Routing Algorithms for Differentiated Services in a Heterogeneous Network Environment

by

WAN LIN

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba, Canada

Thesis Advisor: K. Ferens, Ph.D., P.Eng.

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Routing Algorithms for Differentiated Services in a Heterogeneous Network Environment

BY

Wan Lin

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

MASTER OF SCIENCE

Abstract

Differentiated Services (DiffServ) has been proposed for the next generation network. DiffServ allows the users to pay for and receive differentiated services. We realize that it is impractical to implement DiffServ in all nodes at the same time. Thus, it is reasonable to assume that the network will consist of both DiffServ capable nodes and DiffServ incapable nodes for quite some time. For DiffServ routing, DiffServ capable links are preferred. But a path consisting of only DiffServ capable nodes may not exist in the network. Even if such a path exists, it may not be the "optimal" path.

In this thesis, we give routing algorithms for a DiffServ capable user to establish a route in such a network. One of the routing metrics we consider is DiffServ capability. The other metric is cost. Three routing algorithms are given. In the first algorithm, cost is given higher precedence over DiffServ capability. In the second algorithm, DiffServ capability is given higher precedence over cost. The first two algorithms consider the metrics as independent metrics. In the third algorithm, we assign a DiffServ route selection order number to each link in the network topology. The algorithm computes the optimal path using the assigned DiffServ route selection order numbers.

Simulations in ns show that the third algorithm is more powerful than the first two algorithms. Through simulations, we have proved that the algorithm calculates the correct routing paths in all network topologies.

Acknowledgements

I would like to express my sincerest gratitude to my advisor Prof. K. Ferens for the many suggestions, helpful discussions and comments.

I would also like to say thank you to the other members of my thesis examining committee, Prof. A. S. Alfa and Prof. Y. Zhang. They have spent lots of their valuable time reading my thesis.

Table of Contents

Chapter 1

Introduction

1.1 Need for Quality of Service

The Internet has undergone a tremendous growth. In addition to the traditional applications such as telnet, ftp and gopher, many new applications have emerged in the last decade. WWW is now the primary application for the Internet. Multimedia applications such as Video-on-Demand, Video-Conferencing and IP Telephony applications have also appeared. These applications require various degrees of guarantees regarding bandwidth, delay jitter, latency and loss rate. Network traffic has increased as the number of users and applications have increased. The question is whether increasing bandwidth – the data carrying capacity of the network – is sufficient to accommodate these increased demands. The answer is no, it is not. The nature of traffic through the Internet has changed.

In the traditional IP (Internet Protocol) world, packet delivery is on

a best-effort basis and no guarantees are given with respect to the packet stream's characteristics. A voice stream is more sensitive to packet delays than to packet losses. If a packet experiences a delay of more than 400 milliseconds, the voice becomes unintelligible. But it can typically tolerate packet losses of up to 20%. An application with contrasting demands is the standard ftp. ftp sessions are more concerned about throughput than delays. Many critical applications such as corporate data access applications need reliable and secure network support. So, as we can see, the Internet is filled with traffic from applications with contrasting requirements and the current best-effort style service is no longer sufficient.

Quality of Service (QoS) is the ability of a network element (e.g. an application, host or router) to have some level of assurance that its traffic and service requirements can be satisfied. There are essentially two types of QoS available [71]:

- Resource Reservation: On an application's QoS request, and subject to bandwidth management policy, network resources are reserved for the application's exclusive use. RSVP and Integrated Services (IntServ) provides this service.

- Prioritization: Network traffic is classified and apportioned network resources according to network management policy criteria. An application may request special treatment of its traffic subjecting to network management policy. Differentiated Services (DiffServ) provides this service.

These QoS protocols and algorithms are not competitive or mutually

exclusive, but on the contrary, they are complementary. As a result, they are designed for use in combination to accommodate the varying operational requirements in different network contexts.

These types of QoS can be applied to individual application flows or to flow aggregates [72]:

- Per Flow: A flow is defined as an individual, uni-directional, data stream between a sender and a receiver. A flow is uniquely identified by the flow's transport protocol, source address, source port number, destination address, and destination port number.

- Per Aggregate: An aggregate is two or more flows that have something in common. Usually they have the same source address, destination address and transport protocol. But they may be aggregated in other ways.

To accommodate the need for these different types of QoS, there are a number of different QoS protocols and algorithms:

- ReSerVation Protocol (RSVP): Provides the signaling to enable network resource reservation (also known as Integrated Services or IntServ).

- Differentiated Services (DiffServ): Provides a coarse and simple way to categorize and prioritize network traffic (flow) aggregates.

- Multi Protocol Labelling Switching (MPLS): Provides bandwidth management for aggregates via network routing control according to labels in (encapsulating) packet headers.

- Subnet Bandwidth Management (SBM): Enables categorization and prioritization at Layer 2 (the data-link layer in the OSI model) on shared and switched IEEE 802 networks.

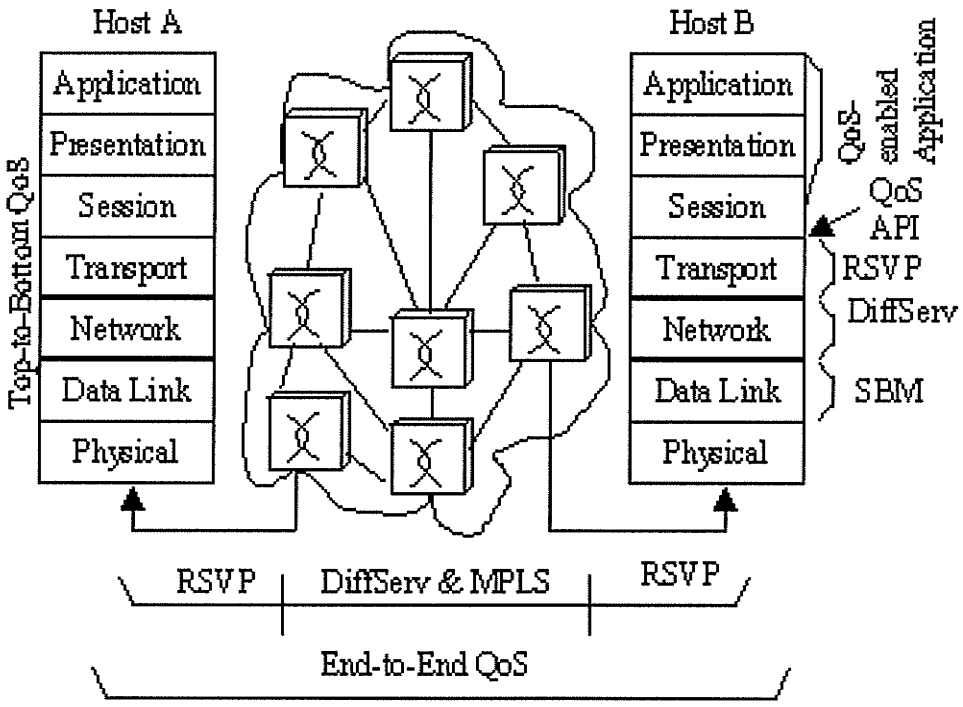


Figure 1.1: Relations among the QoS models

1.2 Integrated Services

Integrated Services (IntServ) is an Internet service model that includes both traditional best-effort service and real-time services. IntServ research

started in the early 1990s, and has generated much interest and discussions by 1994. The problem IntServ research work addresses is how to provide network support for real-time applications as well as for non-real-time applications. The researchers make the observation that many emerging real-time applications – multimedia teleconferencing, remote video, computer-based telephony, remote visualization, etc – will have very different Quality of Service (QoS) requirements than the traditional text-based non-real-time applications like ftp or telnet. IntServ researchers realized that they were designing a service model that is based on conjectures about future applications, institutional requirements, and technical feasibility.

The Integrated Service model is an extension to the original Internet best-effort architecture, and it includes two services targeted towards real-time applications: guaranteed service and predicted service. Guaranteed service involves pre-computed worse-case delay bounds and predicted service uses the measured performance of the network in computing delay bounds [58].

Based on the above considerations, the researchers believe that the Integrated Service model would 1) keep additional flow state in routers, and 2) require an explicit setup mechanism to install and eliminate flow state in routers. The proposed solution, then, is to have end hosts initiate a quality of service (QoS) request prior to the transmission of traffic to networks. Such request will be carried by a reservation protocol called RSVP. If such a request is accepted by networks, then the network will create per-flow state to guarantee such a QoS request during the transmission. If the network does not have enough resources, then the QoS request is denied.

The ReSerVation Protocol (RSVP) is a signaling protocol that provides

reservation setup and control to enable the Integrated Services [IntServ]. RSVP is the most complex of all the QoS technologies, for applications and for network elements. The protocol works as illustrated below:

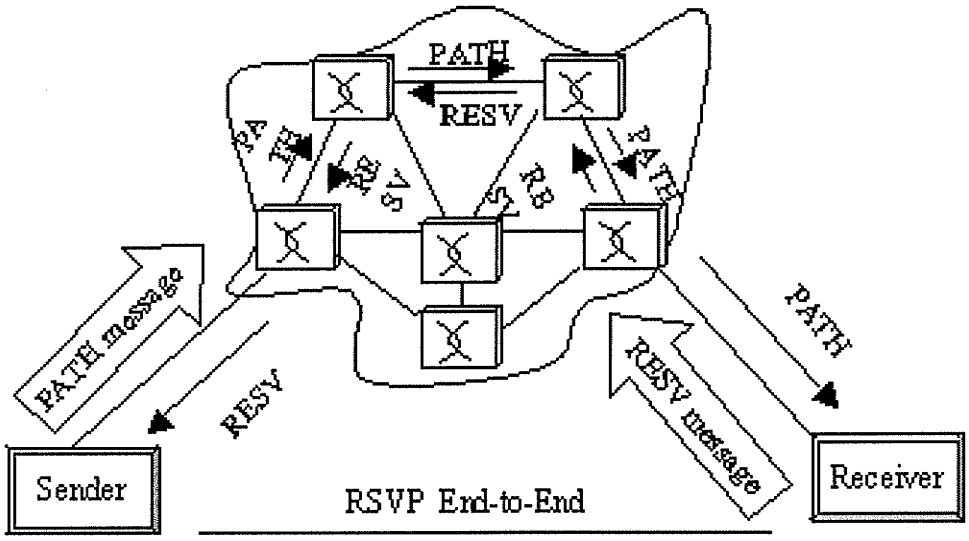


Figure 1.2: RSVP

- Senders characterize outgoing traffic in terms of the upper and lower bounds of bandwidth, delay, and jitter. RSVP sends a PATH message from the sender that contains this traffic specification (TSpec) information to the destination address. Each RSVP-enabled router along the downstream route establishes a path-state that includes the previous source address of the PATH message.

- To make a resource reservation, receivers send a RESV (reservation

request) message upstream. In addition to the TSpec, the RESV message includes a request specification (Rspec) that indicates the type of Integrated Services required – either Controlled Load or Guaranteed – and a filter specification (filter spec) that characterizes the packets for which the reservation is being made. Together, the RSpec and filter spec represent a flow-descriptor that routers use to identify each reservation.

- When each RSVP router along the upstream path receives the RESV message, it uses the admission control process to authenticate the request and allocate the necessary resources. If the request cannot be satisfied (due to lack of resources or authorization failure), the router returns an error back to the receiver. If accepted, the router sends the RESV upstream to the next router.

- When the last router receives the RESV and accepts the request, it sends a confirmation message back to the receiver.

- There is an explicit tear-down process for a reservation when the sender or receiver ends an RSVP session.

Integrated Service effort has done a very good job in 1) analyzing the requirements of real-time applications and 2) arguing for the efficiency of an integrated network offering different kind of services instead of disjoint networks each with a distinct service model. However, the implementation framework IntServ has raised serious concerns. Is it feasible to implement and maintain per-flow state in routers, especially in those which handle a lot of aggregated traffic? The RSVP protocol is complex, and it may be

unrealistic to expect routers to devote much resources to interpreting RSVP requests and handling admission control for established state. Last, there is the partial deployment problem for RSVP itself. If there are routers on a path which do not understand RSVP and can't make reservations, then it would be impossible to make any guarantees on end-to-end delay bounds.

1.3 Differentiated Services

In the Differentiated Services architecture (DiffServ), packets are classified into different classes and given differentiated services. Users can choose from the service level best suited for their applications. They will subscribe to and pay for Service Level Agreements (SLAs) from their Internet Service Providers (ISPs). An SLA specifies the expected service a user will receive. The DiffServ architecture augments the current best-effort Internet architecture. It consists of mechanisms to be implemented in existing Internet devices–network routers and end hosts–but pushes the complexity of the system towards the edge of the network, which makes it more scalable. A variety of services can be constructed using the simple primitives provided by the DiffServ architecture. DiffServ is able to offer very flexible services to users with different requirements.

One reason for IP's tremendous success is its simplicity. Differentiated Services (DiffServ) provides a simple and coarse method of classifying services of various applications. DiffServ is kind of combination of currently used best-effort service model and IntServ.

Currently, there are two standard per hop behaviors (PHBs) defined to

represent two service levels (traffic classes):

- Expedited Forwarding (EF) [DiffServ EF]: Has a single codepoint (DiffServ value). EF minimizes delay and jitter and provides the highest level of aggregate quality of service. Any traffic that exceeds the traffic profile (which is defined by local policy) is discarded.

- Assured Forwarding (AF) [DiffServ AF]: Has four classes and three drop-precedences within each class (so a total of twelve codepoints). Excess AF traffic is not delivered with as high probability as the traffic "within profile," which means it may be demoted but not necessarily dropped.

DiffServ assumes the existence of a service level agreement (SLA) between networks that share a border. The SLA establishes the policy criteria, and defines the traffic profile. It is expected that traffic will be policed and smoothed at egress points according to the SLA, and any traffic out of profile (i.e. above the upper-bounds of bandwidth usage stated in the SLA) at an ingress point have no guarantees (or may incur extra costs, according to the SLA). The policy criteria used can include time of day, source and destination addresses, transport, and/or port numbers (i.e. application Ids). Basically, any context or traffic content (including headers or data) can be used to apply policy.

As illustrated in Figure 1.3, PHBs are applied by the conditioner to traffic at a network ingress point (network border entry) according to predetermined policy criteria. The traffic may be marked at this point, and routed according to the marking, then unmarked at the network egress (net-

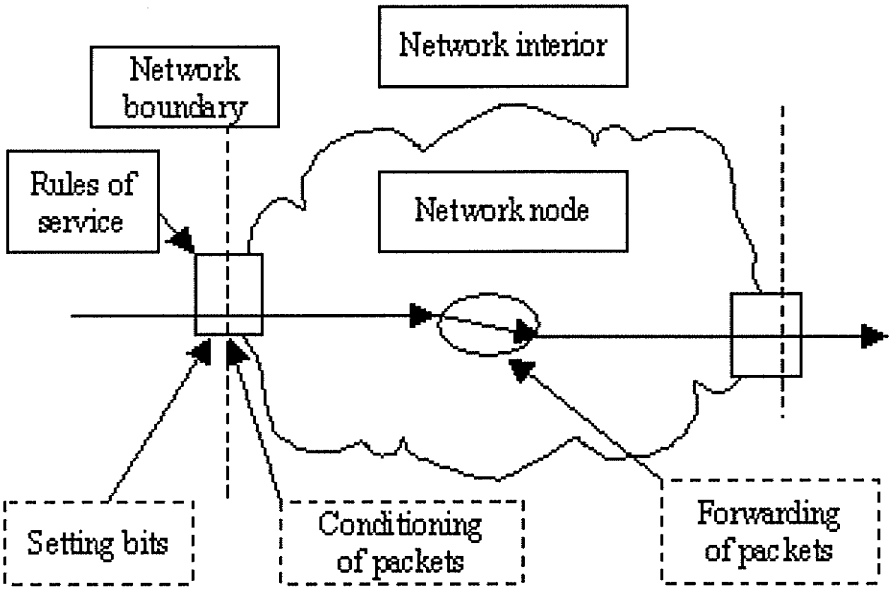


Figure 1.3: Differentiated Services

work border exit). Originating hosts can also apply the DiffServ marking. A detailed description of the architecture of Differentiated Services is given in Chapter 2.

1.4 Multi-Protocol Label Switching

Multi-Protocol Label Switching (MPLS) is similar to DiffServ in some respects, as it also marks traffic at ingress boundaries in a network, and un-marks at egress points. But unlike DiffServ, which uses the marking to determine priority within a router, MPLS markings (20-bit labels) are primarily designed to determine the next router hop. MPLS is not application

controlled (no MPLS APIs exist), nor does it have an end-host protocol component. Unlike any of the other QoS protocols we describe in this thesis, MPLS resides only on routers. And MPLS is protocol-independent (i.e., multi-protocol), so it can be used with network protocols other than IP (like IPX, ATM, PPP or Frame-Relay) or directly over data-link layer as well [MPLS Framework, MPLS Architecture].

MPLS is more of a traffic engineering protocol than a QoS protocol, per se. MPLS routing is used to establish fixed bandwidth pipes analogous to ATM or Frame Relay virtual circuits. The difference is arguable since the end-result is service improvement and increased service diversity with more flexible, policy-based network management control, all of which the other QoS protocols also provide. MPLS simplifies the routing process (decreases overhead to increase performance) while it also increases flexibility with a layer of indirection. The following is a sketch of the process used by MPLS-enabled routers called a Label Switching Router (LSR):

- At the first hop router in the MPLS network, the router makes a forwarding decision based on the destination address (or any other information in the header, as determined by local policy) then determines the appropriate label value – which identifies the Forwarding Equivalence Class (FEC) – attaches the label to the packet and forwards it to the next hop.

- At the next hop, the router uses the label value as an index into a table that specifies the next hop and a new label. The LSR attaches the new label, then forwards the packet to the next hop.

1.5 Need for Routing Algorithms

Routing is essential for data transmission in the network. Standard routing algorithms are typically single objective optimizations, that is, they may minimize the hop-count, or maximize the path bandwidth, but not both. These path computation algorithms are widely used in the current best-effort service network.

For QoS support, we may need to consider two or more metrics. It has been proved that the problem of finding the best path subject to multiple constraints is an NP-hard problem [43]. Thus the QoS routing problem is regarded as not-feasible and, hence, unattractive, though desirable in telecommunications networks. However, many QoS routing algorithms have been developed for Integrated Services. A few papers have addressed the routing problems for Differentiated Services. A review of the existing routing algorithms is given in Chapter 3.

In this thesis, we raise the problem of how to find optimal routing in a network consisting of DiffServ capable nodes and DiffServ incapable nodes (see Definition 4.3.1). We will give routing algorithms for Differentiated Services in such a network. Our algorithms have very little impact on the currently used routing algorithms and protocols and thus are practical.

1.6 Organization of This Thesis

In Chapter 1, we explain why we need Quality of Service (QoS) and give an introduction to the common Quality of Service models.

In Chapter 2, we give a more detailed description of the Differentiated

Services (DiffServ) Architecture. There are currently two Per-Hop Behavior groups defined for DiffServ. The first is Assured Forwarding (AF) PHB. Four AF classes, each with three drop precedence values, are defined and required in a DiffServ implementation. The second is Expedited Forwarding (EF) PHB. EF PHB is not required to be implemented in a DiffServ implementation.

In Chapter 3, we give a review of the existing QoS-based routing algorithms. Theorem 3.1.1, Theorem 3.1.2 and Theorem 3.1.3 show that quality of service routing problems are very hard.

In Chapter 4, we raise the problem of how to find optimal routing schemes in a network consisting of DiffServ capable nodes and DiffServ incapable nodes (see Definition 4.3.1). We develop three routing algorithms for Differentiated Services in such a network. Our algorithms have very little impact on the currently used routing algorithms and protocols.

In Chapter 5, we give a brief introduction to the Network Simulator (ns) that has been developed at the Lawrence Berkeley National Laboratory (LBNL) of the University of California, Berkeley (UCB). Then we show how our new routing algorithm developed in Chapter 4 is implemented and added to ns. Network topologies are then created for simulations in ns. The simulations show that our algorithm is correct and easy to implement. Simulation results are presented in this chapter.

In Chapter 6, we conclude the thesis with a brief summary of our research and raise some issues for future work.

Chapter 2

Differentiated Services Architecture

In the Differentiated Services (DiffServ or DS) architecture [67, 68] , packets are marked into a small number of classes at the edge of the network to be treated differentially according to the Service Level Agreements (SLAs) specified. Users can choose the service level best suited for their applications. They subscribe to and pay for SLAs from their Internet Service Providers (ISPs). An SLA specifies a profile of what a customer's traffic will look like. The customer pays for the SLA under the condition that the ISP delivers the specified forwarding service to traffic within the profile.

