DUBLIN CITY UNIVERSITY

School of Electronic Engineering

A Traffic Engineering System for DiffServ/MPLS Networks

A Thesis Submitted in Fulfilment of Postgraduate
M. Eng. Degree in Electronic Engineering at
Dublin City University

Supervised By
Dr. Tommy Curran

Submitted By
Alexei Chpenst, Dipl.-Eng.

Submission Date: October 2003.

DECLARATION

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of M. Eng. in Electronic Engineering is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: A. Chpenst
Alexei Chpenst

ID No.: 99146185

Date: 16. 02. 2004

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Professor Tommy Curran, for his valuable guidance and support during the writing of the thesis.

My special thanks then go to Sean Murphy who has contributed to my thesis with his knowledge and continued support.

I gratefully acknowledge the valuable suggestions given by Rob Brennan and friendly support of Saman Cooray.

I also wish to thank Robert Burmiston for his kind assistance.

Finally, I would also thank all my colleagues at the lab for friendly atmosphere.

ABSTRACT

A Traffic Engineering system for DiffServ/MPLS Networks

Author: Alex Chpenst

This thesis presents an approach to traffic engineering that uses DiffServ and MPLS technologies to provide QoS guarantees over an IP network. The specific problem described here is how best to route traffic within the network such that the demands can be carried with the requisite QoS while balancing the load on the network. A traffic engineering algorithm that determines QoS guaranteed label-switched paths (LSPs) between specified ingress-egress pairs is proposed and a system that uses such an algorithm is outlined. The algorithm generates a solution for the QoS routing problem of finding a path with a number of constraints (delay, jitter, loss) while trying to make best of resource utilisation. The key component of the system is a central resource manager responsible for monitoring and managing resources within the network and making all decisions to route traffic according to QoS requirements. The algorithm for determining QoS-constrained routes is based on the notion of effective bandwidth and cost functions for load balancing. The network simulation of the proposed system is presented here and simulation results are discussed.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF ABBREVIATIONS

AS	Autonomous System
AF	Assured Forwarding
BA	Behaviour Aggregate
BGP	Border Gateway Protocol
CRM	Central Resource Manager
DiffServ	Differentiated Services
EF	Expedited Forwarding
FEC	Forwarding Equivalence Class
FilterSpec	Filter Specification
IDL	Interface Design Language
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IntServ	Integrated Services
IP	Internet Protocol
ISP	Internet Service Provider
LER	Label Edge Router
LDP	Label Distribution Protocol
LSP	Label Switched Path
LSR	Label Switching Router
MIB	Management Information Base
MPLS	Multi-Protocol Label Switching
NP	Near Polynomial
PHB	Per-Hop Behaviour
QoS	Quality of Service
Rspec	Request Specification
RSVP	ReSource reservation Protocol
SLA	Service Level Agreement
ToS	Type of Service
TSpec	Traffic Specification

UML	Unified Modelling Language
VoD	Video on Demand
VoIP	Voice over IP

1 Introduction

1.1 Introduction

The Internet currently provides only best-effort service that treats all packets equally. However some Internet applications, like voice over IP (VoIP) and video teleconferencing, are very sensitive to the quality of service they receive from the network. Thus, various quality of service (QoS) techniques are being developed and are beginning to be deployed. The Internet Engineering Task Force has introduced several new technologies for this purpose. Some of them that are currently of most interest for network providers are Differentiated services (DiffServ) and Multi-protocol Label Switching (MPLS) technologies. The DiffServ model [RFC2475] is a simple schema to support various types of applications by differentiating classes of service for Internet traffic. DiffServ mechanisms allow network providers to allocate different levels of service to different users to meet their specific QoS requirements. MPLS [RFC3031] is a forwarding scheme, which allows packets to be sent along the specific paths. In combination with DiffServ, MPLS can be used for creation of QoS guaranteed tunnels between a pair of nodes. With MPLS and DiffServ, it is possible to define explicit routes with different performance guarantees. The explicit routing features of the MPLS technology make it very attractive to the process of network traffic engineering, which deals with the problem of network performance optimisation. Traffic engineering can use the explicit routing as a means for making the best use of the network infrastructure.

Traditionally, QoS is provided on top of an existing network by means of over-provisioning, that is, the operator increases the capacity of the links located at the most heavily loaded parts of the network, to ensure that congestion never occurs during the busy periods of any busy day. However, average utilisation on IP networks can be much lower than peak-utilisation, which reaches 100% only for short time periods. This is hardly cost-effective and delivers a very poor return on investment. Also, because traffic growth and

behaviour in today's telecommunication market is unpredictable, it has become almost impossible to forecast how traffic patterns will evolve. This may result in over-investment in areas of the network where capacity will never be used, while the real, unidentified problem areas remain congested. Therefore, today's Internet Service Providers (ISPs) are compelled to offer higher service level guarantees while making more efficient use of their existing infrastructure.

