QoS-based Routing with Dynamic Delay Constraint Blocking Island Algorithm

Bo Cheng

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of**

**Master of Science
in
Information and System Science**

Department of Systems and Computer Engineering

**Carleton University
Ottawa, Ontario**

January 2003

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-83500-6

Our file Notre référence

ISBN: 0-612-83500-6

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

**QoS-based Routing with Dynamic Delay
Constraint Blocking Island Algorithm**

Submitted by Bo Cheng
in partial fulfillment of the requirements for
the degree of Master of Science

Prof. D. Petriu, Thesis Supervisor

Prof. R. Goubran, Chair, Dept. of Systems and Computer Engineering

Department of Systems and Computer Engineering
Carleton University

January 2003

Abstract

The thesis extends an existing QoS-based routing algorithm named Delay Constrained Blocking Island (DCBI). The extended algorithm, named Dynamic DCBI (DDCBI) takes into account dynamic aspects of the network state, such as queuing delay, in order to ensure the computation of routes that meet end-to-end delay constraints. Moreover, DDCBI checks periodically the end-to-end delay of the flows in progress. If the delay rises above the delay constraint, the path is recomputed. This is especially important for long-living demands, such as VPN applications. The performance of the DDCBI algorithm is investigated by simulation. The simulation is performed on the Network Simulator (ns) environment developed at Berkley. The analysis shows that the DDCBI algorithm has, indeed, the potential to achieve better end-to-end delay, and thus better QoS for delay-sensitive applications.

Acknowledgement

First I would like to give my thanks to Prof. Dorina Petriu, my advisor and thesis supervisor, for her guidance and kind support throughout my study in the M.S. program, especially during the research for this thesis.

I would also like to give my thanks to the Department of Systems and Computer Engineering at Carleton University for providing the program of Master of Information and Systems Science (ISS). This gave me a unique opportunity to study in this field and complete my M.S. degree.

Thanks also go to the technical support staff from the computer lab for their help in installing software and solving all kind of practical problems during my thesis work.

Acronyms

AS	Autonomous System
BGP	Border Gateway Protocol
CBR	Constant Bit Rate
DCBI	Delay-constraint Blocking Island
DDCBI	Dynamic Delay-constraint Blocking Island
DiffServ	Differential Services
IntServ	Integrated Services
IDPR	Inter-Domain Policy Routing
ISP	Internet Service Provider
L2F	Layer Two Forwarding
L2TP	Layer Two Tunneling Protocol
<i>ns</i>	Network Simulator
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PPP	Point-to-Point Protocol
PPTP	Point-to-Point Tunneling Protocol
PNNI	Private Network-Network Interface
QoS	Qualify of Service

QOSR	QoS Routing Working Group
RADIUS	Remote Authentication Dial In User Service
RFC	Request for Comments
RIP	Routing Information Protocol
RSVP	Resource reservation Protocol
SDRP	Source Demand Routing Protocol
SLA	Service Level Agreement
VHQP	Viewserver Hierarchy Query Protocol
VPN	Virtual Private Network

Table of Contents

ABSTRACT	i
ACKNOWLEDGEMENTS	ii
ACRONYMS.	iii
TABLE OF CONTENTS	v
LIST OF FIGURES	ix
LIST OF TABLES	xi
CHAPTER 1 INTRODUCTION	1
1.1 Background	1
1.2 Motivation and Objectives	1
1.3 Methodology	2
1.4 Thesis Contributions	3
1.5 Thesis Contents	4
CHAPTER 2 BACKGROUND	6
2.1 Virtual Private Network (VPN)	6
2.1.1 The concept of VPN	6
2.1.2 VPN Architecture	7
2.1.3 The Protocols Behind Internet VPNs	9

2.1.4 VPN Building Blocks.....	10
2.2 Quality of Service Based Routing (QoS-based Routing).....	12
2.2.1 Definition and Objectives of QoS-based Routing.....	12
2.2.2 Main Issues in QoS-Based Routing.....	15
2.2.3 Categories of QoS-based Routing.....	17
2.2.4 QoS-based Routing and Related Techniques	18
2.2.4.1 QoS-based routing and Traffic Engineering.....	19
2.2.4.2 QoS-based Routing and Resource Reservation.....	19
2.2.4.3 QoS-base Routing and MPLS.....	20
2.3.1 History and Status	21
2.3.2 The <i>ns</i> Architecture	21
2.3.3 Network Topology in <i>ns</i>	25
2.3.4 Routing in <i>ns</i>	26
2.3.5 Application and Transport Agents.....	27
CHAPTER 3 DYNAMIC DELAY CONSTRAINED BLOCKING ISLAND...	30
3.1 DCBI Routing.....	30
3.1.1 Network Model and the Blocking Islands Paradigm	31
3.1.2 DCBI Routing Algorithm	36
3.1.3 Description of DCBI Algorithm	38
3.2. The newly proposed DDCBI Routing Algorithm.....	40

CHAPTER 4 SIMULATION MODEL OF DDCBI ROUTING PROTOCOL....	43
4.1 Routing Function Blocks	43
4.2 Internals and Architecture of Routing in <i>ns</i>	44
4.3 Routing Module in <i>ns</i>	47
4.4 The Development of DDCBI routing strategy.....	48
4.4.1 The Functionalities of DCBI routing protocol	49
4.4.2 The Implementation of DCBI Routing Agent	50
4.4.3 Other Extension To The Simulator, Node, Link, and Classifier	55
4.4.4 The Interface to the Simulation Operator	59
4.5 Simulation Design Process	59
4.5.1 Problem Formulation and Objectives	60
4.5.2 Model Conceptualization and Translation	61
4.5.3 Model Verification and Validation	63
4.5.4 Production Runs and Analysis.....	64
4.6 Simulation model for DDCBI	64
4.6.1 Topology Modeling.....	65
4.6.2 Traffic Generation.....	66
4.6.3 Simulation Scenarios	68
CHAPTER 5 ANALYSIS OF SIMULATION RESULTS	72

5.1 Relationship between bandwidth requirements and demands allocation rate	72
5.2 Effects of delay constraints	77
5.3 The impact of queueing delay on demand allocation rate	79
5.4 Link utilization.....	81
5.5 Rerouting effect on end-to-end delay and link utilization.....	83
CHAPTER 6 SUMMARY AND CONCLUSION	92
6.1 Summary and conclusions	92
6.2 Future work	94
REFERENCES.....	95

List of Figures

Figure 2-1: Virtual Private Network	8
Figure 2-2 QoS-based routing example	13
Figure 2-3 Relations Between TE and QoS-based Routing	19
Figure 2-4 Architectural view of <i>ns</i>	22
Figure 2-5 C++ and OTcl Duality	24
Figure 2-6 Partial <i>ns</i> class hierarchy	24
Figure 3-1 A network graph $G = (N, L)$	31
Figure 3-2 The blocking islands hierarchy for bandwidth resource requirements of {56K, 28K, 14K}.....	34
Figure 3-3 The Blocking Islands Hierarchy in DCBI Routing	37
Figure 4-1 Structure of Unicast Node	46
Figure 4-2 Interaction among Node, routing module and routing	47
Figure 4-3 Logic working flow of DCBI routing agent	51
Figure 4-4 Logical working flow of other extension	56
Figure 4-5 Network Topology of AT&T IP Backbone	65
Figure 5-1 Relationship between different bandwidth requirements and the number of successful demands	76
Figure 5-2: Amount of dropped flows under different delay constraints with 100 demands.....	78
Figure 5-3: Amount of dropped flows under different delay constraints	

with 50 demands.....	78
Figure 5-4: Amount of dropped flows under different delay constraints	
with 30 demands	79
Figure 5-5 Difference between delay with and without queuing delay	
and transmission delay with 30 demands.....	80
Figure 5-6 Difference between delays with and without queuing delay	
and transmission delay with 50 demands.....	80
Figure 5-7 Difference between delays with and without queuing delay	
and transmission delay with 100 demands.....	81
Figure 5-8 Link utilization histogram for 100 demands with a delay	
constraint of 0.35 s.....	82
Figure 5-9 Average end-to-end delays under the different delay constraints....	85
Figure 5-10 Link utilization histogram for a delay constraint of 6.0s.....	88
Figure 5-11 Link utilization histogram for a delay constraint of 4.2.....	88
Figure 5-12 Link utilization histogram for a delay constraint of 3.55s.....	89
Figure 5-13 Link utilization histogram for a delay constraint of 2.8s.....	90
Figure 5-14 Link utilization histogram for a delay constraint of 2.0s.....	90

List of Tables

Table 3-1 Definition of DCBI Routing Problem [PH01]	36
Table 3-2 Pseduo-code for DCBI routing algorithm	39
Table 3-3 Function Periodic_Check of the DDCBI routing algorithm.....	42
Table 4-1 a Simplified Forwarding Table with FlowID	58
Table 5-1 Successful demands for different bandwidth requirements: results for replicated experiments.....	73
Table 5-2: The smallest sample size required for 20% ($R_0 = 6$)	75
Table 5-3 Average end-to-end delays with different delay constraints.....	84
Table 5-4 Path length Vs Delay constraint for flow 8.....	86
Table 5-5 Path length Vs Delay constraint for flow 5.....	86
Table 5-6 Path length Vs Delay constraint for flow 3.....	86

CHAPTER 1. INTRODUCTION

1.1 Background

The Internet and the Word Wide Web have expanded dramatically during the last decade. The Internet, as a global public network, provides an ideal backbone for data communication due to its low-cost and ubiquitous access. Nowadays, many companies and agencies are using VPN (Virtual Private Network) technology to build secure networks over the Internet for their private use.

VPN is formally defined as a class of services that use a shared network to emulate the characteristics of a private network. These characteristics can be expressed in terms of performance, reliability, security and quality of service. QoS-based routing is one of the approaches able to enable the provisioning of VPN services with QoS guarantees. As a routing mechanism, QoS-based routing extends the conventional routing and determines paths for flows based on some knowledge of resource availability in the network as well as the QoS requirements of flows.

QoS-based routing usually works together with resource reservation protocols. Resource reservation protocols such as RSVP provide a method for requesting and reserving network resources. QoS-based routing allows the determination of a path that can accommodate the requested QoS. Thus, generally QoS-based routing is used with some form of resource reservation / allocation mechanism.

1.2 Motivation and Objectives

With the tremendous development of Internet, applications are not limited to the traditional Internet application styles like web, email and file transfer. A new breed of applications, including conferencing and telephony, video and audio, appeared. These applications usually require high data throughput (bandwidth) and low-latency when transferring data. In addition, public and private IP networks are also being used increasingly for delivery of mission-critical information that cannot tolerate unpredictable losses. In the current best-effort service model, building routing tables has the only objective of minimizing administrative cost of each path. Thus, unpredictable delays and data loss often appear, and it is difficult to guarantee quality of service.

Current research in communication networks has recognized the significance of QoS-based routing in determining paths for requests with specific QoS requirements. A number of QoS-based routing algorithms have been proposed, but there are many unsolved research problems. The objectives of this research include the development of a new QoS-based routing algorithm that can be used for VPN services, and the study of its performance by simulation.

1.3 Methodology

In this research, the simulation model and experiments are based on the *ns* environment. The following steps describe the methodology employed for the research:

- Study existing QoS-based routing algorithms and propose extensions that will improve them.

- Design and develop a *ns*-based simulation model for the proposed QoS-based routing algorithm. The development of the model and the running of the simulation experiments contains the following sub-steps:
 - Design and implement an *ns* QoS-based routing agent to implement the proposed routing algorithm.
 - Construct proper network topology for the simulation experiments. A realistic network topology, similar to the AT&T IP Backbone network topology, is used in the thesis.
 - Generate traffic with well-chosen characteristics.
 - Generate simulation scenarios that describe the different simulation experiments.
- Conduct performance analysis, investigate the system behavior in different situations, and identify the factors in the protocol that affect the overall performance.

1.4 Thesis Contributions

The contributions of the thesis can be classified in two categories: contributions to knowledge and practical contributions.

A. Contributions to knowledge

- Extend an existing QoS-based routing algorithm named Delay Constrained Blocking Island (DCBI) proposed by Pun and Hamdi [PH01], which in turn extends the Blocking Island concept proposed by Frei and Faltings [Frei00,

FF00]. DCBI applies delay constraints to static delay metrics, which do not reflect the network state changes. Our extended algorithm, named Dynamic DCBI (DDCBI) takes into account dynamic aspects of the network state, such as queuing delay, in order to ensure the computation of routes that meet end-to-end delay constraints. Moreover, DDCBI checks periodically the end-to-end delay of the flows in progress. If the delay rises above the delay constraint, the path is recomputed. This is especially important for long-living demands to guarantee the QoS.

- Performance study of DDCBI routing by simulation: understand what factors have an impact on the routing results and the QoS of the delivered data. The analysis shows that the DDCBI algorithm has, indeed, the potential to achieve better end-to-end delay, and thus better QoS for delay-sensitive applications.

B. Practical Contributions:

- Design and implement an *ns*-based simulation model for the DDCBI routing algorithm. As *ns* is a reusable simulation framework, some of the modules that were developed can be reused by other researchers from the *ns* community who want to simulate other QoS-based routing algorithms.

1.5 Thesis Contents

The thesis comprises six chapters. These chapters are organized as follows:

Chapter 1 gives the introduction of the thesis. It describes the background, motivation and objectives, methodology, main contributions and the structure of the thesis.

Chapter 2 reviews briefly the background literature related to the thesis research in the following areas: Virtual Private Networks, QoS-based routing and the Network Simulator (*ns*).

Chapter 3 describes the original Blocking Island algorithm introduced in [Frei00, FF00], the extended Delay Constrained Blocking Island (DCBI) proposed in [PH01] and our own extension DDCBI proposed in the thesis.

Chapter 4 describes the detailed design and implementation of the DDCBI simulation model and how to integrate the DDCBI routing into the *ns* network simulator.

Chapter 5 describes the simulation experiments we conducted to study the factors that have impact on the DDCBI routing performance, and presents the analysis results.

Chapter 6: Summarizes our findings and identifies directions for the future work.

Chapter 2. Background

This chapter reviews briefly the background literature related to the thesis research in the following areas: Virtual Private Networks, QoS-based routing and the Network Simulator (*ns*).

2.1 Virtual Private Network (VPN)

2.1.1 The Concept of VPN

A Virtual Private Network is a network of virtual circuits for carrying private traffic, possibly over a public network [Kos98]. A virtual circuit is a connection set up on a network between a sender and a receiver in which both the route for the session and the necessary bandwidth are allocated dynamically [Kos98].