DiffServ takes the approach to combine the best of both worlds: a high degree of aggregation from the best-effort model and QoS assurance from the IntServ model. The edge routers keep per-flow state, and monitor and mark traffic using traffic conditioners. The interior routers are still stateless, as in

the current Internet. This approach is more scalable than that of IntServ.

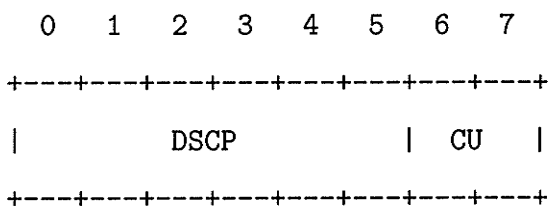
Like IntServ, DiffServ takes advantage of the observation that many real-time applications are adaptive and do not require stringent network guarantees. However, it differs from IntServ in a number of ways. Instead of trying to support a very fine level of QoS specifications in network itself, DiffServ only supports that level of specifications at the edge of the network, where it maps these QoS specifications to a few classes of packets. In the interior of the network, DiffServ treats different classes of packets differently. This design simplifies the design of the interior network and pushes the complexity to the edge of the network.

2.1 The DiffServ Field of an IP Packet

The research work on DiffServ started in late 1997. By early 1998, a working group for Differentiated Services (DiffServ WG) was chartered in the Internet Engineering Task Force (IETF) and active standardization process was in progress. The DiffServ WG attempts to standardize the use of the DiffServ field in both IPv4 and IPv6 packet headers.

The Differentiated Services Field (or DiffServ Field) [67] is the IPv4 header TOS octet [52] or the IPv6 Traffic Class octet [61]. The DiffServ field contains a 6 bit codepoint called the DiffServ codepoint (DSCP). The DSCP field determines the kind of per-hop behavior (PHB) that the flow should receive. All codepoints must be mapped to some PHB and codepoints that are not mapped to a standardized PHB should be mapped to a Default PHB. The PHBs define the forwarding behavior for the flow at that

particular router. The two-bit currently unused (CU) field is reserved. The value of the CU bits are ignored by differentiated services-compliant nodes when determining the per-hop behavior to apply to a received packet. The DiffServ field structure is presented below:



DSCP: differentiated services codepoint

CU: currently unused

2.2 Per-Hop Behavior

A per-hop behavior (PHB) is a description of the externally observable forwarding behavior of a DiffServ node applied to a particular DiffServ behavior aggregate. The PHB is the means by which a node allocates resources to behavior aggregates. The observable behavior of a PHB may depend on certain constraints on the traffic characteristics of the associated behavior aggregate or the characteristics of other BAs. PHBs may be specified in terms of their resource priority relative to other PHBs, or in terms of their relative observable traffic characteristics.

Currently, two types of PHBs for DiffServ have been standardized by the Internet Engineering Task Force (IETF): the Assured Forwarding (AF) PHB

group [69] and the Expedited Forwarding (EF) PHB group [70]. Each PHB group is allocated three bits of the DiffServ field.

2.2.1 Assured Forwarding PHB

Assured Forwarding (AF) PHB was proposed in [69]. In AF PHB, the edge devices of the network monitor and mark incoming packets of either individual or aggregated flows. A packet of a flow is marked IN (in-profile) if the temporal sending rate at the arrival time of the packet is within the contract profile of the flow. Otherwise, the packet is marked OUT (out-of-profile). The temporal sending rate of a flow is measured using TSM (Time Sliding Window) or a token bucket controller. The core routers in the network provide RIO (RED with IN/OUT) drop policy.

AF PHB is a means for a DiffServ domain to offer different levels of forwarding assurances for IP packets received from a customer of a DiffServ domain. Four AF classes are defined, where each AF class in each DiffServ node is allocated a certain amount of forwarding resources. IP packets that wish to use the services provided by the AF PHB group are assigned by the customer or the provider of a DiffServ domain into one or more of these AF classes according to the services that the customer has subscribed to.

Within each AF class, IP packets are marked with one of three possible drop precedence values. In case of congestion, the drop precedence of a packet determines the relative importance of the packet within the AF class. A congested DiffServ node tries to protect packets with a lower drop precedence value from being lost by preferably discarding packets with a higher drop precedence value.

In a DiffServ node, the level of forwarding assurance of an IP packet thus depends on (1) how much forwarding resources has been allocated to the AF class that the packet belongs to, (2) what the current load of the AF class is, and in case of congestion within the class, (3) what the drop precedence of the packet is.

AF PHB must be implemented in a DiffServ implementation.

2.2.2 Expedited Forwarding PHB

Expedited Forwarding (EF) PHB was proposed in [70] as a premium service for the DiffServ network.

In contrast to AF PHB, the EF PHB group is designed to build a low loss rate, low latency, low jitter, assured throughput, end-to-end service through a DiffServ domain. Such a service appears to the endpoints like a point-to-point connection or a "virtual leased line". The departure rate of the EF traffic should equal or exceed a configurable rate independent of the intensity of any other traffic. The departure rate is measured over any time interval equal to or longer than a packet time. Several types of queue scheduling schemes (e.g., a priority queue, a single queue with a weighted round robin scheduler, and class-based queue) may be used to implement the EF PHB.

EF PHB is not required to be implemented in a DiffServ implementation. It is just an option.

2.3 In Packets and Out Packets

At the edge of the network, the ISP's edge routers classify packets and map traffic to their respective SLAs. Traffic sent within the profiles in SLAs are marked by edge routers into different classes of packets. Traffic sent outside the profiles in SLAs are left unmarked by the edge routers and is considered opportunistic traffic. There are two types of packets: IN (in-profile) packets and OUT (out-of-profile) packets. IN packets represent packets within a profile and OUT packets represent packets beyond a profile. The edge routers mark packets as IN packets if the traffic is within a customer's SLA. Anything in excess of the SLA is tagged as OUT packets.

Different conditioning actions may be applied to the in-profile packets and out-of-profile packets. In-profile packets may either be allowed to enter the DiffServ domain without further conditioning or may have their DiffServ codepoint changed. The latter happens when the DiffServ codepoint is set to a non-Default value for the first time, or when the packets enter a DiffServ domain that uses a different PHB group or codepoint to PHB mapping policy for this traffic stream. The out-of-profile packets are either dropped or marked with a different PHB by the egress router. The ingress router classifies traffic into aggregates based on DSCP. This is then policed according to the aggregate profiles. A core router may introduce some burstiness into an in-profile traffic because of queuing or increased aggregation. So, the egress may have to shape the traffic so that the downstream clouds do not police this traffic unfairly.

Inside the network, core routers only need to distinguish between two types of packets and give IN packets preferential treatment in terms of band-

width or delay, or both. In DiffServ, only edge routers need to keep per-flow state and the core routers keep no per-flow state. This way, the complexity of the system is pushed to the edge of a network. Putting per-flow state in only edge routers makes DiffServ architecture more scalable than IntServ. Additionally, SLAs serve as a basis for ISPs to account for the network resource usage.

2.4 Traffic Classification and Conditioning

Differentiated Services are extended across a DiffServ domain boundary by establishing a SLA between an upstream network and a downstream DiffServ domain. The SLA may specify packet classification and re-marking rules and may also specify traffic profiles and actions to traffic streams which are in- or out-of-profile. The Traffic Conditioning Agreement (TCA) between the domains is derived (explicitly or implicitly) from this SLA.

The packet classification policy identifies the subset of traffic which may receive a differentiated service by being conditioned and/or mapped to one or more behavior aggregates (by DiffServ codepoint re-marking) within the DiffServ domain.

Traffic conditioning performs metering, shaping, policing and/or re-marking to ensure that the traffic entering the DiffServ domain conforms to the rules specified in the TCA, in accordance with the domain's service provisioning policy. The extent of traffic conditioning required is dependent on the specifics of the service offering, and may range from simple codepoint re-marking to complex policing and shaping operations. When packets exit the

traffic conditioner of a DiffServ boundary node the DiffServ codepoint of each packet must be set to an appropriate value.

Traffic meters measure the temporal properties of the stream of packets selected by a classifier against a traffic profile specified in a TCA. A meter passes state information to other conditioning functions to trigger a particular action for each packet that is either in- or out-of-profile (to some extent).

2.5 Packet Markers

Packet markers set the DiffServ field of a packet to a particular codepoint and adds the marked packet to a particular DiffServ behavior aggregate. The marker may be configured to mark all packets, that are steered to it, to a single codepoint, or may be configured to mark a packet to one of a set of codepoints used to select a PHB in a PHB group, according to the state of a meter. When the marker changes the codepoint in a packet, it is said to have "re-marked" the packet.

2.6 DiffServ Domain

A DiffServ (or DS) domain consists of a set of DiffServ capable nodes connected together and governed by a common policy for provisioning services. Each DiffServ domain has an ingress DiffServ node that provides fine-grained classification. This node selects a PHB to be associated with the flow and conditions the flow if necessary. A DiffServ domain consists of DiffServ boundary nodes and DiffServ interior nodes. DiffServ boundary

nodes interconnect the DiffServ domain to other DiffServ capable or DiffServ incapable domains, while DiffServ interior nodes only connect to other DiffServ interior or boundary nodes within the same DiffServ domain. Both DiffServ boundary nodes and interior nodes must be able to apply the appropriate PHB to packets based on the DiffServ codepoint. In addition, DiffServ boundary nodes may be required to perform traffic conditioning functions as defined by a Traffic Conditioning Agreement between their DiffServ domain and the peering domain that they connect to.

Traffic enters a DiffServ domain at a DiffServ ingress node and leaves a DiffServ domain at a DiffServ egress node. DiffServ boundary nodes act both as a DiffServ ingress node and a DiffServ egress node for different directions of traffic. A DiffServ ingress node is responsible for ensuring that the traffic entering the DiffServ domain conforms to any TCA between it and the other domain to which the ingress node is connected. A DiffServ egress node may perform traffic conditioning functions on traffic forwarded to a directly connected peering domain, depending on the details of the TCA between the two domains.

2.7 DiffServ Router Components

The basic components of a DiffServ router are classifiers, traffic conditioners, meters, buffer managers, shapers and link schedulers.

2.7.1 Classifiers

A classifier is a module that selects packets based on certain fields of the packet header and according to certain rules. The packets are identified to belong to some flow defined by a flow ID. They are then passed on to a traffic conditioner defined for that particular flow. There are two types of packet classifiers defined in the DiffServ architecture – behavioral aggregate classifier and multi-field classifier.

A behavioral aggregate (BA) classifier selects or classifies packets based on the DSCP field only. If different customers use the same interface to send packets, the BA classifier will not be able to distinguish the packets among the users. In other words, it is impossible to enforce the TCAs on a per-customer basis using a BA classifier. Multi-Field classifiers should be used in such cases.

A Multi-Field (MF) classifier selects packets based on combinations of various fields of the IP packet header. These fields may be the destination address, source address, DSCP, protocol ID, source port number and the destination port number.

2.7.2 Monitor/Meter

A monitoring interface enables collection of statistics regarding traffic carried at various DiffServ service levels. These statistics are important for accounting purposes and for tracking compliance to service level agreements negotiated with customers. Specifically, counter information on how many packets were transferred in-profile vs. out-of-profile would be useful on a

customer-by-customer basis.

Boundary routers use classifiers to identify classes of traffic submitted for transmission through the DiffServ network. Once traffic is classified at the input to the router, traffic from each class is typically passed to a meter. The meter is used to measure the rate at which traffic is being submitted for each class. This rate is then compared against a traffic profile, which is part of the TCA. Based on the results of the comparison, the meter deems particular packets to be conforming to the profile or non-conforming. Appropriate policing actions are then applied to out-of-profile packets.

2.7.3 Traffic Conditioner

A traffic conditioner (TC) is used to make sure the traffic conforms to the negotiated profile. The traffic conditioner may be a marker, dropper or shaper. A traffic profile represents the temporal properties of the traffic stream. The traffic conditioner uses a meter and determines whether the packet is in-profile or out-of-profile. It may decide to drop or re-queue an out of profile packet and normally queues an in-profile packet. The condition-action parameters are decided by the service provider and may be different at each DiffServ router.

2.7.4 Dropper

A dropper discards some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. This is the simplest form of policing that may be supported by ingress routers.

2.7.5 Shaper

A shaper delays some or all of the packets in a traffic stream passing through the router in order to bring the stream into compliance with a traffic profile. A shaper usually has a finite-size buffer, and packets may be discarded if there is not sufficient buffer space to hold the delayed packets. Boundary routers are not required to provide shaping functionality, but may do so for the following reasons: (1) Ingress routers may use shaping as a form of policing – when submitted traffic is deemed non-conforming, it must be policed to protect the DiffServ network. One form of policing is to delay submitted traffic in a shaper until it conforms to the profile specified in the TCA. This is usually referred to as policer shaping. (2) Egress routers may shape behavior aggregate traffic before it is submitted to a subsequent provider's network. This preventative measure avoids policing action in the subsequent network. This is usually referred to as egress shaping.

2.7.6 Buffer Manager

The main function of a buffer manager is to manage the queues. The two main aspects to buffer management are: queue selection and congestion control.

Queue Selection deals with the selection of a queue for the packet. The strategies adopted for this include Class Based Queueing (CBQ) and Microflow Based Queueing (MBQ). The main goal of CBQ is to aggregate traffic into classes. Use of this class-hierarchy enables link sharing too. When the classification is more fine-grained and thus we have more classes, it becomes

a microflow-based queueing.

Congestion Control deals with the issue of controlling congestion. An active queue management strategy helps in controlling congestion. The buffer manager controls the length of the queues by dropping packets at appropriate times. Queues are an integral part of any network as they are required to handle traffic bursts. But long queues can increase the delay in a network. Effective queue management strategies are important.

2.7.7 Link Scheduler

A scheduler is an element which gates the departure of each packet that arrives at one of its inputs, based on a scheduling algorithm. It has one or more input and exactly one output. Every input has an upstream element to which it is connected, and a set of parameters that affects the scheduling of packets received at that input.

Generally, schedulers can be broken into two different classes. The first is known as work conserving. The other class of schedulers are known as non-work conserving.

2.8 Summary

In the DiffServ domain, customers may contract for a certain level of service with the service provider. This contract can be made for individual or aggregated flows. The routers at the edge of the network, called boundary routers or ingress routers, may monitor, measure, shape, classify and mark packets of flows (individual or aggregated) according to the subscribed ser-

vice agreement. The core routers forward packets differently to provide the subscribed service. The core routers only need to provide several forwarding schemes to provide service differentiation, and thus we can deploy the DiffServ to large networks.

Two PHB groups are currently defined in DiffServ: the Assured Forwarding (AF) PHB group [69] and the Expedited Forwarding (EF) PHB group [70]. While Expedited Forwarding is optional, Assured Forwarding is required in an implementation of DiffServ. Expedited Forwarding, however, can be used as a premium service for the DiffServ network like a "virtual leased line."

2.9 Glossary of Terms

The following is a list of the main terms used in the Differentiated Services Architecture [67].

Assured Forwarding (AF) – A specific DiffServ behavior, which divides IP packets into four separate per-hop-behavior (PHB) classes. Using these classes, a provider may offer different levels of service for IP packets received from a customer domain. Each Assured Forwarding class is allocated a specified amount of buffer space and bandwidth.

Autonomous System (AS) – A self-connected set of networks that are generally operated within the same administrative domain.

Behavior Aggregate (BA) – a collection of packets with the same codepoint crossing a link in a particular direction. The terms "aggregate"

and "behavior aggregate" are used interchangeably.

BA classifier – a classifier that selects packets based only on the contents of the DiffServ field.

Best-Effort Service – The default behavior of TCP/IP networks in the absence of QoS measures. TCP/IP nodes will make their best effort to deliver a transmission but will drop packets indiscriminately in the event of congestion managing the bandwidth or assigning priority to delay-sensitive packets. The Internet today uses best-effort service.

Codepoint – Codepoint markings are made in a new implementation of the IP version 4 Type of Service (ToS) header called the DiffServ field (six bits, reserving two for congestion notification) and are used to select a per hop behavior (PHB). This marking takes place on the host or on a boundary or edge device.

Core router – A router on the network service provider WAN that has no direct connections to any routers at customer sites.

DiffServ-compliant – a node enabled to support differentiated services functions and behaviors.

Differentiated Services Boundary – The edge of a DiffServ domain, where classifiers and traffic conditioners are likely to be deployed. A DiffServ boundary can be further sub-divided into ingress and egress nodes, where the ingress/egress nodes are the downstream/upstream nodes of a boundary link in a given traffic direction. A DiffServ boundary may be co-located with a host, subject to local policy.

Differentiated Services Domain – A contiguous portion of the Internet over which a consistent set of DiffServ policies are administered in a coordinated fashion. A DiffServ domain can represent different administrative domains or autonomous systems, different trust regions, different network technologies, hosts and routers, etc.

DiffServ egress node – a DiffServ boundary node which handles traffic as it leaves a DiffServ domain.

DiffServ ingress – a DiffServ boundary node which handles traffic as it enters a DiffServ domain.

DiffServ interior node – a DiffServ node that is not a DiffServ boundary node.

DiffServ field – the IPv4 header TOS octet or the IPv6 Traffic Class octet.

Dropper – a device that discards packets based on specified rules

Expedited Forwarding (EF) – A Per-Hop Behavior (PHB) in the DiffServ standard, used to create a virtual leased line service.

IPv4 (Internet Protocol version 4) – The most widely deployed version of the Internet Protocol. IPv4 provides some basic traffic classification mechanisms with its IP Precedence/CBQ and Type of Service header fields. However, network hardware and software traditionally have not been configured to use them.

IPv6 (Internet Protocol version 6) – An update to the Internet Protocol that is in the early phases of adoption. Most of the refinements

concentrate on basics such as expanding the IP address numbering scheme to accommodate the growth of the Internet. However, IPv6 does include a Class header field that is explicitly intended to designate a Class of Service.

Jitter – The distortion of a signal as it is propagated through the network, where the signal varies from its original reference timing and packets do not arrive at its destination in consecutive order or on a timely basis, i.e. they vary in latency. In packet-switched networks, jitter is a distortion of the interpacket arrival times compared to the interpacket times of the original transmission. Also referred to as delay variance. This distortion is particularly damaging to multimedia traffic.

Latency – Delay in a transmission path or in a device within a transmission path. In a router, latency is the amount of time between when a data packet is received and when it is retransmitted. Also referred to as propagation delay.

Marker – a device that sets the DiffServ codepoint in a packet based on defined rules.

Meter – a device that measures the temporal properties of a traffic stream selected by a classifier.

Microflow – a single instance of an application-to-application flow of packets which is identified by source address, source port, destination address, destination port and protocol id.

MF Classifier – a multi-field (MF) classifier which selects packets based on the content of some arbitrary number of header fields.

Per Hop Behavior (PHB) – The forwarding treatment given to a specific class of traffic, based on criteria defined in the DiffServ field. Routers and switches use PHBs to determine priorities for servicing various traffic flows.

PHB group – A set of one or more PHBs that can only be meaningfully specified and implemented simultaneously, due to a common constraint applying to all PHBs in the set such as a queue servicing or queue management policy. A PHB group provides a service building block that allows a set of related forwarding behaviors to be specified together (e.g., four dropping priorities). A single PHB is a special case of a PHB group.

Policing – Packet-by-packet monitoring function at a network border (ingress point) that ensures a host (or peer or aggregate) does not violate its promised traffic characteristics. Policing means limiting the amount of traffic flowing into or out of a particular interface to achieve a specific policy goal. Policing typically refers to actions taken by the network to monitor and control traffic to protect network resources such as bandwidth against unintended or malicious behavior. Traffic shaping may be used to achieve policing goals or to do congestion management.

Policy – The combination of rules and services where rules define the criteria for resource access and usage to manage the bandwidth made

available to specified traffic. A policy dictates a number of conditions that must be met before a specified action can be taken.

Premium Service – In DiffServ terms, Premium service is a peak-limited, extremely low-delay service, resembling a leased line. At the network edge, where a Premium class is first created, it must be either shaped or policed to a rate with no more than a two-packet burst. A policer for Premium service is set to drop packets that exceed the configured peak rate. For this service, the peak rate of the Premium class aggregate across any boundary must be specified and the rate must be smaller than the link capacity.

Quality of Service (QoS) – A collective measure of the level of service delivered to the customer. QoS can be characterized by several basic performance criteria, including availability (low downtime), error performance, response time and throughput, lost calls or transmissions due to network congestion, connection set-up time, and speed of fault detection and correction. Service providers may guarantee a particular level of QoS (defined by a service level agreement or SLA) to their subscribers.

