems to be worse when the call bandwidth increases. This is because the minhop-shortest routing tended to give the same route to the user request because the link delay was not changed. Therefore, when the call bandwidth increases, the minhop-shortest path tends to be congested and more calls are rejected.

Figure 5.18 shows the call failure rate as a function of average call bandwidth in the light edge-core topology. Overall, the call failure rate is linearly increased on the average of 10% for every 5 Mb increase of the call bandwidth. As in the previous figure, the widest-shortest-minhop routing on average tends to perform better than the others.

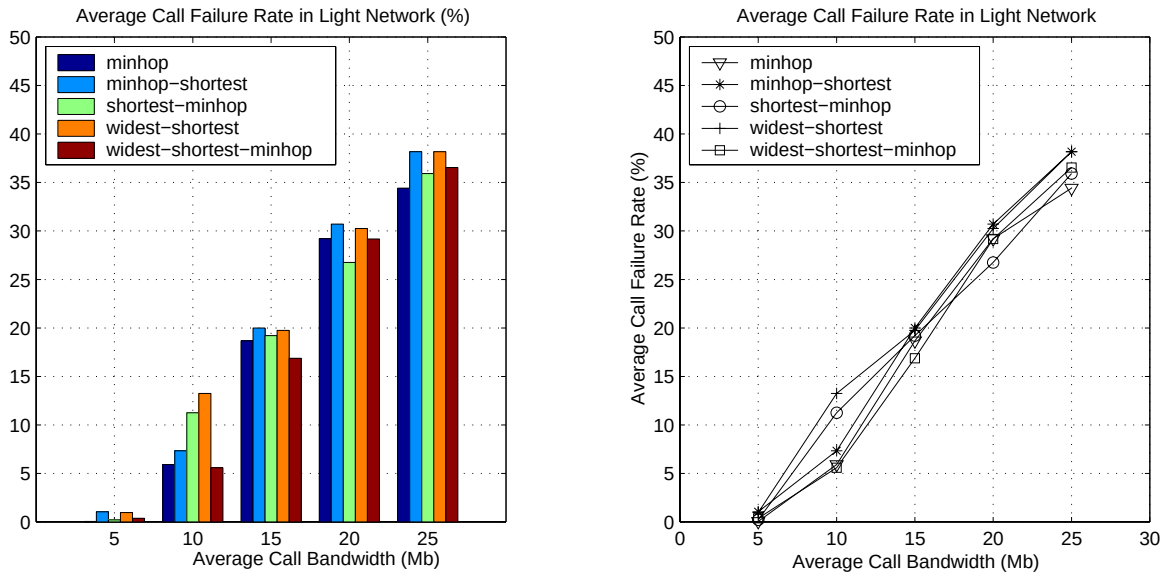

Figure 5.18: Average call blocking rate as a function of average requested bandwidth: Light Edge-Core Network

5.2.3.2 Performance of Multiple Cluster Network

Figure 5.19 shows the average call failure rate as a function of average call bandwidth in the 3-cluster network. Overall, the call failure rate tends to be linearly increased, and the failure is quite high even though the average call bandwidth is about 10 Mb. Furthermore, the slope of the failure rate tends to be decreased when the call bandwidth is higher.

In Figure 5.20, we show the average call failure rate as a function of average call bandwidth in the 8-cluster network. Overall, the performance in the 8-cluster network is similar to the one in the 3-cluster network. However, the average call failure at any value of call bandwidth is about 5% higher than that in the 3-cluster network. The reason might be that the call in the 8-cluster network tends to allocate longer paths because it has the larger diameter, and then the network tends to have more congested links. Consequently, the call requests tend to be rejected more often than those in the 3-cluster network.

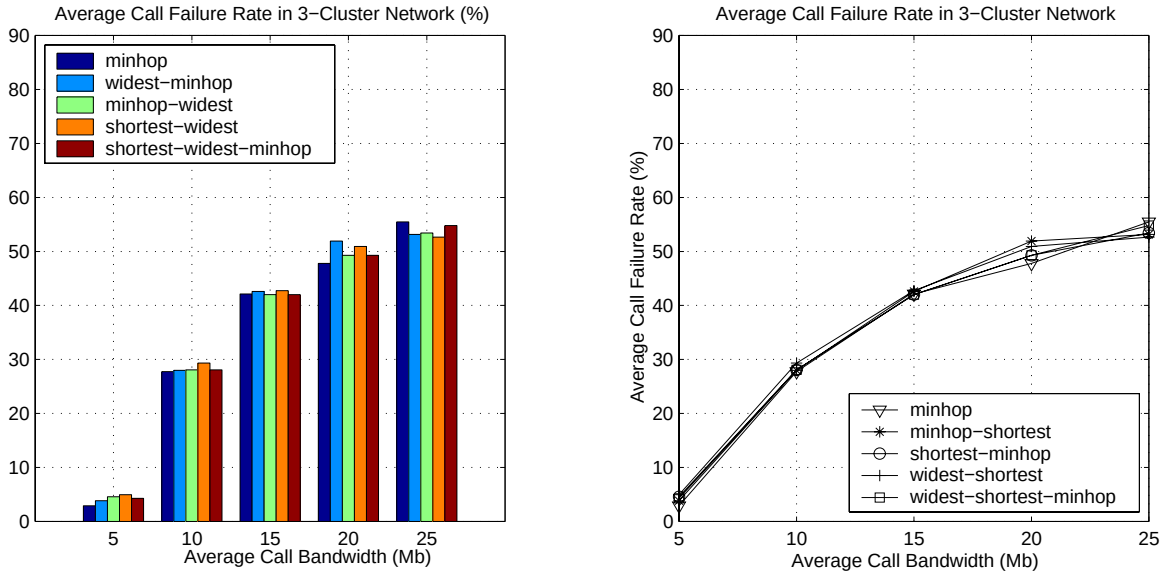

Figure 5.19: Average call blocking rate as a function of average requested bandwidth: 3-cluster Network

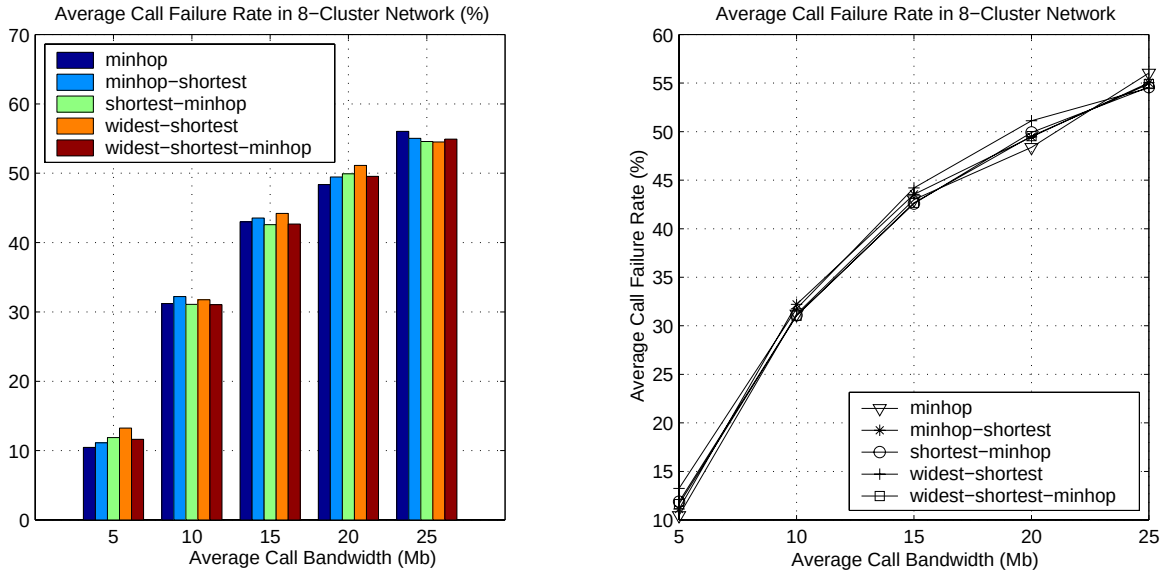

Figure 5.20: Average call blocking rate as a function of average requested bandwidth: 8-cluster Network

5.2.4 Call Blocking Rate as a Function of Call Holding Time

In this section, we examine the call blocking rate of four routing algorithms mentioned in Section 5.2.1 as the function of the average call holding time. We tested our algorithms using four topologies: dense edge-core, light edge-core, 3-cluster, and 8-cluster topologies. The call bandwidth distribution for our experiment is uniform with an average of 10 Mb per call. The call holding time has the Poisson distribution. The call arrival rate also has Poisson distribution, and it is fixed at the average of 5 seconds per call. For the rest of this section, the performance evaluations of our algorithms of the edge-core network and the cluster network are shown in Section 5.2.4.1 and Section 5.2.4.2, respectively.

5.2.4.1 Performance of Edge-Core Networks

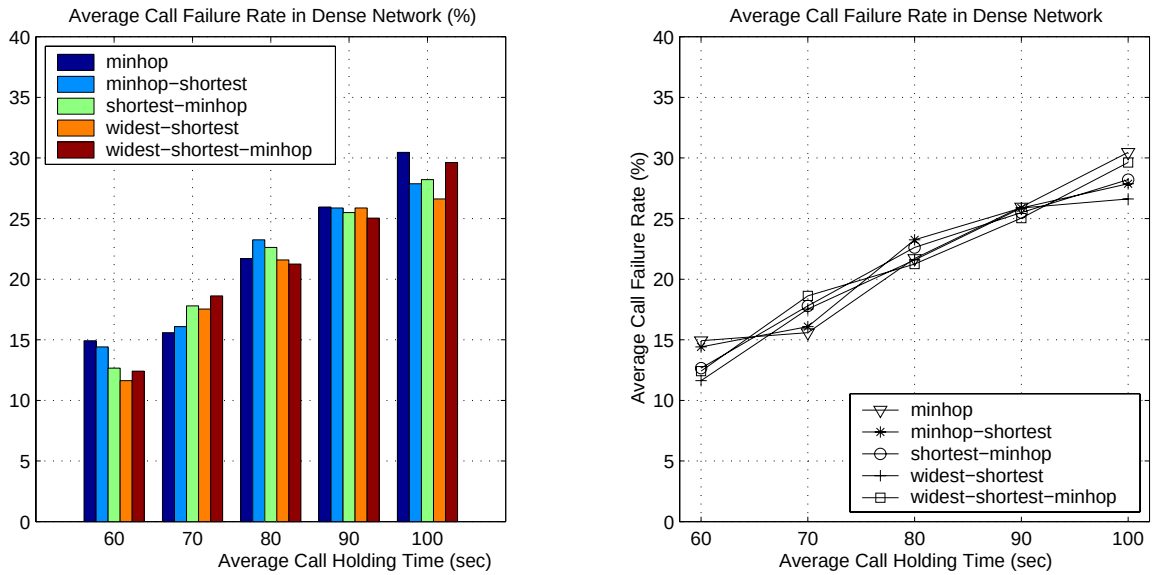

Figure 5.21: Average call blocking rate as a function of average call holding time: Dense Edge-Core Network

Figure 5.21 shows the average call failure rate as a function of the average holding time in the dense edge-core network. Note that the call holding time has the Poisson distribution. Overall, the average call failure rate is linearly increased

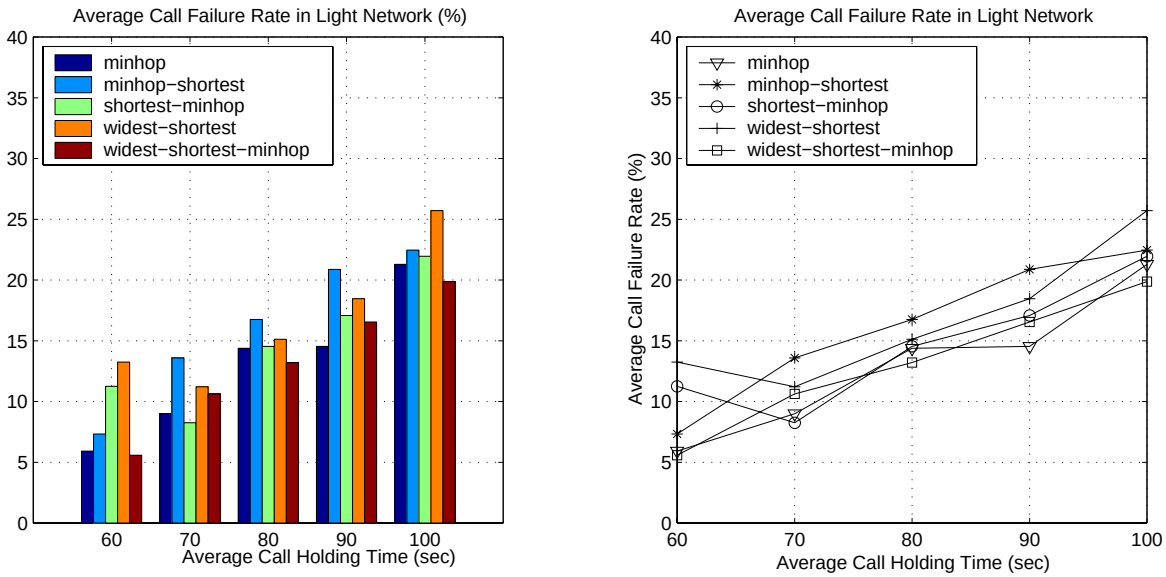

Figure 5.22: Average call blocking rate as a function of average call holding time: Light Edge-Core Network

as the average call holding time increases. The increasing rate of the call failure rate is about 5% for every 10 second increase of the call holding time.