There are many different technologies for implementing virtual private networks. Examples of more traditional VPN approaches are dedicated leased lines and frame relays Private Virtual Circuits. In this thesis, the research and discussion focus only on newer Internet-based VPN technologies, which are using the open distributed infrastructure of the Internet to deliver data between remote sites.

Why use an Internet VPN? Comparing with the traditional VPNs, a number of benefits arise from the use of Internet-based VPN, as follows:

- Cost saving. There is an initial setup cost and an implementation cost for building a VPN. Kosiur discusses in [VPN98] different network scenarios to show how significant cost saving can be.
- Flexibility. VPNs enable flexible communication with remote branches, regional offices, customers and business partners, as long as these are already connected to the Internet [VPN98].
- Scalability. VPNs are able to offer scalability in two dimensions: geographic scalability and bandwidth scalability [Pha00]. For example, if a corporate user needs to grow the bandwidth requirements between two of its sites, Internet Service Providers (ISPs) can quickly provide the user's choice of bandwidths, since the VPN links are not hard wired between the sites.
- Reduced Technical Support. The service providers (ISPs) may take over many support tasks on behalf of the users [Kos98].
- Ubiquitous access. VPNs provide access from anywhere the Internet reaches. Nowadays, Internet points of presence are available worldwide, providing VPN connection points in nearly every country and in most of the major cities in the world [VPN98].

In conclusion, IP-based VPN services are a promising solution for customers that want to replace expensive leased lines and data services with IP networking [Xchange98].

2.1.2 VPN Architecture

VPNs consist of two fundamental parts: a) tunneling mechanisms for building the “virtual” circuits in VPN, and b) security services to ensure the “private” aspects of the communication. A typical VPN architecture is illustrated in Figure 2-1.

Figure 2-1: Virtual Private Network (from [Webserver])

Tunnels: Tunneling is the encapsulation of point-to-point transmission inside IP packets [VPN98]. Using tunneling technologies, packets are transmitted across the public Internet or other commercially available networks within a virtual pipe, which makes the routed network transparent to users [Kos98]. A tunnel is set up dynamically when necessary between two endpoints, unlike the leased lines in traditional VPN which are static. When the connection is no longer needed, the tunnel is torn down, releasing the network resources. A tunnel's endpoints can be either an individual computer, or a LAN with a security gateway, which might be a router or firewall [Kos98].

Security services: Since VPNs use shared open networks to transmit data and associated user information, security is becoming a very important issue in VPN. In order to ensure the security of private data, VPN needs to offer four critical functions: authentication, access control, confidentiality and data integrity.

Authentication: is used to make sure that the two ends involved in the communication are the true parties who want to communicate to each other [SWE99]. PPP Challenge Handshake Authentication Protocol (CHAP) is a common protocol used in the user authentication [McD00].

Access Control: is used to control who has the right to access the VPN, and the network applications. In the present market, there are many products dealing with this issue like Cisco's secure access control server.

Confidentiality: is the ability to ensure that all transmitted data over a link is not read or intercepted by unauthorized individuals.

Data integrity: refers to the ability to verify that all data transmitted and received has not been tampered with or changed.

2.1.3 The Protocols Behind Internet VPNs

Presently there are several protocols making VPNs secure and trusted on the Internet. The most popular three are briefly discussed below.

a) **IPSec** (IP Security protocol). IPSec is a set of extensions to the IP protocol family [OpenBSD]. IPSec provides security services at the IP layer by enabling a system

to select required security protocols, to determine the algorithm(s) for the service(s), and to put in place any cryptographic keys required [KA98]. Security services include authentication, integrity, access control, and confidentiality. Compared with other VPN protocols, IPsec provides the most complete security framework for VPNs, and other protocols are leaning towards using part of IPsec for their security services.

b) **PPTP** (Point-to-Point Tunneling Protocol). The basic idea behind PPTP was to split up the functions of remote access in such a way that individuals and corporations could take advantage of the Internet's infrastructure to provide secure connectivity between remote clients and private networks [Kos98]. PPTP encapsulates Point-to-Point Protocol (PPP) frames into IP datagrams for transmission over an IP-based Internet work [MicroTechinfo]. PPTP is commonly used for dial-up access to the Internet.

c) **L2TP** (Layer 2 Tunneling Protocol). L2TP is one of the key VPN protocols that were developed to carry Point-to-Point Protocol (PPP) traffic over non-point-to-point networks. The L2TP protocol was created from the combination of Microsoft's Point-to-Point Tunneling Protocol (PPTP) and Cisco's Layer-2 Forwarding protocol (L2F) [VLK98]. Today, L2TP has become a standard for Layer 2 tunneling. L2TP provides virtual remote access by tunneling PPP applications. In L2TP, PPP frames are encapsulated in IP packets.

2.1.4 VPN Building Blocks

“Building a VPN can actually be very simple, given the right tools and a basic understanding of how they work together” [MicrobCen]. Usually a VPN is composed of

the following main components: firewalls and routers, VPN software and VPN hardware, ISPs (Internet Service Provider) connection.

a) Firewall and router. The role of firewalls and routers is to inspect and control all the traffics between the local network and the Internet. When firewalls or routers detect potentially "dangerous" traffics coming in or going out from the local network, the "dangerous" traffics will be dropped, which makes the local network achieve security. There are three main classes of firewalls: packet filters, application and circuit gateway, and stateful inspection firewalls [Kos98]. The most common one is perhaps the packet filtration, which blocks specific users (based on userID or hostID) or specific IP services (based on IP address, sub-network address or specific port number). Restrictions can also be based on the type of access (email, Telnet, FTP, etc.), contents of the data accessed, and so on [SWE99].

b) VPN hardware and VPN software: there are many kinds of VPN hardware products in the market. The functions of integrated VPN hardware include packet authentication, tunneling, encryption, and user authentication, filtering as well as key management, in which a single box includes all of the required VPN functionality. VPN hardware can be categorized as LAN-to-LAN VPNs products and remote VPNs products. Comparing with the software counterparts, VPN hardware products usually can offer better throughput. VPN software has similar functions as VPN hardware, only that they offer lower performance. Because it's low cost and easy to collaborate with the existing operating system, if the performance isn't an issue, VPN software products are a good choice to build a small VPN.

c) **ISP connection.** According to [Kos98], the success of a VPN depends greatly on one element – Internet Service Provider (ISP). ISPs are responsible for the transmission of data over the Internet after the data leaves the local network. The customers who want to construct VPN need to sign a Service Level Agreement (SLA) with the ISP in order to set the expectations for the network performance and troubleshooting services. ISPs are usually classified according to the capacities of their networks and the type of Internet connectivity that they provide [Pri98].

2.2 Quality Of Service Based Routing (QoS-based Routing)

QoS is regarded as a key element of any VPN service [ZK00], and QoS-based routing is regarded as a solution for offering VPN services with QoS guarantees [CK98]. QoS-based routing not only improves the efficient utilization of network resources, but also provides flexibility in support for various VPN services.

2.2.1 Definition and Objectives of QoS-based Routing

QoS Routing Working Group (QOSR) of IETF defined a framework of QoS-based routing in the Internet, which guides the research on QoS-based routing techniques. As defined in [RFC2386], a QoS-based routing mechanism is one "under which paths for flows are determined based on some knowledge of resource availability in the network as well as the QoS requirement of the flows". Also, a QoS-based routing protocol is a "dynamic routing protocol that has expanded its path-selection criteria to include QoS parameters such as available bandwidth link and end-to-end path utilization, node resources consumption, delay and latency, and included jitter" [QOSF2]. Another similar concept, policy-based routing, applies when the path selection is based on some

administrative policies, instead of information on network topology and metrics. The combination of QoS-based and policy-based routing is called constraint-based routing.

An example is given in Figure 2-2 to explain how the QoS-based routing works. Suppose Router A wants to send messages to Router C with a requirement of at least 4bs. There are 3 possible paths from Router A to Router C. Path #1 (A-B-C) is the shortest path, but it won't be chosen because its links cannot ensure the required bandwidth. Path #3 (A-D-F-G-E-C) is not only the longest, but cannot ensure the required bandwidth either. The choice is path#2 (A-D-E-C): even though longer than path #1, it has enough bandwidth to satisfy the constraint.

Figure 2-2 QoS-based routing example (From [Sun00])

It is known that some applications, such as multimedia and Internet conferencing, are very strict with the QoS requirements. However, some frequently used Internet protocols such as OSPF, RIP, and BGP cannot ensure any QoS. The main reasons is that

these are "best effort" routing protocols, that try to find a path to reach the destination, but they don't guarantee that the path found can meet the QoS requirements of the applications. Another reason is that the routing tables are built with the single objective of minimizing administrative metrics (hop count, cost) of each path. The traffic is routed to the "shortest path". The drawback of this approach is that some links may become congested, while others are not fully utilized [Sun00]. An undesirable consequence is that some traffic may be discarded due to the congestion. Also, best-effort routing is designed to keep looking for a "better" path, even though the current used path does meet the service requirements of traffic. This mechanism will result in undesirable traffic shift back and forth between the current path and an alternate path if any metrics changes.

QoS-based routing service models are supposed to overcome and solve the drawbacks mentioned above. The main objectives of QoS-based routing as follows:

- Dynamic determination of feasible paths [RFC2386]. Path selection is subject to the user's requirement on bandwidth, end-to-end delay, etc., instead of sticking to the "shortest" path. Different requirements may result in the different paths selected, even though they have the same source and destination.
- Optimization of resource usage [RFC2386]. "The improvement to network efficiency is usually in terms of increase in revenue, where revenue is typically a function of the number of flows or the amount of bandwidth carried by the network" [AST98].

- Graceful performance degradation [RFC2386]. When the network is congested, QoS-based routing is expected to give better performance than best-effort routing.

2.2.2 Main Issues in QoS-Based Routing

i) How often to propagate topology information and how to maintain it.

Similar to the best-effort routing, routers employing QoS-based routing need to exchange periodically routing information. However, the quantity of QoS information (such as available bandwidth, delay, jitter) is larger than in the case of best-effort routing. Moreover, these metrics could change very fast in the network. If routing information is exchanged every time the values of the QoS metrics change, it will cause a great burden for the network links and routers, consuming network bandwidth and CPU cycles in the routers [Wei98]. A careful tradeoff must be made between the precision of the information and the network performance. A common way to deal with this problem is to use a threshold algorithm to trigger a new advertisement whenever the change in available resources exceeds a certain percentage of the previously advertised value. Another alternative is to use hold-down timers to enforce a minimum time interval between consecutive updates.

A related problem is how to maintain the collected information. A method is to keep the routing table for best-effort traffic, and to compute the paths for QoS flows on demand. Another method is to aggregate the flows and maintain the information about aggregated flows, instead of storing information about individual flows.

ii) **Metrics collection and path computation.** As mentioned in [RFC2386], "the metrics must represent the basic network properties of interest". Metrics like delay, delay jitter, loss ratio, bandwidth and so on, are commonly used by the current applications, such as Internet telephony or distributed games, to define the types of QoS of interest. For QoS-based routing, the path computation is usually based on one or a combination of metrics, under multiple constraints. Zheng Wang and Jon Crowcraft have proven in [WC96] that path computation based on certain combinations of metrics is NP-complete. In the recent years, a lot of research effort went into proposing new QoS-based routing algorithms to deal with this problem. Basically, the QoS-based routing algorithms are classified into three categories: a) distributed routing (hop-by-hop routing) algorithms, b) source routing algorithms, and c) hierarchical routing algorithms. (More on the classification of QoS-based routing algorithms will be discussed in the next subsection.) In general, any QoS-based routing algorithm must meet the following requirements: 1) efficient and scalable to large networks, 2) complexity not far greater than the currently used routing algorithms, and 3) suitable to the current Internet architecture [Wei98].

iii) **Scalability to large networks and dealing with imprecise information.** As the size of the Internet keeps growing, the QoS-based routing algorithms are expected to be scalable. They should be able to control and manage an increasing quantity of data, and to deal with the complexity of path computation. Source routing algorithms, for example, are not very scalable. Hierarchical aggregation is a promising way to solve the scalability problem (as in PNNI and OSPF) but this method suffers from information inaccuracy, and may lead incorrect routing decision. It is a trade-off between the

information precision and low cost. QoS-based routing has an imprecise nature in terms of network dynamic, aggregation of routing information, and approximate calculation.

iv) **Integrate QoS-based and best-effort routing.** The networks using QoS-based routing should also support best-effort routing. Usually QoS-based routing has a higher priority to allocate the network resources. The problem is how to distribute fairly the network resources between the two kinds of routing. A control mechanism is needed, (for example, an administrative control) that will allow these two routing approaches to share the network resources in a fair way.

2.2.3 Categories of QoS-based Routing

The QoS-based routing algorithms can be classified according to different criteria.

a) **Inter-domain and intra-domain QoS-based routing** [RFC2386]. Today's Internet consists of domains, also called Autonomous Systems (ASs). Intra-domain QoS-based routing schemes are used to route packets within an AS, while the inter-domain routing scheme are used to route packets between ASs. Inter-Domain Policy Routing (IDPR) protocol, Viewserver Hierarchy Query Protocol (VHQP) and Source Demand Routing Protocol (SDRP) belong to inter-domain QoS-based routing, which is expected to be as simple as possible. According to [Wei98], stability and scalability are the most important issues at the inter-domain level. In contrast, intra-domain QoS-based routing is more complex, since it is based on highly dynamic network state information. Most proposed QoS-based routing algorithms are at the inter-domain level.

b) **QoS-based unicast and multicast routing.** The objective of QoS-based unicast routing is to find a path that can satisfy some specific QoS requirements for one-

to-one communication. An example of QoS-based unicast routing is PNNI (Private Network-Network Interface), which is a hierarchical, dynamic, QoS supporting routing protocol for ATM networks. Today's multicast applications, such as video conferencing, video-on-demand and webcasting are used over the Internet. These applications have relatively strict requirements on bandwidth, delay, delay jitter, etc., and often ask for QoS-based multicast routing. The task of QoS-based multicast routing is to find the minimum cost tree with QoS constraints added for multicast flows. Current research on QoS-based multicast routing mainly focuses on bandwidth-constrained multicast, delay-constrained least-cost multicast, and delay and delay-jitter constrained multicast routing.

c) Distributed routing, Source routing and Hierarchical routing [Chen99].