QoS Routing (QoSR) – 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 induced jitter.

Service Level Agreement (SLA) – a service contract between a customer and a service provider that specifies the forwarding service a cus-

tomer should receive. A customer may be a user organization (source domain) or another DiffServ domain (upstream domain). A SLA may include traffic conditioning rules which constitute a TCA in whole or in part.

Service Provisioning – a policy which defines how traffic policy conditioners are configured on DiffServ boundary nodes and how traffic streams are mapped to DiffServ behavior aggregates to achieve a range of services.

Shaper – a device that delays packets within a traffic stream to cause it to conform to some defined traffic profile.

Traffic Conditioner – An entity that performs traffic conditioning functions and which MAY contain meters, policers, shapers, and markers. Traffic conditioners are typically deployed in DiffServ boundary nodes.

Traffic Conditioning – Control functions that can be applied to a behavior aggregate, application flow, or other operationally useful subset of traffic, e.g., routing updates. These may include metering, policing, shaping, and packet marking. Traffic conditioning is used to enforce agreements between domains and to condition traffic to receive a differentiated service within a domain by marking packets and by monitoring and altering the temporal characteristics of the aggregate where necessary.

Traffic Conditioning Agreement (TCA) – An agreement specifying classifier rules and any corresponding traffic profiles and metering, mark-

ing, discarding and/or shaping rules which are to apply to the traffic streams selected by the classifier. A TCA encompasses all of the traffic conditioning rules explicitly specified within a SLA along with all of the rules implicit from the relevant service requirements.

Traffic Profile – A description of the temporal properties of a traffic stream such as rate and burst size.

Traffic Shaping – A group of techniques that attempt to regulate or meter the flow of packets through the network.

Traffic stream – An administratively significant set of one or more microflows which traverse a path segment. A traffic stream may consist of the set of active microflows which are selected by a particular classifier.

Type of Service (TOS) – A field within an IP header which can be used by the device originating the packet, or by an intermediate networking device, to signal a request for a specific QoS level. TOS uses three bits to tell a router how to prioritize a packet and one bit apiece to signal requirements for delay, throughput, and reliability. TOS is also known as IP precedence bit format and the IP precedence field. However, it has not been used much in practice.

Chapter 3

QoS-Based Routing Algorithms

Routing is an essential component in data transmission over the Internet. A considerable amount of literature has been developed on routing protocols for the "best-effort" service. Quality of Service routing has recently received substantial attention in the context of its possible use in an Integrated Services IP network. A few papers have addressed the routing problems for Differentiated Services.

In this chapter, we give a review of the existing routing protocols and algorithms.

3.1 Basic Concepts

Path computation algorithms for a single metric, such as hop-count and cost, are well known and have been widely used in the current best-effort service network. Multiple metrics can certainly model a network more ac-

curately. However, the problem of finding the best path subject to multiple constraints is an NP-complete problem [43].

The network is modelled as a directed graph G(N, E), where N is the set of nodes representing routers and hosts and E is the set of edges representing links that connect the nodes. Each link e = (u ,v), where u, v in N, is associated with k independent weights, $w_1(e)$, $w_2(e)$, ..., $w_k(e)$, where $w_i(e)$ is a positive real number ($w_i(e) \in R^+$) for all $1 \leq i \leq k$. The notation $w(e) = w(u, v) = (w_1(e), w_2(e), ..., w_k(e))$ is used to represent the weights of a link.

Definition 3.1.1 *Given a directed graph $G(N, E)$ with $k \geq 2$ weight functions $w_1 : E \rightarrow R^+$, $w_2 : E \rightarrow R^+$, ..., $w_k : E \rightarrow R^+$, a path $p = src \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow dst$ is said to be an optimal QoS path from node src to node dst if there does not exist another path q from node src to node dst such that $w(q) < w(p)$, that is, for all $1 \leq i \leq k$, $w_i(q) < w_i(p)$.*

When $k = 1$, the optimal QoS path is the same as the shortest path. When $k > 1$, however, there can be multiple optimal QoS paths between two nodes.

Definition 3.1.2 *Let $d(i, j)$ be a metric for link (i, j). For any path $p = (i, j, k, ..., l, m)$, we say that metric d is additive [43] if*

$$d(p) = d(i, j) + d(j, k) + ... + d(l, m)$$

We say that metric d is multiplicative if

$$d(p) = d(i, j) * d(j, k) * ... * d(l, m)$$

We say that metric d is concave if

$$d(p) = min[d(i,j), d(j,k), ..., d(l,m)]$$

The quality of the service can be estimated and specified in terms of some parameters, called metrics, that are of prime importance to the application under consideration. These parameters are used to express the applications requirements that must be guaranteed by the underlying network. We now look at some parameters that are likely to be considered as QoS routing metrics: delay, delay jitter, cost, loss probability and bandwidth. It is obvious that delay, delay jitter, hop-count and cost follow the additive composition rule, probability of successful transmission follow the multiplicative composition rule, and bandwidth follows the concave composition rule. The composition rule for loss probability is more complicated as seen below:

$$d(p) = 1 - ((1 - d(i,j)) * (1 - d(j,k)) * \ldots * (1 - d(l,m)))$$

However, the loss probability metric can be easily transformed to an equivalent metric (the probability of successful transmission) that follows the multiplicative composition rule.

The main difference between QoS-based routing and traditional routing is that the routing decision is based not only on source and destination addresses for each packet, but also on the traffic characteristics of the related flow. This fact has two major effects on the routing mechanisms (i.e. routing protocol and IP layer). The routing algorithm used by the routing protocol will have to consider several metrics in route computation at the same time,

instead of just one as in traditional routing. This leads to difficulties in the construction of such QoS-based routing algorithms.

Wang and Crowcroft proved that the problem of finding a path subject to two or more independent additive and/or multiplicative constraints in any possible combination is NP-Complete. The only tractable combinations are the concave constraint and the other additive/multiplicative constraints [43]. The proof of NP-Completeness relies heavily on the correlation of the link weight metrics. It has been proved for the chain topology and assumed that the NP-Complete nature will hold for all topologies. However QoS routing in realistic networks may not be NP-Complete in nature as both chain topology and the correlated link metric vectors are unlikely to occur in realistic networks. QoS Routing is NP-Complete when the QoS metrics are independent and are real numbers or unbounded integers.

When both the metrics are additive, [8] suggests that given a weight $w(u, v)$ between nodes u and v, it can be converted to a new weight function $w'(u, v) = \lceil (w(u, v) * x)/c \rceil$. This reduces the constraint $w \leq c$ to $w' \leq x$, where c is a real number or an unbounded integer and x is a bounded integer. It is proved that the solution for the simpler problem is also a solution for the original problem. If L is the length of a path p, and $w(p) \leq 1 - [(L-1) * c]/x$, then p is also a solution for the simpler problem.

We list the three general NP-completeness theorems for additive and multiplicative metrics.

Theorem 3.1.1 *Given a network $G = (N, A)$, n additive metrics $d_1(a)$, $d_2(a)$, ..., $d_n(a)$ for each a in A , two specified nodes i, m, and n positive integers D_1, D_2, ..., D_n, $(n \geq 2, d_i(a) \geq 0, D_i \geq 0$ for $i = 1, 2, ..., n)$, the*

problem of deciding if there is a simple path $p = (i, j, k, ..., l, m)$ *which satisfies the following constraints* $d_i(p) \leq D_i$ *where* $i = 1, 2, ..., n$ *is NP-complete [43].*

Theorem 3.1.2 *Given a network* $G = (N, A)$, n *multiplicative metrics* $d_1(a)$, $d_2(a), ..., d_n(a)$ *for each* a *in* A, *two specified nodes* i, m, *and* n *positive integers* D_1, $D_2, ..., D_n$, $(n \geq 2, d_i(a) \geq 1, D_i \geq 1$ *for* $i = 1, 2, ..., n)$, *the problem of deciding if there is a simple path* $p = (i, j, k, ..., l, m)$ *which satisfies the following constraints* $d_i(p) \leq D_i$ *where* $i = 1, 2, ..., n$ *is NP-complete [43].*

Theorem 3.1.3 *Given a network* $G = (N, A)$, n *additive and* k *multiplicative metrics* $d_1(a), d_2(a), ..., d_{n+k}(a)$ *for each* a *in* A, *two specified nodes* i, m, *and* $n + k$ *positive integers* $D_1, D_2, ..., D_{n+k}$, $(n \geq 1, k \geq 1, d_i(a) \geq 1, D_i \geq 0$ *for* $i = 1, 2, ..., n$, $D_i \geq 1$ *for* $i = n + 1, 2, ..., n + k)$, *the problem of deciding if there is a simple path* $p = (i, j, k, ..., l, m)$ *which satisfies the following constraints* $d_i(p) \leq D_i$ *where* $i = 1, 2, ..., n + k$ *is NP-complete [43].*

The three theorems above show that the problem of finding a path subject to constraints on two or more additive and multiplicative metrics in any possible combination is NP-complete. The results are applicable to any metric that follows additive or multiplicative composition rules, and to any metrics that can be transformed to equivalent metrics that follow the additive or multiplicative composition rule.

In [36], Sobrinho investigated the properties that path weight functions must have so that hop-by-hop routing is possible and optimal paths can be computed with a generalized Dijsktra's algorithm. Sobrinho defined an algebra of weights which contains a binary operation, for the composition of link weights into path weights, and an order relation. Isotonicity is the key

property of the algebra. It states that the order relation between the weights of any two paths is preserved if both of them are either prefixed or appended by a common, third, path.

Isotonicity is proved to be both necessary and sufficient for a generalized Dijkstra's algorithm to yield optimal paths. Likewise, isotonicity is also both necessary and sufficient for hop-by-hop routing. However, without strict isotonicity, hop-by-hop routing based on optimal paths may produce routing loops.

The computation complexity of a routing algorithm is primarily determined by the composition rules of the metrics.

3.2 Extensions to the OSPF Protocol

In [14], Guerin, Orda and Williams discussed path selection algorithms to support QoS routes in IP networks. The work is carried out in the context of extensions to the OSPF protocol [63]. They first review the metrics required to support QoS, and then present and compare several path selection algorithms, which represent different trade-offs between accuracy and computational complexity. They also describe and discuss the associated link advertisement mechanisms, and investigate some options in balancing the requirements for accurate and timely information with the associated control overhead.

In [3], Apostolopoulos, Guerin and Kamat added QoS routing extensions to the OSPF routing protocol [63] implementation (gate daemon or gated) on the UNIX system. They evaluated its performance over a wide range of

operating conditions. The evaluation results provide insight into the respective weights of the two major components of QoS routing costs, processing cost and protocol overhead and establish strong empirical evidence that the cost of QoS routing is well within the limits of modern technology and can be justified by the performance improvements.

3.3 Hop-by-Hop Routing for Premium Traffic in DiffServ

Based on the hop-by-hop routing mechanism, an interesting problem is how to find an optimal routing algorithm for premium class traffic such that (1) it works correctly and efficiently for premium traffic; (2) it reduces negative influences to other classes of traffic (such as bandwidth starvation, excessive delay jitter, etc.). This problem is called the Optimal Premium-class Routing (OPR) problem which is NP-Complete.

In [39], Wang and Nahrstedt analyzed the strength and weaknesses of two existing algorithms (Widest-Shortest-Path algorithm and Bandwidth-inversion Shortest-Path algorithm), and applied to the OPR problem a novel heuristic algorithm, called the Enhanced Bandwidth-inversion Shortest-Path (EBSP) algorithm. They proved theoretically the correctness of the EBSP algorithm, i.e., it is a consistent and loop-free hop-by-hop routing algorithm.

3.4 QoS-Based Routing Algorithms with a Single Metric

3.4.1 Bandwidth

If the single metric is bandwidth, all the links with bandwidth less than the desired bandwidth are removed before the path is selected from the source to the destination. Guerin-Orda [7, 19] considers the imprecision in the network while selecting the path. The imprecision model is based on the probability distribution functions. The heuristic tries to find the path with the best multiplicative probability over all the links making the path, that is, a path that has the highest probability to accommodate a new connection with a given bandwidth requirement. The multiplicative problem is transformed into an additive problem by assigning weight w_l to the link l as $-\log p_l$, where $p_l = p_l(w)$ is the probability of success in the link having the w units of bandwidth.

3.4.2 Delay

If the single metric is delay, Guerin-Orda [7, 19] considers the imprecision model, based on probability distribution functions, to determine the delay-constrained path. The goal of the algorithm is to find a path that has highest probability to accommodate a new connection with a required delay requirement. The heuristic proposes transforming global constraints into local constraints by splitting the end-to-end delay constraint among the intermediate links such that every link in the path has an equal probability

of satisfying the delay constraint. Given a probability function $f_l(d_l)$, [28] defines a cost function $c_l(d_l) = -\log f_l(d_l)$ so that the cost associated with each link is positive and it decreases as the associated delay increases.

3.5 QoS-Based Routing Algorithms with Dual Metrics

3.5.1 Bandwidth and Cost

The Ticket Based Probing (TBP) algorithm proposed for delay constrained least cost routing can be applied to bandwidth constrained least cost routing [9]. Probes are sent from the source, limited in number, towards the destination. Receipt of a probe by the destination ensures availability of a path satisfying the desired resource requirements.

3.5.2 Bandwidth and Delay

Wang-Crowcroft's algorithm [7, 22] based on source routing for bandwidth-delay based routing algorithm first prunes all the links with bandwidth less than the required bandwidth. Next, it finds the shortest path with respect to delay in the modified graph using Djikstra's algorithm. A distributed algorithm based on hop-by-hop routing is proposed that decides precedence among the metrics to determine the best path. Insufficient bandwidth leads to higher queuing delay and loss rate. Improper propagation delay leads to higher overall delay but the increase is predictable and stable. Thus bottleneck bandwidth is given higher precedence over propagation delay. Orda

proposed the quantization of QoS metrics with a rate-controlled earliest deadline first scheduler at each router [50].

3.5.3 Delay and Cost

When both the metrics are additive, [8] suggests that given a weight $w(u, v)$ between nodes u and v, it can be converted to a new weight function $w'(u, v) = \lceil (w(u, v) * x)/c \rceil$, thereby changing the constraint $w \leq c$ to $w' \leq x$, where c is a real number or an unbounded integer and x is a bounded integer. If L is the length of a path p, and $w(p) \leq 1 - [(L-1) * c]/x$, then p is also a solution for the simpler problem (the heuristic condition). The cost-delay constrained QoS routing is reduced to two problems where the link weights are

1. Original cost and new-delay $(u, v) = \lceil (d(u, v) * x)/\Delta_d \rceil$
2. Original delay and new-cost $(u, v) = \lceil (c(u, v) * x)/\Delta_c \rceil$

where x = coefficient * distance (source, destination). coefficient is a given positive number, $d(u, v)$ is the delay of the path from u to v, $c(u, v)$ is the cost of the path from u to v, Δ_d is the delay constraint and Δ_c is the cost constraint. An extended Djikstra's and extended Bellman-Ford algorithm is proposed that is guaranteed to find a solution if any of the paths from the source to the destination satisfies the heuristic condition stated above.

The Delay Constrained Unicast Routing (DCUR) algorithm proposed by Salama chooses between the least cost and least delay paths independent of the choice of the previous nodes [7, 21]. It maintains a cost and delay vector at every node by a distance-vector protocol. Control message is sent from

the source towards the destination to construct a delay-constrained path. Any node at the end of the partially constructed path can select one of the only two alternative outgoing links: least cost or least delay. Loops may occur as control message chooses the least-cost path and the least-delay path alternatively, however it detects a loop if the control message visits the same node twice.

Sun-Landgendorfer improves on Salama's algorithm by avoiding loops instead of detecting and removing loops. The message travels along the least-delay path until it reaches a node from which the delay of the least-cost path satisfies the delay constraint. From that node on, the message travels along the least-cost path all the way to the destination [7].

Delay Scaling Algorithm (DSA) [18] preprocesses the network to prune out nodes that are not feasible. It computes paths from the source to all destinations such that the cost of the path from source to each receiver is at most the cost of the cheapest path between the two with delay T and its delay is at most $(1+\epsilon)T$. A τ-scaling G_τ of graph G is obtained by multiplying the delay on each link in G by τ/T and then truncating the new delay to an integer. DSA is executed on it and if every destination has delay greater than $(1+\epsilon)T$, then τ is doubled and the procedure is repeated.

A Lagrange Relaxation based method has been proposed for QoS Routing called Lagrange Relaxation based Aggregated Cost (LARAC) [21]. LARAC is based on the heuristic of minimizing modified cost function $c_\lambda = cost + \lambda.delay$, where λ is the weight assigned to delay wrt cost. If $\lambda = 0$ and the delay constraint is satisfied, an optimal solution for the original problem has been found. If this is not the case, λ is increased to increase the dominance of

delay in the modified cost function. $L(\lambda) = min\{c_\lambda(p) : p \in P(s,t)\} - \lambda\Delta_d$ is a lower bound to DCLC for any $\lambda \geq 0$. To obtain the best lower bound we need to maximize the function $L(\lambda)$, that is, $L^* = maxL(\lambda)$ for $\lambda \geq 0$. The constraining conditions is neglected and built into the object function (the relaxation). Since solutions feasible to the original problem suit the relaxation conditions as well, a lower bound of the original problem is found. Increasing the dominance enforces the solution to approach the optimal solution and decrease the difference between the obtained lower bound and the optimal of the original problem as well.

A distributed routing algorithm with imprecise state information called Ticket Based Probing (TBP) has been proposed in [9]. Certain number of tickets is issued at the source according to the contention level of network resources. Each probe is required to carry at least one ticket. The total number of tickets bound maximum number of probes at any time. Each probe searches a path; hence the number of tickets also bound the maximum number of paths searched. The Level of imprecision has a direct impact on the number of tickets issued. Probes can only travel along the paths that satisfy the delay requirement. Hence any probe arriving at the destination detects a feasible path.

Delay-Cost-Constrained Routing (DCCR) [20] rapidly generate a near-optimal delay-constrained path, then, it employs the k-shortest path algorithm proposed by Chong et. al. with a new non-linear weight function of path delay and cost to efficiently search for a path subject to both the requested delay constraint and the cost constraint. Starting with the least delay path as a feasible solution, the cost of the least-delay path is selected

as the cost bound. If there is no feasible paths with cost less than this, then the least delay path itself must be the optimal path and this is what the algorithm returns. Path weight (w) has an exponential growth with the path cost (c) and is only linearly proportional to the path delay (d). Thus, the weight function w is designed to give more priority to lower cost paths P_i^u as shown below:

$$w(P_i^u) = \begin{cases} d(P_i^u)/(1 - c(P_i^u)/\Delta_c), & \text{if } d(P_i^u) \leq \Delta_d \text{ and } c(P_i^u) \leq \Delta_c \\ \infty, & \text{otherwise} \end{cases}$$

To search for a tighter cost bound, another heuristic - the Blokh and Gutin (BG) heuristic is used in a new algorithm called Search Space Reduction + DCCR (SSR+DCCR). BG uses a linear function of the link delay and cost to compute link weight. It adjusts the weights given to cost and delay in the weight function according to the quality of the current path, thus it iteratively approaches the optimal (least-cost) solution. It starts with two paths: least-delay-path (LDP) and least-cost-path (LCP). If LCP is a feasible path (delay-bounded) path, then it is the optimal solution. If not, then algorithm maintains 2 paths: The current feasible (delay-bounded) path LDP and the current best infeasible path LCP. $W(p) = \alpha D(p) + \beta C(p)$. If $W(LWP) < \gamma$, where γ is the current least path weight and LWP is feasible, LWP replaces LDP to become the best feasible path, thus the weight given to link cost increases in the next round. If LWP is infeasible, then the weight given to link delay increases. The path found by the BG algorithm may still not be optimal. BG has unbounded time complexity, however SSR+DCCR time complexity is bounded since BG is only used as a prelude to DCCR

with a very small number of iterations.

3.5.4 Two Additive Metrics

Jaffe proposed an intuitive approximation algorithm based on minimizing a linear combination of the link weights $w_1(p) + dw_2(p)$, where $d = 1$ in the first algorithm and $d = \sqrt{(c_1/c_2)}$ in the second. The latter provides better performance in comparison to the former [24].