Traffic engineering allows an ISP to route traffic around known points of congestion in its network, thereby making more efficient use of the available bandwidth. During recent times, work on sophisticated routing strategies which are able to achieve better resource utilisation with the help of MPLS explicit routing has been in progress [RA00], [FR01], [PA98], [TF02].

1.2 Research Objectives

In this thesis, we consider a traffic engineering system for the network which supports DiffServ and MPLS technologies to provide QoS guarantees over an IP network. The traffic engineering system can set up QoS guaranteed label-switched paths (LSPs) between specified ingress-egress pairs in an on-line environment where requests for establishing LSP tunnels arrive one by one and where information about future requests is not available. The main subject of this research is concerned with the problem of how best to route traffic within such a network so that the demands can be carried with the requisite QoS while making the best use of network resources. The research objectives of the thesis include defining the architecture components for the traffic engineering system and developing a routing strategy aimed at making the best use of network resources. The main problem when using current shortest path algorithms for finding QoS routes is that some links between the ingress-egress pairs may get congested while links along possible alternative paths remain free. The primary objective of this thesis is to present a routing algorithm that provides better results with respect to both balancing the load on the network and minimising the resource utilisation.

To achieve the goals of the research it is necessary to carry out a study of various QoS and traffic engineering techniques. This includes gaining an understanding and performing a

survey of technologies such as IntServ, RSVP, DiffServ and MPLS. It is then required to investigate how these technologies can be used to provide QoS guarantees in the network while performing traffic engineering. With respect to developing a routing approach for improving the resource utilisation of the network, it is needed to analyse the existing QoS routing algorithms and their shortcomings. Finally, for validating the proposed traffic engineering system it is necessary to perform network simulations evaluating the system performance.

1.3 Structure of the Thesis

The organisation of the thesis is as follows. Chapter 2 gives a short introduction to traffic engineering; this chapter introduces the main issues of traffic engineering and describes its common concepts along with the fundamental terminology largely used throughout the thesis.

Overview of the projects related to traffic engineering is given in Chapter 3. This contains a description of some projects that have been carried out within the Internet Engineering Task Force (IETF) for the last decade. These include the work related to development of architectures and protocols for providing QoS services in operational networks. Some of the projects described in this chapter are useful tools for traffic engineering and presently of great interest for Internet service providers.

Chapter 4 introduces a traffic engineering management system that addresses the problems of routing QoS guaranteed paths in the network. The system is the integration of several different technologies described in Chapter 3 (DiffServ, MPLS, RSVP) that, used together, achieve the main objectives of the traffic engineering system. The system can set up QoS guaranteed label-switched paths (LSPs) between specified ingress-egress pairs in the DiffServ/MPLS domain. In this chapter, first a network scenario typical for the system is considered and the main objectives of the system are specified. The rest of the chapter is concerned with the architecture considerations and design choices aimed at meeting the system objectives.

In Chapter 5, an optimisation problem of the presented system is defined. The problem is related to optimising the resource utilisation of the network when routing QoS guaranteed paths. The presented solution to the problem is a developed routing algorithm aimed at making the best use of network resources while meeting QoS requirements. The algorithm is described in Section 5.5.3.

Chapter 6 contains a description of the network simulations performed in order to validate the described system. The simulation model is designed for a network scenario running a Voice over IP (VoIP) service across a DiffServ/MPLS network. On the example of voice traffic, it is demonstrated how the routing can be performed in the network by applying the algorithm described in Chapter 5. The simulation results are presented and discussed.

Finally, Chapter 7 concludes the thesis summarising the work carried out and providing some recommendations for future work.

2 Introduction to Traffic Engineering

2.1 Introduction

As the main subject of this thesis is related to developing a traffic engineering solution for IP networks, it is of great interest to briefly describe here the general issues and main principles of Internet traffic engineering. This chapter describes the common traffic engineering concepts along with the fundamental terminology largely used throughout the thesis.

2.2 What is Internet Traffic Engineering?

Traffic engineering deals with the two main problems of network engineering, which are evaluation of the network operation state and subsequent optimisation of the network performance [RFC3272]. Thus, Internet traffic engineering can be defined as a part of Internet network engineering dealing with the problem of performance evaluation and performance optimisation of operational IP networks. This addresses issues of network technologies and scientific approaches such as measurement, modelling, simulation, analysis and optimisation. The general structure of traffic engineering is presented in Figure 2-1.