In Figure 5.22, we show the average call failure rate as a function of average call holding time in the light edge-core network. From the figure, the minhop-shortest routing tended to perform worse than the others. This is because it gives the shortest route, which is congested when the network traffic is high. Similarly, the widest-shortest routing seems to perform worse than other routing algorithms except the minhop-shortest. It performs somewhat better than the minhop-shortest routing because it finds the maximum bandwidth link when more than one route with the same delay is available.

5.2.4.2 Performance of Multiple Cluster Network

Figure 5.23 shows the average call failure rate as a function of average holding time in the 3-cluster network. Note that the call holding time has the Poisson distribution. Overall, the call failure rate is high even though the call holding time

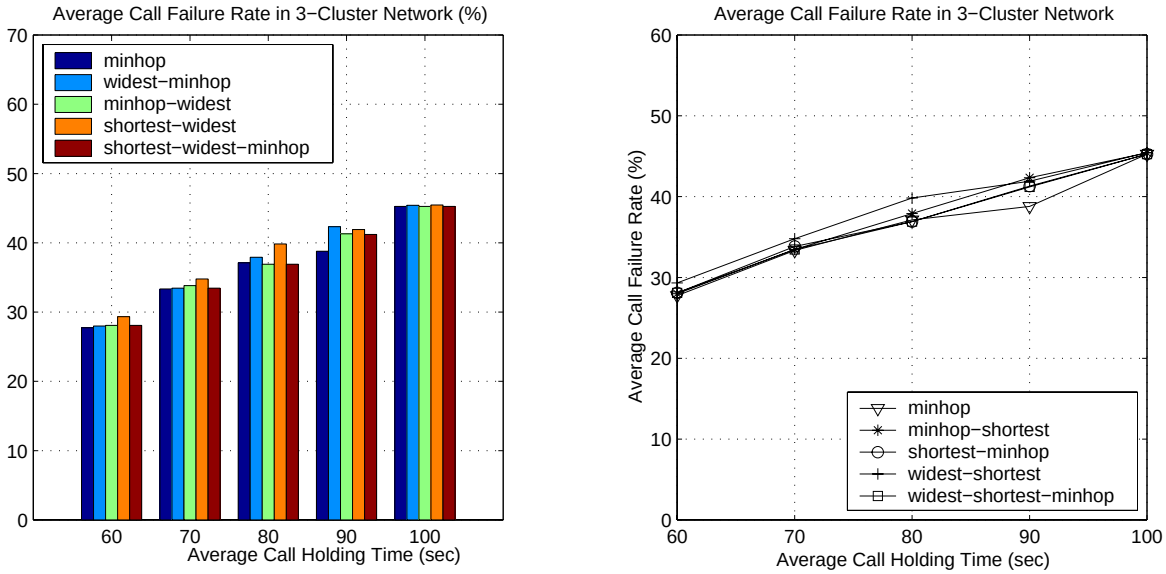

Figure 5.23: Average call blocking rate as a function of average call holding time: 3-cluster Network

is about 60 seconds, and after that the rate linearly increases. The widest-shortest tended to perform worse than the others. However, the minhop routing tended to perform well since it tried to minimize the use of network resources.

In Figure 5.24, we show the average call failure rate as a function of average call holding time in the 8-cluster network. Compared to the one in 3-cluster network, the call failure rate is 5% higher. In addition, the minhop routing tended to perform well when the network traffic is high because it minimizes the use of the network resources.

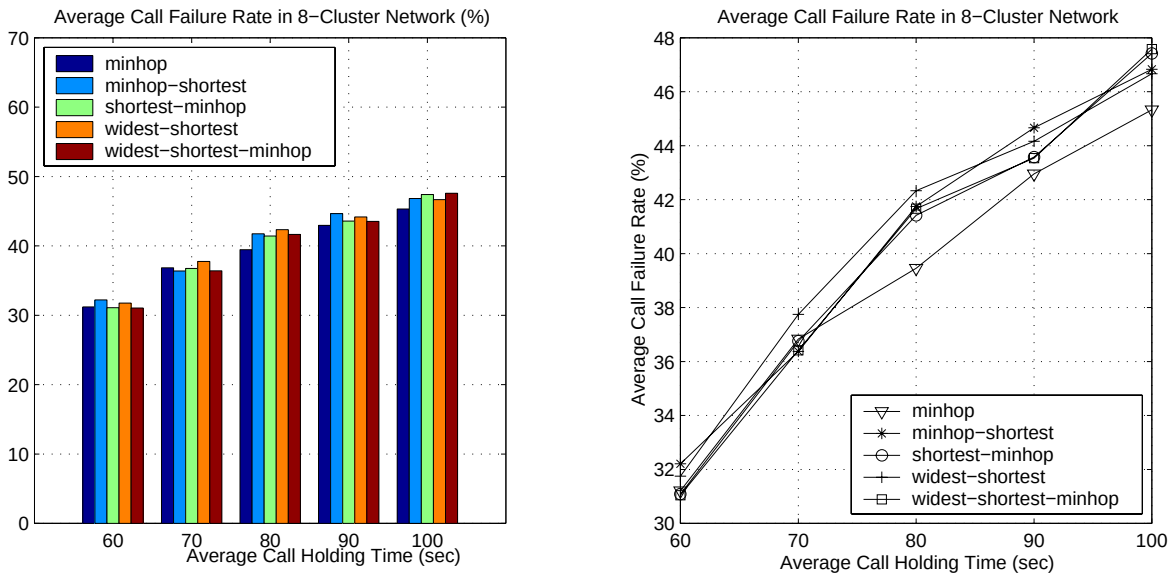

Figure 5.24: Average call blocking rate as a function of average call holding time: 8-cluster Network

5.2.5 Evaluation of Call Setup Time

In this section, we evaluate the impact of the call setup time using our algorithms for the minimum delay service. We use two types of topologies, edge-core and cluster network, and each type has two topologies which are different. These four topologies are described in Section 5.2.2. In our experiments below, the call bandwidth is uniformly distributed, and its average is of 10 Mb. Traffic is also uniformly distributed. The destination of the connection is uniformly selected among other nodes. The total number of calls is 2400 calls in any network. The distribution of call holding time is the Poisson with a mean of 60 seconds, and also the distribution of the call arrival is the Poisson with an average of 5 seconds per call. In the following, the average call setup time of the edge-core topologies is shown in Section 5.2.5.1. In addition, section 5.2.5.2 shows the average call setup of the cluster networks.

5.2.5.1 Performance of Edge-Core Networks

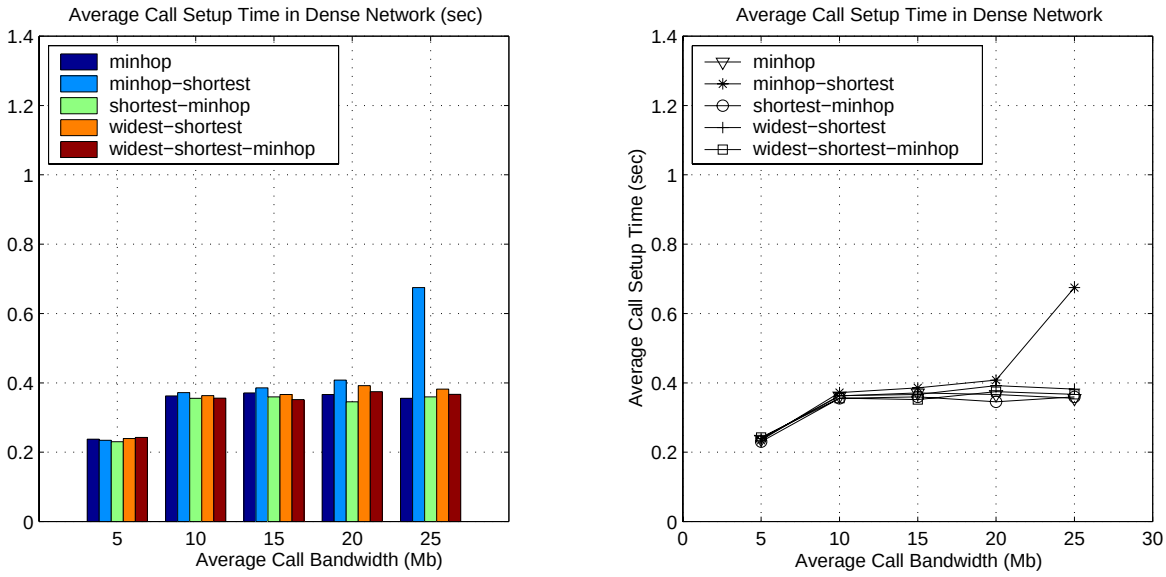

Figure 5.25: Average call setuptime as a function of average requested bandwidth: Dense Network

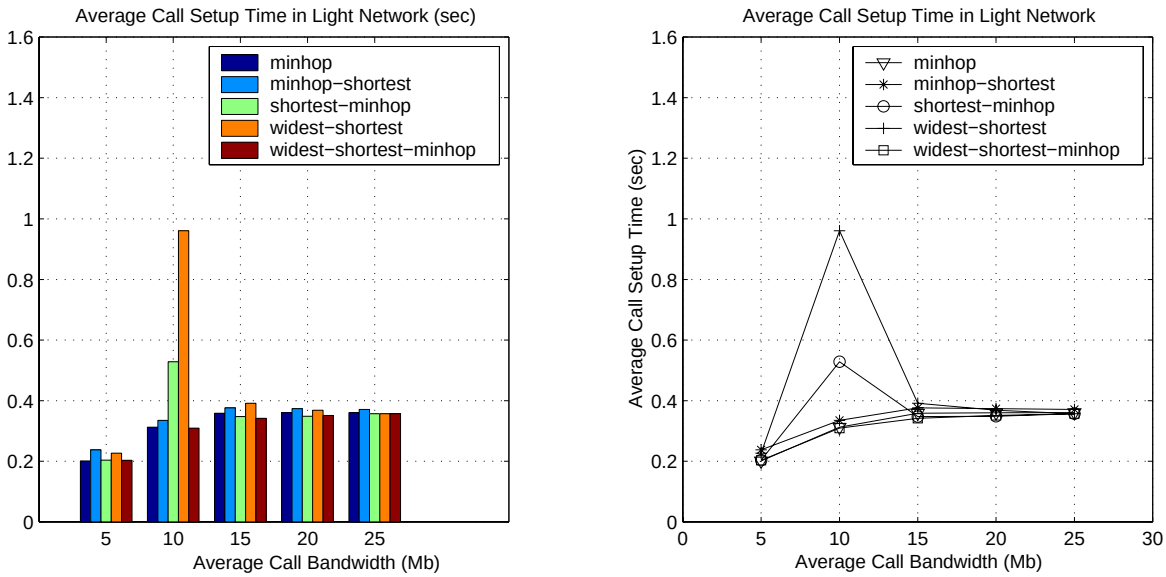

Figure 5.26: Average call setup time as a function of average requested bandwidth: Light Network

In Figure 5.25, we show the average call setup time as a function of average call bandwidth in the dense edge-core topology. Overall, at the low call bandwidth the call setup time is low because there are not many crankbacks. However, when the call bandwidth increases, the call setup time is higher. The failure rate of minhop-shortest routing tends to be more than that of other routing algorithms. In addition, when the call bandwidth is 25 Mb, it becomes even worse because there are many more crankback calls in the network. Thus, the routing algorithm probably spends most of its time finding alternate routes.

Figure 5.26 shows the average call setup time as a function of the average call bandwidth in the light edge-core topology. Overall, the performance in the light edge-core network is similar to the one in the dense edge-core network. However, there are two ”jumps” of the call setup time using the widest-shortest and shortest-minhop routing. The reason might be that most of the calls are successfully routed by using the alternate routing many times. Note that many of the alternate routing retries can significantly increase the average call setup time. This can probably be the transient effect in the network when there are many ”hot”

spots in the network. The hot spot happens because of the effect of the routing algorithm that many calls are routed through the same link.

5.2.5.2 Performance of Multiple Cluster Network

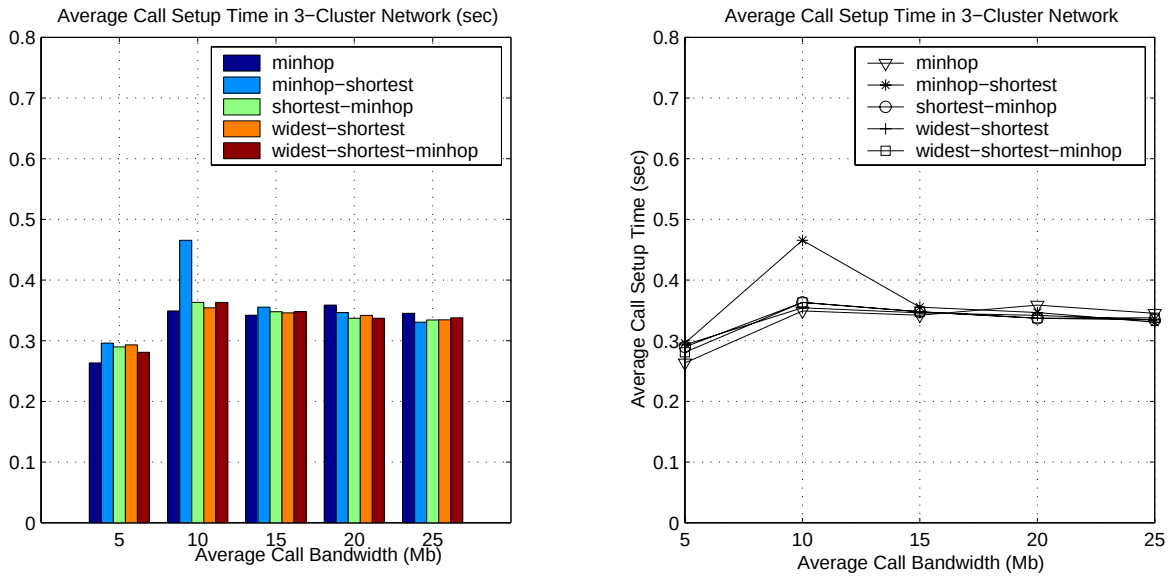

Figure 5.27: Average call setup time as a function of average requested bandwidth: 3-cluster Network

In Figure 5.27, we show the average call setup time as a function of average call holding time in the 3-cluster topology. Overall, the average call setup time does not linearly increase when the call bandwidth increases. The minhop routing tends to perform well when the call bandwidth is low, but does not when the call bandwidth is high. As we explained before, the ”jump” of the call setup time when using minhop-shortest routing happens because there are several crankbacks, and most of the call setup time is spent finding alternate routes.