[24] proposes an algorithm based on the minimization of the linear cost function $\alpha w_1(p) + \beta w_2(p)$ for two additive metrics w_1 and w_2. The paper suggests a binary search strategy (using either $\{\alpha = 1, \beta = k\}$ or $\{\alpha = k, \beta = 1\}$) to find the appropriate value of k. With link weight $l(e) = w_1(e) + w_2(e)$ (i.e., $\alpha = 1, \beta = 1$), the algorithm searches for path p. If both $w_1(p) > c_1$ and $w_2(p) > c_2$, then it is guaranteed that there is no feasible path and the algorithm terminates. It maintains min_$w_1[u]$ and min_$w_2[u]$ that represent the minimum w_1 and w_2 weights among all shortest paths. If min_$w_1[t] \leq c_1$ or min_$w_2[t] \leq c_2$, then there is a possibility of existence of a feasible path. The algorithm executes the binary search using link weights $l(e) = kw_1(e) + w_2(e)$ in Phase I and $l(e) = w_1(e) + kw_2(e)$ in Phase II. k is in the range [1,B], where B = n * max$\{w_j(e)\}$ that is an upper bound on the total cost of the longest path wrt link weight w_j. If the algorithm cannot find a path p for which l(p) is minimum and $w_j(p) \leq c_j$, then such a path p cannot be found with larger values of k. If the binary search fails to return a feasible path wrt both constraints, then it returns a path p that satisfies the c_j constraint and whose $w_i()$ cost is upper bounded as follows:

$$w_i(p) \leq w_i(f) + [w_j(f) - w_j(p)]/k$$

where f is a feasible path, k is the maximum value that the binary search determines at its termination, and the pair (i, j) is either (1,2) or (2,1) depending on the phase. This algorithm provides superior performance to Jaffe's approximation algorithm.

3.6 QoS-Based Routing Algorithms with Multiple Metrics

The Multi-Constrained Optimal Path (MCOP) problem attempts to find a minimal-cost path satisfying the constraints. It is also known as Restricted Shortest Path (RSP) problem and is NP-Complete even for a single constraint. Multi-Constrained Path (MCP) is MCOP problem without the path optimization requirement; it is NP-Complete for more than one constraint [22]. Several techniques have been devised to deal with the multiple metrics routing scenario. We categorize the solutions as shown below:

1. Single Metric representative of the individual metrics: There have been proposals of a single metric that is a linear combination of weighted link metrics and representative of the individual metrics. The individual weights can be statically configured or determined dynamically. The weights can be iteratively changed to get better paths. However, if the path is optimal in the single metric, it is not necessary that it is optimal in terms of the individual metrics. Subsections of shortest paths in multiple dimensions are not necessarily shortest paths [32]. There is information loss in the aggregation process, which needs to be quantified.

2. Fallback Routing approach: QoS metrics are considered one by one in a predetermined fallback sequence hoping that the optimal path wrt a single parameter will also satisfy the other constraints.

3. Dependent QoS Metrics: Multiple metrics dependent on each other can be reduced to one metric and the resulting problem with single metric can be solved in polynomial time.

When all the QoS metrics, except one, take bounded integer values, the multi-constrained QoS routing problem is solvable in polynomial time. Limited path heuristic is when each node maintains a limited number of optimal QoS paths that ensures worst-case polynomial time complexity [50]. Yuan studied the two heuristics for two-constraints problem. [50] extends it for multiple-constraints problems. It was found that limited path heuristic is superior to limited granularity heuristic for multiple constraints problem when the number of constraints is greater than three.

A path vector routing protocol called QoSFinder [38] has been proposed that considers throughput (t, function of bandwidth and load), delay (d) and loss rate (e) metrics for path selection of long-lived multimedia flows. Path vector routing has been derived from distance vector routing. On receiving an update message, the receiving node's address is inserted in the route, cost to reach the neighbor that sent the update is added to the cost of the route and the message is forwarded. If its address is already present, a loop is detected and the message is dropped. Given two paths with QoS (t_1, d_1, e_1) and (t_2, d_2, e_2) respectively, if either one of it satisfies the required constraints, then the one satisfying the demand will be accepted. However if both do/do not satisfy, then the better should be selected. The routing process is required

to return the "best" QoS path, irrespective of whether the path satisfies all the QoS or not. The "best" path is selected based on availability (A), defined as $A_t = (t/t_d)$, $A_d = (d_d/d)$, $A_e = (e_d/e)$. Availability of a parameter with a value of 1 means that the tested parameter is equal to the demanded parameter, while greater than 1 indicates reserve and lesser than 1 indicates that the parameter value does not meet the demand. If two sets are equal, the router with lower hop-count is selected. If hop-counts are also equal, the least recently used route is selected.

Ma-Steenkiste's algorithm [7, 30] states that when WFQ-like scheduling algorithms are used, queueing delay, delay-variation, and loss are not independent metrics; they are a function of bandwidth. This simplifies the problem and makes it solvable in polynomial time.

A heuristic algorithm for multi-constrained optimal path (H_MCOP) problem is proposed in [22]. A non-linear cost function is developed as shown below:

$$g_\lambda(p) = \frac{w_1(p)}{c_1} * \lambda + \frac{w_2(p)}{c_2} * \lambda + \ldots + \frac{w_k(p)}{c_k} * \lambda, l \geq 1$$

where $w_i(p)$ is the total weight of the i^{th} metric in the path p and the constraint is $w_i(p) \leq c_k$. The goal of the algorithm is to determine the path p with minimum cost. It has been proved that $w_k(p) \leq c_k$ for at least one k and $w_k(p) \leq \sqrt[\lambda]{K} c_k$ for the rest. The likelihood of finding a feasible path increases as λ increases. Starting from source, H_MCOP finds the next node based on minimization of $g_\lambda(p)$. If more than one path is feasible since H_MCOP uses the k-shortest path routing approach, the path with minimum cost is selected.

Chen-Nahrstedt's Selective Probing [7] is when after a connection request arrives, probes are flooded selectively along those paths that satisfy the QoS and optimization requirements. This method overcomes the high communication overhead associated with Shin-Chou algorithm: Probes are only forwarded to a subset of outgoing links selected based on the topological distances to the destination and it is done iteratively. Ticket-based Probing is used to improve performance of selective probing. A certain number of tickets are issued at the source according to the contention level of network resources. It can be used for multicast QoS routing as well.

A randomized algorithm [23] has been proposed for multiple constraints QoS routing. It prunes all links that cannot be on any feasible path and uses a randomized BFS search to find a feasible path with minimum hop-count. It may not find all feasible paths, but if it finds a path, that path will satisfy the required constraints.

A scalable internetwork routing architecture for QoS routing has been proposed named Nimrod [62]. The key to its scalability is its ability to represent and manipulate routing related information in the form of maps at multiple levels of abstraction. Different maps can represent the same region of a physical network at different levels of detail. Maps can be used to represent graphs of different metrics and aggregated to determine the optimal path.

As one of the most challenging problems of the next-generation high-speed networks, quality of service routing (QoSR) with multiple (k) constraints is an NP-complete problem. In [11], Cui, Xu and Wu proposed a multi-constrained energy function-based precomputation algorithm, MEFPA. MEFPA cares each QoS weight to b degrees, and computes a number ($B = C_{b+k-2}^{k-1}$)

of coefficient vectors uniformly distributed in the k-dimensional QoS metric space to construct B linear energy functions. Using each LEF, it then converts k QoS constraints to a single energy value. Then it uses Dijkstra's algorithm to create B least energy trees, based on which the QoS routing table is created.

3.7 Multiple Path Routing Algorithms

Most of the numerous studies of QoS routing (QoSR) have been aimed at producing a single optimal path. With this approach, only a single path is established between a source-destination pair, even if there exist some alternative, possibly suboptimal, paths. In case of congestion, this approach is likely to aggravate the problem and may additionally trigger routing oscillations.

With multiple-path routing, the traffic between the same source and destination can be assigned to different paths. Intuitively, this approach will tend to smooth out occasional problems occurring on some paths, including those caused by inaccurate link state information.

Consider a network that is represented by a graph $G = (N, E)$, where N is the set of nodes and E is the set of links. Each link $(i, j) \in E$ is associated with R non-negative and additive QoS values: $w_r(i, j)$, $r = 1, 2, \ldots, R$. A length (or cost) function w_0 is defined as follows:

$$w_0(i, j) = \sum_{r=1}^{R} a_r w_r(i, j)$$

Given a source node s and a target node t, and R constraints $C_r(s, t)$, $r = 1, 2, \ldots R$. The K-Multiple-Constrained-Shortest-Path (KMCSP) problem is

to find either the first K shortest length paths or all the paths (depending on which number is smaller) from a source node s to a target node t such that

$$w_r(p(s,t)) = \sum_{(i,j)\in p(s,t)\ \forall r} w_r(i,j) \leq C_r(s,t)$$

Liu and Ramakrishnan [27] consider the KMCSP problem and give an algorithm called A*Prune for finding k shortest paths subject to multiple constraints. The algorithm constructs paths starting at the source and going towards the destination. But, at each iteration, the algorithm gets rid of all paths that are guaranteed to violate the constraints, thereby keeping only those partial paths that have the potential to be turned into feasible paths, from which the optimal paths are drawn. The choice of which path to be extended first and which path can be pruned depend upon a projected path cost function, which is obtained by adding the cost already incurred to get to an intermediate node to an admissible cost to go the remaining distance to the destination.

In [32], Mieghem and Neve deduce a QoS algorithm called Tunable Accuracy Multiple Constraints Algorithm or TAMCRA. TAMCRA is based on three fundamental concepts: a non-linear measure for the path length, the k-shortest path approach and the principle of non-dominated paths. When m multiple constraints are imposed, a non-linear choice for the definition of the path length proves to be superior to a linear definition (i.e. a weighted sum of the vector components). An important corollary of a non-linear length function is that the subsections of shortest paths in multiple dimensions are not necessarily shortest paths. This corollary suggests to consider in the computation more paths than only the shortest one, leading us naturally to the

k-shortest path approach. Finally, the multi-dimensional character of QoS routing invites the use of state space reduction which has been implemented via the concept of non-dominated paths.

The non-linear concave path length function used in this algorithm [32, 33] is $\max(w_1(P) \,/\, \Delta_1, w_2(P) \,/\, \Delta_2, \ldots, w_m(P) \,/\, \Delta_m)$ where w_1, w_2, ..., w_m are weights of the paths and Δ_1, Δ_2, ..., Δ_m are the constraints in terms of metric $1, 2, \ldots, m$ respectively. TAMCRA possesses tunable accuracy (coupled to the running time) via one integer parameter k that reflects the number of shortest paths taken into account during computation. There always exists a finite value of k for which TAMCRA returns the exact path:

$$k = \begin{cases} min(\Delta_1, \Delta_2) & \text{if } m = 2 \\ \Pi_{1 \leq i \leq m} \Delta_i & \text{if } m > 2 \end{cases}$$

A path is said to be dominated by another path p' iff $w_i(p) \leq w_i(p')$ for $i = 1, 2, \ldots, m$, with an inequality for at least one weight component i. This causes an efficient reduction in the search space. The main advantages of TAMCRA are the following:

- The calculation time of TAMCRA increases only linearly with the value of k and saturates beyond a certain value of k. The level at which the calculation time saturates depends on the size of the graph.

- The value of k needed to solve the multiple constraints problem exactly is a polynomial function of the granularity (i.e., the number of possible values) of the constraints.

- The probability of missing the shortest path is, above a certain thresh-

old, independent of the number of constraints. There is an exponential decrease of the probability of missing the shortest path as the value of k increases. For a constant probability of missing the shortest path, k increases logarithmically in the number of nodes in the graph.

A slight modification of TAMCRA, called Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA), has been proposed [25]. Unlike TAMCRA, SAMCRA is guaranteed to find a path within the constraints, provided such a path exists. The number of alternate paths computed differs from node to node. Memory is required for the storage of the computed alternate paths. SAMCRA allocates memory at the nodes when the need arises, unlike TAMCRA which has a fixed amount of memory allocated at all the nodes.

TAMCRA has higher complexity than H_MCOP due to its extra computation to determine dominated paths. TAMCRA requires a higher value of k than H_MCOP. Due to path optimization feature of H_MCOP, its paths are more resource efficient than TAMCRA. Solution of DCCR is exactly the optimal path while the cost of the TAMCRA path is much higher than that of the optimal solution.

Jia, Nikolaidis and Gburzynski propose a QoS routing scheme in which a connection between a source-destination pair can be assigned to one of several alternative paths [16]. Starting from the well known algorithm by Dijkstra, they develop a collection of K-shortest path routing strategies and investigate their performance under a diverse set of network topologies and traffic conditions.

A. Algorithm for Constructing the K-Shortest Paths

This is a label setting algorithm based on the Optimality Principle and being a generalization of Dijkstra's algorithm. The metrics under consideration are bandwidth and hop-count. The space complexity is $O(Km)$, where K is the number of paths and m is the number of edges. By using a pertinent data structure, the time complexity can be kept at the same level $O(Km)$.

Let a Directed Acyclic Graph (DAG) $G(N, A)$ denote a network with n nodes and m edges, where $N = \{1, \ldots, n\}$, and $A = \{a_{ij} \mid i, j \in N\} = \{(i, j) \mid i, j \in N\}$. The capacity of each link is b_{ij}, where $ij \in A$. Given a bandwidth request B for a connection from the source node s to the destination node t, the bandwidth-constrained multi-path routing problem is to find the top K paths from s to t $\{P_k \mid 0 < k \leq K\}$, such that $b(P_k) \geq B$, $\forall\ 0 < k \leq K$, where

$$b(P_k) = min_{ij \in P_k} b_{ij}$$

is called the bottleneck bandwidth of path P_k.

Define a label set X and a one-to-many projection $h : N \rightarrow X$, meaning that each node $i \in N$ corresponds to a set of labels $h(i)$, each element of which represents a path from s to i. Each label/path has a major weight and a minor weight. For the hop-based algorithm, the major weight is the inverse of the number of hops and the minor weight is the bottleneck bandwidth of the path represented by this label. Those weight are interchanged for the bandwidth-based algorithm.

The algorithm:

- s: the source node
- X: the label set
- b_{vj}: the link bandwidth from v to j
- bw_thrsh: the bandwidth threshold used to trim unqualified links
- $count[v]$: the number of paths from s to v found so far
- $lb0$: the permanent label selected from X, such that $lb0.bw \geq lb.bw$, $\forall\ lb \in X$
- $lb1$: a new label generated from $lb0$
- $lb0.ver$: $h^{-1}(lb0)$, that is, the corresponding node of label $lb0$
- $lb0.bw$: the bottleneck bandwidth of the path from s to $lb0.ver$
- $lb0.parent$: the label which generates $lb0$
- $P_v(count[v])$: the $count[v]$'th path from s to v

$count[i] = 0, \forall\ i \in N$

$lb0 = 1$

$lb0.ver = s$

$lb0.bw = \infty$

$lb0.hops = 0$

$lb0.parent = NULL$

$X = \{lb0\}$

while ($X \neq \phi$ and $\exists\ i$ such that $count[i] < K$, where $0 < i < n$)

do begin

find a permanent label $lb0$ from X, such that $lb0.bw \geq lb.bw, \forall\ lb \in X$

$X = X - \{lb0\}$

$v = lb0.ver$

$count[v] = count[v] + 1$

if ($count[v] \leq K$ and $lb0.hops <$ max_Hop_Num)

then begin

record the path $P_v(count[v])$ by following the $lb0 \rightarrow parent$ link

for each $a_{vj} \in A$

do begin

if (the prospective new label does not result in a loop and

$b_{vj} > bw\_thrsh$)

do begin

$lb1.ver = j$

$lb1 = lb0 + 1$

$lb1.bw = min(lb0.bw, b_{vj})$

$lb1.hops = lb0.hops + 1$

$lb1.parent = lb0$

$X = X \cup \{lb1\}$

end

end

end

end

B. Path Selection Algorithms

There are five path selection algorithms resulting from two different major criteria and different ways of applying the minor criteria.

- Best-K-Widest (BKW): From the K widest paths, select the one whose bandwidth best fits the connection request.
- Random-K-Widest (RKW): From the K widest paths, select one at random.
- Shortest-K-Widest (SKW): From the K widest paths, select the one with the least number of hops.
- Best-K-Shortest (BKS): From the K shortest paths, select the one whose bandwidth best fits the connection request.
- Widest-K-Shortest (WKS): From the K shortest paths, select the one with the largest bandwidth.

Chapter 4

Routing Algorithms for Differentiated Services

Quality of Service (QoS) based routing is one of the main component of the new QoS Internet Architecture. A good and practical routing algorithm provides QoS guarantees to applications and an efficient utilization of the network resources. The QoS routing algorithm should be simple because a complicated procedure is costly to implement and does not scale with the size of the network.

In traditional routing, packets are delivered using a route based on its source and destination addresses. The routing protocols used in IP networks are typically transparent to any particular Quality of Service that different packets/flows may have. As a result, routing decisions are currently made without any awareness of resources availability and requirements. This means that flows are often routed over paths that are unable to support their re-

quirements, while alternate paths with sufficient resources are available [14].

In QoS-based routing, the route for each flow considers also its traffic requirements. The process of selecting a path that can satisfy the QoS requirements of a new flow relies on both the knowledge of the flow's requirements and characteristics, and information about the availability of resources in the network. The path selection process may need to consider several metrics. The common metrics include bandwidth, cost, delay, jitter and loss probability etc.

In general, routing algorithms can be constructed in two ways. One approach is to use independent metrics that will be considered separately by the route computation algorithm [28, 42]. This solution transfers the difficulty to the construction of efficient algorithms. Some authors have shown that the computation of routes with more than two metrics is very difficult because of its complexity. It has been proved that multi-constrained path routing is known to be NP–hard [15, 43].

The second approach is to consider the dependency between the used metrics [30]. This solution puts the complexity on the definition of the relations between metrics, imposing minor changes to traditional algorithms.

4.1 Source Routing And Hop-By-Hop Routing

Source routing and hop-by-hop routing are the two basic routing architectures for data networks. Hop-by-hop routing is the common form of general-purpose routing in current networks while source routing is mainly used for

network diagnosis and special policy routes.

In hop-by-hop routing, packets are forwarded hop-by-hop at each node. Each node has a routing table with next hops for all destinations, and this table is usually computed periodically in response to routing updates. When a packet is received, hop-by-hop routing only requires a table lookup to find the next hop and send the packet to it. The packet header is smaller compared with source routing as the packets do not have to carry the full forwarding path.

In source routing, a forwarding path is computed on-demand at the source and listed in the packet header. Packets are forwarded according to the path in the packet. Since the computation is done for each individual request in a centralized fashion, source routing is very flexible. However, a source must have access to full routing information for each link for path computation, and packets have a larger packet header.

Since hop-by-hop routing pre-computes forwarding entries for every destination, it has to accommodate all possible resource requirements. The usual approach in current hop-by-hop routing algorithms is to compute the best path to every destination. With a single metric, the best path can be defined easily. With multiple metrics, however, the best path with all parameters at their optimal values may not exist at all. Thus, we must decide the precedence among the metrics in order to define the best path.

4.2 Dijkstra's Algorithm

The famous Dijkstra's algorithm (named after its discover, E.W. Dijkstra) solves the problem of finding the shortest path from a point in a graph (the source) to another point in the graph (the destination). It turns out that one can find the shortest paths from a given source to all points in a graph at the same time, hence this problem is sometimes called the single-source shortest paths problem.

We state Dijkstra's algorithm in network terminology instead of formal mathematical terminology below. We denote a network topology by G = (N,L), where N is a set of nodes, and L is a set of links.

Dijkstra's algorithm [37] keeps two sets of nodes:

S the set of nodes whose shortest paths from the source have already been determined and

N-S the set of the remaining nodes

The other data structures needed are:

d an array of best estimates of shortest path to each node

pi an array of predecessors for each node

The basic mode of operation is:

1. Initialise d and pi
2. Set S to empty

3. While there is still node in N-S,

   (a) Sort the nodes in N-S according to the current best estimate of their distance from the source

   (b) Add u, the closest node in N-S, to S

   (c) Relax all the nodes still in N-S connected to u

The relaxation process updates the costs of all the nodes v connected to a node u if it is possible to improve the best estimate of the shortest path to v by including (u,v) in the path to v. The relaxation procedure proceeds as follows:

```
InitialiseSingleSource(NetworkTopology T, Node s)
   for each node v in nodes(T)
       T.d[v] = infinity
       T.pi[v] = nil
       T.d[s] = 0;
```

This sets up the network topology so that each node has no predecessor (pi[v] = nil) and the estimates of the distance (or cost) of each node from the source (d[v]) are infinite, except for the source node itself (d[s] = 0).