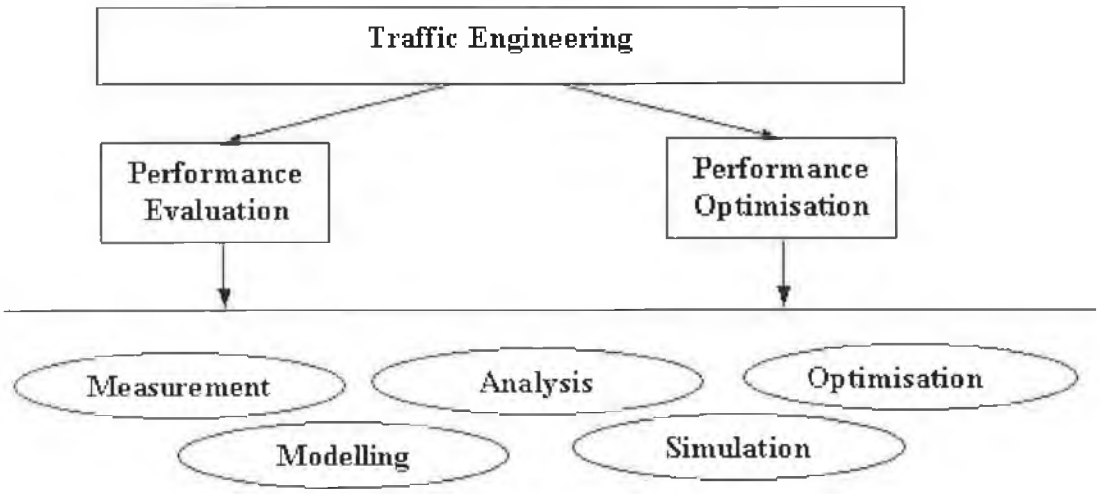


Figure 2-1: Structure of Traffic Engineering

2.2.1 Network performance evaluation problem

One of the significant aspects of Internet traffic engineering is network performance evaluation. The primary tasks of network performance evaluation are monitoring and assessment of the operational state of the network. This helps to identify existing problems in the network, predict potential future problems and undertake the necessary steps for network optimisation.

Performance evaluation can be achieved with the help of measurement, modelling, analysis and simulation. Measurement is used to determine the operational state of the network and the quality of network services installed in the network. Modelling is used when performing analysis or simulation of the network to model network nodes and links to capture relevant operational features such as topology, bandwidth, buffer space, and nodal service policies (link scheduling, packet prioritisation, buffer management, etc). Analytical traffic models can be used to depict dynamic and behavioural traffic characteristics, such as burstiness and statistical distributions.

In this thesis, we do not consider the problem of the network performance evaluation. With regard to evaluating the current operational state of the network, we assume that we collect all necessary performance measures by means of some link state protocol or measurement

and monitoring techniques [KY02, SL02, RFC1157] not discussed here. The network optimisation problem generally defined in the next section is the main interest of this work.

2.2.2 Network performance optimisation problem

The other major objective of traffic engineering is performance optimisation of the network. This can be achieved by different methods of modelling, analysis, simulation and optimisation techniques. Network modelling may facilitate analysis and/or simulation, which can be used to predict network performance under various conditions and to validate the effectiveness of planned solutions with respect to optimisation of the network operational sate.

Performance optimisation of the network is achieved by meeting traffic oriented performance requirements related to delay, delay variation, loss, and throughput, while utilising network resources economically and reliably. For example, this thesis includes development of an algorithm described further (Chapter 5) that follows this objective and routes traffic within the network such that the demands can be carried while balancing the load on the network.

To optimise the network performance at the traffic level some traffic oriented performance measures can be considered. These include:

- delay
- delay variation (jitter)
- packet loss
- throughput

To optimise the network performance at the resource level some measures of load balancing can be used (Section 5.3.1).

With respect to our system described later we analyse how each of these measures can be calculated and measured and how they can be used for optimising the network performance.

The optimisation aspects of traffic engineering can be analysed from the point of view of *capacity management* and *traffic management*.

As used in [RFC3272], capacity management includes:

- capacity planning (ranging from days to possibly years);
- routing control (milliseconds to days);
- resource management (link bandwidth, buffer space, computational resources).

Likewise, as used in [RFC3272], traffic management includes:

- nodal traffic control functions (ranging from picoseconds to milliseconds):
  - traffic conditioning;
  - queue management;
  - scheduling.
- other functions that regulate traffic flow through the network or that arbitrate access to network resources between different packets or between different traffic streams.

These main aspects of capacity and traffic management noted above are of great interest for the work presented here and they are discussed in more detail further.

From the control point of view, the traffic engineering system can be *pro-active* and *reactive*. In the pro-active system, some preventive actions are to be taken by the system to avoid predicted undesirable future network states or to induce a more desirable state in the future (e.g congestion avoidance). In the reactive system, the traffic engineering control system responds correctively and adaptively to events that have already happened in the network (e.g congestion control). Different combinations of capacity and traffic management aspects (cited above) can be used to achieve the goals of pro-active and

reactive systems. We analyse the traffic engineering system presented here from the reactive/proactive point of view in Chapter 4.

It is worth describing the inputs of the traffic engineering control system. These can be defined as:

- network state variables;
- policy variables;
- decision variables.

Network state variables include the network parameters that the system takes into account for evaluating the current state of the network. Policy variables are the set of rules currently installed in the network and currently governing the operation of the network. Finally, decision variables include the network parameters that the system can change to optimise the network. We define the inputs of our system in Chapter 4.