Figure 5.28 shows the average call setup time as a function of average call holding time in the 8-cluster topology. Overall, the performance is similar to the one in the 3-cluster network. The minhop routing tends to perform well at the low call bandwidth, but not at the high call bandwidth.

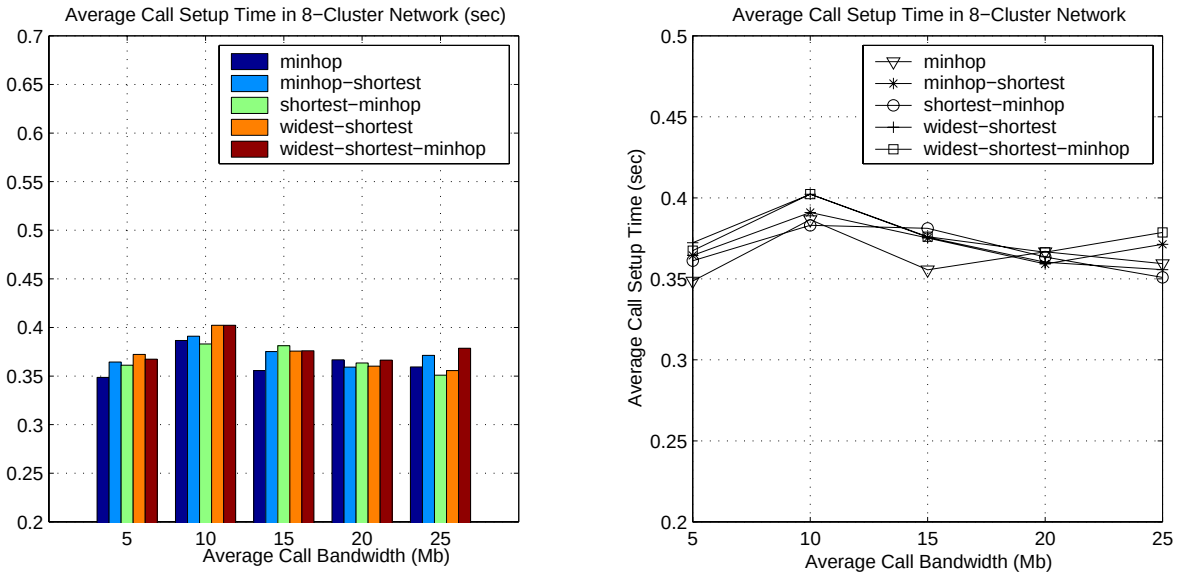

Figure 5.28: Average call setup time as a function of average requested bandwidth: 8-cluster Network

5.2.6 Evaluation of Routing Inaccuracy

This section shows the results of the routing inaccuracy by using our routing algorithms for minimum delay services. Unlike routing with bandwidth guaranteed services, the delay of the link is not dynamically changed over time. Therefore, the link delay information is the same for routing any call request. However, the routing has to consider other QoS parameters of the link such as bandwidth, which is dynamically changed. So the routing algorithm might find an ”incorrect” path, and our algorithms can perform differently in different topologies. For the rest of this section, we test our algorithms in two types of the topologies. The routing inaccuracy results of the edge-core topologies and the cluster network are shown in Section 5.2.6.1 and Section 5.2.6.2, respectively.

5.2.6.1 Performance of Edge-Core Networks

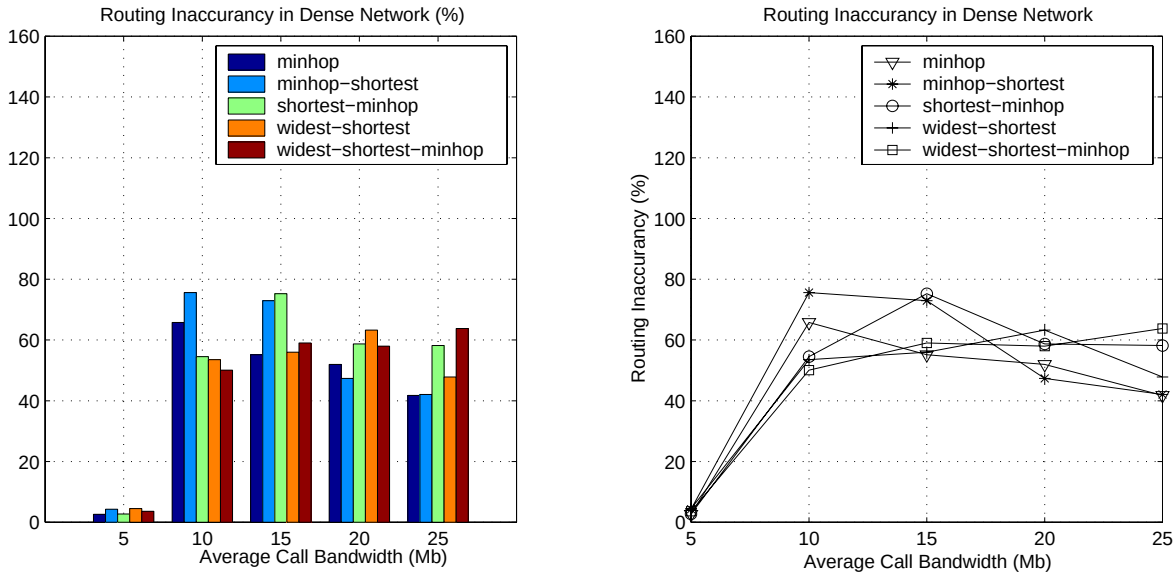

Figure 5.29: Routing Inaccuracy as a function of average requested bandwidth: Dense Network

In Figure 5.29, we show the routing inaccuracy as a function of the average call bandwidth in the dense edge-core network. Overall, at the low call bandwidth,

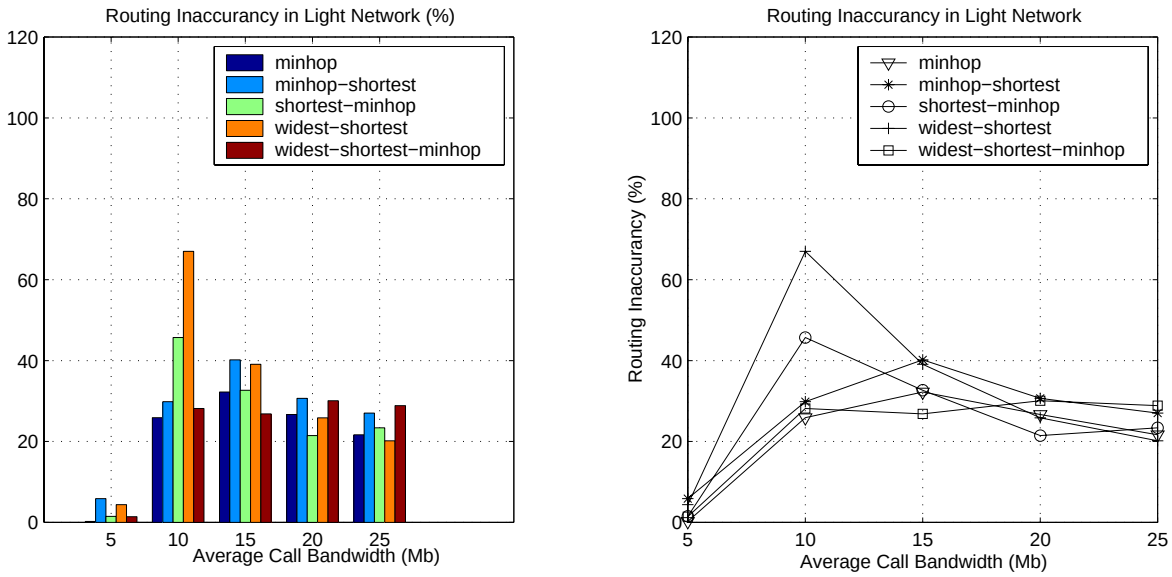

Figure 5.30: Routing Inaccuracy as a function of average requested bandwidth: Light Network

all routing algorithms seem to perform well because there are not many crankbacks nor is there much alternate routing. At the high-level of call bandwidth, the routing inaccuracy tends to increase. The average routing inaccuracy is about 60% when the call bandwidth is more than 10 Mb. Thus, the routing with the shortest criterion, such as the minhop-shortest routing, is probably not appropriate in the dense network since it tends to give the same route to the user request because the routing does not consider the available bandwidth of the links, and then it tends to create more crankbacks.

Figure 5.30 shows the routing inaccuracy as a function of the average call bandwidth in the light edge-core network. Overall, the performance is similar to the one in the dense network, but the average of routing inaccuracy is about 40% when the average call bandwidth is more than 10 Mb. In addition, there are two ”jumps” of the routing inaccuracy when the widest-shortest or shortest-minhop routing is used. That is probably a transient effect in the network as described before. In addition, these two jumps cause the jumps of the call setup time in the light edge-core network.

5.2.6.2 Performance of Multiple Cluster Network

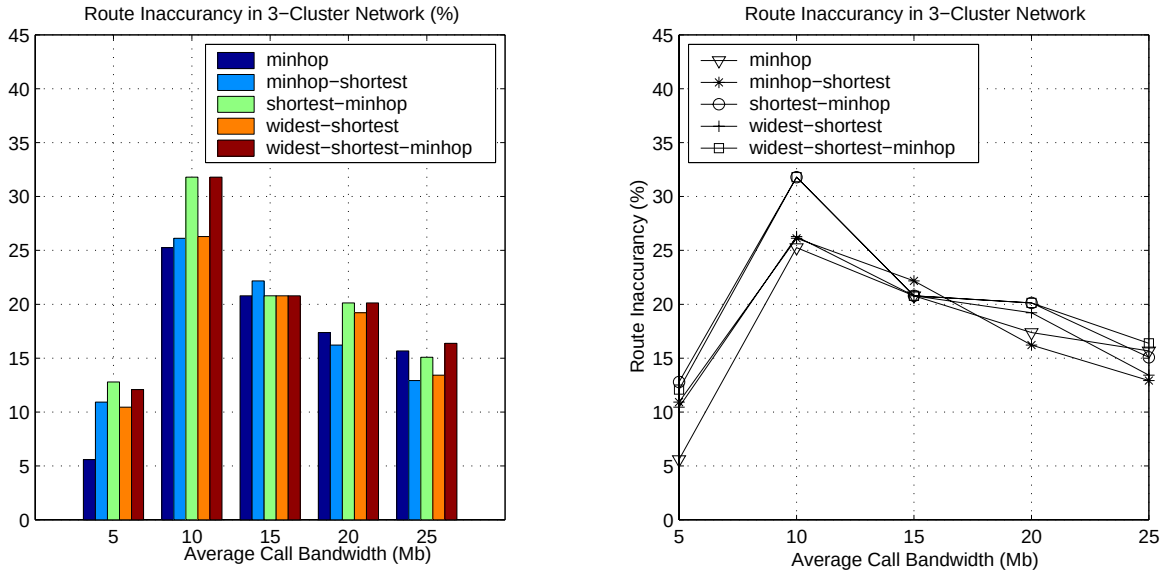

Figure 5.31: Routing Inaccuracy as a function of average requested bandwidth: 3-cluster Network

Figure 5.31 shows the routing inaccuracy as a function of the average call bandwidth in the 3-cluster network. For the 5 Mb call bandwidth, the routing inaccuracy is low because the traffic is not high, and the network is not congested. However, the routing inaccuracy tends to be increased when the call bandwidth is more than 10 Mb, and it slightly goes down when the call bandwidth increases. This can be explained that when the call bandwidth is higher, the network is more congested. Therefore, the call tends to be rejected because there is no feasible path to support the call request. The routing inaccuracy is slightly decreased because the number of rejected calls tends to be more than the number of crankbacked calls. Because the routing inaccuracy is based on the number of crankbacks, the decrease of the crankbacked calls makes the routing inaccuracy slightly decreased.