The relaxation procedure checks whether the current best estimate of the shortest distance to v (d[v]) can be improved by going through u (i.e. by making u the predecessor of v):

```
Relax(Node u, Node v, double w[][])
    if d[v] > d[u] + w[u,v] then
```

```
        d[v] = d[u] + w[u,v]
        pi[v] = u
```

The Dijkstra algorithm is now:

```
DijkstraShortestPaths(NetworkTopology T, Weight w, Node s)
    InitialiseSingleSource(T, s)
    S = {0}
    Q = nodes(T)
    while not Empty(Q)
        u = ExtractCheapest(Q);
        AddNode(S, u);
        for each node v in Adjacent(u)
            Relax(u, v, w)
```

4.3 Algorithms

4.3.1 Problem Definition

Although extensive research has been done in QoS routing, most of them are focused on IntServ [31, 32, 36, 42]. A few papers have addressed the routing problems for DiffServ [39]. In this chapter, we develop routing algorithms for DiffServ in a heterogenous network.

Definition 4.3.1 *If a network node supports DiffServ, we call it DiffServ capable. If a network node does not support DiffServ, we call it DiffServ incapable. A link is DiffServ capable if and only if its originating node is DiffServ capable.*

We realize that DiffServ is needed in the next generation network, but it is impossible to implement DiffServ in all nodes at the same time. Thus, it is reasonable to assume that the network will consist of both DiffServ capable nodes and DiffServ incapable nodes. Here we raise the problem of how to find optimal routing schemes in a network consisting of DiffServ capable nodes and DiffServ incapable nodes. We will give routing algorithms for a DiffServ capable user to establish a route in such an heterogeneous network. Our algorithm has very little impact on the currently used routing algorithms and protocols. It is easy to implement.

For DiffServ routing, DiffServ capable links are preferred. But a path consisting of only DiffServ capable nodes may not exist in a heterogenous network. Even if such a path exists, it may not be the "optimal" path. The problem is how to find the optimal path for DiffServ in such a network.

One of the routing metrics we consider is DiffServ capability. The other metric is the cost to use a link. If the cost to use the link is set to be unity, then cost is the same as hop-count. The number of hops determines the amount of resources used along the path, which is an important factor from the viewpoint of efficiency and performance.

We denote a network by a directed graph $G = (N, L)$, where N is a set of nodes representing routers and hosts, and L is a set of edges representing links that connect the nodes. L is also simply called the set of links. We associate each link $e = (u, v)$, where $u, v \in N$, with a real variable c_{uv} and a boolean variable $dcapable_{uv}$ (1 or 0). c_{uv} is the cost to use the link e and $dcapable_{uv}$ indicates whether e is DiffServ capable (value 1) or not (value 0).

We use $p = (1, 2, \ldots, i, j)$ to denote a path from 1 to j via 2, ..., i. $C(1, 2, \ldots, i, j)$ denotes the total cost of the path from 1 to j. $D(1, 2, \ldots, i, j)$ denotes the number of DiffServ capable nodes along the path from 1 to j.

The current best-effort internet model depends on hop-count as a measure of path cost (each link has unit cost). If each link is assigned a weight greater than unity, the algorithm is a cost-constrained routing algorithm.

In order to find the proper routing algorithms for such a network, we need to decide the precedence between the two metrics.

4.3.2 Algorithm 1

In this section, cost is given higher precedence over DiffServ capability. In this case, we can use the existing routing algorithms to find the minimum cost routing paths, and use a selecting algorithm to get the one with the maximum number of DiffServ capable links. For example, the current routing algorithms have given the K-shortest paths. We can simply select the one with the most DiffServ capable routers. But this is not efficient in computation. We give a single algorithm to solve the routing problem.

Our strategy is to find a path with minimum cost (shortest path), and when there are more than one shortest path, we choose the one with maximum number of DiffServ capable nodes (widest path).

Suppose that node 1 is the source node and h is the number of hops away from the source node. Let $D_i^{(h)}$ and $C_i^{(h)}$ be the number of DiffServ capable nodes and the total cost of the chosen widest-shortest path from node 1 to node i within h hops. Let S be the set of nodes whose optimal paths from the source node 1 have already been determined, and N-S the set of the re-

maining nodes. The following algorithm finds optimal paths from the source node 1 to all nodes in the network topology.

Algorithm 1:

Step 1 Initialization. Set $c_{ij} = \infty$ if there is no direct link from node i to node j. $dcapable_{ij} = 0$ if there is no direct link from node i to node j. For all other links, the values of c_{ij} and $dcapable_{ij}$ are assigned. Set $C_i^{(0)} = 0$ and $D_i^{(0)} = 0$, for all $i \neq 1$. $h = 0$ and $S = \{1\}$.

Step 2 For each $i \in N - S$, find the set K of all the nodes k such that $C(\overbrace{1, ..., k, i}^{h+2 \ nodes}) = min_{j \in N}\{C_j^{(h)} + c_{ji}\}$

Step 3 If $C(1, ..., k, i) \neq \infty$, $\forall k \in K$, find $k \in K$ such that $D(1, \ldots, k, i) = max_{j \in K}\{D_j^{(h)} + dcapable_{ji}\}$. Record the optimal path $(1, \ldots, k, i)$, and set $S = S \cup \{i\}$.

Step 4 $D_i^{(h+1)} = D(1, ..., k, i)$ and $C_i^{(h+1)} = C(1, ..., k, i)$.

Step 5 If $N - S \neq \phi$, set $h = h + 1$ and go to Step 2.

Step 2 finds all paths from node 1 to each node i with the minimum cost. If there are more than one such path found, Step 3 chooses the one with the maximum number of DiffServ capable nodes. Step 4 sets the values for later use. Step 5 checks to see if it is necessary to continue the loop.

Example. In the network topology depicted in Figure 4.1, node 2 is DiffServ incapable, nodes 1, 3, 4 and 5 are DiffServ capable. We assign a cost of 1 to each link in the network topology. We will show how the algorithm finds

optimal paths from the source node 1 to each of the nodes 2, 3, 4 and 5.

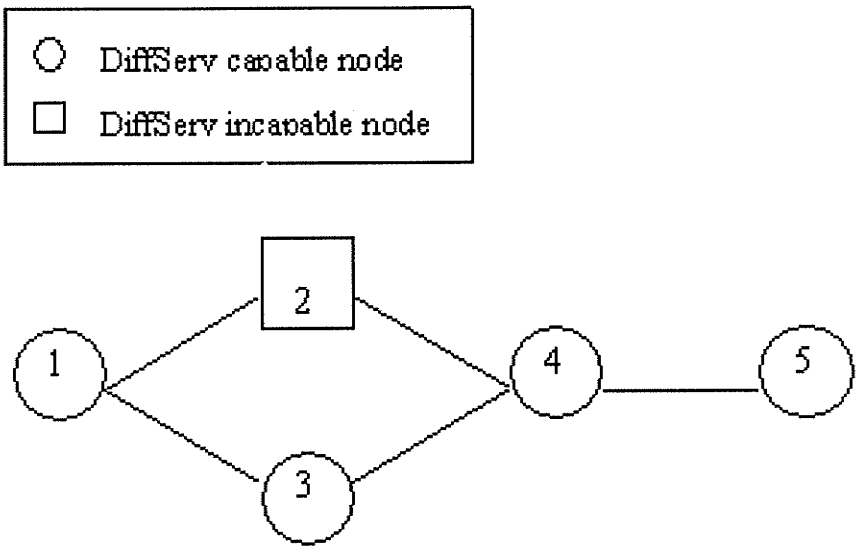


Figure 4.1: Example Network

Step 1 After initialization. We have $c_{12} = c_{13} = c_{24} = c_{34} = c_{45} = 1$, $c_{ij} = \infty$, for all other pairs (i, j), $i \neq j$.

$dcapable_{12} = dcapable_{13} = dcapable_{34} = dcapable_{45} = 1$, $dcapable_{24} = 0$, $dcapable_{ij} = 0$, for all other pairs (i, j), $i \neq j$.

$C_i^{(0)} = 0$ and $D_i^{(0)} = 0$, for all $i \neq 1$.

$h = 0$ and $S = \{1\}$.

Step 2 ($h = 0$, $i = 2$) Step 2 finds paths with minimum cost from 1 to each node i within h hop(s). In the algorithm, each node i goes through steps 2–4 separately.

Since $c_{12} = 1$, the path $(1, 2)$ has cost 1 and is the only path from 1 to 2 with minimum cost within 1 hop. $K = \{\phi\}$.

Step 3 ($h = 0$, $i = 2$) Since $C(1,2) = 1$, $(1,2)$ is the optimal path from 1 to 2.

Now $S = \{1,2\}$.

Step 4 ($h = 0$, $i = 2$) $D_2^{(1)} = 1$ and $C_2^{(1)} = 1$.

Step 2 ($h = 0$, $i = 3$) Since $c_{13} = 1$, the path $(1,3)$ has cost 1 and is the only path from 1 to 3 with minimum cost within 1 hop. $K = \{\phi\}$.

Step 3 ($h = 0$, $i = 3$) Since $C(1,3) = 1$, $(1,3)$ is the optimal path from 1 to 3.

Now $S = \{1,2,3\}$.

Step 4 ($h = 0$, $i = 3$) $D_3^{(1)} = 1$ and $C_3^{(1)} = 1$.

Step 2 ($h = 0$, $i = 4$) Since $c_{14} = \infty$, the path $(1,4)$ has cost ∞. That is, $(1,4)$ is not a physical link in the topology.

Step 3 ($h = 0$, $i = 4$) No optimal path from 1 to 4 has been found within 1 hop.

Step 4 ($h = 0$, $i = 4$) $D_4^{(1)} = 0$ and $C_4^{(1)} = \infty$.

Step 2 ($h = 0$, $i = 5$) Since $c_{15} = \infty$, the path $(1,5)$ has cost ∞.

Step 3 ($h = 0$, $i = 5$) No optimal path from 1 to 5 has been found within 1 hop.

Step 4 ($h = 0$, $i = 5$) $D_5^{(1)} = 0$ and $C_5^{(1)} = \infty$.

Step 5 ($h = 0$) Since $N - S = \{4,5\} \neq \phi$, continue with $h = 1$.

Step 2 ($h = 1$, $i = 4$) There are two paths from 1 to 4 within 2 hops: $(1, 2, 4)$ and $(1, 3, 4)$. Both of these two paths have a minimum cost of 2. $K = \{2, 3\}$.

Step 3 ($h = 1$, $i = 4$) Since $d_{34} = 1$ and $d_{24} = 0$, the only optimal path is $(1, 3, 4)$. Record the path $(1, 3, 4)$.

Now $S = \{1, 2, 3, 4\}$.

Step 4 ($h = 1$, $i = 4$) $D_4^{(2)} = 2$ and $C_4^{(2)} = 2$.

Step 2 ($h = 1$, $i = 5$) All paths from 1 to 5 within 2 hops have cost ∞.

Step 3 ($h = 1$, $i = 5$) No optimal path from 1 to 5 has been found within 2 hops.

Step 4 ($h = 1$, $i = 5$) $D_5^{(2)} = 0$ and $C_5^{(2)} = \infty$.

Step 5 ($h = 1$) Since $N - S = \{5\} \neq \phi$, continue with $h = 2$.

Step 2 ($h = 2$, $i = 5$) Since $c_{45} = 1$ and $(1, 3, 4)$ is the only optimal path from 1 to 4, $(1, 3, 4, 5)$ is the only possible path from 1 to 5 considered by the algorithm. Further calculation finds that $(1, 3, 4, 5)$ is the only path from 1 to 5 with minimum cost within 3 hops. $K = \{4\}$.

Step 3 ($h = 2$, $i = 5$) Since $C(1, 3, 4, 5) = 3$, $(1, 3, 4, 5)$ is the optimal path from 1 to 5.

Now $S = \{1, 2, 3, 4, 5\}$.

Step 4 ($h = 2$, $i = 5$) $D_5^{(3)} = 3$, and $C_5^{(3)} = 3$.

Step 5 ($h = 2$) Since $N - S = \phi$, stop.

The complexity of the algorithm is equal to those of the traditional widest-shortest algorithm and shortest path algorithm. The algorithm is scalable.

4.3.3 Algorithm 2

In this section, we give an algorithm to find the optimal path when DiffServ capability is given higher precedence over cost.

Our strategy is to find a path with maximum number of DiffServ capable nodes (widest path), and when there are more than one widest path, we choose the one with minimum cost (shortest path).

Suppose that node 1 is the source node and h is the number of hops away from the source node. Let $D_i^{(h)}$ and $C_i^{(h)}$ be the number of DiffServ capable nodes and the total cost of the chosen shortest-widest path from node 1 to node i within h hops. Let S be the set of nodes whose optimal paths from the source node 1 have already been determined, and N-S the set of the remaining nodes. The following algorithm finds optimal paths from the source node 1 to all nodes in the network topology.

Algorithm 2:

Step 1 Initialization. Set $c_{ij} = \infty$ if there is no direct link from node i to node j. $dcapable_{ij} = 0$ if there is no direct link from node i to node j. For all other links, the values of c_{ij} and $dcapable_{ij}$ are assigned. $C_i^{(0)} = 0$ and $D_i^{(0)} = 0$, for all $i \neq 1$. $h = 0$ and $S = \{1\}$.

Step 2 For each $i \in N - S$, find the set K of all the nodes k such that $D(\overbrace{1, ..., k, i}^{h+2 \ nodes}) = max_{j \in K}\{D_j^{(h)} + dcapable_{ji}\}$

Step 3 If K has more than one element, find $k \in K$ such that $C(1,\ldots,k,i) = min_{j \in N}\{C_j^{(h)} + c_{ji}\}$. If K has only one element k, $(1,\ldots,k,i)$ is optimal. Record the optimal path $(1,\ldots,k,i)$, and set $S = S \cup \{i\}$.

Step 4 $D_i^{(h+1)} = D(1,...,k,i)$ and $C_i^{(h+1)} = C(1,...,k,i)$

Step 5 If $N - S \neq \phi$, set $h = h + 1$ and go to Step 2.

Step 2 finds all paths from node 1 to each node i with the maximum number of DiffServ capable nodes. If there are more than one such path found, Step 3 chooses the one with the minimum cost. Step 4 sets the values for later use. Step 5 checks to see if it is necessary to continue the loop.

The complexity of the algorithm is equal to those of the traditional shortest-widest algorithm and widest path algorithm. The algorithm is scalable.

4.3.4 Algorithm 3 – A Different Approach

In the first two algorithms, the two metrics are independent. The first algorithm treats cost as the more important metric. It can find the optimal path in most network topologies. But in some cases, the found path may not be the best path. For example, for the network topology in Figure 4.2, the path with the lower cost is not the optimal path for DiffServ.

The second algorithm treats DiffServ capable as the more important metric. Again it can find the optimal path in most network topologies. But in some special cases, the found path may not be the best path. For example, in the network topology depicted in Figure 4.3, apparently, the found path is not optimal in the overall topology.

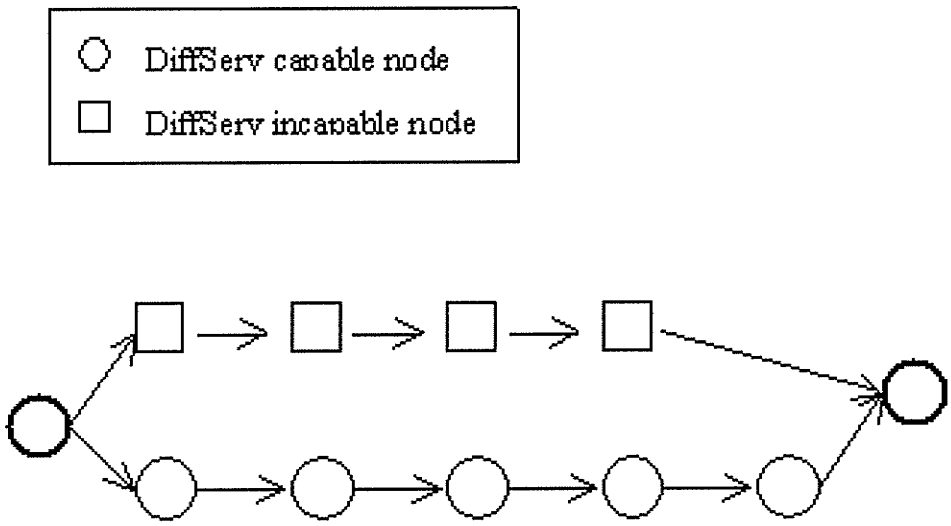


Figure 4.2: Network 1

If most of the nodes are DiffServ capable, choosing DiffServ as the precedence is a good choice. If most of the nodes are DiffServ incapable, it is better to choose cost as the precedence. If all nodes are DiffServ capable, it does not matter which metric is chosen to be the precedence. They have the same result. Neither of the two algorithms performs well in all network topologies. This prompts us to find a better algorithm for DiffServ in such a network.

In this section, we will assign a DiffServ route selection order number to each link. The order number is a real number. Usually a DiffServ capable link has an order number smaller than that of a DiffServ incapable link. The order number can be different in each area (or AS) according to the user's request and the service charging policy. The order number can also be changed by the Internet Service Provider (ISP) to reflect the service charging

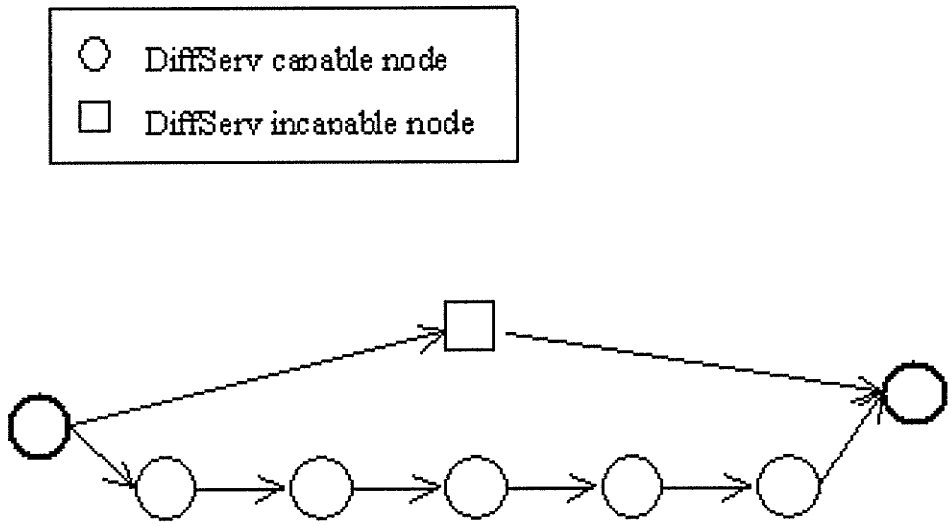


Figure 4.3: Network 2

policy.

The algorithm we give in this section is for source routing. It is an extended Dijkstra's algorithm. One generic aspect of the algorithmic complexity of computing QoS paths is the efficiency of the algorithm used. Dijkstra's algorithm has traditionally been considered more efficient for standard shortest path computations because of its lower worst case complexity.

Suppose the network topology is $T = (N, L)$, where N is a set of DiffServ capable nodes and DiffServ incapable nodes, and L is a set of links. Each link $e \in L$ is assigned a real number w_e, called the DiffServ route selection order number of the link.

We define the DiffServ route selection order number $o(p)$ of a path $p = (1, 2, ..., i)$ to be the sum of the DiffServ route selection order number of each

link along the path. That is,

$$o(p) = w_{1,2} + w_{2,3} + \cdots + w_{i-1,i}$$

For DiffServ, the desired path is the one with the lowest DiffServ route selection order number.

Let S be the set of nodes whose optimal paths from the source have already been determined, and $N - S$ be the set of the remaining nodes. Let o denote the array of the DiffServ route selection order numbers of the best estimates of optimal paths to each node, and r an array of predecessors for each node. Q is a priority queue used to store the network topology along with the DiffServ route selection order numbers. w is the DiffServ route selection order numbers of the links.

```
InitialiseSingleSource(NetworkTopology T, Node s)
   for each node v in nodes(T)
       T.o[v] = infinity
       T.r[v] = nil
       T.o[s] = 0;

DSR_Dijkstra(NetworkTopology T, SelectionOrder w, Node s)
    // s is the source node
    InitialiseSingleSource(T, s)
    S = {0}              // Initialise S to empty
    Q = nodes(T)         // Store the nodes in Q
    while not Empty(Q)
```

```
        u = ExtractLowestOrder(Q);
        AddNode(S, u);  // Add the lowest order node u to S
        for each node v in Adjacent(u)  // v is an adjacent node of u
            Relax(u, v, w)

Relax(Node u, Node v, double w[][])
    if o[v] > o[u] + w[u,v] then
       o[v] = o[u] + w[u,v]    // Replace the previous order
       r[v] = u    // u is now the predecessor
```

The above procedures can be combined to give the following easy to understand algorithm.