The classification is based on the way to maintain the state information and the way to search for feasible paths. In distributed routing, each router knows only the next hop towards the destination. When a packet comes, it gets forwarded to the next hop, and it gets to the destination step by step. In source routing, every router maintains global state information about the network, which is used for path selection. Once the path is determined, the source router informs the other routers along the path how to forward the traffic flow. Hierarchical routing is most suitable for large network. Its routing structure consists of multiple levels.

2.2.4 QoS-based Routing and Related Techniques

QoS-based routing is a main component to guarantee QoS in the VPN. Besides it, there are other techniques, such as traffic engineering, resource reservation that can help VPN gain QoS.

2.2.4.1 QoS-based routing and Traffic Engineering

“Traffic Engineering (TE) means the ability of the network to dynamically control traffic flows, to optimize the availability of resources, to choose routes for traffic flows while taking into account traffic loads and network state, and to move traffic flows toward less congested paths” [SZ02]. The purpose of traffic engineering is to avoid uneven network utilization. QoS-based routing is an important part of traffic engineering, which can help make the traffic engineering process automatic.

Figure 2-3 Relations Between TE and QoS-based Routing

2.2.4.2 QoS-based Routing and Resource Reservation

Resource Reservation Protocol has been designed to provide end-to-end quality of service to Internet data flows. In order to make a resource reservation at a node, the RSVP daemon communicates with admission control and policy control to check whether the node has sufficient available resources to supply the requested QoS and whether the user has administrative permission to make the reservation. Only if both checks succeed, then RSVP can be applied. RSVP is a signaling and control protocol that doesn't carry application data. It operates on top of IP in the transport layer of the Open Systems

Interconnection (OSI) protocol stack. QoS-based routing is to find a feasible path that meets the QoS requirements. RSVP is to reserve the resources along the path, and then the reservation message will affect the next routing decision of QoS-based routing. "Consequently, QoS-based routing is usually used in conjunction with some form of resource reservation or resource allocation mechanism... Combining a resource reservation protocol with QoS-based routing allows fine control over the route and resources at the cost of additional state and setup time. For example, a protocol such as RSVP may be used to trigger QoS-based routing calculations to meet the needs of a specific flow." [RFC2386]

2.2.4.3 QoS-base Routing and MPLS

MPLS (Multiprotocol Label Switching) is a more recent industry-standard approach developed by the Internet Engineering Task Force (IETF) for reducing the complexity of forwarding in IP network [BAN01]. "MPLS integrates a label swapping framework with the network layer routing. The basic idea involves assigning short fixed length labels to packets at the ingress to an MPLS cloud (based on the concept of forwarding equivalence classes)" [AMAO99]. Inside of an MPLS domain, the labels attached to packets are used to make forwarding decisions. There are two characteristics of MPLS that are needed by QoS explicit (source) routing. First of all, MPLS allows decoupling of the information used for forwarding from the information carried in the IP header. Second, the decision as to which IP packets will take a particular explicit route is completely confined to the LSR (Label Switching Router) that computes the route. MPLS can be used as the forwarding mechanism for QoS explicit routing.

In conclusion, QoS-based routing is a very important enhancing mechanism for deploying different quality classes into the VPN networks. Combined with other techniques, QoS-based routing helps to implement QoS-VPN, which “provides guaranteed service levels to VPN trunks across the backbone while maintaining high network performance” [West99].

2.3 Network Simulator (*ns*)

The simulation experiments in the thesis were conducted with *ns* (Network Simulator) [FV00]. In this section, the main principles and the structure of *ns* will be briefly described. The description will focus on the *ns* components related to the thesis research.

2.3.1 History and Status

The Network Simulator *ns* is an object-oriented discrete event simulator targeted at networking research. It began as a variant of the REAL network simulator in 1989, which was originally intended for studying the dynamic behavior of flow and congestion control schemes in packet-switched data networks. Since then, *ns* has evolved substantially. Researchers from UC Berkeley, LBL, USC/ISI, and Xerox PARC contributed to its development. Besides these researchers, “*ns* got substantial contributions from other researchers all over the world. At present, *ns* is still not a finished product, but the result of an on-going effort of research and development” [FV00].

2.3.2 The *ns* Architecture

In a simplified user's view, the *ns* architecture consists of three parts: a simulation event scheduler, network component object libraries, and network setup (plumbing) module

libraries. The event scheduler is used to keep track of the simulation time and fire all the events in the event queue. It schedules events for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with the event. In *ns*, an event refers to a packet ID with scheduled time and the pointer to an object that handles the event. The network component object libraries include nodes, links, routing protocols, transportation protocols and some network applications as well as network topologies, etc. The users can choose one or several of these components to build the network environment they want. The network setup module libraries are used to setup and run a simulation by writing an OTcl script.

Figure 2-4 Architectural view of *ns*

Figure 2-4 shows the general architecture of *ns*. A user who wants to design and run simulations will use a Tcl script to write a simulation file, and will make use of the simulator objects in the OTcl library. Tccl, the middle part in Fig. 2-4, is needed as the linkage between OTcl and C++. *ns* supports two class hierarchies: the interpreted hierarchy (OTcl) and the compiled hierarchy (C++). The event scheduler and the network components implemented in C++ make use of the compiled hierarchy.

The *ns* software is constructed in a way intended to promote its extension by users. *ns* is written in Otcl and C++, since “different simulation functions require different programming models to provide adequate flexibility without unduly constraining performance” [FV00]. Tasks like the event scheduler or packet forwarding are written in C++, because they require high performance and are modified infrequently once put into place. On the other hand, tasks like the dynamic configuration of protocol objects and the specification and placement of traffic sources are often iteratively refined and undergo frequent change. Thus, they are implemented in a flexible and interactive scripting language like OTcl. The compiled objects are made available to the OTcl interpreter through an OTcl linkage. For establishing the linkage between an instance variable on one side and the corresponding peer on the other side, the user can create a bi-directional binding. Thus, both the compiled instance variable and interpreted instance variable will access the same data. The binding is created in the constructor of a C++ class as follows:

```
LinkDelay::LinkDelay() {
    bind ("transDelay_", &transDelay);
}
```

In tclcl, a class named Tcl is defined. It encapsulates the actual instance of the OTcl interpreter, and provides methods to access the interpreter. Through some methods defined in this class, such as `tcl.eval()` and `tcl.result()`, users can also call an interpreted procedure (either in the Tcl script or in OTcl classes) or access an interpreted variable from the compiled side. Correspondingly, compiled classes use a specific method named `command()` to provide a gate to Tcl/OTcl interpreter. Thus, the users can accomplish the

communication between compiled classes and interpreted classes. Figure 2-5 shows the relationship between C++ and OTcl objects.

Figure 2-5 C++ and OTcl Duality

The *ns* class hierarchy starts from the root `TclObject`, which is the base class for most of the other classes in the interpreted and compiled hierarchy. The C++ class hierarchy is illustrated in Figure 2-6.

Figure 2-6 Partial *ns* class hierarchy

In this hierarchy, class `Simulator` provides the general interface to users. From this interface, the user can configure and implement a simulation. The OTcl class named

Simulator is defined in directory `~ns/tcl/lib/`. Each simulation requires a single instance of the class `Simulator` to control and operate the simulation. Through a `Simulator` instance, the user can create a scheduler, change a scheduler, schedule events, and trigger the scheduler to start the simulation and so on. Some examples for the uses of this interface are the following:

```

set ns [new Simulator]      # create a scheduler

$ns use-scheduler Heap      # change the default scheduler
                             # into Heap scheduler

$ns at <time> <event>       # schedule a event

$ns run                     # start scheduler

```

2.3.3 Network Topology in *ns*

In order to carry out a meaningful study of different networking problems like protocol interaction, effect of network dynamics, scalability etc, it is very important to build the right kind of network topology. *ns* supports both pre-defined and automatically generated network topologies. Pre-defined topologies may be created manually, that is users write the code to create nodes and links, connectivity of links, bandwidth and delay, and so on. Another alternative is to use a topology generator package called GT-ITM (Georgia Tech Internet work Topology Models), which can generate automatically flat random networks using a variety of edge distribution models, as well as hierarchical and transit-stub networks. The random topologies are generated according to a set of specified parameters, such as number of levels in the hierarchy, degree of connectivity, and other features.

2.3.4 Routing in *ns*

From the routing strategy point of view, *ns* may perform static, session, and dynamic routing computations. The static routing strategy uses Dijkstra's all-pairs shortest-path-first (SPF) algorithm to compute the routes with an adjacency matrix and link costs of all the links in the topology. It is considered the default route computation mechanism in a user level simulation script, if no specific routing strategy is required.

Session routing is almost identical to static routing, in that it runs Dijkstra's all-pairs SPF algorithm prior to the start of the simulation. The difference is that session routing will rerun the same algorithm to recompute routes if the topology changes while the simulation is in progress. Session routing provides complete and instantaneous routing changes in the presence of topology dynamics.

Dynamic routing has two modes, *rtProtoDV* and *rtProtoLS*. The first implements the distributed Bellman-Ford (i.e. Distance Vector) algorithm. The implementation sends update messages periodically; at the same time, each agent sends triggered update when the topology changes or an agent receives route updates. Each agent also advertises its routes to adjacent peers. The second mode, *rtProtoLS* implements a Link State routing algorithm. In terms of the actions it performs, *rtProtoLS* is designed to be a simplified OSPF-like protocol. Functionally, *rtProtoLS* does what *rtProtoDV* does - dynamically calculate routes in a distributed fashion. They both support only flat topologies and point-to-point links. However, *rtProtoLS* offer the advantages of being more reliable due to acknowledgement, and faster to converge.

No matter what routing protocol is used, the routing implementation in *ns* consists of three functional blocks: routing agent, route logic and classifier. The routing agent is responsible for exchanging routing information with its neighbors. The route logic performs the route computation by using the information gathered by routing agents. The classifiers, which sit inside a Node, use the computed routing table to perform packet forwarding.

In addition, *ns* may perform unicast routing or multicast routing. They are implemented through different types of nodes. A node for unicast routing contains only one address classifier and a port classifier, while a node for multicast routing must be extended with extra components. These components include a multicast classifier and several replicators. The multicast classifier is used to classify packets according to both source and destination group addresses. It maintains a table mapping source/group pairs to slot numbers. The replicator is responsible for producing additional copies of the packet, one for each entry in the group. Thus, a copy of packet can be delivered to each of the nodes listed in the classifier's table.

2.3.5 Application and Transport Agents

An application agent includes two parts, a traffic generator and simulated applications. An application agent in *ns* is defined as an abstract C++ class. It provides basic application behaviors, such as `send()`, `recv()`, `resume()`, etc. Currently two simulated applications are derived from Application: Application/FTP and Application/Telnet. An FTP object, implemented in *Otcl*, produces the bulk data for the TCP object to send; a Telnet objects produces individual packets with inter-arrival times.

The traffic generator is used to generate traffic of the following types:

- Exponential -generates traffic according to an exponential on/off distribution,
- Pareto – generates traffic according to a Pareto on/off distribution,
- CBR traffic - generates traffic according to a deterministic rate.

Although these kinds of traffic generators are implemented in C++, when simulating them, we should use Otcl to create them first, and then assign their parameters. Here is an example:

```
set e [new Application/Traffic/Exponential]
$e set packetSize_ 210
$e set burst_time_ 500ms
$e set idle_time_ 500ms
$e set rate_ 100k
```

The main transport agents defined in *ns* are reliable delivery transport agents (e.g. TCP and SRM) and unreliable transports (e.g. UDP and RTP). They are subclasses of class Agent, and inherit the packet generation and reception from Agent. A UDP agent accepts data in variable size chunks from an application and segments the data if needed. A TCP agent can be a one-way agent or two-way agent. A TCP sender obeys different congestion and error control techniques. The transport agents should be created in pairs (one is the sender, and the other is the receiver). They must be attached to a pair of nodes respectively: the sender to the source node, and the receiver to the destination node.

These two agents are then connected to each other for delivering packets. Applications sit on top of transport agents and are attached to transport agents.

This section gives background information on *ns*. Chapter 4 will describe the process of building a *ns* simulation model, and the implementation of the model for the DDCBI routing protocol.

Chapter 3. Dynamic Delay Constrained Blocking Island Routing Protocol

This chapter describes first the Delay Constrained Blocking Island (DCBI) routing protocol proposed by Pun and Hamdi [PH01], which extends the Blocking Island concept proposed by Frei and Faltings [Frei00, FF00]. Then we will introduce our own extension to DCBI, named the Dynamic Delay Constrained Blocking Island (DDCBI) algorithm.

3.1 DCBI Routing

The current QoS-routing algorithms found in the literature focus on variations of the shortest path method, with different metric combinations for the link length and different ways to explore the network for path computation. Although the shortest path routing method can ensure the best possible route for every single demand, Wang and Crowcroft have proved that it can lead to highly congested links and uneven link utilization [WC96]. Therefore, researchers started looking for new methods for searching an optimal path to meet the QoS requirements. Frei and Faltings [Frei00, FF00] introduced a new concept, named Blocking Island abstraction for efficient control and management of Virtual Private Networks. Based on their work, Pun and Hamdi [PH01] developed the Delay Constrained Blocking Island (DCBI) routing algorithm that is more efficient for the computation of routes with a required QoS. The research of this thesis concentrates on the BI and DCBI routing algorithms, and further extends them. More exactly, we propose an extension to the DCBI routing algorithm. While DCBI applies

delay constraints to static delay metrics, we propose to apply the constraints to dynamic delay metrics. Our extended algorithm named DDCBI is a true QoS-based routing algorithm, and will be described later in the chapter.

3.1.1 Network Model and the Blocking Islands Paradigm

A communication network consists of a collection of nodes (e.g. routers, switches, computers, etc.) and transmission links (e.g. copper wires, optical fibers, etc.). Each link connects two nodes and is commonly characterized by its bandwidth capacity, and other properties such as delay, loss probability, cost, and security level. The network is modeled as a non-directed network graph $G = (N, L)$ without loops as depicted in Fig.3-1

Figure 3-1 A network graph $G = (N, L)$ [PH01]

- $N_i, i=1,n$ are nodes that represent routers or processing units.
- $I_k, k=1,L$ are links that correspond to bi-directional communication medias.

The links are characterized here by the link bandwidth measured in [bits/second], and the delay measured in [ms].

Note that the delay metric considered in DCBI refers to the propagation delay.