In Figure 5.32, we show the routing inaccuracy as a function of the average call bandwidth in the 8-cluster network. Overall, the routing inaccuracy is high even at the low bandwidth. This is because the 8-cluster network has a larger net-

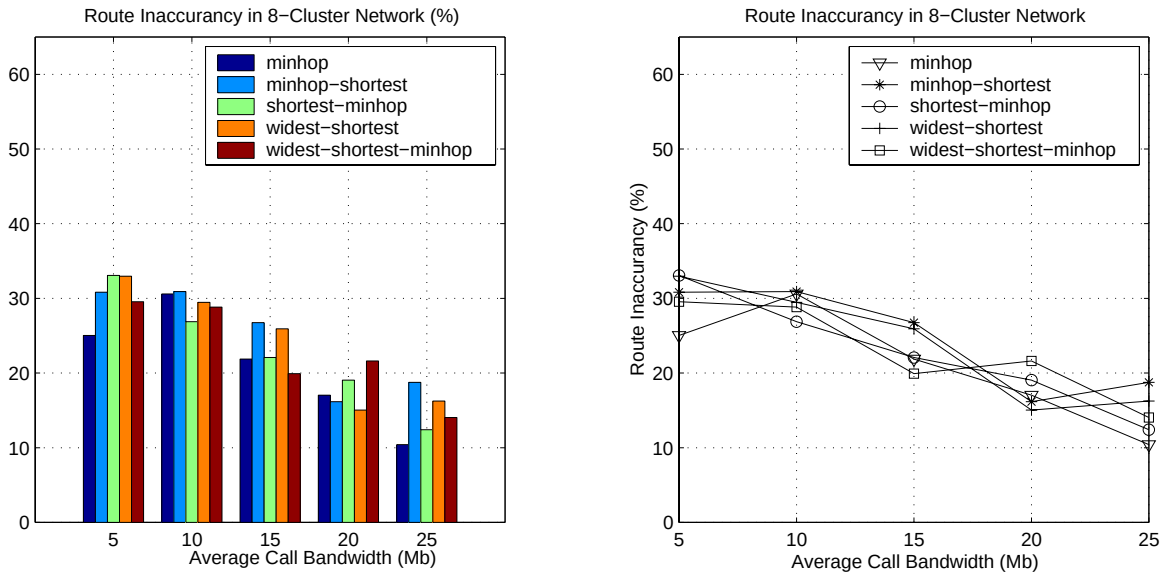

Figure 5.32: Routing Inaccuracy as a function of average requested bandwidth: 8-cluster Network

work diameter than the 3-cluster network. Therefore, in the 8-cluster network, call connections need to reserve longer paths than those in the 3-cluster network. This causes the 8-cluster network to be more congested than the 3-cluster network even at 5 Mb call bandwidth. Thereafter, the routing inaccuracy is slightly decreased. This is because of the same reason described above.

From Figure 5.31 and Figure 5.32, the routing inaccuracy is shown to be lower when the call bandwidth is higher. This does not mean the higher the call bandwidth, the better the network performance. If the call bandwidth is higher, the call failure rate which is proportional to the increase of the call bandwidth will be increased also. This means increasing the call bandwidth will slightly decrease the routing inaccuracy, but it significantly increases the call failure rate. The decrease of the routing inaccuracy is because calls in the network tend to be rejected at the source node because of no available route rather than crankbacked at an intermediate node because of inaccurate network state information at the source node.

5.3 Link Utilization

In this section, we evaluate the utilization of links in the network topology. We used the dense edge-core topology and the 3-cluster network in our experiments. In the edge-core topology, links are divided into two kinds, edge-core links and core-core links. An edge-core link is a link that connects between edge node and core node. The edge-core link normally has short delay and high capacity. Similarly, a core-core link is a link that connects between core node and core node. The core-core link commonly has a longer delay and higher capacity than the edge-core link. In the 3-cluster topology, there are two kinds of links, inside links and outside links. The inside link is a link that connects two nodes within the same peer group. On the other hand, the outside link is a link that connects between a node in one peer group and a node in another peer group. The delay of the inside links is uniformly distributed between 10 msec and 20 msec. On the other hand, the delay of outside links is uniformly distributed between 20 msec and 40 msec. Both inside links and outside links are OC-12 links.

The traffic used in our experiments is explained here. The call bandwidth was uniformly distributed with the average of 10 Mb. The call destination was uniformly selected among all other nodes. The call arrival distribution is the Poisson with an average of 5 second between calls. The call duration distribution is also the Poisson with an average of 60 seconds for each call. All the calls are CBR-typed. In the edge-core topology, there are 24 edge nodes, and each of them is connected to two hosts. Each host makes 1000 calls. Thus, the total number of calls in each experiment is 48,000 calls. In the cluster network, there are 24 nodes, but only 12 nodes are connected to host systems. Each of those nodes is connected to two hosts, and each host makes 1000 calls. Thus, the total number of calls in each experiment is 24,000 calls.

The rest of this section shows the results from our experiments. Section 5.3.1 explains the topologies we used and our routing algorithms. The link utilization of

the edge-core topology is shown in Section 5.3.2. Section 5.3.3 shows and explains the results of our experiments in the cluster network.

5.3.1 Routing Algorithms and Topology Used

For our experiments, we selected two types of topologies to evaluate the link utilization of the networks using our routing algorithms. We chose the dense edge-core topology and the 3-cluster topology. The edge-core topology is somewhat symmetrical. It is composed of the edge switch which is connected to many end-user systems. The analog of the edge switch can probably be the gateway in the network. The edge-core topology is also composed of core switches which are connected to only the core switch or the edge switch. The analog of the core switch would be the backbone of the network. The details of the dense edge-core topology are described in Section 4.1.2. Furthermore, the cluster topology is used to represent an asymmetrical network. It is composed of several peer groups, and each peer group is connected to other peer groups with high capacity and high delay links, called outside links. Within the peer group, there are switches connected to other switches using low capacity and low delay links, called inside links. The details of the 3-cluster network are described in Section 4.1.1.

For our experiments, we selected three routing algorithms to be evaluated: the minhop, shortest-minhop, and widest-minhop routing algorithms. The purpose of our experiments is to prove that the multiple criteria routing can improve the link utilization of the network. Therefore, we select the shortest-minhop and widest routing to compare their performance with the minhop routing. The rest of this section is organized as follows: the link utilization of the dense edge-core network is shown in Section 5.3.2, and Section 5.3.3 shows the link utilization of the 3-cluster network.

5.3.2 Link Utilization in Edge-core Topology

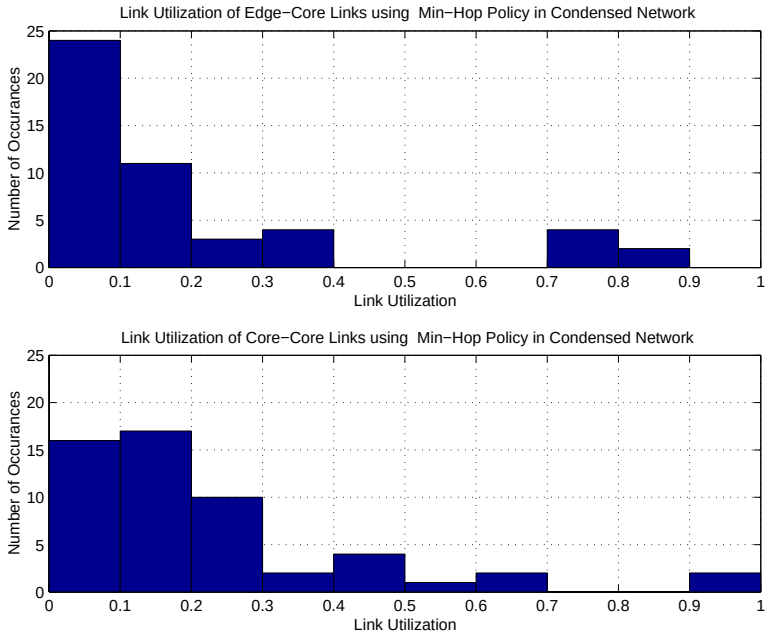

Figure 5.33: Link Utilization of Links Using Minhop Routing in Dense Edge-core Topology

In Figure 5.33, we show the histogram of the link utilization of edge-core and core-core links when the minhop routing is used. In the upper figure, it shows that most of the edge-core links have low utilization. However, there are about 6 edge-core links that have high link utilization. The lower figure shows that most of the core-core links have about 0 to 0.3 utilization, and there are two links that have a very high utilization. The links with high utilization are congested, and they are bottleneck links.

Figure 5.34 shows the histogram of the link utilization when the shortest-minhop routing is used. Compared to the minhop routing, it shows that the minhop-shortest slightly improves the utilization of edge-core links and core-core links. This is because the minhop-shortest routing does not consider the available bandwidth of links when routing. Therefore, the routing with the widest criterion would improve the link utilization of the network.

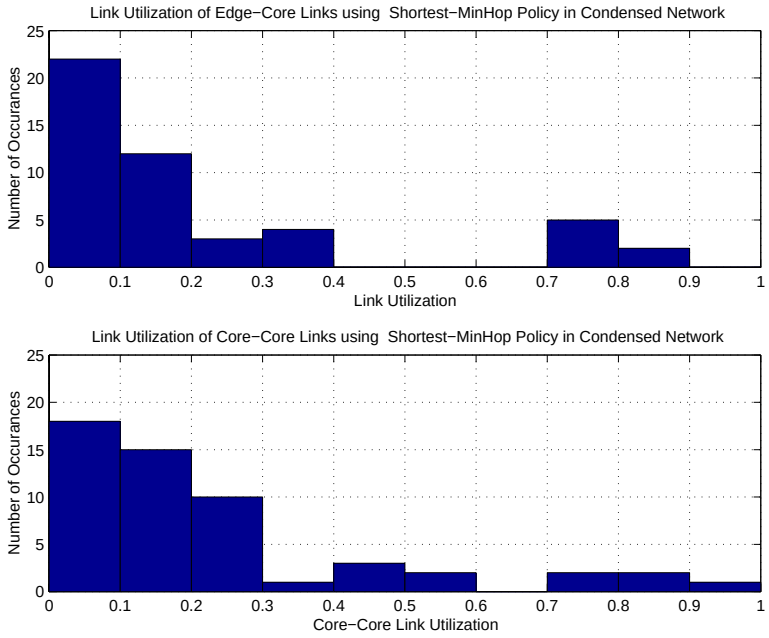

Figure 5.34: Link Utilization of Links Using Shortest-minhop Routing in Dense Edge-core Topology

In Figure 5.35, we show the link utilization histogram of the edge-core and core-core links when the widest-minhop routing is used. It shows that the widest-minhop routing can reduce the number of overly congested edge-core links and core-core links. In addition, it balances the utilization of all the links at the highest possible level.

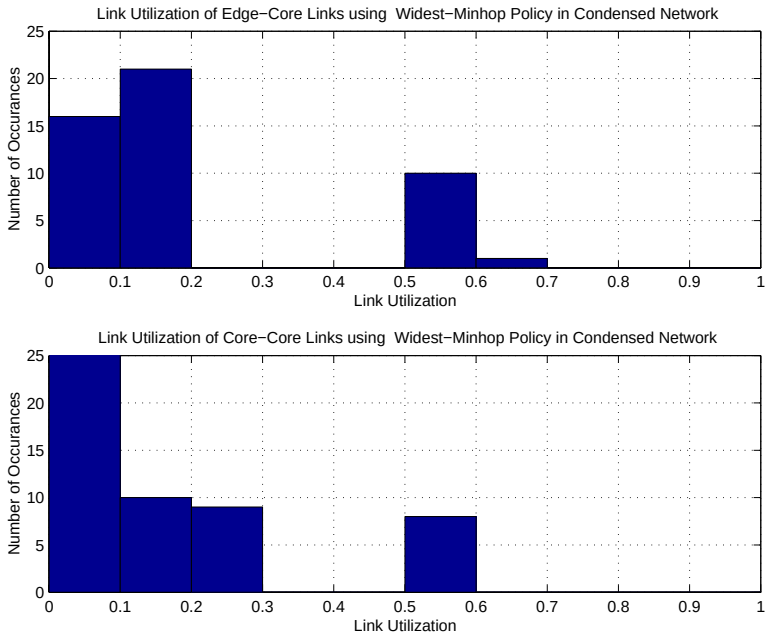

Figure 5.35: Link Utilization of Links Using Widest-minhop Routing in Dense Edge-core Topology

5.3.3 Link Utilization in Cluster Network

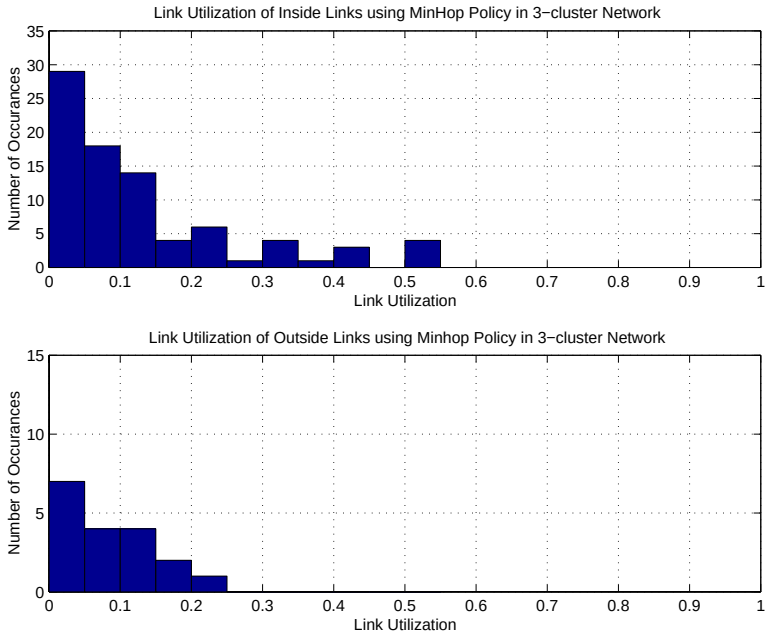

Figure 5.36: Link Utilization of Links Using Minhop Routing in 3-cluster Network

In Figure 5.36, we show the histogram of the link utilization of edge-core and core-core links when the minhop routing is used. It shows that there are many edge-core links which have a 0.5 utilization. Furthermore, Figure 5.37 shows the histogram of the link utilization when the shortest-minhop routing is used. Compared to using the minhop routing, the link utilization of edge-core links was slightly improved. The link utilization of core-core links is also improved to some degree.