Algorithm 3:

```
DSR_Dijkstra(NetworkTopology T, SelectionOrder w, Node s) {
    initialize o(u) for all u to infinity
    initialize o(s) to 0
    initialize Q with all u using o(u) as keys
    while (Q is not empty) {
        u = Q.extractLowestOrder()
        for each node v that is adjacent to u {
            if o(v) > o(u) + w(u,v) {
                o(v) = o(u) + w(u,v)
                r(v) = u
```

```
            }
        }
    }
}
```

4.4 Implementation Issues

In order to deploy source routing, one must implement a link state type of routing algorithm (e.g., OSPF). With link state, every router in the network can maintain a local copy of the entire network topology and resource utilization information. This prevents the scheme from being scalable since the size of network information increases sharply as the network size grows.

Thus, large networks are usually composed hierarchically of several domains (AS or areas in OSPF networks), and hierarchical source routing is considered as the most promising scalable QoS routing approach. In hierarchical QoS routing, network topology and resource information about a specific domain are summarized before being exchanged with other domains. This process is called topology aggregation [7, 13, 17].

The currently used version of OSPF does not support QoS. OSPF routes IP packets based solely on the destination IP address found in the IP packet header. The paper [14] gives an approach to provide QoS support while imposing the least possible impact on the existing OSPF protocol. The reader is referred to [14] for more details.

None of the existing routing protocols has the ability to identify whether a router is DiffServ capable or not. In the simulations in ns, we distinguish

the routers by assigning different DiffServ route selection order numbers to the routers. In a real network, however, we need a routing protocol that is capable of identifying if a router is DiffServ capable or not.

Chapter 5

Simulations

5.1 Introduction to ns

The Network Simulator (ns) [1] has been developed at the Lawrence Berkeley National Laboratory (LBNL) of the University of California, Berkeley (UCB), for network research. The current version of ns is generally referred to as ns-2 (Network Simulator Version 2). ns has an extensible background engine implemented in C++ that uses OTcl (an object oriented version of Tcl) as the front end (command and configuration interface etc.). It is a discrete event-driven simulator that derives its functionality through an OTcl interpreter, which runs in the background.

A simulation is defined by a Tcl script. Running a simulation involves creating and executing a file with a ".tcl" extension. A Tcl script file:

1. Defines a network topology (including the nodes, links, scheduling and routing algorithms of a network).

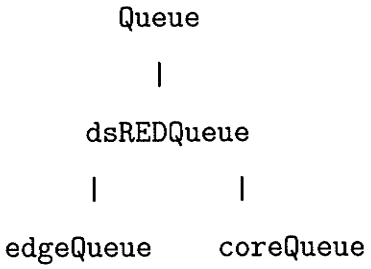
2. Defines a traffic pattern (including the start and stop time of the simulation).

3. Collects statistics and outputs the results of the simulation. Results are usually written to files, including files for Nam [2], the Network Animator program that comes with the full ns download.

We use ns-2.1b8 for our simulation in this chapter.

5.1.1 DiffServ Support in ns

The Differentiated Services module that comes with ns is provided by Nortel Networks. The DiffServ functionality is captured in a queue object. The position of dsREDQueue in the class hierarchy is shown below:



5.1.2 Routing in ns

There are three routing strategies in ns: Static, Session and Dynamic. Static and Session routing use Dijkstra's all-pairs SPF algorithm. One type of dynamic routing is currently implemented: Distributed Bellman-Ford algorithm, which is the algorithm implemented by distance-vector routing protocols such as RIP [59].

In ns, we blur the distinction between strategy and protocol for static and session routing, considering them simply as protocols.

There is no specific routing algorithm specified or implemented for DiffServ. A DiffServ capable router acts according to the code point. A DiffServ incapable router simply ignores the code point. Routing strategies in ns apply the same rules to all packets, whether they are from a DiffServ capable node or not.

5.2 Adding Our New Module to ns

Our algorithm (Algorithm 3) is implemented as an additional module to ns-2 [1]. In this section, we show how our DiffServ Routing Module is added to ns. Adding the module to ns consists of the following steps:

`Step 1` Creating the header file "rtDSRouting.h". This file includes class specifications, as well as other definitions needed by the new class.

`Step 2` Creating the main C++ file "rtDSRouting.cc". This file includes implementations of each of the new class's methods and other routines defined in the header file. To incorporate the new class into ns and make it accessible through Tcl scripts, the class must be linked to the ns class hierarchy. The following code is used in "rtDSRouting.cc" to add DSRAlgorithmClass to the class hierarchy:

```
static class DSRAlgorithmClass : public TclClass {
public:
        DSRAlgorithmClass() : TclClass("DSRAlgorithm") {}
```

```
        TclObject* create(int, const char*const*) {
                return (new DSRAlgorithm);
        }
} class_dsralgorithm;
```

Step 3 Modifying the "Make" file. In order for the new class to be included in the ns compilation, a reference to the new module must be added to the file "Makefile." We add the following line

rtDSRouting.o

to the object files section of "Makefile."

Step 4 Make a new version of ns with the new module. After recompiling ns with the make command, Tcl scripts can use the new class. To declare an instance of the new module, use the following Tcl command:

set dsr [new DSRAlgorithm]

5.3 Other Modifications to ns

We also modify the Source Routing Agent Module that comes with ns. The command method defined in the file sragent.cc (source files for source routing are in the subdirectory src_rtg under the main ns directory) treats every argument passed to it as a node name. When we get the optimal path

from the DiffServ Routing Algorithm Module, we must pass the complete path as one argument. Thus we must add code to the command method to accommodate our simulation. Prototype of the command method is shown below:

```
int SRAgent::command(int argc, const char*const* argv)
```

5.4 Simulations

Simulation scripts are written in Tcl. Each script defines a topology, sets attributes of network devices, defines parameters, and describes network events during a period of time frame.

Three topologies are constructed, all of them with a single traffic source. This source sends packets at a constant bit rate (CBR). By convention, the CBR source is set at node 0 (source node) sending packets to the last node (destination node) in the topology.

5.4.1 Network Topology One

The first network topology we use for simulations is shown in Figure 5.1. This topology has 18 nodes, including 5 traditional nodes or DiffServ incapable nodes. Each link has a speed of 1Mbps and a delay of 2ms.

The source node and destination node (hexagon) are supposed to be DiffServ capable. Since DiffServ is captured in queues, DiffServ capable nodes are either DiffServ capable edge routers or DiffServ capable core routers in ns. Packet marking is done by a DiffServ capable edge router, so the source node

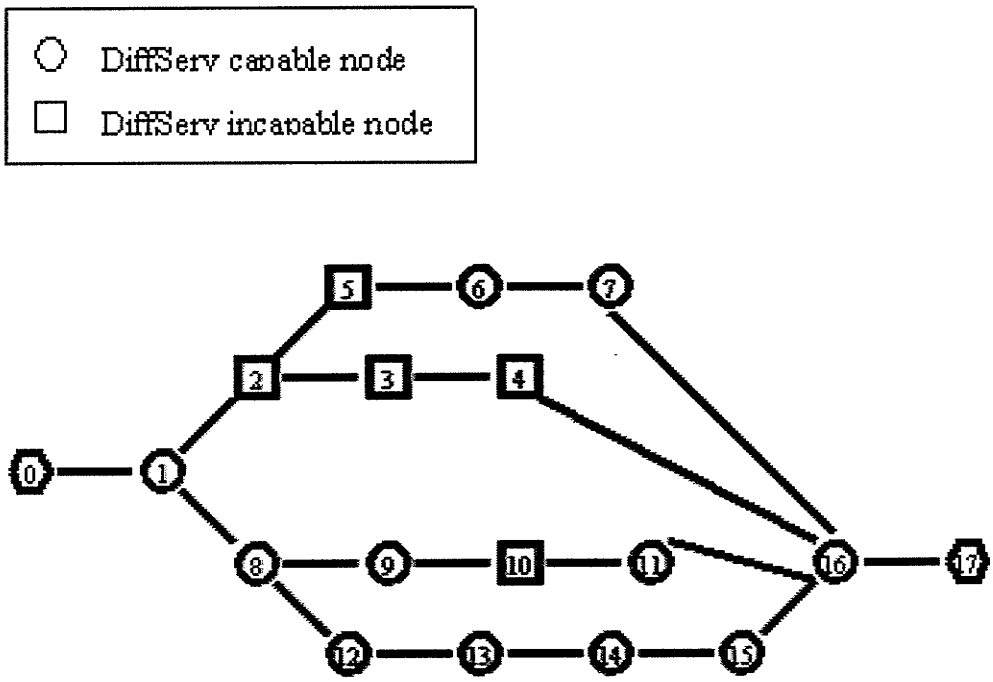


Figure 5.1: Network 3

and destination node can be modelled as traditional nodes. They support DiffServ through the DiffServ capable edge routers.

The source node starts sending at the beginning of the simulation and stops at time 2.5 sec.

We examine some lines of the Tcl script and explain what they do below:

```
set ns [new Simulator]
```

```
$ns src_rting 1
```

This creates a Simulator instance and enables source routing in the sim-

ulation.

```
set dsr [new DSRAlgorithm]

proc DSRCompletePath {ReturnedPath} {
    global gstrPath
    set gstrPath $ReturnedPath
}

set cpble 1
set incpble 5
set infinity 100
```

dsr is an instance of our DiffServ Routing Module, which must be created in the Tcl script. Through dsr, we can use the DiffServ Routing Algorithm to calculate the optimal path for DiffServ in the simulation. If called with the procedure *DSRCompletePath*, the calculated optimal path will be stored in the global variable *gstrPath*. *gstrPath* is accessible in the whole script.

The DiffServ route selection order number are declared here. To simplify the simulation, all DiffServ capable nodes are assigned the same route selection order number (cpble), and DiffServ incapable nodes are assigned another route selection order number (incpble). If the value of cpble or incpble changes, the calculated optimal path will change.

The variable infinity is used to indicate that a link is down.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

```
proc finish {} {
    global ns nf
    $ns flush-trace

    #Close the trace file
    close $nf

    #Execute nam on the trace file
    puts "\nStarting nam..."
    exec nam out.nam &

    exit 0
}
```

We record the simulation results in the file out.nam for the Network Animator program Nam [2]. The procedure finish is called at the end of the simulation.

```
set src [$ns node]
set e_src [$ns node]
set r1 [$ns node]
set r2 [$ns node]
set r3 [$ns node]
set r4 [$ns node]
set c5 [$ns node]
```

```
set c6 [$ns node]
set c7 [$ns node]
set c8 [$ns node]
set r9 [$ns node]
set c10 [$ns node]
set c11 [$ns node]
set c12 [$ns node]
set c13 [$ns node]
set c14 [$ns node]
set e_dst [$ns node]
set dst [$ns node]

$src shape "hexagon"
$dst shape "hexagon"
$r1 shape "box"
$r2 shape "box"
$r3 shape "box"
$r4 shape "box"
$r9 shape "box"

$ns duplex-link $r1 $r2 1Mb 2ms DropTail
$ns simplex-link $e_src $c7 1Mb 2ms dsRED/edge
$ns simplex-link $c7 $c11 1Mb 2ms dsRED/core
$ns simplex-link $c14 $e_dst 1Mb 2ms dsRED/core
```

The above lines of code set up the network topology, set the link speed

and delay, and declare if a node is DiffServ capable or not. dsRED/edge indicates that the first node in the link is a DiffServ capable edge router. dsRED/core indicates that the first node in the link is a DiffServ capable core router.

```
set qESC7 [[$ns link $e_src $c7] queue]
set qC7C8 [[$ns link $c7 $c8] queue]
set qC14ED [[$ns link $c14 $e_dst] queue]
```

Since DiffServ is captured in queues in ns, we must set queues corresponding to the DiffServ core routers and edge routers.

Other parameters from one node to another must also be set before the simulation begins. We set the packet size, policy and other parameters as follows:

```
$qESC7 meanPktSize $packetSize
$qESC7 set numQueues_ 1
$qESC7 setNumPrec 2
$qESC7 addPolicyEntry [$src id] [$dst id] TokenBucket 10 $cir0 $cbs0
$qESC7 addPolicerEntry TokenBucket 10 11
$qESC7 addPHBEntry 10 0 0
$qESC7 addPHBEntry 11 0 1
$qESC7 configQ 0 0 20 40 0.02
$qESC7 configQ 0 1 10 20 0.10
```

To call the DiffServ Routing Algorithm, we pass the node ids and their DiffServ route selection order number to the C++ method as shown below:

```
set nt "[$src id] [$e_src id] $cpble:[$e_src id] [$r1 id] $cpble:
    [$r1 id] [$r2 id] $incpble:[$r2 id] [$r3 id] $incpble:
    [$r3 id] [$e_dst id] $incpble:[$e_dst id] [$dst id] $cpble:
    [$r1 id] [$r4 id] $incpble:[$r4 id] [$c5 id] $incpble:
    [$c5 id] [$c6 id] $cpble:[$c6 id] [$e_dst id] $cpble:
    [$e_src id] [$c7 id] $cpble:[$c7 id] [$c8 id] $cpble:
    [$c8 id] [$r9 id] $cpble:[$r9 id] [$c10 id] $incpble:
    [$c10 id] [$e_dst id] $cpble:[$c7 id] [$c11 id] $cpble:
    [$c11 id] [$c12 id] $cpble:[$c12 id] [$c13 id] $cpble:
    [$c13 id] [$c14 id] $cpble:[$c14 id] [$e_dst id] $cpble"
$dsr GetDSRCompeletePath [$src id] [$dst id] $nt
puts "The complete path is: $strPath \n"
```

The last line prints out the complete path returned from the C++ method.

When a link in the topology goes down at a specific time, the module is called to recalculate the new route. If the route is not updated, the source continues to send traffic on a failed route. Some packets will be lost, at a quantity proportional to the time the routes are not updated, since the traffic generated is CBR. A DiffServ capable node ia able to adapt to the network changes.

```
$temp install_connection [$udp0 set fid_] [$src id] [$dst id]
    $strPath
```

This line of code installs the returned complete path for source routing. Once the path is installed, all traffic will use the installed path.

At the end of the simulation, Nam [2] is executed on the trace file out.nam.

Optimal paths returned from the algorithm in the simulations along with the corresponding values of cpble and incpble are shown below:

cpble	incpble	Optimal Path
1	5	(0, 1, 8, 12, 13, 14, 15, 16, 17)
2	5	(0, 1, 8, 12, 13, 14, 15, 16, 17)
3	5	(0, 1, 8, 9, 10, 11, 16, 17)
4	5	(0, 1, 2, 3, 4, 16, 17)
5	5	(0, 1, 2, 3, 4, 16, 17)

cpble	incpble	Optimal Path
1	9	(0, 1, 8, 12, 13, 14, 15, 16, 17)
2	9	(0, 1, 8, 12, 13, 14, 15, 16, 17)
3	9	(0, 1, 8, 12, 13, 14, 15, 16, 17)
4	9	(0, 1, 8, 12, 13, 14, 15, 16, 17)
5	9	(0, 1, 8, 9, 10, 11, 16, 17)
6	9	(0, 1, 2, 3, 4, 16, 17)
7	9	(0, 1, 2, 3, 4, 16, 17)
8	9	(0, 1, 2, 3, 4, 16, 17)
9	9	(0, 1, 2, 3, 4, 16, 17)

The following are some snapshots of the simulation with $cpble = 1$ and $incpble = 5$.