In a virtual private network, applications such as Internet telephony and video-conferencing often put forward communication requests with specified QoS requirements. These communication requests are called demands, and the QoS requirements are typically expressed in terms of minimal bandwidth, maximal delay, maximal delay jitter, maximal loss probability and cost. Among these QoS parameters, any combination of two or more metrics regarding the delay, delay jitter, cost, or loss probability, was proven to lead to an NP-complete path computation algorithm [WC98]. The only feasible combination that is not NP-complete is a constraint on bandwidth and on one of the other four metrics (delay, delay jitter, cost and loss probability). Bandwidth and delay are chosen as the metrics of interest for the DCBI routing algorithm. On one hand, bandwidth is the primary QoS parameters for most applications; on the other hand, many networking applications are, indeed, delay-sensitive.

A demand is defined as a 4-tuple: $d_k = (x_k, y_k, \beta_k, D_k)$

- x_k and y_k are the demand's endpoints, the source and the destination of the traffic flow, respectively.
- β_k is the amount of bandwidth needed for the demand. It means that each link along the route should have at least β_k bandwidth to meet the bandwidth constraint.
- D_k is the delay constraint of the demand. That is, for any route path, the total propagation delays of all the links in the path should not exceed D_k .

- k is the number of demands.

The *Blocking Island* (BI) abstraction is a clustering scheme that clusters a part of the network into a single node (a BI) to represent the available bandwidth (called also residual bandwidth) of a communication network. Inside each BI, any routing demand requiring a certain amount of bandwidth can be accommodated. By definition, a β -*blocking island* (β -BI) for a node x is the set of all nodes of the network that can be reached from x using links with at least β available resources, including x . Figure 3-2(a) shows all 56K-BIs for an example network. The β -BI for a given node x of a network graph can be obtained by a simple greedy algorithm. Starting with an initial set $S = \{x\}$, every node that can be reached by a link adjacent to a node from S which has at least β available bandwidth is recursively added to S . S is the β -BI sought when we cannot add any more nodes into S . The detailed description and implementation of building BIs will be given in section 4.4.2.

The blocking islands are used to build β -*blocking island graph* (β -BIG). A β -BIG is an abstract view of the available resources: each β -BI can be clustered into a single node, and the links between two BIs can be clustered into an abstract link. The available resources of an abstract link are equal to the maximum available resources (for example, bandwidth) of the links it clusters. Fig. 3-2(b) is an example of a 56K-BIG. These abstract links denote the critical links, since their available resources do not suffice to support a demand requiring β resources.

Another important definition is Blocking Island Hierarchy (BIH). It is built by a recursive decomposition of BIGs in decreasing order of the bandwidth requirements. It is a layered structure that can be used to identify bottlenecks for different possible bandwidths.

Figure 3-2 The blocking islands hierarchy for bandwidth resource requirements of {56K, 28K, 14K}

The computation of a route for a demand follows the so-called rule of the Lowest Level (LL). LL states that a path uses only links that are clustered at the lowest possible BIH level and uses critical links as little as possible. Here lower level means larger β .

This is because the lower a BI is in the BIH, the less the critical links are inside the BI, and hence a better bandwidth connectivity preservation effect can be achieved. Let us consider a demand $d = (a, e, 12K)$ as an example. Following the BI abstraction and the LL rule, the path $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ is selected. It only uses the links that are clustered at the lowest level in the BIH and didn't use any critical links. If we would use the classical shortest path method, the route $a \rightarrow f \rightarrow e$ would be selected. From the previous discussion, we know that this path uses two critical links (a, f) and (f, e) which joins the 56K-BIs N_1 and N_2 in the 56K-BIG from Figure 3-2(b). Once allocated, no other demands with bandwidth requirements equal to or greater than 3K can be allocated inside N_1 and N_2 . So the LL rule can reduce the risk of future allocation failures. Bandwidth connectivity can therefore be viewed as a kind of overall load balancing.

However, the BI technique has several possible problems. Firstly, BI is originally designed for abstracting bandwidth available in the network. It is not designed to deal with multiple QoS requirements. Although a Connectivity Cluster method is developed to take multiple metrics into account, these metrics are limited to the concave metrics. The concave metric refers that the path metric is the minimum value of its passed link's metric. Some QoS requirements like delay, loss probability, are not concave. Secondly, the BI technique is applied in an off-line manner. The demands are known in advance and allocated one after the other centrally. But, in real situations, on-line (a.k.a. on demand) routing is more realistic and flexible; usually the demands cannot be predicted in advance. Lastly, the possible bandwidth requirements of the demands must be known ahead of time for building a BIH according to the BI technique. Building BIH always starts from its lowest level BI. However, bandwidth requirements are made by different

applications in a large variety of values. This can lead to the numerous BI Graphs in the BIH and make the BIH more time-consuming to maintain.

3.1.2 DCBI Routing Algorithm

Given the problems of the BI method, Pun and Hamdi improved it and proposed a new routing algorithm, called Delay Constrained Blocking Island (DCBI). DCBI allocates network resources on-line to demands with QoS requirements coming one at a time [PH01]. The problem is formally defined as follows:

Given a network composed of nodes and bi-directional links, which has a given bandwidth capacity and delay, and a set of communication demands, which is defined by a source and destination node, and QoS requirements in terms of bandwidth requirement and maximal delay.

Find one and only one route for each demand so that the QoS requirements of demands are simultaneously satisfied within the available network resources.

Table 3-1 Definition of DCBI Routing Problem [PH01]

The first modification is the BIH building method. The BIH will be built only when a demand arrives. That is, if a demand requests β_k bandwidth, all useless nodes screened out based on the required bandwidth and the right level β_k -BIs is built, instead of building a complete BIH hierarchy with many levels. This approach reduces the overhead of maintaining BIH, and makes DCBI applicable to on-line demands.

DCBI routing algorithm still uses the Lowest Level (LL) heuristic to build the lower level BIs. It adapts the multiplicative-increasing approach to build a lower BI. A

factor M is used to multiply the bandwidth requirement ($M * \beta$) for building a lower BI as shown in Figure 3-3 (b). For example, $M=2$ for 100% wider path. The building process will be stopped until no route can be found in any lower BIs. The building of lower level BI is only restricted to the current BI rather than the whole network. This gives a better computational complexity than the original one. The lower level BI has a smaller size that makes the process of searching the best route more efficient. If a new lower BI can be found, the old BI becomes useless and will be thrown out.

Figure 3-3 The Blocking Islands Hierarchy in DCBI Routing

As many networking applications or services have a strict requirement on delay, delay becomes very important and can't be easily neglected for QoS-based routing. By using the least cost heuristic based on delay criteria, DCBI routing algorithm takes delay into account. So DCBI can cope not only with concave metric (bandwidth), but also with

non-concave metrics (delay). Once the lowest level BI is found for β bandwidth requirement by using Dijkstra's algorithm, the shortest delay path can be selected from the qualified links. In summary, there are two steps in DCBI routing to find a path satisfying bandwidth and delay constraints at the same time: one is to find the links that have at least β available resources in the possible lowest BI; another is to find the shortest delay path among the links that were found in the first step.

Next, let's give a simple example to illustrate how the DCBI routing algorithm works. We assume all of the links in Fig. 3-3(a) have 100ms propagation delay. A new routing demand $d_u = (a, e, 12K, 500ms)$ is a communication demand from a to e with 12K bandwidth requirement and 500ms delay bound. Applying DCBI routing, a 15K-BI is first created as shown in Fig. 3-3(a). We know that there is at least one route satisfying the bandwidth requirement within the 15K-BI. This is the route existence property. The shortest delay path $a \rightarrow f \rightarrow e$ with 200ms then is found. Since both bandwidth requirement and delay bound are satisfied, we build the lower 30K-BI as shown in Fig. 3-3(b). The path $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ is the shortest delay path in 30K-BI. The building process won't be terminated until some QoS parameter can't be satisfied. When trying to build 60K-BI, we found the destination e is not in the 60K-BI. So we terminate the building process at 30K-BI. Therefore, the path $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ in 30K-BI is the final path for the demand. In the following section, we will discuss DCBI routing algorithm in more detail.

3.1.3 Description of DCBI Algorithm

The DCBI routing algorithm calls two functions. One is *Find_BI()* whose input parameters are the network graph, bandwidth request and source, and whose output is a

dedicated blocking island. Another is *shortest_Delay()*, whose inputs are the network graph, delay bound, source and destination nodes. It finds the shortest delay path and returns the total delay of the path. In order to avoid the overhead of multiple route computation and make the routing result precise, Pun and Hamdi chose factor M as 2 for 100% wider path. The DCBI routing algorithm is summarized in Table 3-2.

```

island ← Find_BI (whole network, x, source);
if (destination is not in Island)
    return route does not exist;
[delay, route] ← Shortest_Delay();
if (delay > delaybound)
    return route does not exist;
do {
    solution ← route;
    x ← x * M;
    island ← Find_BI (island, x, source);
    if (destination is not in island)
        break;
    [delay, route] ← Shortest_Delay ();
} while (delay <= delaybound)
return solution;

```

Table 3-2 pseduo-code of DCBI routing algorithm [PH01]

Pun and Hamdi evaluated the performance of DCBI routing algorithm in [PH01] and reached the following conclusions:

1) DCBI can achieve a better allocation rate than some other routing algorithms, like least hop count, least delay and shortest-widest algorithms. This is because DCBI only route the demand in the lowest level BI, in which the available bandwidths are largest. It can get the better remaining bandwidth connectivity and hence can solve more routing problems.

2) DCBI is more efficient in applying the shortest path algorithm, as it is restricted to the nodes in a Blocking Island, while other algorithms search the shortest delay in the whole network.

3) DCBI intends to find a route just fit for the delay bound requirement in the lowest level-blocking island, instead of finding a shortest delay path in the whole network. This method can result in larger probability of allocating an additional new demand.

4) DCBI running time is faster than other algorithms, due to the route existence property of the BI technique. Before even starting to compute a new route, we know if a route with the desired properties exists or not between source and destination.

3.2 The newly proposed DDCBI Routing Algorithm

QoS-based routing is mostly used by the real-time networking applications and services, like Internet Audio/Video, conferencing, etc. These applications or services are all delay-sensitive and have tight requirement on delay. So it's very important to obtain accurate and complete delay information when trying to route paths for these applications. However, the original DCBI routing only considers a static delay metric, given by the propagation delay, which is based on the distance between the sender and the receiver. Through the literature review, we found the most QoS-based routing schemas just take the propagation delay into account, and often ignore the queuing delay. Under the assumption of small packet sizes and limited buffer space, the link propagation delay is dominant and the queuing component of the link delay can be neglected. However, the propagation delay is a static value and does not really reflect changes in the

network state (such as congestion). On the contrary, the queuing delay (i.e., the time a packet has to wait in a queue for service) is dynamic and traffic sensitive, and can accurately reflect the network's load. Due to this concern, we redefined the delay metric in DCBI as a dynamic delay given by the sum of the propagation delay, transmission delay and queuing delay. We named the extended algorithm Dynamic DCBI (DDCBI). DDCBI is a "true" QoS-based routing algorithm because it takes into account the changes in network congestion that can have an impact on the QoS of delivering the demand after the demand was originally scheduled. This is important especially for long-living demands, such as in VPN applications. DDCBI routing algorithm can ensure that VPN provides better QoS for its customers.

For the DDCBI routing, we need two new mechanisms to realize the dynamic checking and enforcing of the delay constraints. A mechanism is needed to collect the mean value for the queuing delay at each node over a short period of time t_q . Given that the queuing delay is very dynamic and changes from packet to packet, we cannot use instantaneous values for the evaluation of the end-to-end delay. We use instead a function $\text{CollectAvgQDelay}(n_i, t_q)$, which returns the mean queuing delay at a node n_i over a small interval of time t_q .

Another mechanism is needed for comparing periodically the end-to-end delay with the corresponding delay constraint for each flow, in order to ensure the desired QoS for the communication demands. If the end-to-end delay is larger than the delay constraint, then a rerouting procedure will be triggered, which will try to find a new route that satisfies the constraint. The pseudo-code for the new function $\text{Periodic_Check}(t_c)$ of

the DDCBI routing algorithm is shown in Table 3-3. The input parameter t_c of the function is the time between two consecutive executions of the checking mechanism, which is triggered periodically for each flow. In general, t_c should be much greater than t_q (i.e., $t_c \gg t_q$).

```

Periodic_Check( $t_c$ ) {
    delay  $\leftarrow$  0;
    for (all links along the path)
        delay  $\leftarrow$  propagation delay + transmission delay
            + CollectAvgQDelay ( $n_i$ ,  $t_q$ );
    if (delay  $\leq$  delaybound )
        keep the route;
    else
        reroute;
} // end Periodic_Check

```

Table 3-3 Function Periodic_Check of the DDCBI routing algorithm

In chapter 5 we describe the simulation model for DDCBI, and in chapter 6, we analyze the simulation results to show the effect of queuing delay and other factors on the DDCBI routing results.

Chapter 4 Simulation Model of DDCBI Routing Protocol

This chapter describes the detailed implementation of DDCBI routing in *ns*, and all the components of the DDCBI simulation model.

4.1 Routing Function Blocks

In general, every routing implementation in *ns* consists of three function blocks:

- Routing agent: exchanges routing information with neighbors.
- Route logic: uses the information gathered by routing agents (or the global topology database in the case of static routing) to perform the actual route computation.
- Classifier: sits inside a Node. They use the computed routing table to perform packet forwarding. [FA00]

When implementing a new routing protocol into *ns*, we should follow these function blocks. However, it doesn't mean all of these three blocks must be implemented at once. For instance, when a distance vector routing protocol exists and one implements a link state routing protocol, one only needs to implement a routing agent that exchanges topology information in a link state manner. It can then use the same route logic that does Dijkstra on the resulting topology database and the same classifiers as other unicast routing protocols.

For the implementation of DDCBI routing strategy, route logic and classifier parts are newly implemented based on QoS-based routing properties and DDCBI routing characteristics. Routing agent is used as the same as link state routing protocol existing in *ns*. Detailed implementation is coming in section 4.4.

4.2 Internals and Architecture of Routing in *ns*

In section 3.2.4, we have briefly introduced the existing routing protocols in *ns*. Although the detailed implementations are different, those routing protocols have to follow the *ns* routing architecture. In *ns*, every routing protocol has its own routing logic to perform its functionalities and behaviors. In addition, there are various classes that deal with various aspects of routing, e.g. Simulator, Node, Link, and Queue etc. Let's introduce these basic classes involved in routing strategies first.