3 Enabling Technologies for Advanced Traffic Engineering

3.1 Introduction

This chapter presents an overview of some Internet Engineering Task Force (IETF) activities relevant to traffic engineering. These activities include the work carried out on development of architectures and protocols for providing QoS services in operational networks. Some of the technologies described here are presently of great interest for contemporary network providers due to the availability of flexible tools for performing traffic engineering.

3.2 Integrated Services

Integrated Services (IntServ) is an architecture developed within the IETF to provide QoS capabilities for delay-sensitive applications such as real-time voice and video. IntServ provides the QoS for specific user packet streams or flows. To guarantee the requested QoS, the IntServ model requires network resources, such as bandwidth and buffers, to be reserved in advance for a given traffic flow.

Currently, the IntServ architecture offers two service models [RFC2212]:

- Guaranteed Service
- Controlled Load

Guaranteed Service guarantees a deterministic upper limit on delay for a packet travelling through the network. Guaranteed Service is designed for applications requiring strict QoS, applications that are intolerant of delays, and thus requiring that a packet arrive no later than a certain length of time after it was transmitted from the source. For example, a real-

time voice application requires end-to-end delay to be no longer than 100ms, otherwise there may be a great amount of distortion which would make the application worthless. The guaranteed service is accomplished by controlling the queuing delay on network elements along the data flow path. The queuing delay for a particular flow can be controlled with the help of scheduling mechanisms (e.g. WFQ, CBQ, Priority Queuing) [FR00]. Scheduling is a method to differentiate traffic into a network, which is used to treat packets in buffers according to different rules.

Controlled Load Service provides a probabilistic upper limit on delay. This service is designed for applications that are more flexible and more tolerant of delays than the applications requiring Guaranteed Service. Examples of such applications include interactive access and non-realtime audio and video. The idea behind the Controlled Load Service is that the applications should obtain similar service to that of a lightly loaded network. Controlled Load Service can be compared with the best-effort service in a lightly loaded network where the quality of service is not affected by the actual network conditions. Controlled Load Service can be described qualitatively or quantitatively. An example of the qualitative description of Controlled Load Service can look like: "the transit delay experienced by a very high percentage of the delivered packets will not greatly exceed the minimum transmit delay experienced by any successfully delivered packet" [RFC 2211]. Another option would be a quantitative definition: "99 % of the packets will experience a maximum delay of 100 ms".

The integrated services model includes three logical components needed to be installed in a network node for providing QoS services:

- packet classifiers

  The classifier maps each arriving packet into a class. A class corresponds to a flow, which includes all packets having the same QoS requirements and network parameters (e.g. source and destination address, protocol, port number).

- packet schedulers

Schedulers manage the output queues to provide the desired forwarding of different flows for different QoS. All packets from the same flow receive the same treatment for onward forwarding.

- admission control

  Admission control examines whether a request for QoS services should be accepted or rejected (depending on resource availability and authorisation results).

The main disadvantage of the IntServ architecture is a scalability problem. As was mentioned above, IntServ provides QoS services for a traffic flow. In large public IP networks there can be millions of active micro-flows traversing the network concurrently. This results in a huge storage and processing overhead on routers. Besides, all routers must have RSVP, classification, packet scheduling, and admission control. Altogether, this makes IntServ architecture difficult to deploy in practice; although there can be some scenarios of deploying it in conjunction with other technologies such as DiffServ [RFC2475].

The IntServ model requires explicit signalling of QoS requirements from end systems to routers. To provide this signalling functionality, the RSVP protocol can be used. In the prevailing model for the RSVP/IntServ architecture, RSVP is responsible for signalling per-flow resource requirements to network elements using IntServ protocol parameters. In turn, these network elements would then apply IntServ admission control to the reservation requests. In this model RSVP carries IntServ information. Because RSVP is designed to be used with a variety of QoS control services, the RSVP specification does not define the internal format of the protocol fields, or objects, which are related to invoking particular QoS control services. Rather, RSVP treats these objects as opaque. For the RSVP/IntServ architecture, the objects necessary to carry IntServ information within RSVP fields are defined in [RFC2210]. The RSVP protocol is described in more detail next.

3.3 RSVP

The ReSerVation Protocol (RSVP) [RFC2205] is a signalling protocol that provides reservation setup and control to the integrated services. RSVP can be used by hosts and

routers. A host may use RSVP to request specific qualities of service from the network for particular application data streams or flows. A router may use RSVP to deliver QoS requests to all nodes along the paths of the flows and to set up and maintain a state to provide requested service. Successful RSVP requests result in resources being reserved in each node along the data path.

An overview of how the protocol works is illustrated in Figure 3-1. The reservation of the network resources is performed as follows:

1. RSVP sends a PATH message from the sender that contains the *traffic specification* (TSpec) information to the destination address. TSpec may include such specifications as bandwidth, delay and jitter.