In Figure 5.38, we show the link utilization histogram of the edge-core and core-core links when the widest-minhop routing is used. The link utilization of edge-core links is improved much more compared to using a minhop and shortest-minhop routing. Also, the core-core link utilization is improved much more. It reduces the number of high utilization links and increases the number of low utilization links.

In summary, by using the shortest-minhop routing, the utilization of the

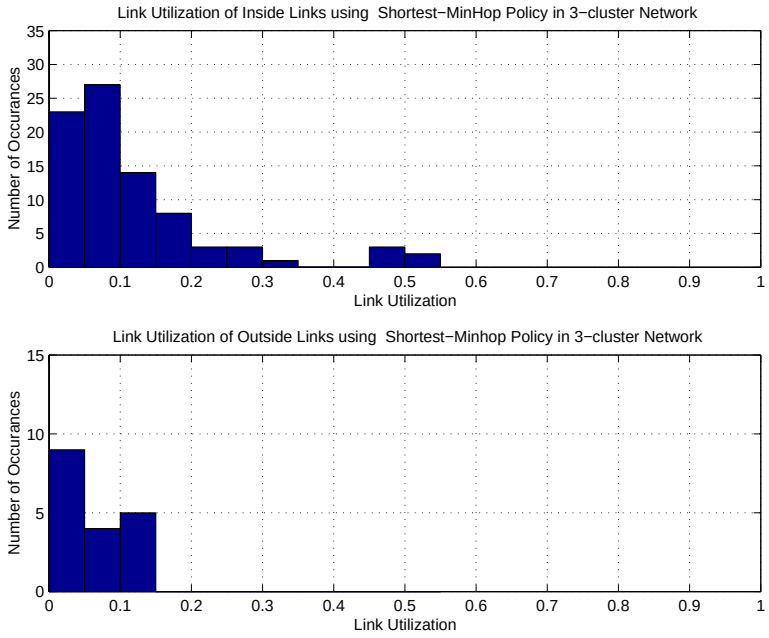

Figure 5.37: Link Utilization of Links Using Shortest-minhop Routing in 3-cluster Network

core-core link can be slightly improved, but that of the edge-core link is not improved much. Furthermore, the widest-minhop routing can significantly improve the edge-core and core-core link utilization. By using the widest-minhop routing in the edge-core or cluster networks, there will not be many congested links or bottle necks. Therefore, the widest-minhop routing can efficiently utilize the network resource and improve the overall performance of the network.

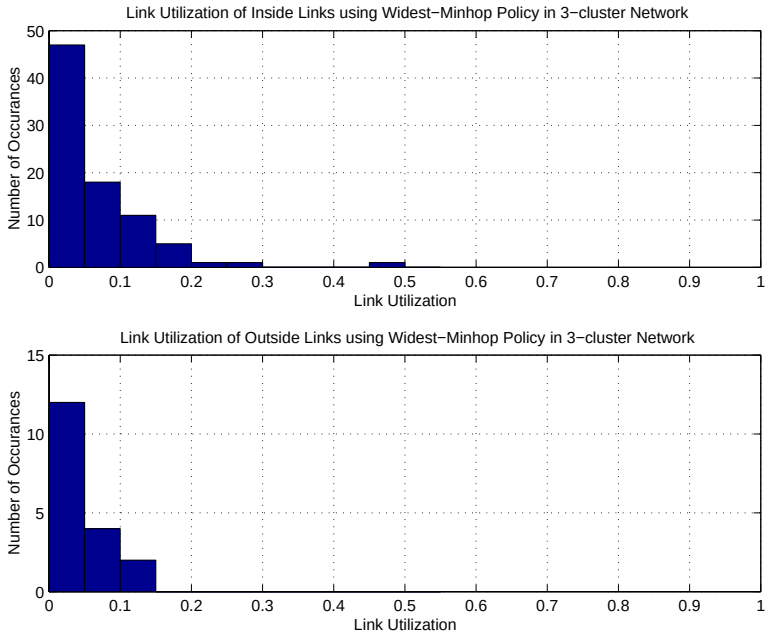

Figure 5.38: Link Utilization of Links Using Widest-minhop Routing in 3-cluster Network

5.4 Alternate Routing with MCRAs

In a connection-oriented network like the ATM, a route for a call is calculated by the PNNI routing protocol before the call request starts signaling to the destination following the pre-calculated route. At the source node, the routing protocol takes the resource information from its database and finds a feasible route that can support the call request. After the call request gets a feasible path, it will reserve the resource along the way to the destination. Since the resource information contained at the source node is not ”perfect”, the information may be out-of-date, so a link within the path given by the routing protocol might not be able to support the call request while the call is routing through the network. The call will be *”crankbacked”* to the source node in order to get an *alternate route* for a retry.

Therefore, alternate routing plays an important role by increasing the chance that the user call will succeed. The alternate routing can probably decrease the call failure rate. However, doing alternate routing tends to increase the call setup time. Thus, in this section, we investigate the impact of increasing the number of alternate route retries on the call failure rate and call setup time in two kinds of networks using our multiple criteria routing algorithms. The rest of this section is organized as follows. Section 5.4.1 explains our test scenarios. Section 5.4.2 shows the performance of routing algorithms in the edge-core network topology as a function of the number of alternate routing retries. In addition, the performance of routing algorithms in the cluster network is shown in Section 5.4.3.

5.4.1 Routing Policies and Topologies

To evaluate the number of alternate routing retries, we used two topologies in our experiments, the edge-core topology and the cluster topology. The details of these two topologies are explained in Section 4.1. We used many of our multiple criteria routing algorithms for the experiments. We divided those algorithms into

two groups, the shortest group and the widest group. In the shortest group, we use the shortest-minhop, widest-shortest-minhop, widest-shortest and minhop-shortest routing to perform in two topologies, and we compared their performance to that of the minhop routing. This is because we also wanted to compare the single criterion to multiple criterion routing when the number of alternate routing retries was increased.

On the other hand, in the widest group, we used the widest-minhop, shortest-widest-minhop, shortest-widest, and minhop-widest routing to be used in two topologies. We compared the results using those routing algorithms to those using the minhop routing algorithm. We wanted to evaluate our multiple criteria routing algorithm to the single criterion routing such as minimum hop count routing when the number of alternate routing retries is increased. In Section 5.4.2, we show the results from our experiments in the edge-core topologies and Section 5.4.3 shows the results in the multiple cluster networks.

5.4.2 Performances of Dense Edge-core Topology

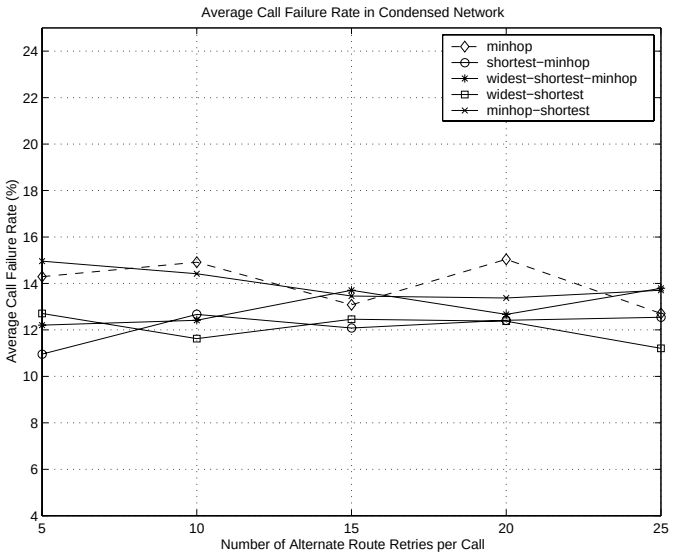

Figure 5.39: The Call Blocking Rate in the Dense Edge-core Topology Using Shortest Group Routings

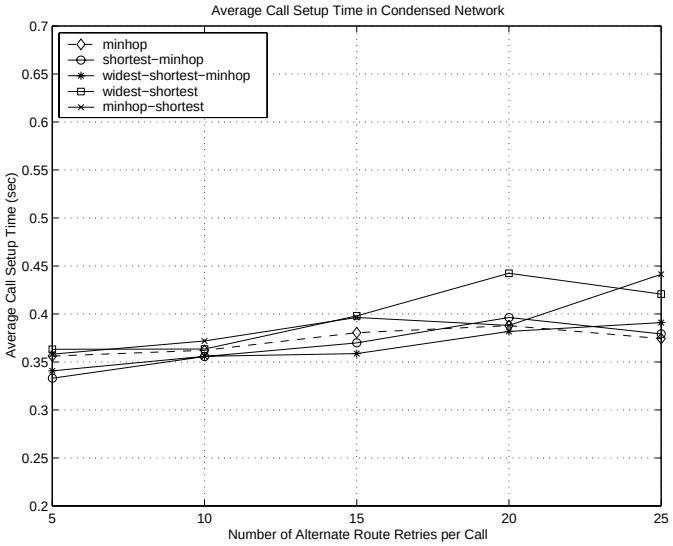

Figure 5.40: The Call Setup Time in the Dense Edge-core Topology Using Shortest Group Routings

In Figure 5.39 and Figure 5.40, we show the average call failure rate and the average call setup time as a function of the number of alternate routing retries of the shortest group respectively. In Figure 5.39, the average call failure rate is slightly improved when the number of alternate routing retries increases. The call failure rate is slightly improved because some failed calls have a chance to find another route using alternate routing, but it is difficult to find an alternate path when the network traffic is high. Also giving an opportunity of a failed call to be routed again for many times probably increases the network traffic, and the call failure rate may not be improved at all. In addition, the average call setup time tends to increase as shown in Figure 5.40, but the average call failure cannot be improved much. Therefore, the algorithms in the shortest group are probably not appropriate to use for improving the call failure rate by using alternate routing.

Figures 5.41 and 5.42 show the average call failure rate and average setup time as a function of the number of alternate routing retries of the widest group respectively. In Figure 5.41, an increase in the number of routing retries tends to reduce the call failure rate when the shortest-widest routing is used, but its failure

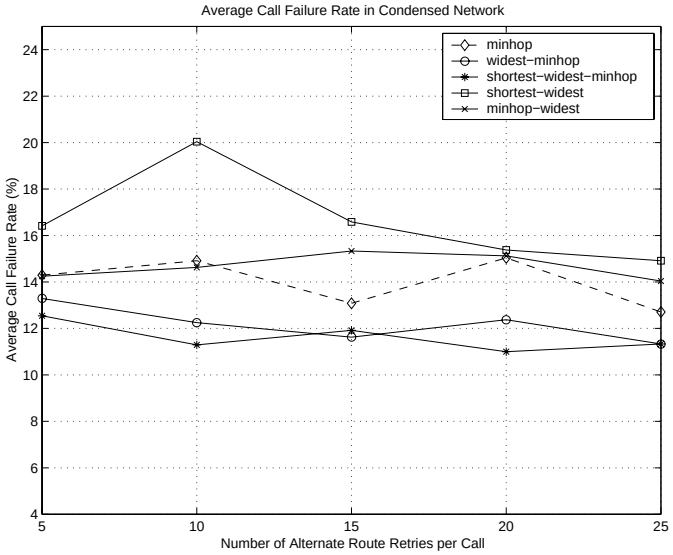

Figure 5.41: The Call Blocking Rate in the Dense Edge-core Topology Using Widest Group Routings

is still high compared to the others. When the widest-minhop and the shortest-widest-minhop routing are used, the failure rate tends to be reduced at about 3% (72 calls), and they perform better than the minhop routing. The shortest-widest-minhop routing tends to perform slightly better than the widest-minhop routing. In addition, the average call setup time of the widest-minhop routing is slightly increased as shown in Figure 5.42. Furthermore, the call setup time using shortest-widest-minhop routing is almost the same as the widest-minhop routing. Therefore, the shortest-widest-minhop routing performs slightly better than the widest-minhop routing, but it does not spend more time than the widest-minhop routing. Thus, the shortest-widest-minhop would be the best algorithm to use with alternate routing for improving the average call failure rate in edge-core networks.

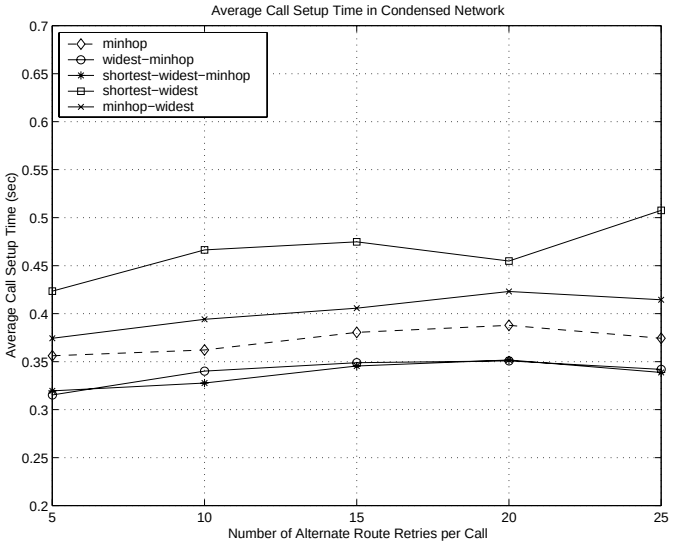

Figure 5.42: The Call Setup Time in the Dense Edge-core Topology Using Widest Group Routings

5.4.3 Performances of the 3-cluster Network

Figures 5.43 and 5.44 show the average call failure rate and the average call setup time as a function of the number of alternate routing retries of the shortest routing groups, respectively.