Class RouteLogic: It is essentially defined as the routing table that is created and maintained centrally for every simulation. There are three procedures for unicast routing. Procedure of register {} is for registering routing protocol and a list of nodes that will run this routing protocol. Configure {} is used to invoke route protocol to perform the appropriate initializations. Lookup {} performs the action of querying route information.

Class rtPeer: this is a container class used by the protocol agents. Each object stores the address of the peer agent, and the metric for each route advertised by that peer. The class maintains several variables, and procedures for accessing the variables.

Class Agent / rtProto: In *ns*, class Agent/rtProto is a base class from which all-routing protocol agents are derived. Our DDCBI routing agent will be a subclass of it and inherit the most procedures of it. It defines the following procedures:

1. `init-all {}` to initialize the complete protocol. The centralized routing protocols such as static and session routing use all nodes in the topology as an argument. For dynamic routing protocols, they will use the argument list to instantiate protocols agents at each of the nodes specified.
2. `init {}` is the constructor for protocol agents that are created. The base class constructor initializes the default preference for objects in this class, identifies the interfaces incident on the node and their current status. Centralized routing protocols do not create separate agents per node while the dynamic routing protocols create separate agents for each node.
3. `compute-routes {}` computes the actual routes for the protocol. The computation is based on the routes learned by the protocol, and varies from protocol to protocol. The procedure is invoked whenever the topology changes, or when the node receives an update for the protocol.
4. `send-updates {}` is invoked when the node routing tables have changes, and fresh updates have to be sent to other nodes. This procedure may also be invoked when there are no changes to the routes, but the topology incident on the node changes state.

Class Node: A node in *ns* is represented by an instance of Class

Node. Node itself is an aggregate object consisting of a set of classifiers. It contains an address or id that monotonically increase by 1 as new nodes are created, a list of neighbors and a list of agents as well as a list of routing modules.

Class Classifier: When receiving a packet, a node first examine the packet's fields, usually its destination address, and on occasion, its source address. It should then map the values to an outgoing interface object that is the next downstream recipient of this packet. In *ns*, a classifier object performed this task, and it has several types that are Address Classifiers, Multicast Classifiers, Hash Classifiers and MultiPath Classifiers for different purposes. Each classifier contains a forwarding table indexed by slot number. The job of a classifier is to determine the slot number associated with a received packet and forward that packet to the object referenced by that particular slot.

Figure 4-1 Structure of Unicast Node ([FA00])

Figure 4-1 shows the structure of the simplest node (unicast) containing only one address classifier and one port classifier. Address classifier decides, on the basis of destination address, to which link should the packet be forwarded. If it is destined for the current node, it is forwarded to port classifier, which decides to which agent should the packet be forwarded based on port number.

4.3 Routing Module in *ns*

Figure 4-2 Interaction among Node, routing module and routing ([FA00])

Routing Module is a uniform interface to organize the classifiers and to bridge these classifiers to the route computation blocks. As we just mentioned in 4.1, every routing implementation consists of routing agent, route logic and classifier. A routing module manages all these function blocks and interfaces with node to organize its classifiers. The routing modules of *ns* include *RtModule/Base*, *RtModule/Mcast*,

RtModule/Hier, and so on. Figure 4-2 shows the interaction among node, routing module and routing.

A routing module contains three major functionalities:

- `register {}/ unregister {}`: Through `register {}`, a routing module initializes its connection to a node to tell the node whether it interests in knowing route updates and transport agent attachment; And it creates its classifiers and install them in the node. `unregister {}` is used to tear down the connection and to delete its classifiers and remove its hooks on routing update in the node.
- `RTModule::add-route {dst, target}` and `RTModule::delete-route {dst, target}`: The node informs the module routing updates via them.
- `RTModule:: attach {agent, port}` and `RTModule::detach {agent, nullagent}`: The node lets the module know transport agent attachment and detachment in a node.

4.4 The Development of DDCBI routing strategy

A routing strategy is a general mechanism by which *ns* will compute routes for the traffics. *Ns* provides the users 3 routing strategies: Static, Session and Dynamic routing strategy. They are all best-effort routing strategies. In this research, a QoS-based routing mechanism called DCBI routing is designed and implemented into *ns*, which riches the routing protocol family of *Ns*.

4.4.1 The Functionalities of DDCBI routing protocol

The main functionalities of DDCBI routing strategy can be summarized as follows:

- Accepting the traffic demands from a simulation interface.
- Computing routes for flows through DDCBI routing algorithm. The routes should satisfy the requirement of bandwidth and delay constraints.
- Reserving the resources along the path. It guarantees the availability of resource allocated.
- Inserting qualified paths into a forwarding table, such that when packets come, they are directly forwarded to the next hop.
- Identifying each flow by a unique ID. The packets belonging to the same flow has the same ID that is stored in the packets' headers. When a packet arrives at a router, the router first checks its flowID, instead of the destination, then looks up the forwarding table to see which is the next hop corresponding to this flowID. It works like the label of MPLS.
- Sending and receiving update information periodically.
- Checking delay of a path periodically. If the current delay exceeds the delay constraint, a re-computation event will be triggered. Otherwise, the current path will be kept until the next periodical checking. The periodic time t_c is 2s.

4.4.2 The Implementation of DDCBI Routing Agent

Class `Agent/rtrProto/DDCBI`, which is a subclass of `Agent/rtrProto`, is a QoS-based routing protocol we brought into *ns*. DDCBI routing is designed as a distributed routing, so the implementation completely followed the structure of dynamic distributed routing in *ns*.

Figure 4-3 shows the states and workflow of `Agent/rtrProto/DDCBI`. In this diagram, thick circles represent end states; thin circles are for middle states; actions are shown as rectangles; events triggered by other agents are shown as ellipses; boolean decisions are shown as diamonds. The meaning of the states is given below:

- Initialize protocol agent. When a routing demand comes, it is accepted by `instproc init-all{}`. The routing demand contains the requirement for bandwidth and delay, and the starting point (source) as well as the end point (destination). `init-all{}` abstracts these information, then assign them to the corresponding instance variables.

Figure 4-3 Logic working flow of DDCBI routing agent

- Build BI blocks. After getting bandwidth requirement, `instproc buildBI{}` will check the available bandwidth of all links to see which links are satisfied with the required bandwidth. The variable `bandwidth_` in class `Link` records the available bandwidth of each link. Through `LinkDelay::bandwidth{}`, link's bandwidth can be accessed. The algorithm of building BI blocks from [Frei00] is presented below.

```

function BuildBI( $G = (N, \mathcal{L}), \Gamma, \beta, x$ )
   $S \leftarrow \{x\}$ 
   $C \leftarrow \emptyset$ 
   $Q \leftarrow \{\text{links where } x \text{ is one endpoint}\}$ 
   $W \leftarrow \emptyset$ 
  while  $Q \neq \emptyset$  do
     $l \leftarrow \text{pop}(Q)$ 
    if both endpoints of  $l$  are in  $S$  then
       $W \leftarrow W \uplus \{l\}$ 
    else
      if Supports( $l, \Gamma, \beta$ ) then
         $y \leftarrow \text{the endpoint of } l \text{ that is not in } S$ 
         $S \leftarrow S \cup \{y\}$ 
         $Q \leftarrow Q \cup \{\text{links where } y \text{ is one endpoint}\} \setminus \{l\}$ 
      else
         $C \leftarrow C \cup \{l\}$ 
   $W_2 \leftarrow \text{all links of } C \text{ that have both endpoints in } S$ 
   $C \leftarrow C \setminus W_2$ 
   $W \leftarrow W \uplus W_2$ 
  return ( $S, C, W$ )

```

Algorithm: Construction of the β -BI for node x , given the network graph g and the established connections Γ . It returns 3 values: the node S contained in the β -BI, its cocycle C , and the links W that are inside the BI. Operator $\uplus$ denotes the union of two sets such that there are no duplicates in the result.

- BI blocks found? We got two arrays after building BI blocks. One holds all qualified links while another one holds the qualified nodes. If they are not empty, it means that the BI block is found, then go to the next step. Otherwise, it means there is no link that has enough bandwidth to meet the bandwidth constraint.

- Find shortest delay. Once the qualified links are found, the next step is to find the shortest delay path from source to destination among the qualified links. The instproc `shortest_delay{}` performs this functionality. The cost of each link is expressed as delay that is the sum of propagation delay, transmission delay and queuing delay. By using Dijkstra algorithm, the least cost path can be calculated.
- Compare the shortest delay with delay bound. The shortest delay may or may not exceed the delay bound.
- Exceed? If not, go back to “build BI blocks” to try to build a lower level BIs. As mentioned in 3.1.2, DCBI routing should build as low as possible level BIs. The multiplicative factor M is assigned to 2 for 100% wider path. The value 2 is from [PH01]. If the answer is yes, then go to the next condition test.
- First time? If it’s the first time to compare the shortest delay with delay bound, and the shortest delay is greater than delay bound, then go to allocate failure state.
- Allocate failure. It is used to notify the user that the current network condition can’t satisfy this traffic demand.
- Go back to `prev_path`. There is a while loop to try to find a feasible path in a lower level BIs for a demand. The loop will not stop until the lower level BI can’t meet the demand. So the last successful BI is the lowest level BI we can

find and the path that came from that BI is the best path for the demand at this time.

- Send updates. `instproc send-periodic-update{}` sends update messages every *adverInterval*. The default value `tc` of *adverInterval* is 2 second. In addition to periodic updates, each agent also sends triggered update messages (i.e. `send-update{changes}`). It does this whenever the forwarding tables in the node change. This occurs either due to changes in the topology, or because an agent at the node received a route update, and recomputed and installed new routes.
- Receive updates. Agents use `instproc recv-update{}` to receive the update messages.
- Check delay. After receiving update messages, the agent will check whether the current delay making up a path is able to meet the delay constraint.
- Exceed delay bound? *Cur_delay* records the most recent delay along a path. We need to compare the current delay with delay bound.
- Keep path. If the current delay doesn't exceed delay bound, the path will continue to be used to forward packets.
- Compute routes. If the change of delay is major and it makes the present delay along a path exceed the delay bound, it means the current used path can't satisfy QoS requirement any more, and a new path need to be recomputed.

- Path found? Is there a qualified candidate path found for the demand? The answer is either allocation failure or allocation success.
- Add routes. The path will be added into the forwarding table. `instproc add-routes{}` will invoke the related methods of class `node` to add routes into a classifier.

4.4.3 Other Extension To The Simulator, Node, Link, and Classifier

Class `Agent/rProto/DDCBI` performs the main work of DDCBI routing. Besides it, class `Simulator`, `Node`, `Link` and `Classifier` are also involved. The implementation of these classes is much simpler than that of class `Agent/rProto/DDCBI`. They are illustrated in Figure 4-4. The meaning of those activities can be described as follows:

- Idle. This is the initial state. Before the agent is triggered and after a route is allocated, the agent enters the idle state.
- Receive demands. The `instproc rproto{}` in class `Simulator` accepts a demand. The first argument is the name of a routing protocol. The second argument is a list that contains other information, like nodes, links and so on.
- Register demands. After receiving demands, they will be registered into class `RouteLogic` that acts as a routing table. The index of an array `rprotos_` is the name of the protocol, and the value is the contents of a traffic demand.

Figure 4-4 Logical working flow of other extension

- Configure routes. The routes for all demands should be set up before traffics come. An object of class Simulator constructs the instances of class Agent/rtProto/DDCBI and performs the appropriate initializations.
- Instantiate Agent/rtProto/DDCBI. The behaviors and workflow of this class are already interpreted in detail in previous section.
- Add routes. The path should be added into class Node after a qualified path is found for a demand. Node::add-route-DDCBI {} performs this action. It takes destination, target (next hop) and flow ID as arguments. Moreover, Node will inform routing module by RtModule/Base::add-route {dest, target, flowID}.

- Install routes. Routing module RtModule/Base add routes into forwarding table for packets forwarding. In *ns*, class Classifier acts as a forwarding table. For most routing protocols when a node receives a packet, it examines the destination address of the packet to get a slot number, and then forwards that packet to the object referenced by that particular slot. In our implementation of DDCBI routing, there are differences from other routings at the structure of a forwarding table and checking a packet's field. As shown in Table 4-1, there is a one more item -- flowID in a forwarding table. The flowID is a label that represents which flow the path belongs to. Why here we use a label to identify traffic flows instead of destination address? The reason is: for QoS-based routing, which path is chosen depends not only on the source and destination of a traffic flow, but also on the qualities of services of a traffic flow required. For example, two flows I and II (flowID = 1 and flowID = 2) start from the same source and end to the same destination D. Assume that they have different bandwidth or delay constraints. The paths for these two flows are shown in Table 4-1. If only examining the packet's destination when flow I came, node will confuse with which path should be used to forward the I's packets. That is, *destid* is not unique identifier in DCBI forwarding table, but flowID is. Corresponding to flowID in a forwarding table, the header of each packet should contain flowID too. IPv6 defines a 20-bit flow label [PD96] in a packet's header, so we can use it directly.

Destination	flowID	nh
D	1	A
D	2	B

Table 4-1 a Simplified Forwarding Table with FlowID

- Reserve resources. After the routes are set up, resource reservation mechanism will work to reserve the bandwidth. It originates from the traffic sender and carries reservation request to the nodes along the path between the sender and the receiver. The procedures `reserveBW()` / `reserveBW{}` of class `Simulator` in interpreted hierarchy and compiled hierarchy perform this function.
- Update available bandwidth. Variable `bandwidth_` in class `Link` records available bandwidth. It will be visited when trying to allocate new traffic demand, and updated after new route is set up.

Another thing need to be mentioned is the implementation of queuing delay. Unlike propagation delay and transmission delay, queuing delay varies from time to time. Thus, we need a statistical method to get a mean value of it. Fortunately, class `Sample` and `Integrator` in *ns* are for statistical purpose. We first create the objects of class `Sample` and `Integrator`, and then connect them to the object of class `QueueMonitor` that is used to monitor packets' arrival, departure and drop. This process is done in class `Link` when a link is built. After that, a handler of `QueueMonitor` can be used to access a mean queuing delay when computing the shortest-delay path. The small interval t_q is 0.01s.