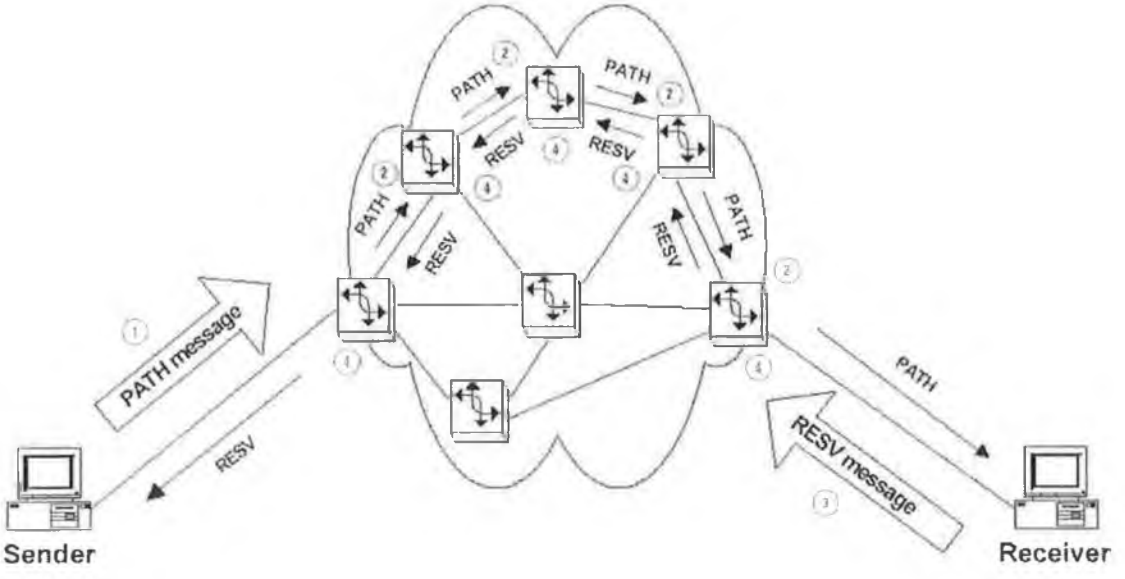


Figure 3-1: RSVP signalling process

2. Each RSVP-enabled router along the downstream route establishes a source route that includes the previous source address of the PATH message (i.e. the next hop "upstream" towards the sender).

3. Upon receiving the PATH message, the receiver sends the RESV message "upstream" (following the source route provided in PATH message) to make a resource reservation. A RESV message (reservation request) contains resource reservation request, which contains TSpec from sender, request specification

(Rspec) with QoS level and filter specification (FilterSpec) specifying the packets (by the transport protocol or port number) for which the reservation will work. Together, the RSpec and FilterSpec represent a flow-descriptor that routers use to identify each reservation.

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

5. When the last router (the closest to the sender) receives the RESV and accepts the request, it sends a confirmation message back to the receiver. After that, the resources along the path are reserved and the receiver is ready to accept data from the sender.

The resource reservation can be cancelled directly or indirectly. In the first case, the request for cancellation is initiated either by sender or by receiver and performed by the corresponding messages of the RSVP protocol. In the second case, the cancellation may happen in the case of time-out. That is, each state configuration of the router is a short-term configuration that automatically expires after a given time, unless refreshed by another set-up command message. This is designed to prevent the misbehaviours such as when a receiver had set up a reservation and then "forgot" to free those resources, and disappeared from the network (e.g. logged off). To make sure that routers will not keep that reservation forever, all reserved resources have a “time to live” and if the reservation is not refreshed by the receiver in time, it is cancelled.

A number of IETF working groups are engaged in activities related to the RSVP protocol. It has been recently modified and extended in several ways to reserve resources for aggregation of flows, to set up explicit routes with QoS requirements (e.g. within MPLS architecture), and to do some other signalling tasks for traffic engineering [GU97, AW98, PA98]. As a result, RSVP can be used with a variety of QoS control services.

3.4 DiffServ

Since there is difficulty in implementing and deploying IntServ, the simpler (and hence cheaper) Differentiated Services (DiffServ) architecture is designed for implementing service differentiation in the Internet. Service differentiation is necessary to meet various application and user requirements, and to differentiate pricing of services. The main objective of the architecture is to provide scalable mechanisms for classification of traffic, which ultimately allows each class of traffic to be treated differently to meet its specific requirements. The main advantage of the architecture is that no resource reservation is necessary.

The traffic entering a network is classified and conditioned at the boundaries of the network and assigned to different *behaviour aggregates*. To each behaviour aggregate corresponds a single DS codepoint. All packets are forwarded within the network according to the *per-hop behaviour* associated with the DS codepoint. There is a DS field defined in the IP header for containing the DS codepoint. The DS field consists of six bits of the part of the IP header formerly known as Type of Service (ToS) octet. There are a number of standardised Per-Hop Behaviour (PHB) groups. Using the PHB groups, several classes of services can be defined using different classification, policing, shaping and scheduling rules.

The DS domain is a contiguous set of DS nodes, which operate with a common service provisioning policy and set of PHB groups implemented on each node. A DS domain consists of DS boundary nodes and DS interior nodes (Figure 3-2).