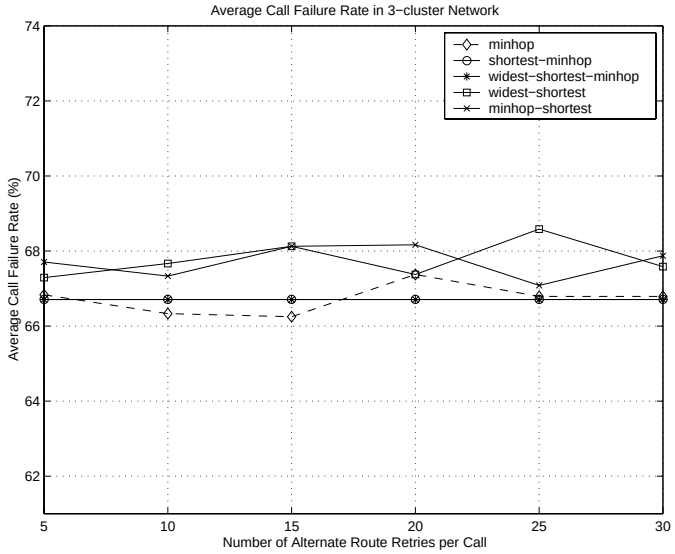

Figure 5.43: The Call Blocking Rate in the 3-cluster Network Using Shortest Group Routing

Overall, the call failure rate of the 3-cluster network topology is hardly improved using any routing algorithm. In addition, the call setup time of this topology is also hardly increased. This would mean that the routing in the shortest group is not very successful in finding an alternate route for the user request. The routing algorithm keeps finding an alternate route until the number of retries exceeds a limit, and the call is rejected. Note that the call setup time of the rejected call is not calculated in the average call setup time. Thus, the average call failure rate is hardly improved and the average call setup time is hardly increased by using the routing in the shortest group to find an alternate route.

In Figure 5.45 and Figure 5.46, we show the average call failure rate and the average call setup time as a function of the number of alternate routing re-

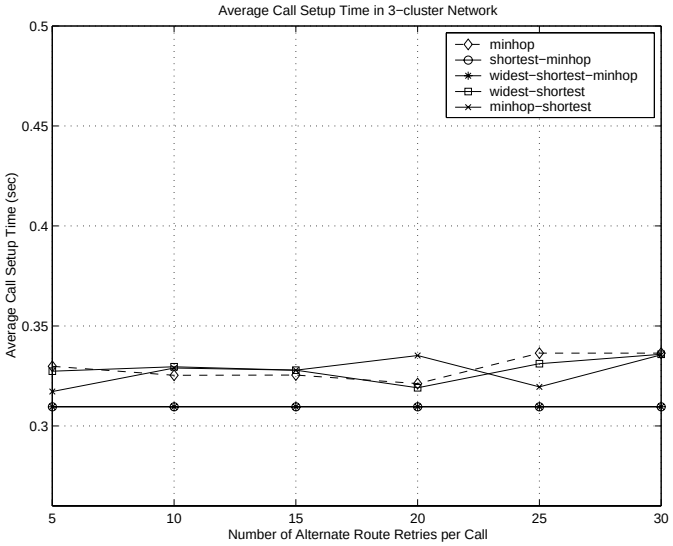

Figure 5.44: The Call Setup Time in the 3-cluster Network Using Shortest Group Routing

tries of the widest routing groups, respectively. On average, the call failure rate is slightly improved when the number of alternate route retries increases. The shortest-widest routing tends to slightly improve the failure rate of about 2% (48 calls). However, the call failure rate is still higher than that using the minhop routing. Overall, an increase of the number of routing retries hardly improves the call failure rate. In addition, the call setup time is slightly increased as shown in Figure 5.46. This would mean that there is a small number of alternate routes which the alternate routing can successfully find. In summary, within the 3-cluster network, an increase in the number of alternate routing retries probably does not improve the average call failure rate much.

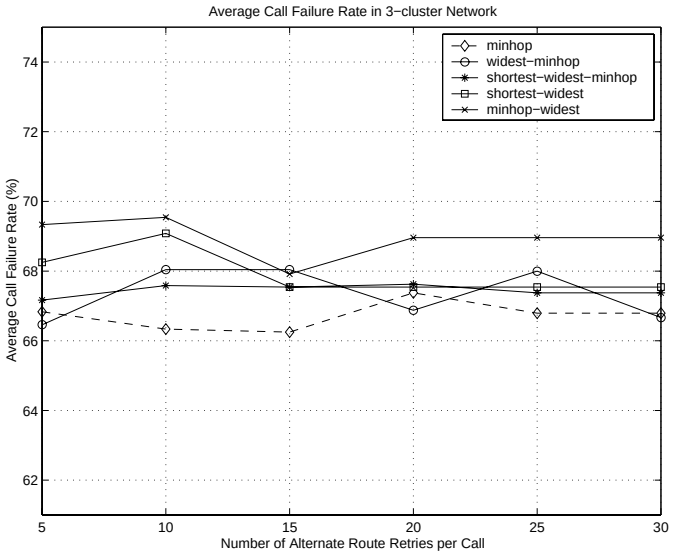

Figure 5.45: The Call Blocking Rate in the 3-cluster Network Using Widest Group Routing

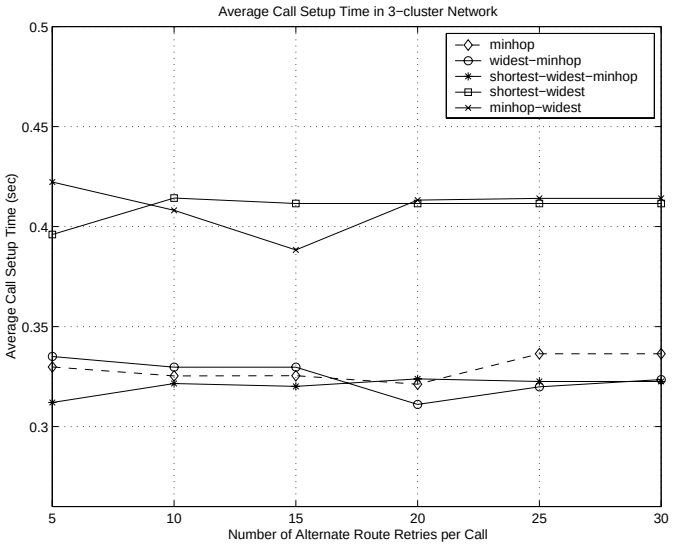

Figure 5.46: The Call Setup Time in the 3-cluster Network Using Widest Group Routing

5.5 Effects of Changing the Network Core Density

The call failure rate in a network can be improved by installing a new core link in order to share traffic from a congested link. The user call request will be more successful if we have fewer congested links. Therefore, increasing the number of core links, which also increases the *connectivity* of the network, should improve the average call failure rate. However, adding as many new links as possible may not always improve the call failure rate because the new links may have a low utilization. Therefore, the *optimal* number of new links to be installed to improve the average call failure rate needs to be carefully determined.

To determine the optimal number for best set of links, we selected the edge-core network topology to be used in our experiments because the edge-core topology is quite similar to many enterprise networks. We use three edge-core topologies, low-dense, medium-dense, and high-dense, each of which has 12 core nodes and 12 edge nodes. One edge node is connected to two core nodes by the edge or local links. Each core node is connected to other core nodes by the core links. The number of core links of these three networks is different, which also makes their connectivity different. The link characteristics of these three networks are summarized in Table 5.1. Note that the S–>L link means the link between ”small” capacity core node (S) and the ”large” capacity core node (L). The more details of edge-core topology are described in Section 4.1.2.

In our experiments, we used all of our multiple criteria routing algorithms for evaluation. All the multiple criterion routing algorithms are explained in Section 3.3.3. For the rest of this section, the average call blocking rate and the average call setup time of three edge-core networks are shown in Section 5.5.1.

Type of Links	Capacity	Type of Networks		
		Low-Dense	Medium-Dense	High-Dense
S–>L	OC-3	12	12	12
S–>L	OC-12	0	9	12
L–>L	OC-12	6	6	9
S–>S	OC-12	0	0	3
Total Core Links		18	27	36
Total Edge Links		24	24	24
Connectivity		1.75	2.125	2.5

Table 5.1: The Characteristics of Three Edge-core Networks with Different Connectivities

5.5.1 Average Call Blocking Rate and Average Call Setup Time

Figure 5.47 shows the average call failure rate as a function of the connectivity when the widest group routing is used. Overall, it shows that the call failure rate is decreased when the connectivity increases from 1.75 to 2.125. However, when the connectivity is further increased from 2.125 to 2.5, the call failure rate is not improved. In addition, the call setup time as shown in Figure 5.48 is slightly increased when the network connectivity increases. The reason could be the fact that when the connectivity increases, the routing tends to give a longer feasible path which will decrease the possibility of later arrivals being successfully routed.

In addition, Figure 5.49 shows the average call failure rate as a function of network connectivity when the routing algorithms in the shortest group are used. Overall, the average call failure rate was significantly decreased when the connectivity increased from 1.75 to 2.125. This means at a 1.75 connectivity, there are many congested links in the network, which are likely unable to support many call requests, and the requests are rejected. When we increase the number of core links, which make the connectivity change to 2.125, the call failure rate is rapidly decreased. However, when we further increase the connectivity to 2.5, it seems that the call failure rate is not improved. Also the failure rate seems to be slightly

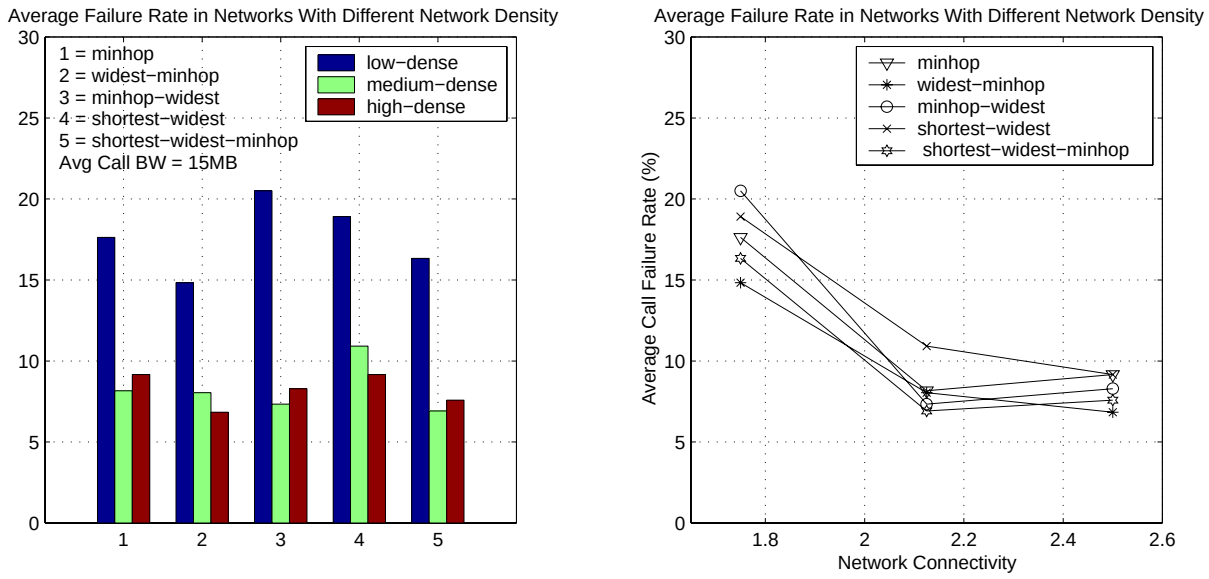

Figure 5.47: The Call Blocking Rate using Routing with Widest Criteria

increased. The reason is that the routing tends to give a longer feasible path when the number of links is increased. The longer path for the call request will penalize later call arrivals and increase the average call setup time as shown in Figure 5.50.

In summary, the average call failure rate can be reduced by increasing the network connectivity. However, a very large connectivity can deteriorate the performance of the network. The average call setup time increases and the average link utilization of the network is low. In addition, at some level of increasing the connectivity, the average call failure rate may not decrease more because the routing algorithm tends to give a longer path which will penalize later call arrivals.