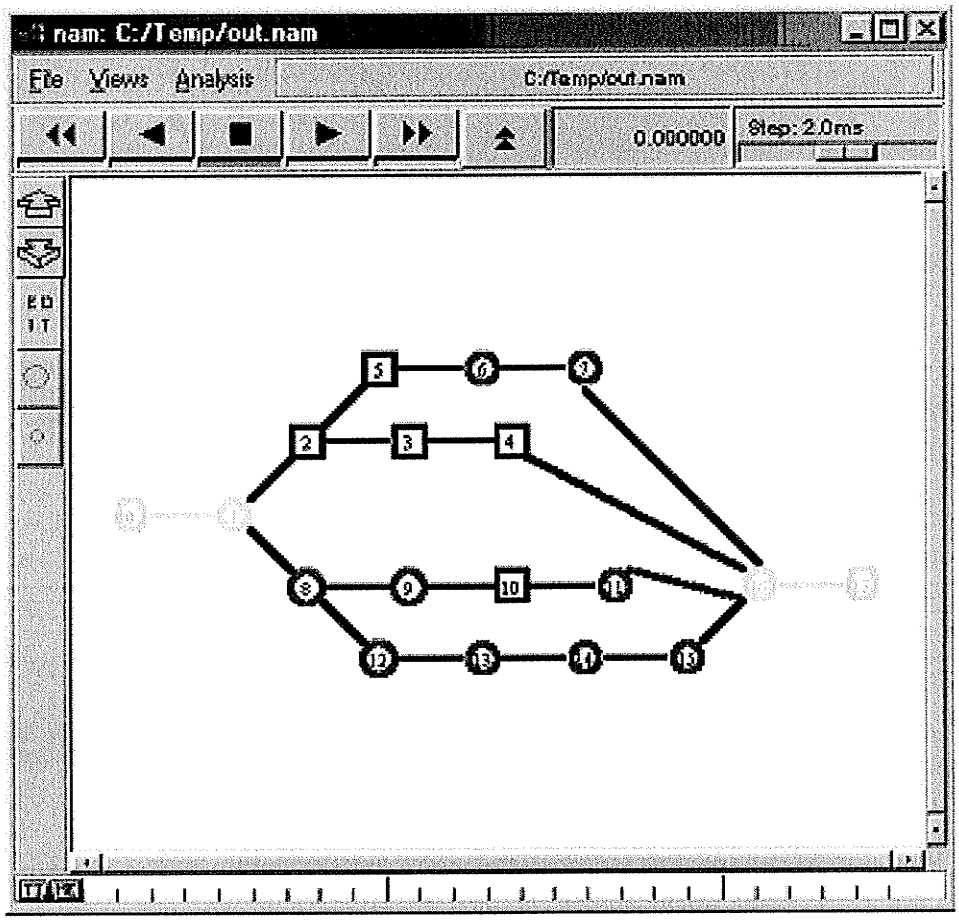


Figure 5.2: Layout

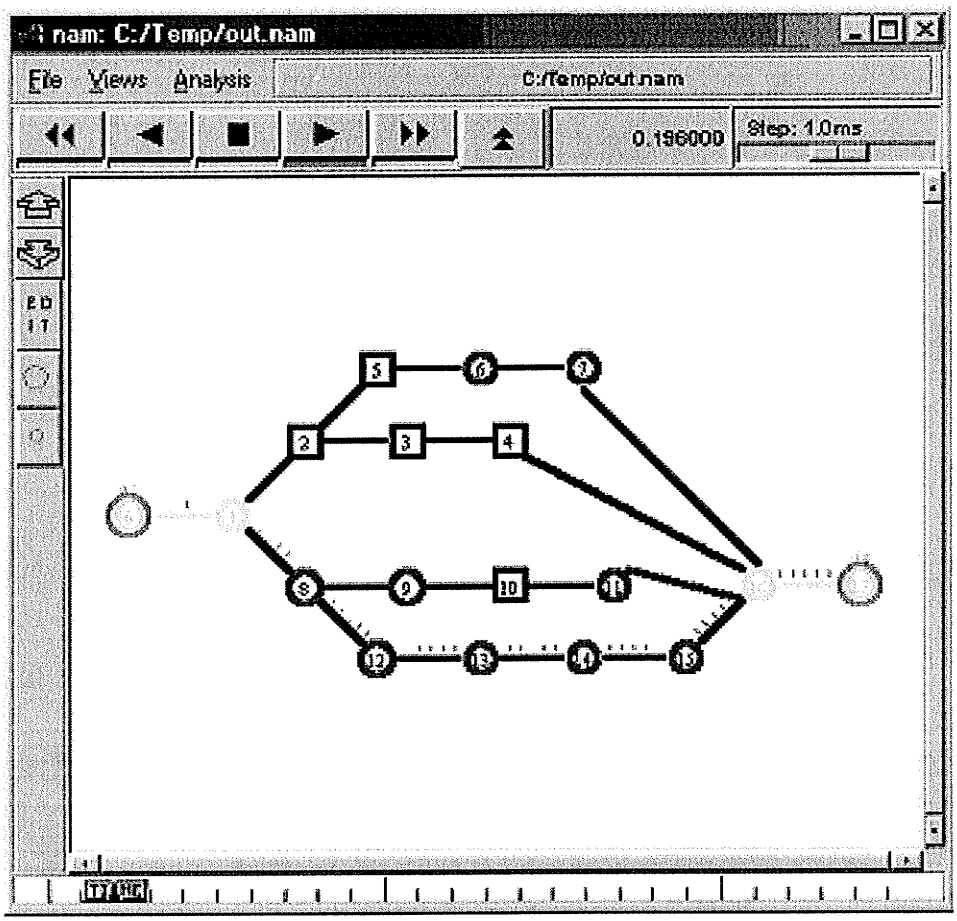


Figure 5.3: Snapshot at 0.196s

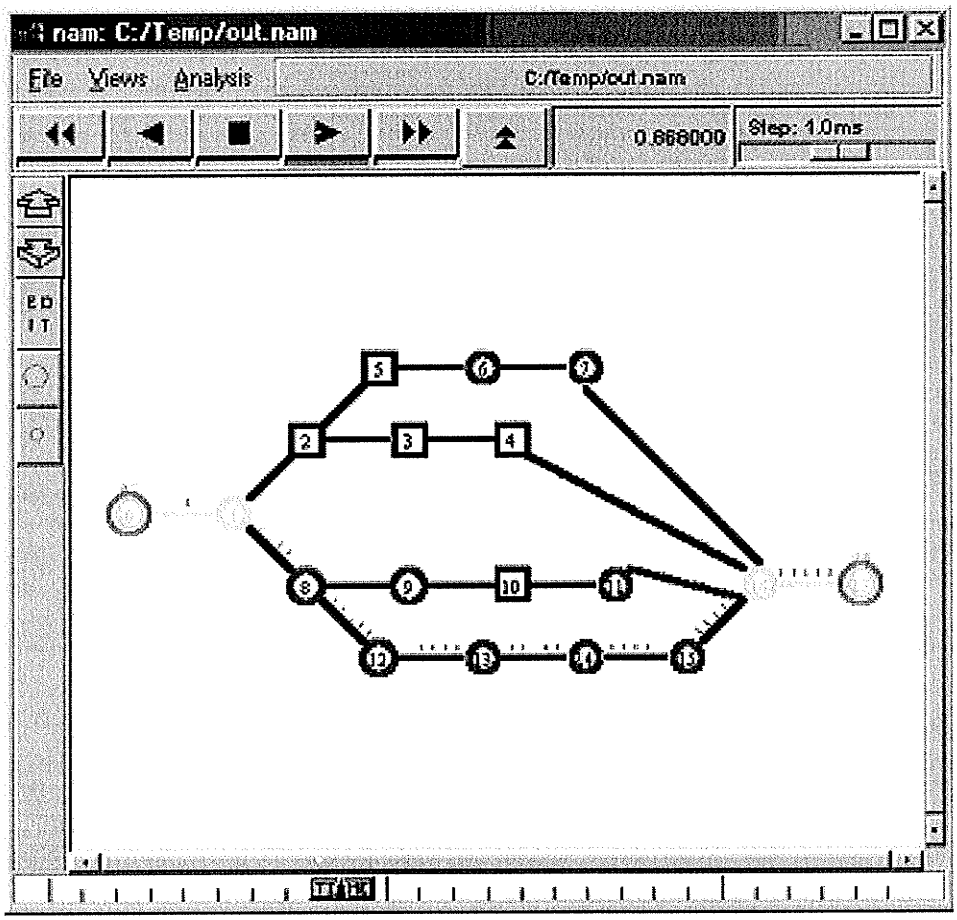


Figure 5.4: Snapshot at 0.868s

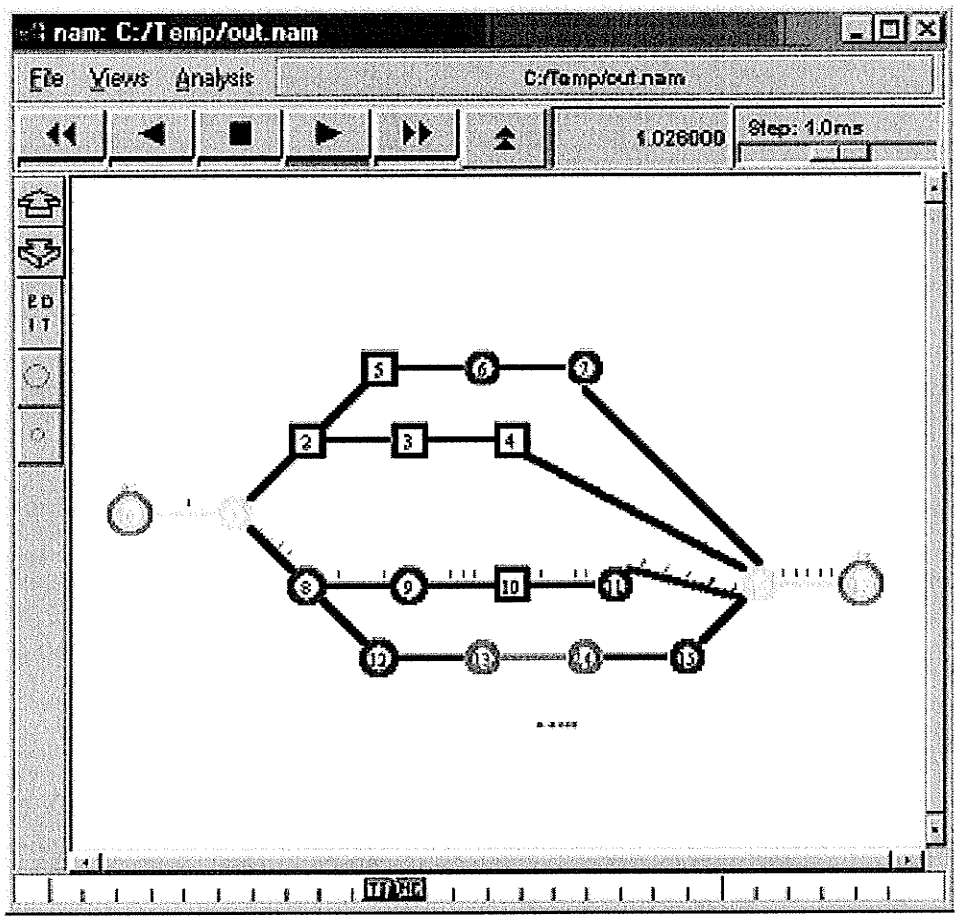


Figure 5.5: Snapshot at 1.026s. A link is down. New route is used.

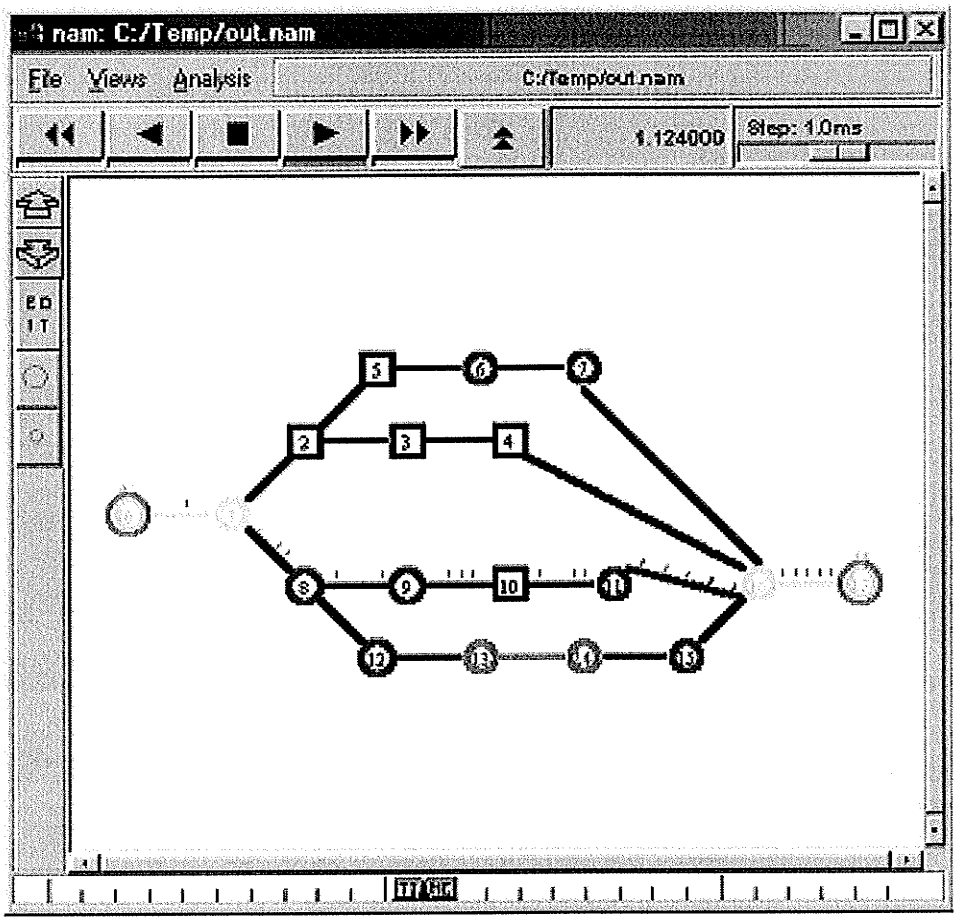


Figure 5.6: Snapshot at 1.124s

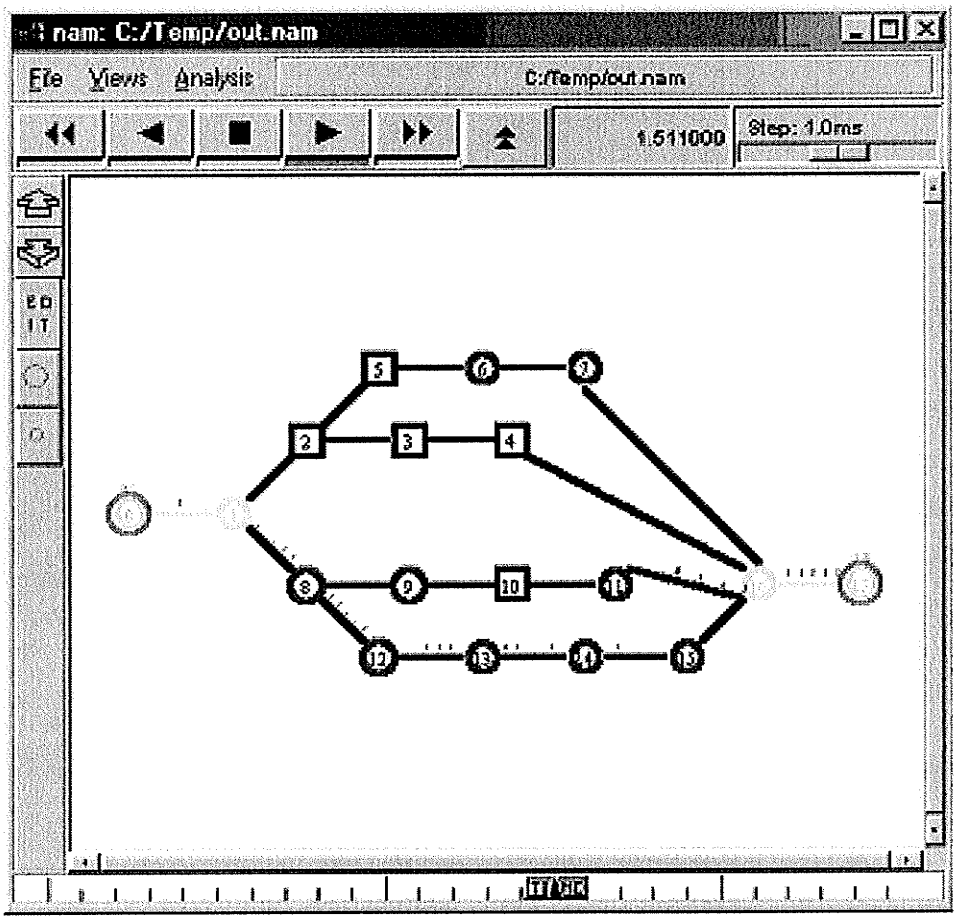


Figure 5.7: Snapshot at 1.511s. The link is back. The previous route is used.

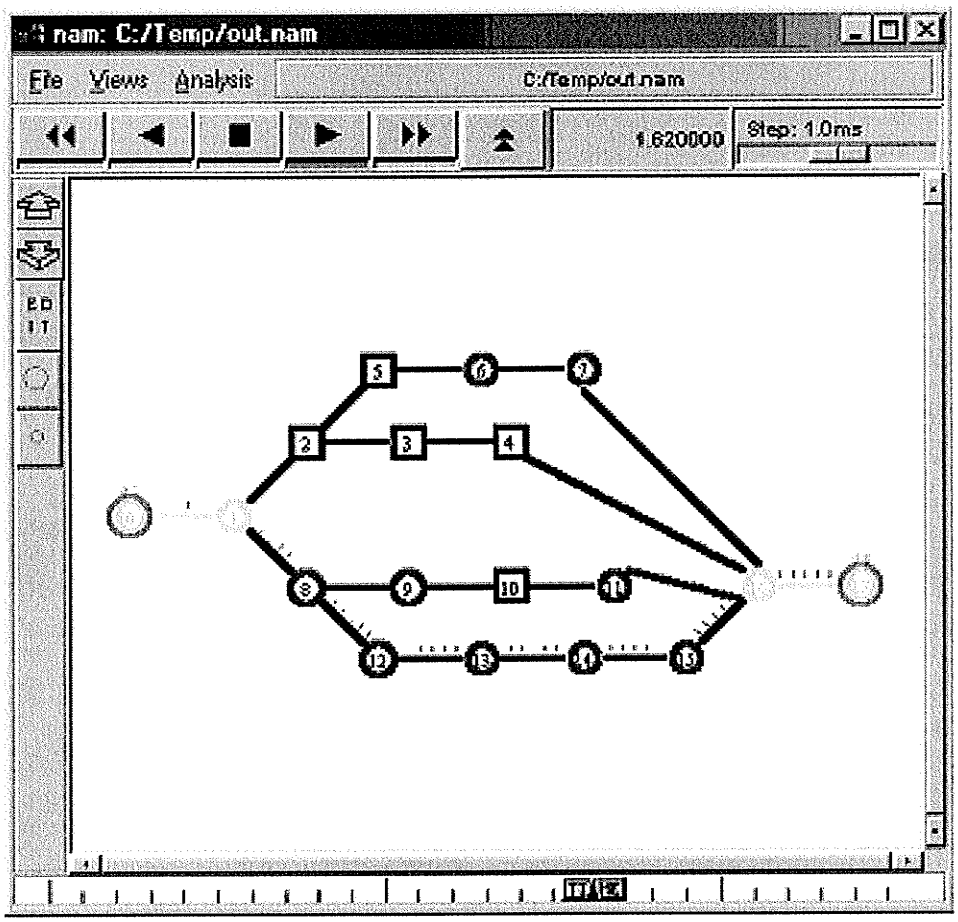


Figure 5.8: Snapshot at 1.620s

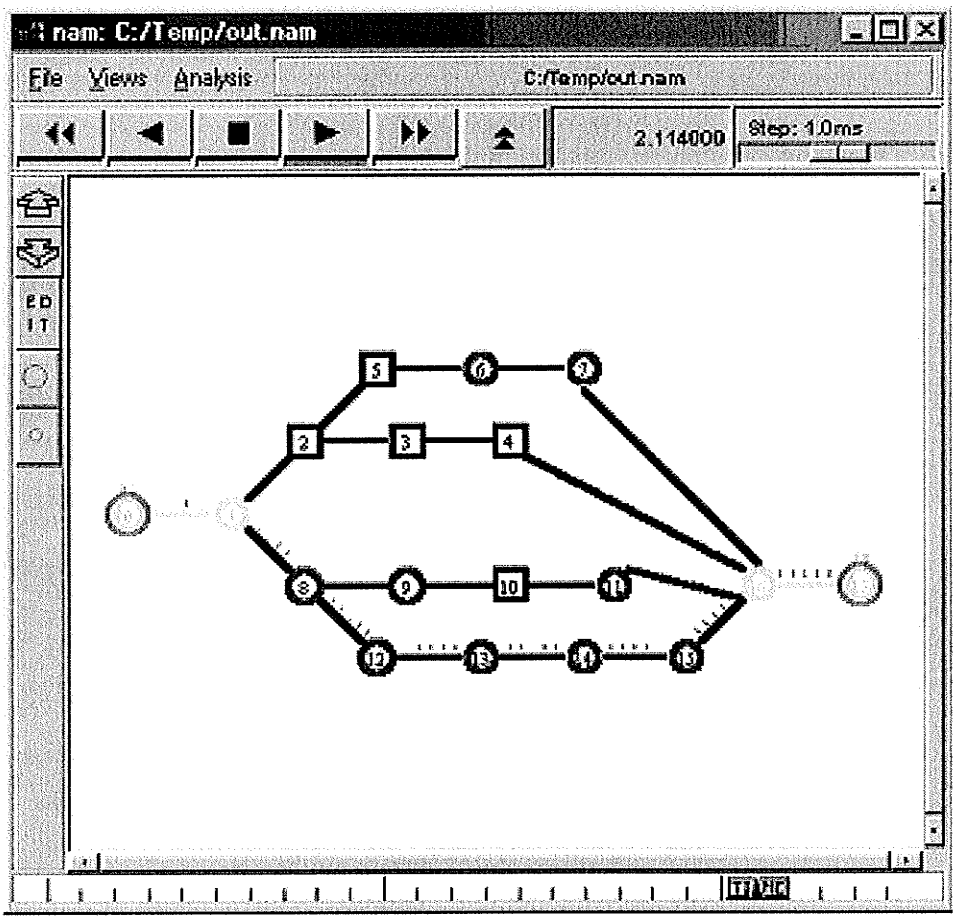


Figure 5.9: Snapshot at 2.114s

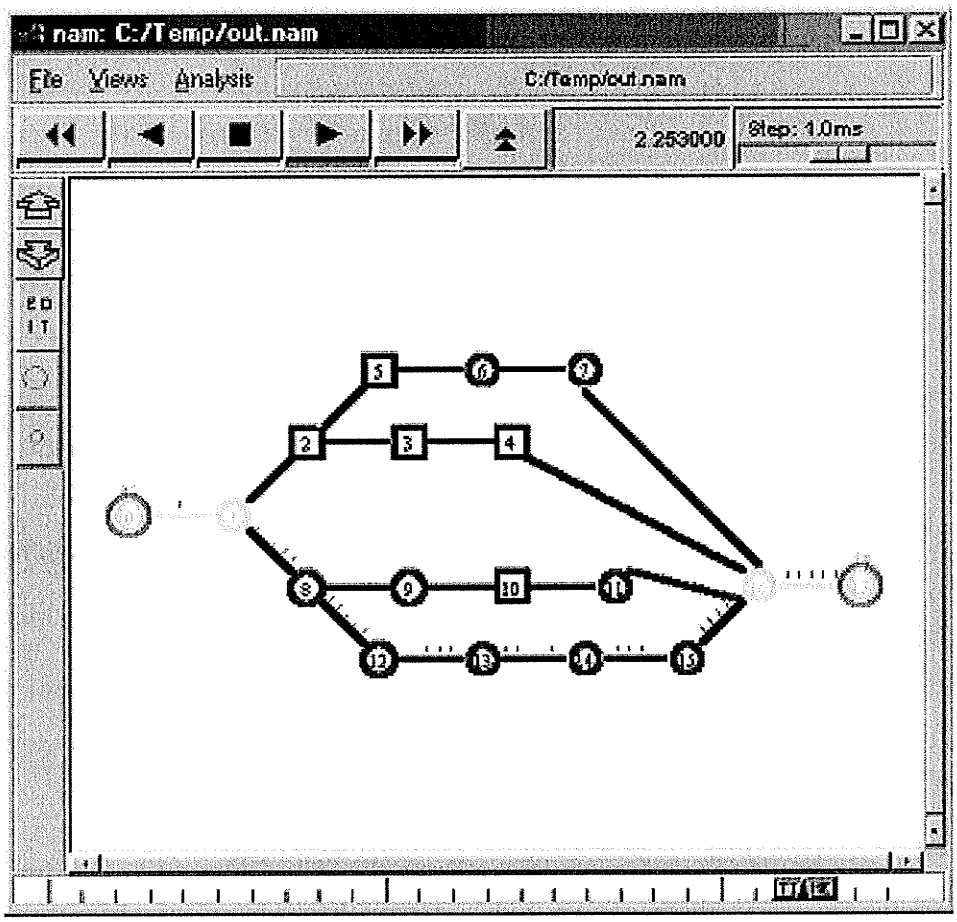


Figure 5.10: Snapshot at 2.253s

5.4.2 Network Topology Two

The second network topology we use for our simulations is shown in Figure 4.2. In ns, DiffServ capable nodes are either DiffServ capable edge routers or DiffServ capable core routers. We add a sending host (source, node 0) and a receiving host (destination, node 12) to the above network topology. Two DiffServ capable edge routers are attached directly to them. The resulting topology has 13 nodes. If Algorithm 1 is used to find the optimal routing path in this topology, the found path will be the one with four traditional routers (represented by squares). Obviously, the best path for DiffServ is the one with five DiffServ capable routers (represented by circles) in this network topology. So it is better to use our Differentiated Services Routing Algorithm (Algorithm 3) to find the optimal path.