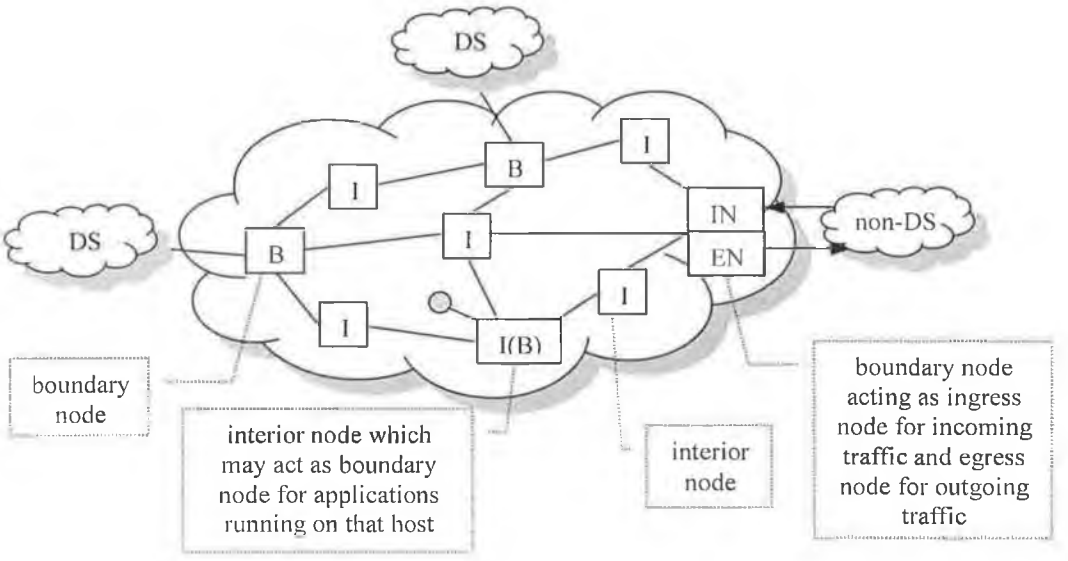


Figure 3-2: DiffServ domain

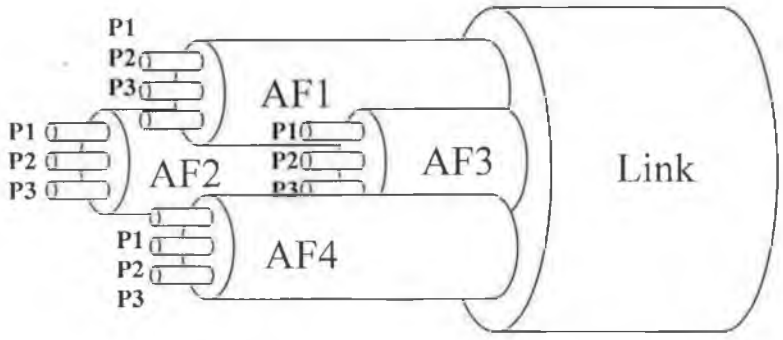
A host within a DS domain may act as a DS boundary node for traffic from applications running on that host. DS boundary nodes act both as a DS ingress node and as a DS egress node for different directions of traffic. A DS ingress node is responsible for conditioning traffic entering the DS domain and a DS egress node is responsible for conditioning traffic leaving the DS domain.

Per-Hop-Behaviour Groups

The PHB groups are the actual mechanism to implement a service differentiation in the networks. These are the means by which a node allocates resources to behaviour aggregates. The classification of the different behaviour aggregates to a particular group may be specified in terms of the needed resources (e.g. bandwidth, buffer) or in terms of traffic oriented performance measures (e.g. delay, loss, jitter). For example, a PHB group may guarantee a minimal bandwidth allocation of a link to a particular behaviour aggregate. As too many PHB groups would complicate efficient router design [RFC2475], currently there are proposals for two PHB groups:

- *Assured Forwarding PHB Group*
- *Expedited Forwarding PHB Group*

An Assured Forwarding PHB group (AF) provides four independently forwarded traffic classes, each with three drop precedences [RFC2597]. Graphically the classification of AF traffic is depicted in Figure 3-3. Each of four classes is assigned some amount of bandwidth and buffer capacity. In case of congestion, the drop precedence of a packet determines its probability within the AF class of being discarded. One way to use classes is *Olympic service*. That is when packets are assigned to *gold*, *silver*, and *bronze* classes. The *gold* class has lighter load than the other two classes. The customer may select one of these classes (which each have a different cost).



AF1, AF2, AF3, AF4 – four classes
P1, P2, P3 – three drop precedences

Figure 3-3: Classification of AF traffic

An Expedited Forwarding PHB Group (EF) can be used to build a low loss, low latency and low jitter assured bandwidth end-to-end service through DS domains. The EF PHB is defined as a forwarding treatment for a particular DiffServ aggregate where the departure rate of the aggregate's packets from any DiffServ node must equal or exceed a configurable rate [RFC2598]. The implementation supporting the EF traffic must provide this rate independently of the intensity of any other traffic attempting to transit the node. This makes it possible to provide end-to-end *virtual leased lines* or *premium service* [RFC2638].