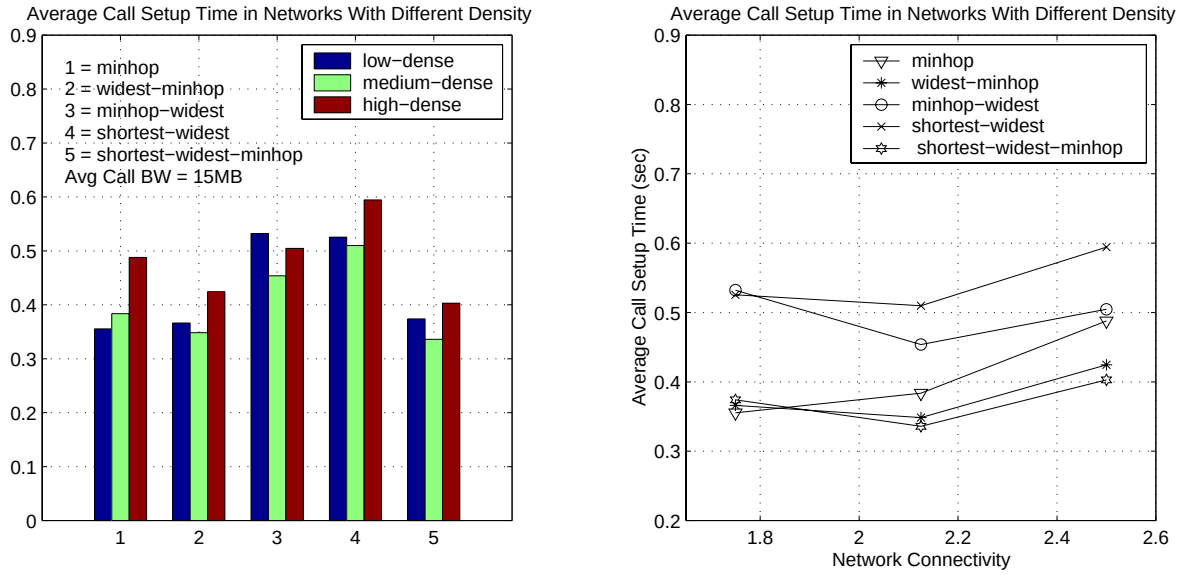

Figure 5.48: The Call Setup Time using Routing with Widest Criteria

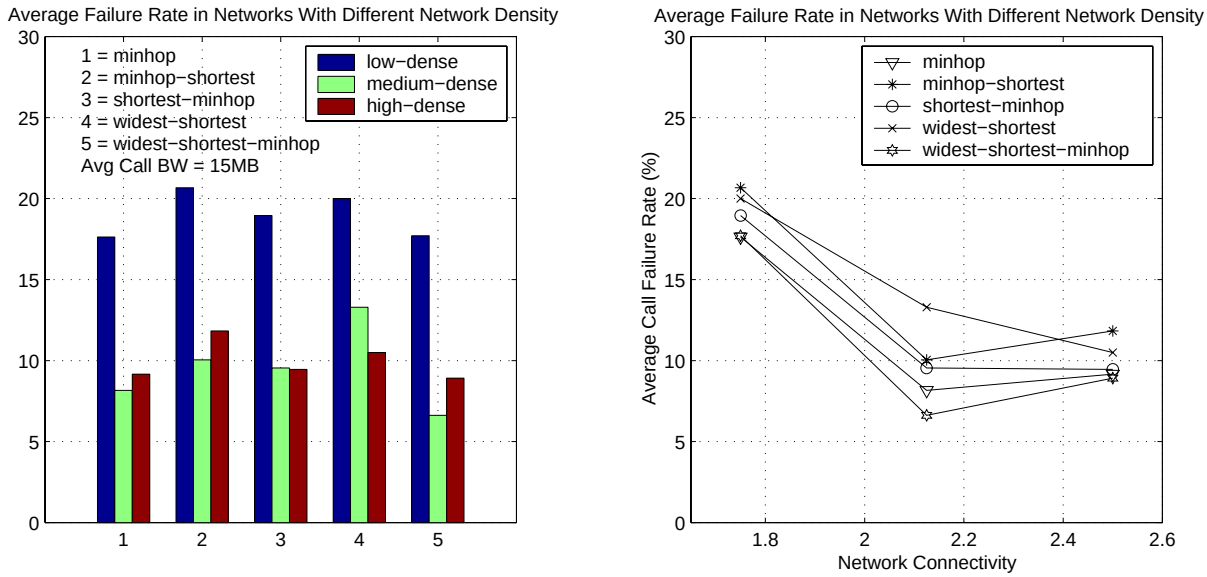

Figure 5.49: The Call Blocking Rate using Routing with the Shortest Criteria

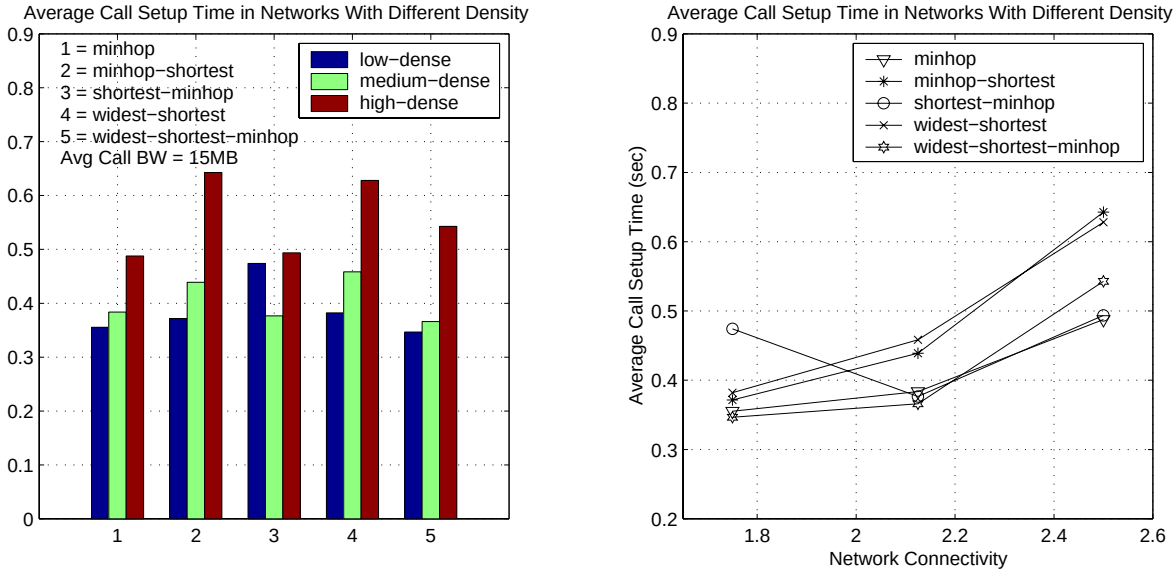

Figure 5.50: The Call Setup Time using Routing with the Shortest Criteria

Chapter 6

Conclusion and Future Work

Asynchronous Transfer Mode (ATM) technology has been used in high-speed network communication. The ATM network is expected to become the backbone network for high-speed multimedia services because it is able to support a large scale network with robustness, scalability, and Quality of Service (QoS). To support a large-scale network, a dynamically automatic network configuration mechanism which can automatically control a topology of switches and links is required. The ATM forum [6], therefore, has introduced a dynamic configuration protocol for supporting private networks called Private Network to Network Interface (PNNI) protocol. The PNNI standard has introduced two kinds of protocols, the signaling protocol and the routing protocol. The PNNI signaling protocol was designed for a call connection setup using message exchanges between switches. The PNNI routing protocol was designed for discovering the network elements, exchanging network resource information, and selecting feasible paths that are able to support user requests using a routing algorithm.

The routing algorithm plays an important role in finding a feasible path, and it can affect both the connection setup delay and call blocking probability and also influence both the quality of service for users and network utilization which affects the network efficiency for providers. To select a feasible path which is efficient for not only the user but also the provider, a routing scheme which supports both

user and provider constraints is necessary. Since the user and provider constraints address different properties, *a routing algorithm must select a route based on multiple criteria*.

6.1 Problem Statement and Our Implementation

In traditional data networks, the network is usually characterized by a single metric such as hop count or delay, and the shortest-path algorithm is used for path computation. When we want to do multi-QoS routing, we face a basic problem of finding a path that satisfies the multiple constraints imposed by the QoS requirement contained in the user's call request. The problem is that routing with more than one requirement is an *NP-complete* problem [24]. For this reason, we introduce a *multiple criteria routing algorithm* (MCRA) based on a heuristic approach to avoid the NP-complete problem. We introduce several routing algorithms as follows:

- Double-criteria Routing Algorithms
  - widest-shortest and shortest-widest algorithms
  - widest-minhop and minhop-widest algorithms
  - shortest-minhop and minhop-shortest algorithms
- Triple-criteria Routing Algorithms
  - widest-shortest-minhop algorithm
  - shortest-widest-minhop algorithm

A traditional routing algorithm that is widely used is the *single source shortest path* (SSSP) from Dijkstra. We modified the SSSP algorithm to have two or three cost functions instead of just one. The reason is that we want to improve the SSSP algorithm to be able to find a path with more than one criterion. We also proposed the widest algorithm to be used in a part of the shortest-widest and minhop-widest

algorithms. The D_widest algorithm is created based on the idea of Dijkstra's algorithm with the exception that it can take the cost such as the bandwidth for routing. Furthermore, we tried to prove that routing with three criteria might perform even better than that with two criteria.

To evaluate our algorithms, we proposed various kinds of networks such as the edge-core topology and the cluster network, and we also used several kinds of metrics such as call blocking rate, call setup time, and routing inaccuracy. The conclusion of our results is discussed in Section 6.2.

6.2 Our Results of the Performance Evaluations

In our experiments, we evaluated the performance of routing algorithms with a bandwidth guarantee, and with a minimum delay service. The average call blocking probability, average call setup time, and routing inaccuracy were measured to show the performance of the different algorithms.

For the bandwidth guaranteed routing, the average call failure rate was found to increase when the average call bandwidth or call holding time increased. Furthermore, the widest-minhop and the shortest-widest-minhop routing tended to perform better than minhop routing. However, the minhop-widest and the shortest-widest routing tended to perform worse than the minhop routing because they tend to give a longer path at the user's request. In addition, a longer path penalizes later arrivals.

Moreover, the average call setup time was also increased when the call bandwidth increased. At the low call bandwidth, the call setup time tended to be low. However, the call setup time seemed to be higher when the call bandwidth increased because the number of crankbacks was increased. When the crankback occurs, the call setup time is increased by the alternate routing time. The widest-minhop and the shortest-widest-minhop routing tended to have the lower call set-

up time than the minhop routing. The shortest-widest and the minhop-widest routing have the higher setup time than other routing algorithms because they run the D_widest algorithm and the modified Dijkstra's algorithm in order to find the maximum bandwidth and the minimum additive cost.

For the minimum delay service routing, the average call failure rate is increased when the call bandwidth or the call holding time increases. The widest-shortest-minhop performs slightly better than other routing algorithms. However, in some cases the minhop routing tends to perform slightly better because those algorithms that do not consider the maximum bandwidth criterion to be a primary criterion or do not consider it at all. This means the maximum bandwidth is a crucial criterion in the multiple criteria routing.

We also evaluated the link utilization, and we have shown that we can improve the link utilization by using multiple criteria routing algorithms such as the widest-minhop routing instead of using a single criteria routing algorithm such as the minhop routing.

In addition, we evaluated the effects of the routing performance in the networks when the number of alternate routing retries increases. We found that an increase of the number of routing retries hardly reduces the call failure rate. The reduction is about 2% - 3%. Also, the call setup time was increased when the number of route retries increased.

Finally, we evaluated the performance of the network when the core density of the network changed. We found that increasing the number of core links reduces the call failure rate but not always. At a certain point, an increase in the network density does not reduce the call failure because the amount of resource information is larger. The performance of the flooding mechanism in the PNNI network which is used to exchange information is deteriorated by the large amount of resource information. Therefore, the resource information provided to the routing function tends to be inaccurate, and it degrades the routing performance. In summary,

the improvement of the network performance does not continue beyond a certain point as the number of core links in the edge-core network increases.

6.3 Future Work

There are several issues here about PNNI ATM networks that can support a large-scale network. One of the most important issues about the PNNI is the scalability. To be able to scale a large network, the PNNI introduces a hierarchical level of multiple peer group networks. The issues regarding the hierarchical PNNI network for future consideration are listed as follows:

- Aggregating topology information using different aggregation methods in the multiple peer group network.
- Finding a feasible path in the hierarchical network with ”imperfect” topology information using different routing schemes in the aggregated multiple peer group network.
- The pre-computation routing schemes for the hierarchical network that compromise between a decrease of call setup times and an increase of the call success rate.

Bibliography

[1] ATM Forum Technical Committee. *Traffic Management Specification version 4.0 (af-tm-0056.000).* ATM FORUM, April 1996.

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

[3] ATM Forum Technical Committee. *ATM Private Network-Network Interface specification version 1.0 (af-pnni-0055.000).* ATM Forum, March 1996.

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

[5] Thomas Cormen and et al. *Introduction to Algorithms.* McGraw-Hill Book Company, New York, USA, 1989.

[6] ATM FORUM. http://www.atmforum.com/.

[7] R. M. Fujimoto, S.R. Das, and K.S. Panesar. Georgia tech time warp (gtw version 2.3) programmer's manual. Technical report, College of Computing, Georgia Institute of Technology, 1994.

[8] Private NNI Working Group. Simple gcac errata. ATM Forum Contribution/96-0896, June, 1006.

[9] R. Guerin, A. Orda, and D. Williams. Qos routing mechamisms and ospf extensions. IETF Internet Draft (draft-guerin-qos-routing-ospf-00 .txt), November 1996.

[10] F. Hao, E. W. Zegura, and S. Bhatt. Performance of the pnni protocol in large networks. *IEEE ATM WORKSHOP*, pages 315–323, May 1998.

[11] A. Iwata, R. Izmailov, H. Suzuki, and B. Senggupta. Pnni routing algorithm for multimedia atm internet. *NEC Research and Development*, 38, January 1997.

[12] J. M. Jeffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[13] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. *Proceeding of IEEE International Conference on Network Protocols*, October 1997.

[14] Q. Ma, P. Steenkiste, and H. Zhang. Routing high bandwidth traffic in max-min fair share networks. *ACM SIGCOMM'96*, 123:206–217, Aug 1996.

[15] Qingming Ma and Peter Steenkiste. Routing traffic with quality-of-service guarantees in integrated services networks. In *Proceedings of Workshop on Network and Operating Systems Support for Digital Audio and Video*, July 1998.

[16] J. Moy. Ospf version 2. Request for Comment (RFC) 1583, March 1994.

[17] H. D. Neve and P. V. Mieghem. A multiple quality of service routing algorithm for pnni. *IEEE ATM WORKSHOP*, pages 324–328, May 1998.