All of the implementations mentioned above are written either in TCL or in C++. Class `Agent/rProto` and its existing subclasses are written in TCL, so our `Agent/rProto/DDCBI` is implemented in TCL as well.

```
Agent/rProto/DDCBI instproc init-all demand {
    global rtglibRNG start
    $self instvar ns_ demand_ node_ bw_avail_ count_recompute
    set value [$self compute-routes]
    --
    if { $value == "unsuccess_bw" || $value == "unsuccess_delay" }
    { return $value }
    --
}
```

4.4.4 The Interface to the Simulation Operator

The user level simulation script requires one command to specify the routing strategy for the simulation. Instproc `rtproto{}` in class `Simulator` is such a command to specify routing protocol.

```
$ns rtproto DCBI $n1 $n2 $bw $delay $flowID
# Run DCBI agent; flow is from node 1 to node2; QoS requirements are bandwidth
and delay
```

4.5 Simulation Design Process

“A simulation is the imitation of the operation of a real-world process or system over time” [BCNN01]. From the simulation, data are collected as if a real system were being observed. These simulation-generated data can help us to understand, estimate or

evaluate the process or system. In order to gain accurate simulation results, the simulation process should be carefully designed. In our research, the simulation design procedures exactly followed the steps mentioned in book “Discrete-Event System Simulation”. There are four design steps introduced in detail in the following subsections.

4.5.1 Problem Formulation and Objectives

The real-world system being simulated in this research is a QoS-based routing protocol – DDCBI routing. As mentioned in section 3.1.3, comparing with the other common used routing algorithms, DDCBI considerably improves the online QoS-routing performance and results in substantially more economical networks. However, as discussed in the section 3.1.4, DDCBI routing algorithm didn't take other various kinds of delays, e.g. transmission delay and queuing delay, into consideration. Also we found many paper ignored to discuss transmission delay and queuing delay without providing strong reasons. So, one of objectives of this research is to measure and evaluate how much the transmission delay and queuing delay will affect the routing results. Besides that, another objective of the simulation is to provide some insight into the issue of DDCBI routing protocol, such as how QoS parameter settings impact the routing results and how the QoS-based routing achieves global efficiency in network resource utilization etc. The following factors are varied in the simulation experiments:

- Demand's contents: there is 4-tuple in each demand. Source node and destination node are randomly generated for each simulation. The amount of bandwidth and delay represent QoS service level the network users chose.

- Different traffic workload: We chose the different traffic workload, for instance a small amount of traffics (around 30 demands), medium amount of traffics (around 50 demands) and high amount of traffics (100 demands).

4.5.2 Model Conceptualization and Translation

“The construction of a model of a system is probably as much art as science.” [BCNN01]. Usually it starts from a simple model by abstracting the essential features of a problem and by selecting and modifying basic assumptions that characterize the system. After making sure it works well, adding more features to enrich and elaborate the model until a useful approximation results. In this research we focus on several fundamental features of QoS-based routing, which includes the following building blocks:

- Topology database: routers periodically flood routing information to the other routers (link state based). The information allows routers to construct a whole picture for their area(s) and is used to compute the qualified path. For QoS-based routing, information contains not only the contents as usual (e.g. hop number), but also the more detail QoS information about the network state, e.g. amount of unreserved bandwidth, mean delays, etc., which allows determination of paths that satisfy more complex constraints. This information may be quite dynamic, and it needs increase the amount of control traffic to keep information fresh.
- Routing mechanism: QoS-based routing algorithm is an important component of QoS-based routing protocol. The detailed design and advantages of DCBI algorithm have already been described in section 3.2.2. Due to simple complexity

and better allocation rate, we introduced it to *ns* to share a good QoS mechanism with other *ns* users.

- FIB (Forwarding Information Base): For the best effort routing, routes in the RIB (Routing Information Base) are used to generate the router's FIB for efficient lookup. QoS-based routing departing from the best effort routing is where QoS routes are selectively inserted in the FIB. The insertion of a QoS route in the FIB usually happens when a signaling protocol tries to establish a QoS path for some traffic. [AST98]. When a demand comes, DCBI routing attempts to allocate a path for it. Once an appropriate path that meets the QoS requirement is found, the FIB is suitably modified. So that when packets arrive in the data path, they get forwarded along the corresponding path.
- Resource reservation will send reservation information along a path to reserve the network resources against the other usages.
- Network topology: in this paper, we adopted AT&T IP backbone network as our network topology, since it's more realistic and QoS service is provided by ISPs. It is composed with 27 nodes and 45 duplex links, and created manually.
- Traffic generation: The traffics were generated according to a Constant Byte Rate (CBR) distribution at the different sending rates. The sending rate is decided on the required bandwidth of each demand. It'll be further discussed in chapter 5.
- Simulation scenarios: After all of the components were developed, they were combined together. A scenario, which itself is a building block, acts as a frame

which integrates all building blocks and makes everything ready to work. According to the combinations of all experimental factor values, 13 scenarios were constructed for the simulation. They will be described in section 5.2.

4.5.3 Model Verification and Validation

In order to guarantee the credibility of the simulation model, model verification and validation must be done. It's the most important and difficult tasks facing a model developer. Verification includes the examination of input parameters (the system configuration) and the examination of logical structure of the model. The verification may be done in different stages, i.e., after each component is developed, it should be debugged and tested thoroughly. In this research, the logical examination was done in two ways: structural examination and functional examination.

The functional examination means to trace each functional feature individually, from its starting point to the ending point. For example, for the sake of examination of computing route mechanism, we had to trace the entire computation process, from building BI blocks to searching the shortest delay, from the first allocating route to the route re-computation. Functional examination can make sure the functions of the feature are fully fulfilled.

In structural examination, the model was examined as a whole. The purpose of structure examination is to check whether the model contains all features required in the simulation, and whether those components can work together (interact with each other)

correctly. The structural examination guarantees that the model works properly as a whole.

Validation is concerned with building the right model. It is the determination that the model is an accurate representation of the real system. In our case, a real system does not exist yet, as the DCBI and DDCBI routing algorithms have never been implemented on a real network. Therefore, it was impossible to validate the simulation model by comparing it with real results. The most we could do is to compare the DDCBI model results with DCBI results published in literature. For this, we disabled the DDCBI model feature that compute the dynamic delay and considered only the propagation delay as in DCBI. In these conditions, the results were very close with those published for DCBI.

4.5.4 Production Runs and Analysis

Production runs and their subsequent analysis are used to estimate measures of performance for the system designs that are being simulated. We design 12 simulation scenarios and get various measured values from them. All measured values are run repeatedly until confidence intervals of less than 5% of the mean value, using a 95% confidence level in this research.

4.6 Simulation model for DDCBI

The components involved in the simulation model are grouped into four basic building blocks: DDCBI routing agent, network topology, traffic generation and simulation scenarios. The integration of the DDCBI routing into the ns framework was

described in detail in section 4.4. In this section we will describe the other three components.

4.6.1 Topology Modeling

To simulate the behavior of a protocol, a topology of network needs to be generated. The topology is an abstraction or a model of the actual network structure. The topology of a network is typically modeled using a graph in which nodes represent switches or routers, and edges represent direct connections (links) between switches or routers. The selection and generation of a proper topology is very important in a simulation since different topologies may lead to different results.

Figure 4-5 Network Topology of AT&T IP Backbone

In that QoS-based routing is able to make resource utilization more efficient and ensure QoS requirements for users, ISP can greatly benefit from this kind of routing approach. Actually, the utility of QoS-based routing is currently restricted to use for routing of large traffic aggregates such as traffic trunks that are part of one ISP. So in this

research, the topology we chosen is similar to the AT&T IP Backbone Network topology shown in Figure 4-5 (the same topology was used to assess the DCBI algorithm in [PH01]).

Totally, there are 27 nodes and 45 full duplex links in Figure 4-5 that roughly represent the area of the continental USA. The capacity of each link is considered to be of 155Mbps (OC3). The propagation speed through the links are taken to be two-thirds the speed of light. Under this assumption, we defined the propagation delay for each link is in the range from 8ms to 10ms. This topology is created manually in *ns*.

```
set nodenum 27
for {set i 0} {$i < $nodenum } {incr i} {
set n($i) [$ns node]
} // create nodes

$ns duplex-link $n(0) $n(1) $bw(0)Mb $delay(0)ms DropTail
$ns duplex-link $n(0) $n(4) $bw(1)Mb $delay(1)ms DropTail
$ns duplex-link $n(4) $n(5) $bw(2)Mb $delay(2)ms DropTail // create
links
```

4.6.2 Traffic Generation

In order to emulate the situation of a real network, we need to bring traffic into the simulation, rather than only conduct the establishment of DDCBI routing in a free network. These traffics share the routers (nodes), links, and compete for the network resources with each other.

ns provides a traffic generator to the users. It allows users to select different modes (EXPOO_Traffic, POO_Traffic and CBR_Traffic) and use different rates to generate and send traffic. In this simulation, the CBR_Traffic mode is used. The CBR_Traffic mode generates traffic according to a deterministic rate. Packets are constant size. Optionally, some randomizing dither can be enabled on the interpacket departure intervals.

A CBR object is embodied in the OTCL class Application/Traffic/CBR. The member variables are:

- `rate_`: the sending rate.
- `interval_`: interval between packets.
- `packetSize_`: the constant size of the packets generated.
- `random_`: flag indicating whether or not to introduce random “noise” in the scheduled departure times (default is off).
- `maxpkts_`: the maximum number of packets to send.

By initializing those instance variables with proper values, users can configure the traffic generator and design their specific traffic pattern. The values used for our simulation are:

- `packetSize_ 240bytes`. ISPs use small packet size to send traffic. 240 bytes size is used by AT&T IP backbone network.

- `interval_` is equal to $(\text{packetSize_} * 8) / \text{bw_demand}$, in which `bw_demand` is the amount of bandwidth needed by a demand. In the formula, the interval is decided by required bandwidth of each demand. In terms of `random_` and `maxpkts`, we use the default values.

The script for CBR traffic is:

```
set cbr($index) [new Application/Traffic/CBR]
$cbr($index) set packetSize_ 240
$cbr($index) set interval_ [expr 240 * 8 / $bw_demand ]
```

In addition CBR distribution, the exponential distribution traffic generator is also used as background traffic in the simulation. The Otcl class name and associated parameters are given below:

Exponential On/Off:

`packetSize_` the constant size of the packets generated
`burst_time_` the average “on” time for the generator
`idle_time_` the average “off” time for the generator
`rate_` the sending rate during “on” time

4.6.3 Simulation Scenarios

After the basic building blocks are developed, they have to be put together to form the simulation system. A simulation scenario, which itself is a basic building block, is responsible for this task. All components used in the simulation are

integrated in the scenario so that they can come into play. A set of scenarios is created in this research. They are used in simulations to examine the performances in different situations. The specification and implementation of those scenarios is discussed in the following section. In addition, a set of parameters is used to define the behaviors of the system. The determination of the parameters' value will be discussed too.

Scenario 1

Node: 27

Links #: 45 (duplex)

Bandwidth of each link: 155Mbps

Delay constraint: 3.55s

Number of demands: 100

Different Bandwidth: 5% / 15% / 25% / 35% / 50% / 60% / 70% / 80% / 90% /

(Different bandwidth is defined as the ratio of bandwidth required for the demands divided by the capacity of a link.)

Scenario 2-4:

Node: 27

Links #: 45 (duplex)

Bandwidth of each link: 155Mbps

Bandwidth required: [64k – 20M] bps

Delay constraint: 0.01s/ 0.03s/ 0.05s/ 0.08s/ 0.15s/ 0.3s/ 0.85s/ 1.05s/ 2.25s/ 3.55s

Least delay (cost): propagation delay + transmission delay + queuing delay

Number of demands: 30 (for scenario 2)

50 (for scenario 3)

100 (for scenario 4)

Scenario 5-7:

Node: 27

Links #: 45 (duplex)

Bandwidth of each link: 155Mbps

Bandwidth required: [64k – 2 0M] bps

Delay constraint: 0.01s/ 0.03s/ 0.05s/ 0.08s/ 0.15s/ 0.3s/ 0.85s/ 1.05s/ 2.25s/ 3.55s

Least delay (cost): propagation delay

Number of demands: 30 (for scenario 5)

50 (for scenario 6)

100 (for scenario 7)

Scenario 8

Node: 27

Links #: 45 (duplex)

Bandwidth of each link: 155Mbps

Bandwidth required: [64k – 40M] bps

Least delay (cost): propagation delay + transmission delay + queuing delay

Number of demands: 100, Delay constraint: 0.06s (for scenario 8)

Scenario 9

Node: 27

Links #: 45 (duplex)

Bandwidth of each link: 55Mbps (OC1)

Bandwidth required: [64k – 25M] bps

Delay constraints: various

Number of demands: 10, plus background traffic

To perform the simulation of a scenario, a simulator object must be firstly created. This simulator object will handle all simulation activities. Let's take Scenario 1 as an example to describe the process of the simulation implementation. The network topology consists of 27 nodes, 45 duplex links. The topology is created manually. Thirty pairs of traffic agents (a pair of traffic agents consists of a source agent and a sink agent) are created. The responsibility of these agents is to create the traffics. For each pair of agents, we randomly select a pair of nodes and attach the agents to them. Since there are 27 nodes and 30 pairs of traffic agents, each node, on the average, has the same opportunity to play the role of both the source node and the destination node at the same time. The source agents and the sink agents are connected in pairs. Once the simulation starts, they begin to generate and send traffic packets in a CBR mode. Besides these, the following command tells the *ns* that DCBI routing protocol will be run on all the nodes:

```
$ns demands $n($Id1) $n($Id2) $bw_required $delay_bound $flowID
```

Up to now, the basic building blocks of the simulation have been modeled and developed. Integrating those building blocks into a whole, we can build up the simulation output system. The different scenarios of the system are configured through assigning proper values to the parameters. After that, we can implement the simulation. In next chapter, the simulation results will be discussed.

Chapter 5 Analysis of Simulation Results

The simulation system has been modeled in chapter 4. After the model is translated into computer programs (in C++, Tcl and OTcl), we can start to perform the simulation experiments. There are 9 different simulation scenarios in total. The output data of all these scenarios will be analyzed in this chapter.

5.1 Relationship between bandwidth requirements and demands allocation rate

Scenario 1 is designed to evaluate the relationship between different bandwidth requirements and successful number of demands. A random set of 100 route demands characterized by two endpoints, a bandwidth constraint and delay bound are generated. The workload can be adjusted by varying the mean bandwidth requirement. This scenario is run repeatedly and part of the output data is given in table 5-1. The output is the number of successfully allocated demands.