Main components of DiffServ architecture

PHBs are implemented in nodes by means of some buffer management and packet scheduling mechanisms. In general, a variety of implementation mechanisms may be

suitable for implementing a particular PHB group. The main components of DiffServ architecture, supporting the implementation of PHB groups, are presented in Figure 3-4.

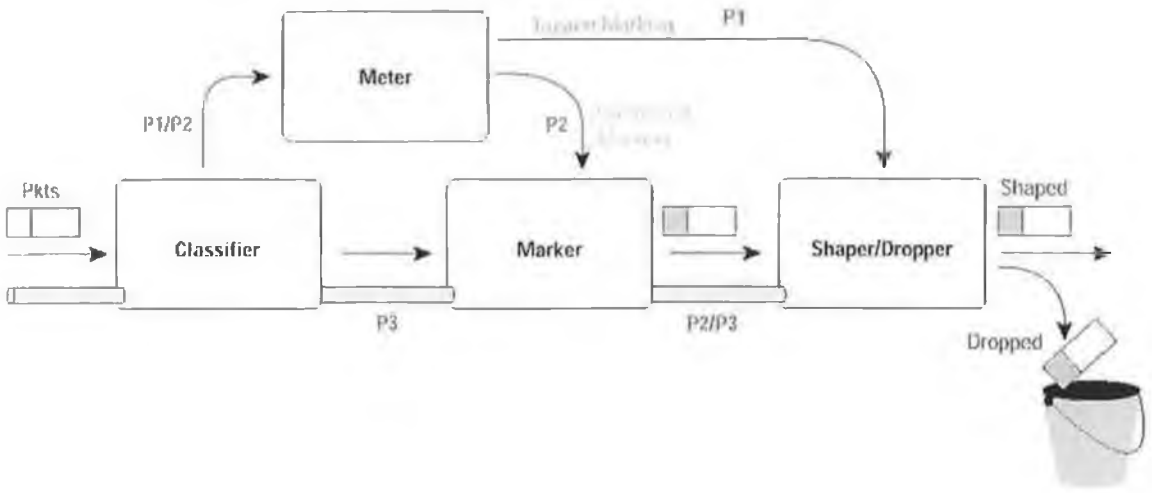


Figure 3-4: Main components of DiffServ architecture

Classifiers

Classifiers are used to "steer" packets matching some specified rule to an element of a traffic conditioner for further processing. There are two types of classifiers defined. The BA (Behaviour Aggregate) Classifier classifies packets based on the DS codepoint only. The MF (Multi-Field) classifier selects packets based on the value of a combination of one or more header fields, such as source address, destination address, DS field, protocol ID, source port and destination port numbers, and other information such as the incoming interface.

Meters

Traffic meters measure the temporal properties of the stream of packets selected by a classifier against a traffic profile specified in a traffic conditioning agreement. The example of meter is token bucket meter.

Markers

Packet markers set the DS field of a packet to a particular codepoint, adding the marked packet to a particular DS behaviour aggregate. When the marker changes the codepoint in a packet it is said to have "re-marked" the packet.

Shapers

Shapers delay some or all of the packets in a traffic stream 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.

Droppers

Droppers discard some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile. This process is known as "policing" the stream. Note that a dropper can be implemented as a special case of a shaper by setting the shaper buffer size to zero (or a few) packets.

3.5 MPLS

Multiprotocol Label Switching (MPLS) [RFC3031] is very important for Traffic Engineering because it provides great possibilities for routing the traffic and therefore optimising the network resource utilisation. MPLS was developed within the IETF as a forwarding scheme, which offers the ability to explicitly control routing based on information carried in packets' headers, such as destination and source addresses. Even though MPLS is quite a complex architecture and difficult to deploy, it has many advantages over conventional routing techniques.