As in the last simulation, all DiffServ capable nodes are assigned the same route selection order number (cpble), and all DiffServ incapable nodes are assigned another route selection order number (incpble). In the Tcl script, we can set different values for the variables *cpble* and *incpble* to achieve different paths.

The following are optimal paths returned from the algorithm in the simulations along with the corresponding values of cpble and incpble:

cpble	incpble	Optimal Path
1	5	(0, 1, 6, 7, 8, 9, 10, 11, 12)
2	5	(0, 1, 6, 7, 8, 9, 10, 11, 12)
3	5	(0, 1, 6, 7, 8, 9, 10, 11, 12)
4	5	(0, 1, 2, 3, 4, 5, 11, 12)
5	5	(0, 1, 2, 3, 4, 5, 11, 12)

cpble	incpble	Optimal Path
1	9	(0, 1, 6, 7, 8, 9, 10, 11, 12)
2	9	(0, 1, 6, 7, 8, 9, 10, 11, 12)
3	9	(0, 1, 6, 7, 8, 9, 10, 11, 12)
4	9	(0, 1, 6, 7, 8, 9, 10, 11, 12)
5	9	(0, 1, 6, 7, 8, 9, 10, 11, 12)
6	9	(0, 1, 6, 7, 8, 9, 10, 11, 12)
7	9	(0, 1, 6, 7, 8, 9, 10, 11, 12)
8	9	(0, 1, 2, 3, 4, 5, 11, 12)
9	9	(0, 1, 2, 3, 4, 5, 11, 12)

5.4.3 Network Topology Three

The third network topology we use for our simulations is shown in Figure 4.3. We add a sending host (source, node 0) and a receiving host (destination, node 9) to the above network topology. The resulting topology has 10 nodes. If Algorithm 2 is used to find the optimal routing path in this

topology, the found path will be the one with five DiffServ capable routers (represented by circles). Apparently, the best path is the one with only one traditional router (represented by a square) in this network topology. Again, it is better to use our Differentiated Services Routing Algorithm (Algorithm 3) to find the optimal path.

As usual, all DiffServ capable nodes are assigned the same route selection order number (cpble), and all DiffServ incapable nodes are assigned another route selection order number (incpble). In the Tcl script, we can set different values for the variables *cpble* and *incpble* to achieve different paths. This can be controlled by the ISPs according the subscribed service agreement.

The following are optimal paths returned from the algorithm in the simulations along with the corresponding values of cpble and incpble:

cpble	incpble	Optimal Path
1	6	(0, 1, 3, 4, 5, 6, 7, 8, 9)
2	6	(0, 1, 2, 8, 9)
3	6	(0, 1, 2, 8, 9)
4	6	(0, 1, 2, 8, 9)
5	6	(0, 1, 2, 8, 9)

cpble	incpble	Optimal Path
1	20	(0, 1, 3, 4, 5, 6, 7, 8, 9)
2	20	(0, 1, 3, 4, 5, 6, 7, 8, 9)
3	20	(0, 1, 3, 4, 5, 6, 7, 8, 9)
4	20	(0, 1, 2, 8, 9)
5	20	(0, 1, 2, 8, 9)

5.5 Data Structures and Prototypes in the Implementation

In this section, we list some of the main data structures and method prototypes in our implementation of the Differentiated Services Routing Algorithm on the Network Simulator.

```
class DSR_Node
{
public:
    int intName;
    int intOrder;
    int intParent;
    DSR_Node *next;

    DSR_Node(int n, int p, int d, DSR_Node *pointer=NULL);
    void setName(int intNm);
```

```
    void setTheOrder(int intOrd);
    void setParent(int intNewParent);
    void operator =(DSR_Node& b);
    int operator < (DSR_Node& b);
    int operator <= (DSR_Node& b);
    int operator > (DSR_Node& b);
    int operator >= (DSR_Node& b);
};


class DSR_HashTable
{
private:
    int intHSize;
    int intCurrVal;

    struct HashNode {
        unsigned int intID;
        HashNode *Next;

        HashNode(unsigned int id = 0, HashNode *N = NULL) :
            intID (id), Next (N) { }
    } **Hash_Table;

    struct HashNodeConsec {
```

```
        unsigned int uintHash;
        char *Word;

        HashNodeConsec(char *W = NULL, unsigned int id = 0) :
            Word (StrSave(W)), uintHash (id) { }
    } **Hash_Table_Consec;

    unsigned Hash (const char *s);

public:
    DSR_HashTable(int intSize = intHDefaultSize);
    ~DSR_HashTable( );
    int GetID (char *word);
    char *GetName (int id);
    int GetNodes ();
    int Find (char *word);
};

class DSR_PriQueue
{
private:
    DSR_Node *head;
    int intSize;
    int *intMapper;
    int intMaxSize;
```

```
public:
    DSR_PriQueue(int number = intQDefaultSize);
    ~DSR_PriQueue();

    int IsThere(int nodeName);
    void Insert(DSR_Node& add);
    DSR_Node DeleteMin();
    int DecreaseKey(int nodeName, int amt, int newparent);
};


class DSR_AdjacentList
{
private:
    int intNodes;
    int intSize;
    int intLinks;

    struct AdjNode {
        DSR_Node *curpos;
        DSR_Node *head;
        short intKnown;
        int intOrd;
    } *List;
```

```
public:
    DSR_AdjacentList(int intNewSize = intListDefaultSize);
    ~DSR_AdjacentList();

    void AddLink(int ptA,int ptB, int link);
    int operator ()(int ndA,int ndB);

    int Empty();
    void setNodes(int nv);
    int NumLinks();
    int NumNodes();
    void setKnown(int i,short status);
    short Known(int i);
    int intOrder(int i);
    void setOrder(int i, int intOrder);
    DSR_Node *start(int i);
};


class DSRAlgorithm : public TclObject
{
public:
    DSRAlgorithm() {};
    void DSR_Dijkstra(int intSource, DSR_AdjacentList& AL,
```

```
        DSR_Node *ND, DSR_HashTable& HT);

    void DSR_GetCompletePath(int intSource, int intDestination,
        DSR_Node *ND, DSR_AdjacentList& AL,
        DSR_HashTable& HT, int *intTotalOrder, char *pathBuf);

    void CalPathForSim(char *strSource, char *strDestination,
        char *strNT);

    virtual int command(int argc, const char*const* argv);
};
```

Chapter 6

Conclusions

6.1 Conclusions

This thesis studies the routing problem for Differentiated Services in a network consisting of DiffServ capable nodes and DiffServ incapable nodes (see Definition 4.3.1). We give three routing algorithms. The routing metrics we consider in this thesis are DiffServ capability and cost. The first algorithm assumes that cost is given higher precedence over DiffServ capability. The second algorithm assumes that DiffServ capability is given higher precedence over cost. The first two algorithms do not perform well in their worst cases. This leads us to the third algorithm. In the third algorithm, we assign a DiffServ route selection order number to each link in the network topology and the algorithm computes the optimal path using the assigned DiffServ route selection order numbers. Simulations show that the third algorithm is more powerful than the first two algorithms. Through simulations in ns,

we have proved that the algorithm calculates the correct routing paths in all network topologies. Since the route selection order number can be changed by the Internet Service Provider, it can be used to create the service packages and charging policy. Based on this, we recommend that the third algorithm be used for all network topologies.

6.2 Recommendations for Future Work

The following areas require further attention:

1. In the algorithms, we use only two metrics: DiffServ capability and cost. Bandwidth, delay and other metrics are not considered in the route computation. In some QoS situations, these metrics may be important in route selections. We need to consider these metrics in future work.

2. We need to find a way to identify whether a router is DiffServ capable or not.

References

[1] ns-2 network simulator, http://www-nrg.ee.lbl.gov/ns/.

[2] http://www.isi.edu/nsnam/nam/.

[3] G. Apostolopoulos, R. Guerin and S. Kamat, Implementation and Performance Measurements of QoS Routing Extensions to OSPF, *Proceedings of Infocomm '99*, New York, March 1999.

[4] George Apostopoulos, Roch Guerin, Sanjay Kamat, Satish K. Tripathi, Quality of Service Based Routing: A Performance Perspective, *Proceedings of ACM SIGCOMM*, September 1998, pp. 17-28.

[5] Cristina Aurrecoechea, Andrew T Campbell, Linda Hauw, A survey of QoS architectures. *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, Vol. 6, No. 3, May 1998, pp. 138-151.

[6] Roland Bless, Klaus Wehrle, Evaluation of Differentiated Services using an Implementation under Linux, *Proceedings of IWQoS'99*, London, June 1999. IEEE Press, 1999.

[7] Shigang Chen, Klara Nahrstedt, An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and

Solutions, *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video*, vol. 12, num. 6, pp. 64-79, November-December, 1998.

[8] Shigang Chen, Klara Nahrstedt, On Finding Multi-constrained Paths (MCP), *International Journal of Computational Geometry and Applications*, 1998, pp. 874-879.

[9] Shigang Chen, Klara Nahrstedt, Distributed QoS Routing with Imprecise State Information, *Proceedings of 7th IEEE International Conference on Computer, Communications and Networks*, Lafayette, LA, October 1998, pp. 614-621.

[10] Boris V. Cherkassky and Andrew V. Goldberg and Tomasz Radzik, Shortest Paths Algorithms: Theory and Experimental Evaluation, *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.

[11] Yong Cui, Ke Xu and Jianping Wu, Precomputation for Multi-contraned QoS Routing in High-speed Networks, *Proceedings of IEEE Infocomm*, 2003.

[12] C. Dovrolis, D. Stiliadis and P. Ramanathan, Proportional Differentiated Services, *Proceedings of SIGCOMM (October 1999)*, vol. 29, 109-120, 1999.

[13] N. Fujita and A. Iwata., A Hierarchical Multilayer QoS Routing System with Dynamic SLA Management, *IEEE Journal on Selected Areas in Communication*, 18(12), December 2000.

[14] Roch A. Guerin, Ariel Orda and Doughlas Williams, QoS Routing Mechanisms and OSPF Extensions, *IETF Internet Draft*, November 1996.

[15] J.M. Jaffe, Algorithms for Finding Paths with Multiple Constraints, *Networks*, 14:95–116, 1984.

[16] Yanxia Jia, Ioanis Nikolaidis and P. Gburzynski, Multiple Path QoS Routing, *Proceedings of ICC'01*, Helsinki, Finland, June 11–15, 2001, pp. 2583–2587.

[17] S. Kamat, S. Tripathi, G. Apostolopoulos and R. Guerin, Improving QoS Routing Performance under Inaccurate Link State Information, *ITC-16*, June 1999.

[18] Ashish Goel, K.G.Ramakrishnan, Deepak Kataria, Dimitris Logothetis, Efficient Computation of Delay-Sensitive Routes from One Source to All Destinations, *IEEE INFOCOM 2001*, pp. 148-152.

[19] R. Guerin and A. Orda, QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms, *IEEE/ACM Transactions on Networking*, Vol. 7, No. 3, June 1999, pp. 350-364.

[20] Liang Guo, Ibrahim Matta, Search Space Reduction in QoS Routing, *College of Computer Science, Northeastern University, Technical Report NU-CCS-98-09*, October 1998.

[21] Alpar Juttner, Balazs Szviatovszki, Ildiko Mecs, Zsolt Rajko, Lagrange Relaxation Based Method for the QoS Routing Problem, *IEEE INFOCOM 2001*, pp. 782-791.

[22] Turgay Korkmaz and Marwan Krunz, Multi-Constrained Optimal Path Selection, *Proceedings of the IEEE INFOCOM 2001 Conference*, Vol 2, Anchorage, Alaska, April 2001, pp. 834-843.

[23] Turgay Korkmaz, Marwan Krunz, A Randomized Algorithm for Finding a Path Subject to Multiple QoS Constraints, *Computer Networks Journal*, Vol. 36, No. 2/3, 2001, pp. 251 268.

[24] Turgay Korkmaz, Marwan Krunz, Spyros Tragoudas, Efficient Algorithm for Finding a Path Subject to Two Additive Constraints, *Computer Communications Journal*, Vol. 25, No. 3, February 2002, pp. 225-238.

[25] F. A. Kuipers and P. Van Mieghem, QoS Routing: Average Complexity and Hopcount in m dimensions, *Proceedings of 2nd International Workshop on Quality of Future Internet Services*, QofIS2001, Coimbra, Portugal, September 24-26, 2001, pp. 110-126.

[26] Longsong Lin, Tianji Jiang and Jeffrey Lo, A Generic Traffic Conditioning Model for Differentiated Services, *ICC (3)*, 1305-1309, 2000.

[27] Gang Liu, K.G. Ramakrishnan, A*Prune: An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints, *INFOCOM*, 743-749, 2001.

[28] D. H. Lorenz and A. Orda, Qos routing in networks with uncertain parameters, *IEEE INFOCOM'98*, 1998.

[29] Q. Ma, Quality-of-Service Routing in Integrated Services Networks, Ph.D. thesis, CMU-CS-98-138, January, 1998.

[30] Q. Ma and P. Steenkiste, Quality of service routing for traffic with performance guarantees, *IFIP Fifth International Workshop on Quality of Service*, (New York), pp. 115–126, May 1997.

[31] Ibrahim Matta and Azer Bestavros, QoS Controllers for the Internet, *Proceedings of the NSF Workshop on Information Technology*, Cairo, Egypt, March 2000.

[32] Piet Van Mieghem, Hans De Neve, Aspects of Quality of Service Routing, *SPIE'98*, Nov. 1-6, Boston (USA), 3529A-05.

[33] Hans De Neve, Piet Van Mieghem, TAMCRA: A Tunable Accuracy Multiple Constraints Routing Algorithm, *Computer Communications*, Vol. 23, pp. 667-679

[34] Kesava Prasad Narasimhan, An Implementation of Differentiated Services in a Linux Environment, M.Sc. Thesis, North Carolina State University, 2000.

[35] Satyabrata Pradhan, QoS-Aware Hierarchical Multicast Routing on Next Generation Internetworks, UM M.Sc. thesis, 2000.

[36] Joao Luis Sobrinho, Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet, *IEEE INFOCOM'01*, vol.2, pp. 727–735, 2001.

[37] W. Stallings, High-Speed Networks: TCP/IP and ATM Design Principles, Prentice Hall Inc., NJ, USA, 1998.

[38] Ronny Vogel, Ralf G. Herrtwich, Winfried Kalfa, Hartmut Wittig and Lars C. Wolf, QoS Based Routing of Multimedia Streams in Computer Networks, *IEEE JSAC*, Vol. 14, No. 7, 1996, pp. 1235-1244.

[39] Jun Wang and Klara Nahrstedt, Hop-by-Hop Routing Algorithms For Premium-class Traffic In DiffServ Networks, *IEEE INFOCOM'02*, 2002.

[40] June Wang, Klara Nahrstedt and Yuxin Zhou, Design and Implementation of DiffServ Routers in OPNET, *Proceedings of OPNETWORK'00*, Washington D.C., Aug 28 - Sep. 1, 2000.

[41] Shengquan Wang, Dong Xuan, Riccardo Bettati and Wei Zhao, Providing Absolute Differentiated Services for Real-Time Applications in Static-Priority Scheduling Networks, *INFOCOM*, 669-678, 2001.

[42] Z. Wang and Jon Crowcroft, Qos Routing for Supporting Resource Reservation, *IEEE JSAC*, September 1996.

[43] Z. Wang and J. Crowcroft, Quality of service routing for supporting multimedia applications, *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1228–1234, Sept. 1996.

[44] Zheng Wang and Jon Crowcroft, Quality of Service Routing for Supporting Multimedia Applications, *IEEE Journal on Selected Areas in Communications*, vol.14, pp. 1228-1234, Sept. 1996.

[45] X. Xiao and L. M. Ni, Internet QoS: a big picture, *IEEE Network*, vol. 13, no. 2, pp. 8–18, March–April, 1999.

[46] Ikjun Yeom and A. L. Narasimha Reddy, Realizing throughput guarantees in a differentiated services network, *ICMCS*, Vol. 2, 372-376, 1999.

[47] Ik-Jun Yeom, Bandwidth Assurance In A Differentiated Services Network, Ph.D. Dissertation, Texas A & M University, May 2001.

[48] I. Yeom and A. L. N. Reddy, Marking for QoS Improvement, *Computer Communications*, vol. 24, No. 1, 35-50, 2001.

[49] Xin Yuan, On the Extended Bellman-Ford Algorithm to Solve Two-Constrained Quality of Service Routing Problems, *Technical Report, TR-990701, Department of Computer Science, Florida State University*, July 1999.,

[50] Xin Yuan, Xingming Liu, Heuristic Algorithms for Multi-Constrained Quality of Service Routing, *IEEE INFOCOM 2001*, pp. 355-364.

[51] RFC 768, User Datagram Protocol, August 1980.

[52] RFC 791, INTERNET PROTOCOL, September 1981.

[53] RFC 792, INTERNET CONTROL MESSAGE PROTOCOL, September 1981.

[54] RFC 793, TRANSMISSION CONTROL PROTOCOL, September 1981.

[55] RFC 826, An Ethernet Address Resolution Protocol, November 1982.

[56] RFC 951, BOOTSTRAP PROTOCOL (BOOTP), September 1985.

[57] RFC 1058, Routing Information Protocol (RIP), June 1988.

[58] RFC 1633, Integrated services in the internet architecture: an overview, 1994.

[59] RFC 1721, RIP Version 2 Protocol Analysis, 1994.

[60] RFC 1771, A Border Gateway Protocol 4 (BGP-4), March 1995.

[61] RFC 1883, Internet Protocol, Version 6 (IPv6) Specification, December 1995.

[62] RFC 1992, The Nimrod Routing Architecture, August 1996.

[63] RFC 2178, OSPF Version 2, July 1997.

[64] RFC 2205, Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification, September 1997.

[65] RFC 2208, Resource ReSerVation Protocol (RSVP) Version 1 Applicability Statement Some Guidelines on Deployment, September 1997

[66] RFC 2210, The Use of RSVP with IETF Integrated Services, September 1997.

[67] RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, December 1998.

[68] RFC 2475, An Architecture for Differentiated Services, December 1998.

[69] RFC 2597, Assured Forwarding PHB Group, June 1999.

[70] RFC 2598, An Expedited Forwarding PHB, June 1999.

[71] Stardust, White Paper – The Need for QoS, www.stardust.com, www.qosforum.com.

[72] Stardust, White Paper – QoS protocols & architectures, www.stardust.com, www.qosforum.com.

[73] IETF "Differentiated Services" working group, See http://www.ietf.org/html.charters/diffserv-charter.html and http://www.ietf.org/ids.by.wg/diffserv.html.

[74] Y.Bernet, R.Yavatkar, P.Ford, F.Baker, L.Zhang, K.Nichols, M.Speer and R. Braden, Interoperation of RSVP/Int-Serv and Diff-Serv Networks, February 1999, ⟨draft-ietf-diffserv-rsvp-02.txt⟩.

[75] IETF "Integrated Services" working group, See http://www.ietf.org/html.charters/intserv-charter.html and http://www.ietf.org/ids.by.wg/intserv.html.

[76] IETF "Multiprotocol Label Switching" working group, See http://www.ietf.org/html.charters/mpls-charter.html and http://www.ietf.org/ids.by.wg/mpls.html.

[77] E. Rosen, A. Viswanathan and R. Callon, Multiprotocol Label Switching Architecture, April 1999, ⟨draft-ietf-mpls-arch-05.txt⟩.

[78] J. Heinanen, Differentiated Services in MPLS Networks, June 1999, ⟨draft-heinenen-diffserv-mpls-00.txt⟩.

[79] F. Baker, Aggregation of RSVP for IPv4 and IPv6 Reservations, June 1999, ⟨draft-baker-rsvp-aggregation-01.txt⟩.

[80] Y. Bernet, Usage and Format of the DCLASS Object with RSVP Signalling, February 1999, ⟨draft-bernet-dclass-00.txt⟩.