In order to make the output data convincing, we need to do statistical analysis of the output data and to compute confidence intervals. Several simulation scenarios were used in this thesis. Unless otherwise stated, each simulation scenarios was run repeatedly with new random numbers until confidence intervals of less than 5% of the mean value, at 95% confidence level were obtained. We will use the results from Table 5-1 to explain how the confidence intervals were calculated.

Replication, r	Different Bandwidth Requirement				
	5%	15%	20%	30%	40%
1	100	86	84	53	39
2	100	91	81	50	39
3	100	86	87	56	38
4	100	94	77	54	42
5	100	91	76	51	48
6	100	95	79	58	39
7	100	89	76	47	36
8	100	91	80	47	36
9	100	82	69	51	35
10	100	89	80	49	38
11	100	89	79	53	44
12	100	85	80	59	35
13	100	91	81	55	38

Table 5-1 Successful demands for different bandwidth requirements: results for replicated experiments

Step 1: Calculating the sample mean $\bar{Y}$ and the sample variance S_0 of the 13 replications.

Firstly, the sample mean is calculated using formula 5-1.

$$\bar{Y} = \frac{1}{R_0} \sum_{r=1}^{R_0} \bar{Y}_r \quad (5-1)$$

Where

$R_0 = 13$; Y_r : the r^{th} observation within replication r .

For example, the value of the sample mean for the 20% case is: $\bar{Y} = 79.2$.

Secondly, calculate the sample variance of R_0 runs. The sample variance is defined to be:

$$S_0^2 = \sum_{r=1}^{R_0} \frac{(\bar{Y}_r - \bar{Y})^2}{R_0 - 1} \quad (5-2)$$

The sample variance is $S_0^2 = 10.56$.

Step 2: Estimating the sample size R .

Given a half-length for a confidence interval, we can estimate the sample size required for meeting this criterion. Confidence interval at the 95% confidence level is used for all simulation scenarios. This means that it is desired to estimate the measured values to within $\pm 5\%$ error with probability 0.95. Based on this criterion, we first roughly estimate, and then, accurately compute the sample size through analyzing the initial sample. The rough estimate for R is given by

$$R \geq \frac{(z_{\alpha/2}, S_0)^2}{\varepsilon^2} \quad (5-3)$$

Where

$\phi(z_\alpha)$: cumulative normal distribution of confidence intervals. The value of z_α can be found in Table A.3 in Banks *et al.* [BCN96].

$1-\alpha$: confidence interval, $1 - \alpha = 0.95$, $\alpha = 0.05$.

ε : the half-length error, $\varepsilon = 0.05 * \bar{Y} = 79.2 * 0.05 = 3.96$.

Bring the values of above parameters into inequality (5-3). The estimate of the sample size can be obtained:

$$(Z_{0.025} * S_0 / \varepsilon)^2 = (1.96)^2(10.56)/(3.96)^2 = 2.59$$

Therefore, the final sample size must be at least as large as 3.

Step 3: Determining the sample size R .

The sample size calculated in Step 2 is just an estimate. It gives a lower limit of the possible value of the sample size. We can take this value as the start point to compute the true value of the required sample size. Inequality (5-3) gives the smallest sample size that satisfies the given criterion.

$$R \geq \frac{(t_{\alpha/2, R-1} S_0)^2}{\varepsilon^2} \quad (5-4)$$

Where

$t_{\alpha/2, R-1}$ is the percentage point of the Student t distribution with $R-1$ degree of freedom. The value of t can be found in Table A.5 in Banks *et al.* [BCN96].

The results calculated from inequality (5-4) are shown in Table 5-2.

R	3	4	5	6
$(t_{0.025, R-1} * S_0 / \varepsilon)^2$	12.45	6.81	5.20	4.45

Table 5-2: The smallest sample size required for 20% ($R_0 = 6$)

From Table 5-2, it can be seen that $R = 6$ is the smallest integer satisfying inequality (6-3). $R_0 = 13 > 6$, so we don't need run more. The half-length for 95% confidence interval is $t_{0.025, 12} Y = 1.964$. The resulting 95% confidence interval for successful demands, which the required bandwidth of each demand is equal to 20% of link bandwidth, is 79.2 ± 1.964 .

We repeated the same calculation for all the case studies, and found out that 13 replications are enough, so we did not need to run more.

Figure 5-1 Relationship between different bandwidth requirements and the number of successful demands

Different bandwidth requirements are defined as the ratio of bandwidth required for a demand divided by the capacity of each link in the network. As shown in Figure 5-1, with the increase in bandwidth requirements, the number of successfully allocated demands trends to drop rapidly. This is due to the increased competition between demands for network resources. From figure it can be seen that the success rate tends to

level off for bandwidth requirement over 60%. This result will guide us to select the bandwidth bound for the different purposes of the simulations.

5.2 Effects of delay constraints

Each demand includes two QoS parameters. One of them is the delay constraint, which represents a bound on the total delays of all links along a path. On one hand, the delay constraints are dependent on the specific network applications, but on the other hand, are limited by the network condition. How does one choose a reasonable delay constraint? This is a challenging question that requires more investigation. Figure 5-2, 5-3 and 5-4 will show us how different delay constraints impact the traffics entering the network under different network loads.

In this experiment, we use 3 different kinds of traffic loads. 100 traffic demands represent heavy traffic load, due to each node averagely attaching more than 3 pairs of traffic agents that send packets following the CBR distribution, and receive packets at the same time. 50 traffic demands represent moderate traffic load, since each node has almost 2 pairs of traffic agents. Comparing with these two traffic loads, 30 traffic demands bring a light traffic load to the network; every node has only one pair of traffic agents attached. We assume that all traffic demands in the network requires the same delay constraint.

From Figure 5-2 to Figure 5-4, we see first that the larger the delay constraint is, the smaller the chance that a flow gets to be dropped from the network. Secondly, when the network is getting busier (from 30 demands to 100 demands), the values of delay constraints should become larger in order to achieve same level of throughput. The reason is that packets need more time to traverse from the source to the destination.

Figure 5-2: Amount of dropped flows under different delay constraints with 100 demands

Figure 5-3: Amount of dropped flows under different delay constraints with 50 demands

Figure 5-4: Amount of dropped flows under different delay constraints with 30 demands

5.3 The impact of queueing delay on demand allocation rate

For many QoS-based routing algorithms, the propagation delay is the base of the delay metric. This assumption is reasonable for long links, small packet sizes and limited buffer space, where the link propagation delay is dominant and the queueing component of the link delay is neglected. However, QoS-based routing schema is usually used for those applications that are sensitive to network delays, such as Internet conferencing and Internet Audio. Considering a delay metrics that includes the queueing delay can help the routing algorithms to find a better path to meet the QoS requirements. In our experiments, the buffer space can hold 50 packets, which is the queue size adopted by the most routers in the market. We still run three kinds of workloads: 30 traffic demands, 50 traffic demands and 100 traffic demands. Through this experiment, we want to measure how the static and dynamic delay constraints impact the routing results.

In Figure 5-5, Figure 5-6 and Figure 5-7, represent the percentage of unsuccessful demands versus the delay constraint value with and without considering queuing delay.

Figure 5-5 Difference between delay with and without queuing delay and transmission delay with 30 demands

Figure 5-6 Difference between delays with and without queuing delay and transmission delay with 50 demands

Figure 5-7 Difference between delays with and without queuing delay and transmission delay with 100 demands

The number of unsuccessful flows when considering queuing delay is always greater than when queuing delays are not considered. Furthermore, the difference becomes more clear when the traffic is getting heavier. In a network with light workload, the queuing delay can reasonably be ignored. However, when the workload is getting heavier, the queuing delay becomes larger and it cannot be simply neglected. This results show that the DDCBI algorithm pays a price in terms of flow acceptance with respect to the static DCBI algorithm. Fewer flows can be allocated, but their QoS will be better, as shown in section 5.5.

5.4 Link utilization

Pun and Hamdi have shown in [PH01] that the DCBI routing algorithm can carry out a better average allocation rate than some other routing algorithms, such as shortest-widest, least hop-count and least delay algorithms. Besides the allocation rate, the link utilization is another important aspect to be investigated under QoS-based routing

schemes, as it is expected that the network resources be more evenly utilized. In this section we will investigate the average link utilization under DDCBI routing. In the following experiments, the delay metrics is always the sum of propagation delay, transmission delay and queuing delay, as defined in the DDCBI algorithm.

In an ideal case, all links covered by paths would be utilized at the same level, and no links would be over-utilized. In this experiment, we randomly generate 100 traffic demands as background traffics to consume the link bandwidth, each with a random source and destination. The required bandwidth of each demand is a random variable uniformly distributed within the range of 64kb/s to 40Mb/s (this values were taken from [AST98]). In terms of delay constraint, we chose a value that ensures that no flow will be dropped. (Figure 5-2 suggests that a delay constraint of 0.35 s is appropriate.) We know from Figure 5-1 that in these conditions, around 93% demands can be allocated. The combination of these parameters makes for what we consider a heavy workload. The link utilization histogram is shown in Figure 5-8.

Figure 5-8 Link utilization histogram for 100 demands with a delay constraint of 0.35 s

The X-axis is the percentage of the different link utilizations and Y-axis is the number of links. The mean value of link utilization is 0.548 and the variance 0.257. From Figure 5-8, we see that 13 links (14.4% of total links) are utilized below 20%, while 19 links (21.1% of total links) are utilized above 80%. The reason for some links being under-utilized or over-utilized is the position of links in the network topology. For example, border links have a lower chance to be used, whereas links located in the middle have more opportunities to be covered by paths. Figure 5-8 does suggest that under heavy workload, the links are not as evenly utilized as expected. More investigation is necessary in order to draw any conclusions. The next section will study further the issue of link utilization in conjunction with other factors, such as rerouting.

5.5 Rerouting effect on end-to-end delay and link utilization

Our DDCBI routing protocol has a re-computation mechanism. Once the current end-to-end delay of a flow exceeds the delay constraint, it implies that the currently used path cannot satisfy the QoS requirements of the flow. The re-computation mechanism is then triggered to find a new path for the flow (a detailed description was given in section 4.4.2). In order to investigate the effect of rerouting on the end-to-end delay of a flow and the link utilization, we designed the following experiment.

We use the same network topology in Figure 4-5, but decrease the capacity of the link bandwidth from 155Mbps (OC3) to 50Mbps (OC1) in order to have simulation experiments that are using less memory and are running faster. 10 traffic demands are randomly generated; each demand has a bandwidth requirement uniformly distributed between 64kbps to 25Mbps, and a delay constraint taking decreasing values (6.0s, 4.2s,

3.55s, 2.8s and so on). The traffic of each flow follows the CBR distribution, the packet size is 240B, and the sending rate is dependent on the bandwidth requirement of each flow (the formula is given in section 5.1.2). In addition, background traffic is added to the network, to produce a high-enough level of congestion that will trigger rerouting of the flows. The background traffic follows an exponential distribution that makes the network environment more realistic. The parameters of packet size, burst time, idle time and sending rate are 500B, 0.5s, 0.1s 1500kps respectively. Table 5-3 contains the average end-to-end delay of each flow under different delay constraints.

FlowID	Delay Constraint								
	6.0s	4.2s	3.55s	2.8s	2.0s	1.4s	1.2s	1.1s	1.0s
8	2.5444367	2.177889*	1.415437*	1.092201*	0.842219*	---	---	---	---
4	0.010106	0.010106	0.010106	0.010106	0.010106	0.010106	0.010106	0.010106	0.010106
10	0.0206214	0.020621	0.020621	0.020621	0.020621	0.020621	0.020621	0.020621	0.020621
9	0.0404276	0.047896	0.059411	0.159665	0.50695	0.347836	0.347836	---	---
5	2.5241584	1.830949*	1.394169*	1.148076*	0.977027*	0.313656*	0.31559*	---	---
1	0.0304444	0.030444	0.030444	0.030444	0.030444	0.078812	0.078812	0.235736*	0.235736*
6	0.0201634	0.020803	0.0552	0.112024	0.348678	---	---	---	---
2	0.0101475	0.02256	0.064202	0.010148	0.524511	0.312305*	0.312305*	---	---
7	0.0101264	0.010592	0.035427	0.07766	0.252588	0.440077*	0.305836*	0.294936*	0.294936*
3	2.5615075	1.738974*	1.373679*	1.472523*	0.990272*	0.711691*	0.711691*	---	---

Table 5-3 Average end-to-end delays with different delay constraints

Note: “*” means rerouting occurred for this flow.

For the first column, the delay constraint of 6 s is so tolerant that no flow’s end-to-end delay exceeds the delay constraint during the period of simulation. It’s a base case of end-to-end delay for flows. Then we introduce smaller and smaller delay constraints to see if rerouting happens and how the end-to-end delay will behave. From chapter 5, we

know that the rerouting can have two consequences: either a new path is found that satisfies the QoS requirement, or no path can be found to meet the delay constraint, and the flow will be dropped. In Table 5-3, “ --- ” means that the flows was dropped. The flow 8, 5 and 3 are more demanding, and so they suffer rerouting more often. In order to show the impact of rerouting on the end-to-end delay more clearly, Figure 5-9 illustrate the average end-to-end delay for the three more demanding flows and the average end-to-end delay of all ten flows in function of the delay constraint.

Figure 5-9 Average end-to-end delays under the different delay constraints

Figure 5-9 shows that DDCBI has the advantage that the average end-to-end delay becomes shorter when delay constraints gets more tight. This is due to the fact that, for stricter delay constraints, the rerouting computation is triggered earlier in order to find a new path that avoids network congestion.

From Table 5-3 it can be observed that cases with more tolerant delay constraints (such as 6s) do not require any rerouting, which makes such cases similar to the static

DCBI algorithm. As the delay constraints get tighter (e.g. from 4.2s to 2s), the corresponding end-to-end delay gets smaller because the network resources are better used due to rerouting; no flows are dropped yet. For even tighter constraints (e.g. 2s and less) the more demanding flows begin to be dropped. In such cases, the improvement in end-to-end delay of the remaining flows comes with a high price that may be unacceptable in practice.