[18] K. Perumalla and R. Fujimoto. A c++ instance of ted. Technical Report GIT-CC-96-33, College of Computing, Georgia Institute of Technology, 1996.

[19] K. Perumalla and R. Fujimoto. Gtw++: An object-oriented interface in c++ to the georgia tech time warp system. Technical Report GIT-CC-96-09, College of Computing, Georgia Institute of Technology, September 1996.

[20] K. Perumalla, A. Ogielski, and R. Fujimoto. Metated: A meta language for modeling telecommunication networks. Technical Report GIT-CC-96-32, College of Computing, Georgia Institute of Technology, 1996.

[21] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, Inc., 1996.

[22] 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 Telecommunication Union, 1995.

[23] Quan Sun and Horst LangenDőrfer. A new distributed routing algorithm for supporting delay-sensitive applications. *Computer Communications*, 21(6):572–578, May 1998.

[24] Z. Wang and J. Crowcroft. Quality of service routing for supporting multimedia applications. *IEEE J. Selected Areas of Comm*, 14:1228–1234, 1996.

[25] R. Widyono. The design and evaluation of routing algorithms for real-time channels. Technical Report TR-94-024, Department of EECS, University of California, Berkeley, 1994.

Appendix A

Sample Scripts

This section shows the simulation scripts describing our network using in our experiments. These scripts are used to run our simulator developed at University of Kansas.

A.1 Dense Edge-core Topology Script

```
parameter_block node  spartan{
      crankback_retries       =     10,
      flooding_threshold      =     2,
      default_flooding_period =     1800,
      prop_constant           =     25,
      default_flooding_factor =     5,
      hello_timer             =     5,
      summary_timer           =     5,
      ptsp_timer              =     5,
      ack_timer               =     3,
      request_timer           =     5,
      routing_policy          =     min_hop_widest,
      numports                =     40,
      process_time            =     5.0,
      queuesize               =     5000
};

parameter_block host newton{
      duration              =     400,
```

```
      calltype              =     cbr,
      call_bw               =     [uniform 1000 100000],
      calls                 =     100,
      queuesize             =     5000,
      host_process_time     =     3.0,
      destination           =     uniform_any,
      arrival_distribution  =     poisson,
      arrival_mean          =     5,
      duration_distribution =     poisson,
      duration_mean         =     60
};

node  S1{ parameter_block spartan,
          address = 0x4705ffef5600000000000000a100000000000000
        };
node  S2{ parameter_block spartan,
          address = 0x4705ffef5600000000000000a200000000000000
        };
node  S3{ parameter_block spartan,
          address = 0x4705ffef5600000000000000a300000000000000
        };
node  S4{ parameter_block spartan,
          address = 0x4705ffef5600000000000000a400000000000000
        };
node  S5{ parameter_block spartan,
          address = 0x4705ffef5600000000000000a500000000000000
        };
node  S6{ parameter_block spartan,
          address = 0x4705ffef5600000000000000a600000000000000
        };
# --------------- Large Nodes --------------- #

node  L1{ parameter_block spartan,
          address = 0x4705ffef5600000000000000b100000000000000
        };
node  L2{ parameter_block spartan,
          address = 0x4705ffef5600000000000000b200000000000000
        };
node  L3{ parameter_block spartan,
          address = 0x4705ffef5600000000000000b300000000000000
        };
node  L4{ parameter_block spartan,
          address = 0x4705ffef5600000000000000b400000000000000
        };
node  L5{ parameter_block spartan,
```

```
          address = 0x4705ffef5600000000000000b500000000000000
        };
node  L6{ parameter_block spartan,
          address = 0x4705ffef5600000000000000b600000000000000
        };

# --------------- Edge Nodes ------------------- #

node  E16{ parameter_block spartan,
           address = 0x4705ffef5600000000000000100000000000000000
         };
node  E17{ parameter_block spartan,
           address = 0x4705ffef5600000000000000110000000000000000
         };
node  E18{ parameter_block spartan,
           address = 0x4705ffef5600000000000000120000000000000000
         };
node  E19{ parameter_block spartan,
           address = 0x4705ffef5600000000000000130000000000000000
         };
node  E20{ parameter_block spartan,
           address = 0x4705ffef5600000000000000140000000000000000
         };
node  E21{ parameter_block spartan,
           address = 0x4705ffef5600000000000000150000000000000000
         };
node  E22{ parameter_block spartan,
           address = 0x4705ffef5600000000000000160000000000000000
         };
node  E23{ parameter_block spartan,
           address = 0x4705ffef5600000000000000170000000000000000
         };
node  E24{ parameter_block spartan,
           address = 0x4705ffef5600000000000000180000000000000000
         };
node  E25{ parameter_block spartan,
           address = 0x4705ffef5600000000000000190000000000000000
         };
node  E26{ parameter_block spartan,
           address = 0x4705ffef56000000000000001a00000000000000
         };
node  E27{ parameter_block spartan,
           address = 0x4705ffef56000000000000001b00000000000000
         };
# ----------------- Hosts ---------------------- #
```

```
host Host1{ parameter_block newton,
            address = 0x4705ffef56000000000000001000ec3011000100
        };
host Host2{ parameter_block newton,
            address = 0x4705ffef56000000000000001000ec3011000200
        };
host Host3{ parameter_block newton,
            address = 0x4705ffef56000000000000001100ec3011000300
        };
host Host4{ parameter_block newton,
            address = 0x4705ffef56000000000000001100ec3011000400
        };
host Host5{ parameter_block newton,
            address = 0x4705ffef56000000000000001200ec3011000500
        };
host Host6{ parameter_block newton,
            address = 0x4705ffef56000000000000001200ec3011000600
        };
host Host7{ parameter_block newton,
            address = 0x4705ffef56000000000000001300ec3011000700
        };
host Host8{ parameter_block newton,
            address = 0x4705ffef56000000000000001300ec3011000800
        };
host Host9{ parameter_block newton,
            address = 0x4705ffef56000000000000001400ec3011000900
        };
host Host10{ parameter_block newton,
             address = 0x4705ffef56000000000000001400ec3011001000
        };
host Host11{ parameter_block newton,
             address = 0x4705ffef56000000000000001500ec3011001100
        };
host Host12{ parameter_block newton,
             address = 0x4705ffef56000000000000001500ec3011001200
        };
host Host13{ parameter_block newton,
             address = 0x4705ffef56000000000000001600ec3011001300
        };
host Host14{ parameter_block newton,
             address = 0x4705ffef56000000000000001600ec3011001400
        };
host Host15{ parameter_block newton,
             address = 0x4705ffef56000000000000001700ec3011001500
        };
```

```
host Host16{ parameter_block newton,
             address = 0x4705ffef56000000000000001700ec3011001600
        };
host Host17{ parameter_block newton,
             address = 0x4705ffef56000000000000001800ec3011001700
        };
host Host18{ parameter_block newton,
             address = 0x4705ffef56000000000000001800ec3011001800
        };
host Host19{ parameter_block newton,
             address = 0x4705ffef56000000000000001900ec3011001900
        };
host Host20{ parameter_block newton,
             address = 0x4705ffef56000000000000001900ec3011002000
        };
host Host21{ parameter_block newton,
             address = 0x4705ffef56000000000000001a00ec3011002100
        };
host Host22{ parameter_block newton,
             address = 0x4705ffef56000000000000001a00ec3011002200
        };
host Host23{ parameter_block newton,
             address = 0x4705ffef56000000000000001b00ec3011002300
        };
host Host24{ parameter_block newton,
             address = 0x4705ffef56000000000000001b00ec3011002400
        };

port  genericport { bw = OC12, delay = 4 } ;

#-------------- Connection S -> L ---------------- #
#
# delay S -> L uniform low = 10 high = 25 (24 links)

connection  S1->L6 { bw = OC3,  ad_weight = 50, delay = 14 } ;
connection  S1->L1 { bw = OC3,  ad_weight = 60, delay = 13 } ;
connection  S1->L2 { bw = OC3,  ad_weight = 70, delay = 11 } ;
connection  S1->L5 { bw = OC3,  ad_weight = 50, delay = 13 } ;
connection  S2->L1 { bw = OC3,  ad_weight = 60, delay = 20 } ;
connection  S2->L2 { bw = OC3,  ad_weight = 70, delay = 11 } ;
connection  S2->L3 { bw = OC12, ad_weight = 50, delay = 14 } ;
connection  S2->L6 { bw = OC12, ad_weight = 60, delay = 22 } ;

connection  S3->L2 { bw = OC3,  ad_weight = 70, delay = 21 } ;
connection  S3->L3 { bw = OC3,  ad_weight = 50, delay = 10 } ;
```

```
connection  S3->L4 { bw = OC12, ad_weight = 60, delay = 10 } ;
connection  S3->L1 { bw = OC12, ad_weight = 70, delay = 11 } ;
connection  S4->L3 { bw = OC3,  ad_weight = 50, delay = 22 } ;
connection  S4->L4 { bw = OC3,  ad_weight = 60, delay = 24 } ;
connection  S4->L5 { bw = OC3,  ad_weight = 70, delay = 21 } ;
connection  S4->L2 { bw = OC3,  ad_weight = 70, delay = 25 } ;

connection  S5->L4 { bw = OC3,  ad_weight = 70, delay = 17 } ;
connection  S5->L5 { bw = OC3,  ad_weight = 60, delay = 17 } ;
connection  S5->L6 { bw = OC12, ad_weight = 50, delay = 15 } ;
connection  S5->L3 { bw = OC12, ad_weight = 70, delay = 14 } ;
connection  S6->L5 { bw = OC3,  ad_weight = 60, delay = 17 } ;
connection  S6->L6 { bw = OC3,  ad_weight = 50, delay = 20 } ;
connection  S6->L1 { bw = OC12, ad_weight = 70, delay = 23 } ;
connection  S6->L4 { bw = OC12, ad_weight = 60, delay = 16 } ;

# delay L -> L uniform low = 25 high = 40  (3 links)

connection  L1->L4 { bw = OC12, ad_weight = 70, delay =  37 } ;
connection  L2->L5 { bw = OC12, ad_weight = 70, delay =  31 } ;
connection  L3->L6 { bw = OC12, ad_weight = 70, delay =  40 } ;

#delay E -> S unifrom low = 5 high = 10 (12 links)

connection  E16->S2 { bw = OC3, ad_weight = 70, delay = 9  } ;
connection  E17->S1 { bw = OC3, ad_weight = 70, delay = 6  } ;
connection  E18->S3 { bw = OC3, ad_weight = 70, delay = 7  } ;
connection  E19->S2 { bw = OC3, ad_weight = 70, delay = 6  } ;
connection  E20->S4 { bw = OC3, ad_weight = 70, delay = 10 } ;
connection  E21->S3 { bw = OC3, ad_weight = 70, delay = 9  } ;
connection  E22->S5 { bw = OC3, ad_weight = 70, delay = 9  } ;
connection  E23->S4 { bw = OC3, ad_weight = 70, delay = 9  } ;
connection  E24->S6 { bw = OC3, ad_weight = 70, delay = 9  } ;
connection  E25->S5 { bw = OC3, ad_weight = 70, delay = 6  } ;
connection  E26->S1 { bw = OC3, ad_weight = 70, delay = 6  } ;
connection  E27->S6 { bw = OC3, ad_weight = 70, delay = 7  } ;

#delay E -> L unifrom low = 5 high = 10 (12 links)

connection  E16->L6 { bw = OC12, ad_weight = 70, delay = 8  } ;
connection  E17->L2 { bw = OC12, ad_weight = 70, delay = 8  } ;
connection  E18->L1 { bw = OC12, ad_weight = 70, delay = 6  } ;
connection  E19->L3 { bw = OC12, ad_weight = 70, delay = 8  } ;
connection  E20->L2 { bw = OC12, ad_weight = 70, delay = 6  } ;
connection  E21->L4 { bw = OC12, ad_weight = 70, delay = 8  } ;
```

```
connection  E22->L3 { bw = OC12, ad_weight = 70, delay = 10 } ;
connection  E23->L5 { bw = OC12, ad_weight = 70, delay = 8  } ;
connection  E24->L4 { bw = OC12, ad_weight = 70, delay = 8  } ;
connection  E25->L6 { bw = OC12, ad_weight = 70, delay = 8  } ;
connection  E26->L5 { bw = OC12, ad_weight = 70, delay = 9  } ;
connection  E27->L1 { bw = OC12, ad_weight = 70, delay = 6  } ;

# connection Host to Edge (E)

#delay Host -> E fixed = 4  (24 links)

connection  Host1->E16  { bw = 2000 };
connection  Host2->E16  { bw = 2000 };
connection  Host3->E17  { bw = 2000 };
connection  Host4->E17  { bw = 2000 };
connection  Host5->E18  { bw = 2000 };
connection  Host6->E18  { bw = 2000 };
connection  Host7->E19  { bw = 2000 };
connection  Host8->E19  { bw = 2000 };
connection  Host9->E20  { bw = 2000 };
connection  Host10->E20 { bw = 2000 };
connection  Host11->E21 { bw = 2000 };
connection  Host12->E21 { bw = 2000 };
connection  Host13->E22 { bw = 2000 };
connection  Host14->E22 { bw = 2000 };
connection  Host15->E23 { bw = 2000 };
connection  Host16->E23 { bw = 2000 };
connection  Host17->E24 { bw = 2000 };
connection  Host18->E24 { bw = 2000 };
connection  Host19->E25 { bw = 2000 };
connection  Host20->E25 { bw = 2000 };
connection  Host21->E26 { bw = 2000 };
connection  Host22->E26 { bw = 2000 };
connection  Host23->E27 { bw = 2000 };
connection  Host24->E27 { bw = 2000 };

schedule{
duration = 6000
};
```