FlowID 8	Path Length	
Delay Constraints	Before	After
6.0s	5	5 (no rerouting)
4.2s	5	4
3.55s	5	4
2.8s	5	4
2.0s	5	4

Table 5-4 Path length Vs Delay constraint for flow 8

FlowID 5	Path Length	
Delay Constraints	Before	After
6.0s	3	3 (no rerouting)
4.2s	3	5
3.55s	3	4
2.8s	3	5
2.0s	3	5

Table 5-5 Path length Vs Delay constraint for flow 5

FlowID 3	Path Length	
Delay Constraints	Before	After
6.0s	5	5 (no rerouting)
4.2s	5	2
3.55s	5	2
2.8s	5	9
2.0s	5	2

Table 5-6 Path length Vs Delay constraint for flow 3

Table 5-4, 5-5 and table 5-6 show the path length before and after rerouting for different delay constraints. By looking at the results in more detail, we realized that in most of the cases, the same new path was selected when an old path was found unsatisfactory and rerouting was triggered. However, since rerouting is triggered earlier for smaller delay bounds, the new path helps to avoid the congestion sooner and yields better end-to-end results. That's why stricter delay constraints lead to shorter end-to-end delays for the same flows. It is clear from this example that the delay constraints must be chosen very carefully. A constraint that is too loose will not trigger rerouting and thus will not help to obtain better QoS. On the other hand, a constraint that is too tight will lead to dropping flows, which means dissatisfied users. More study is necessary in order to understand how to choose delay constraints that are optimal or close to optimal. From the above analysis, we can draw the conclusion that the rerouting mechanism has indeed the potential to achieve better end-to-end delay, and thus better QoS, when properly used.

Next, we will discuss the impact of rerouting on the link utilization. It is expected that the rerouting mechanism is able to prevent certain links from being heavily loaded by spreading the workload more evenly over the network. This not only leads to a better end-to-end response times, but also to more uniform link utilization. Figure 5-10, Figure 5-11 and Figure 5-12 support this expectation.

Figure 5-10 Link utilization histogram for a delay constraint of 6.0s

Mean (E) link utilization $E=0.0968$, variance $V=0.033$, standard deviation $\sigma = 0.182$

Figure 5-11 Link utilization histogram for a delay constraint of 4.2s

Mean link utilization $E=0.09623$, variance $V=0.031$, standard deviation $\sigma = 0.175$

Figure 5-12 Link utilization histogram for a delay constraint of 3.55s

Mean link utilization $E=0.09434$, variance $V=0.02832$, standard deviation $\sigma = 0.1683$

Figure 5-10, 5-11 and 5-12 give the histograms for the link utilization for different delay constraints. From the previous discussion, we know that when the delay constraint is 4.2s or 3.55s, 3 flows out of 10 flows get rerouted to meet the delay constraint, while no flow is rerouted when the delay constraint is 6.0s. It is worth to mention that in all the cases the network workload is the same (i.e., no demand is dropped). Let's compare these three figures. With the similar mean values of link utilization, Figure 5-11 and 5-12 get smaller variances than that of Figure 5-10, which implies the links in Figure 5-11 and 5-12 are used more evenly. Also, the highest utilization drops from 60% in Figure 5-11, to 50% in Figure 5-12, to 40% in Figure 5-13. Thus, we can conclude that rerouting in the DDCBI algorithm can help to obtain a better link utilization.

Another observation from these three figures is that the link utilization becomes more uniform with the decrease of delay constraints. Is it true? Let's look at two more cases before drawing a conclusion.

Figure 5-13 Link utilization histogram for a delay constraint of 2.8s

Mean link utilization $E=0.102$, variance $V=0.0305$, standard deviation $\sigma = 0.1747$

Figure 5-14 Link utilization histogram for a delay constraint of 2.0s

Mean link utilization $E=0.0945$, variance $V=0.0326$, standard deviation $\sigma = 0.181$

Figure 5-13 and 5-14 show that with the further decrease of delay constraints, the link utilization starts to become more uneven. This phenomenon can be explained as the price to pay for the unequal usage of links that results when we need to attain very short delays in order to meet very strict constraints. Let's consider an example with two paths from a to b. Path 1 has a larger bandwidth (30k) and a longer end-to-end delay (16ms). Comparing with path 1, path 2 has a smaller bandwidth (10k) and a shorter end-to-end delay (10ms). Assume that a delay constraint for some flow from a to b is chosen as 18ms

and the bandwidth constraint is 10k. Our DDCBI routing protocol will select path 1, because it has a better allocation rate and link utilization ($10k/30k=33.3\%$ for the links making up path1). However, in the same condition but with a stricter delay constraint of 13ms, only path 2 can satisfy the QoS requirement. It means that the links along path 2 are fully utilized ($10k/10k=100\%$ utilization). The consequence is that other flows can't use path 2 to send traffic. So in certain situations, too tight delay constraints can lead to bad link utilization. Hence, the delay constraints should be chosen carefully. Section 5.2 provides some experimental values for the delay constraints for some levels of traffic load in AT&T IP backbone network.

In this chapter, we discussed the relationship between bandwidth constraints, delay constraints and routing results, showed the effect of queuing delay on the demand allocation rate, and discussed the effect of delay constraints and rerouting on end-to-end delays and average link utilization in the DDCBI routing scheme.

An aspect that was not investigated in the thesis is the rerouting overhead of the DDCBI algorithm. Rerouting has the potential to lead to better QoS, but it comes at the price of increasing the complexity of the network control. Additional operations are necessary, such as the collection and monitoring of dynamic measures, the advertising of these measures to the interested parties, the rerouting computations per se, the increased frequency of routing table changes, etc. All these are matters for future work, together with the question on how to choose an optimal delay constraint.

Chapter 6 SUMMARY AND CONCLUSION

6.1 Summary and conclusions

The main contribution of the thesis is the extensions proposed to the DCBI routing protocol, which lead to a new QoS-based routing algorithm named Dynamic Delay Constrained Blocking Island (DDCBI) algorithm. A simulation model was built and several DDCBI characteristics were studied by simulation. The thesis work can be summarized as follows:

- Bibliographic research on VPN problems and solution, and on QoS-based routing concepts and principles that enables the provisioning of QoS for VPN. This lead to a better understanding of the challenges raised by QoS-based routing, and to the development of the DDCBI algorithm.
- Proposed an extension to the DCBI routing algorithm, which transforms it into DDCBI, a truly QoS-based routing algorithm that takes into account dynamic aspects of the network state, such as queueing delay, in order to ensure the computation of routes that meet end-to-end delay constraints.
- Development of an *ns*-based DDCBI simulation model. As a part of the thesis, we have developed new modules in *ns* to provide QoS-based routing support. These modules can be made available through the web to other users from the *ns* community, as they can be used to model other routing algorithms, as well. This is a side-benefit of the thesis work.

- Performance analysis of the DDCBI algorithm by simulation. The analysis shows that the algorithm has indeed the potential to achieve better end-to-end delay, and thus better QoS for delay-sensitive applications, when applied properly. However, this algorithm comes at a price, as explained below.

Through the thesis work, the following conclusions related to the DDCBI routing can be drawn:

1. DDCBI has the potential to yield better end-to-end response times, and thus better QoS, but only if the delay constraints are carefully chosen. A constraint that is too loose will not trigger rerouting and thus will not help in obtaining better QoS. On the other hand, a constraint that is too tight will lead to dropping flows, which means dissatisfied users. More study is necessary in order to understand how to choose delay constraints that are optimal or close to optimal.
2. The rate of successful allocations drops as expected with the increase in bandwidth requirement of the demands, given that the demands compete for the network resources. The impact of including the queuing delay in the delay metrics is stronger when the network works under a higher load.
3. From our analysis, DDCBI routing is able to distribute the load more evenly over the network and to obtain relatively good link utilization if the delay

constraints are well chosen. Constraints too tight or too low tend to undo this effect.

4. Another price paid by DDCBI routing is the re-routing overhead. More study is necessary to assess this overhead and its effect on the overall performance of the DDCBI algorithm.

6.2 Future work

We believe that the first item for future work is the investigation of how to choose delay constraints that are optimal or close to optimal. In general, the delay constraints depend on the applications, but they can be also used as a control mechanism for triggering rerouting to avoid network congestion.

Another aspect that was not investigated in the thesis is the rerouting overhead of the DDCBI algorithm. Rerouting has the potential to lead to better QoS, but it comes at the price of increasing the complexity of the network control. Additional operations are necessary, such as the collection and monitoring of dynamic measures, the advertising of these measures to the interested parties, the rerouting computations per se, the increased frequency of routing table changes, etc.

Another issue is to integrate QoS-based routing and Best-effort routing. For compatibility, these two routing schemes must be able to coexist. The main problem is how to allocate the network resources. QoS-based routing usually has a higher priority. There must be an overall control mechanism to fairly distribute the resources between DDCBI routing and other best-effort routing schemes.

References

- [AMAO99] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell,
"Requirements for Traffic Engineering Over MPLS", RFC2702
- [AST98] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Quality of
service based routing: a performance perspective".
In Proceedings of ACM SIGCOMM'98 Conference, September 1998.
<http://citeseer.nj.nec.com/apostolopoulos98quality.html>
- [BAN01] Tamrat Bayle, Reiji Aibara, Kouji Nishimura, "Performance
Measurements of MPLS Traffic Engineering and QoS", Hiroshima
University Japan, <http://www.isoc.org/isoc/conferences/inet/01>
- [BCNN01] Jerry Banks, John S. Carson, Barry L. Nelson, David M. Nicol, "Discrete-
Event System Simulation", Upper Saddle River, New Jersey, 2001.
- [Black99] Uyless Black, "Advanced Internet Technologies" Prentice Hall PTR,
Upper Saddle River, NJ 07458, 1999.
- [Chen99] Shigang Chen, "Routing Support for Providing Guaranteed End-to-End
Quality-of-Service", Ph.D. thesis, 207 pages, UIUC, May 1999,
<http://cairo.cs.uiuc.edu/papers/SCthesis.ps>
- [Cisco] Cisco IOS Release 12.0(7)T "MPLS Traffic Engineering",
http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120t/120t7/te120_7t.htm

- [CK 98] Chen, S., Nahrstedt, K.: "An Overview of Quality of Service Routing for Next-Generation High-Speed Networks: Problems and Solutions." IEEE Networks, Vol. 12, No. 6 (1998) 64-79
- [DC90] S. Deering and D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs" ACM Transactions on Computer Systems, pages 85-111, May 1990.
- [FF00] C. Frei, B. Faltings, "Planning Bandwidth Allocation in Communication Networks" Special Issue on Multimedia Network and Service Control, ICON Journal, Baltzer Science Publisher, 2000.
- [Frei00] C. Frei, "Abstraction Techniques For Resource Allocation in Communication Networks", PhD Thesis, Swiss Federal institute of Technology, Lausanne (EPFL), 2000.
- [FV00] Fall, Kevin and Kannan Varadhan, Eds, "The *ns* Manual (Notes and Documentation)", 2000, <http://www.isi.edu/nsnam/ns/doc/index.html>
- [KA98] S. Kent, R. Atkinson, 1998, "Security Architecture for the Internet Protocol", RFC 2401.
- [Kos98] Dave Kosiur, "Building and Managing Virtual Private Networks", John Wiley & Sons, New York, 1998.
- [McD00] McDonald, Christopher, "Feature Virtual Private Networks, An Overview", Intranet Journal, 2000, <http://idm.internet.com/foundation/vpn-2.shtml>
- [MicrobCen] "Networking 101: Building Blocks and Essentials", Networking Center at Microsoft bCentral, <http://www.bcentral.com/articles/networking/102.asp>

[MicroTechinfo] "Virtual Private Networking Overview", White Paper, Sept. 2001,
<http://www.microsoft.com/windows2000/techinfo/howitworks/communications/remotaccess/vpnoverview.asp>

[OpenBSD] "Using IPsec (IP Security Protocol)", OpenBSD on-line tutorial,
<http://www.openbsd.org/faq/faq13.html> - what

[PD96] Larry L. Peterson, Bruce S. Davie, "Computer Networks" Morgan Kaufmann Publisher Inc., San Francisco, CA, 1996.

[PH01] Konghong Pun, Mounir Hamdi, "Provision of Quality of Service for Virtual Private Networks", Proceedings of SPIE Vol.4211 (2001)

[Pha00] Phaltankar, M. Kaustubh, 2000, "Practical Guide for Implementing Secure Intranets and Extranets", Artech House Inc. Norwood.

[Pri98] "Private Use of Public Networks for Service Provider"
<http://www.3com.com/>

[QOSF2] "Quality of Service - Glossary of Terms", May 1999, 26 pages,
<http://www.qosforum.com/white-papers/qos-glossary-v4.pdf>

[RFC2386] E. Crawley, R. Nair, B. Rajagopalan, H. Sandick, "A Framework for QoS-based Routing in the Internet", August 1998

[Semeria00] Chuck Semeria, "Traffic Engineering for the New Public Network", White Paper, Juniper Networks Inc., 2000,
<http://www.juniper.net/techcenter/techpapers/200004.html>

[Sun00] Wei Sun, "QOS/Policy/Constraint Based Routing"
http://www.cis.ohio-state.edu/~jain/cis78899/qos_routing/index.html

- [SWE99] Scott, Charlie, Paul Wolfe and Mike Erwin, 1999, *Virtual Private Networks (second edition)*, O'REILLY, Sebastopol, CA
- [SZ02] Roberto Sabella, Hui Zang, CFP: IEEE Network Special Issue on Traffic Engineering in Optical Networks, IEEE Network, 2002
- [VPN98] VPNet Technologies Inc., 1998, "What's a VPN Anyway? A Virtual PrivateNetworking Primer" <http://www.vpnet.com/>
- [WC96] Zheng Wang and Jon Crowcrafft, "Quality of Service Sensitive Multicast Internet Protocol", IEEE JSAC, 14(7):pp. 1288-1234, Sep. 1996
- [Webserver] VPN Architecture, http://www.webserver.com.my/pdf/vpn_final.pdf
- [West99] "QVPN™ - VPN Services with Built-in QoS", Westcon company,
<http://www.westcon.com/prodinfo/vertical/verticalesp/xedia/appnotes/qvpn>
- [White97] Paul P. White, "RSVP and Integrated Services in the Internet: A Tutorial", 0163-6804/97 IEEE
- [Xchange98] Peter Mead, "Lighting up the Local Loop", posted: 08/1998
<http://www.x-changemag.com/articles/881cove2.html>
- [XN99] Xipeng Xiao and Lionel M. Ni, "Internet QoS: the Big Picture", IEEE Network Magazine, pp.8-18, March/April, 1999
- [ZK00] Peng Zhang, Raimo Kantola, 2000, "Building MPLS VPNs with QoS Routing Capability", Laboratory of Telecommunication Technology, Helsinki University of Technology