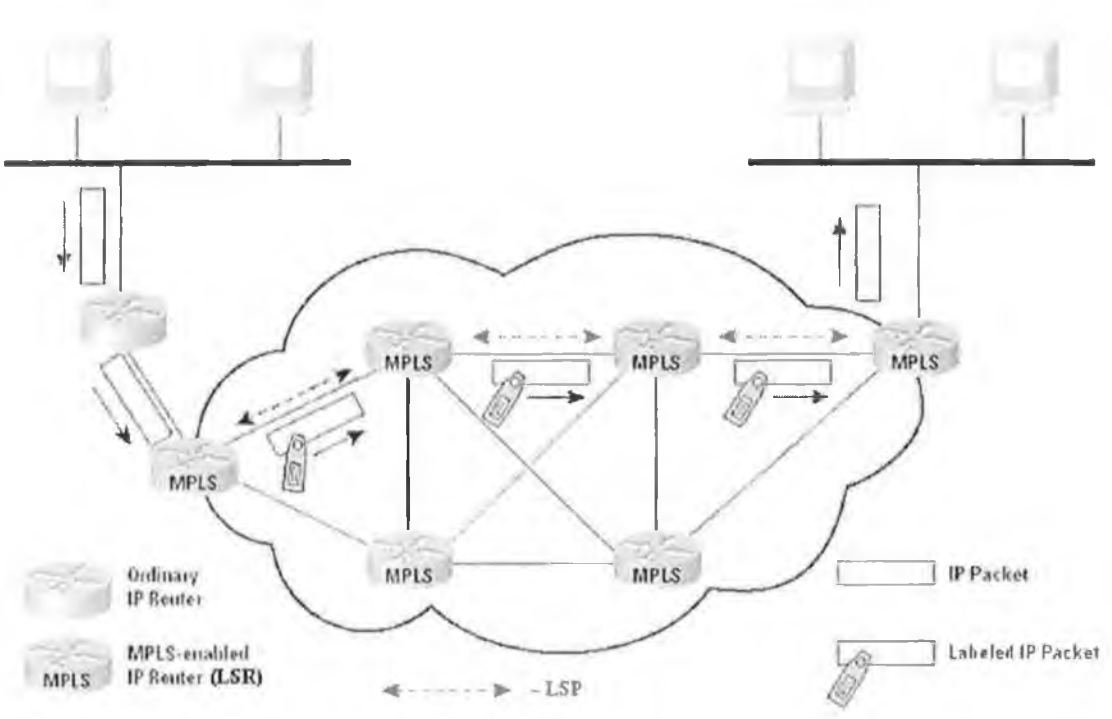


Figure 3-5: MPLS architecture

In conventional IP networks, as the packet travels from one router to the next, each router makes an independent decision for that packet. That is, each router chooses the next hop for the packet according to its analysis of the packet's header and the results of running the routing algorithm. In MPLS, unlike in conventional IP networks, a routing decision is made when the packet enters the network. By that time, based on the analysis of the packet's header, a particular Forwarding Equivalence Class (FEC) is assigned to the packet, which is then encoded into the packet as a label and is transmitted within it to the next hop. The next hop doesn't make an analysis of the packet's header again, rather, it chooses the subsequent hop analysing the label. Eventually, it replaces the old label with the new one and forwards the packet to the next specified hop. That is, once a packet is assigned to a FEC, no further header analysis is performed by the proceeding hops. All forwarding is done only by analysing the labels.

This decoupling of forwarding and routing, where the route is determined once and simple forwarding happens for each subsequent packet, makes the MPLS label based approach much better than IP routing, where a routing decision is made individually for each packet.

In Figure 3-5 an MPLS domain is presented. At the ingress to the domain are Label Switching Routers (LSRs). They classify all packets entering the network into FECs. This can be done by analysing a variety of factors such as information included in the packets' headers or information about local routing policies. An MPLS label then is attached to the packets according to their FECs. All other routers in the domain are MPLS capable routers (or LSRs). Each LSR analyses the label of the traversing packet and makes a forwarding decision. When the packet leaves an MPLS domain, the MPLS label is usually removed. A path between an ingress LSR and an egress LSR through which a packet traverses is known as a Label Switched Path (LSP). A particular LSP is defined for each packet at the ingress LSR. To support all LSPs defined in the network, the correct processing of labels should be carried out at all the LSRs within the network. For this purpose the Label Distribution Protocol (LDP) was developed [RFC3036]. The LDP protocol is a set of procedures and messages by which one LSR informs another of the meaning of labels used to forward traffic passing between and through them. The MPLS architecture allows for the possibility of more than a single method for distributing labels. There are proposals to use some other protocols for label distribution such as RSVP and BGP [RFC1267].

Traffic engineering is expected to be one of the important MPLS applications. MPLS support for traffic engineering makes use of explicitly routed LSPs that can be set up by some extensions to the existing label distribution protocols. Extensions to the LDP protocol to support explicitly routed LSPs are specified in [RFC3212]. Extensions to RSVP to support instantiation of explicit LSP are discussed in [RFC3209]. Extensions to BGP to support explicit LSPs are presented in [RFC3107].

3.6 DiffServ over MPLS

To provide QoS services, a solution combining DiffServ and MPLS technologies can be used. In DiffServ, packets are marked differently to create several packet classes. Packets in different classes receive different services. When IP packets enter a DiffServ domain, they are classified and marked at the ingress node. Afterwards, at each transit node, the packets are served accordingly to their assigned class. The service mainly includes scheduling treatment and drop probability for each packet. However, DiffServ architecture

does not provide any control mechanism for how traffic is routed in the network. The routing requirements of the system providing QoS service can be fulfilled by using the advantages of the MPLS technology.

MPLS is a forwarding scheme. MPLS can be used in combination with DiffServ for creation of tunnels between a pair of nodes that directly connect to a single autonomous system. MPLS will specify a next hop and DiffServ will specify the treatment of a packet waiting to make that next hop. The network diagram in Figure 3-6 illustrates the two distant hosts that are connected via a DiffServ/MPLS domain.

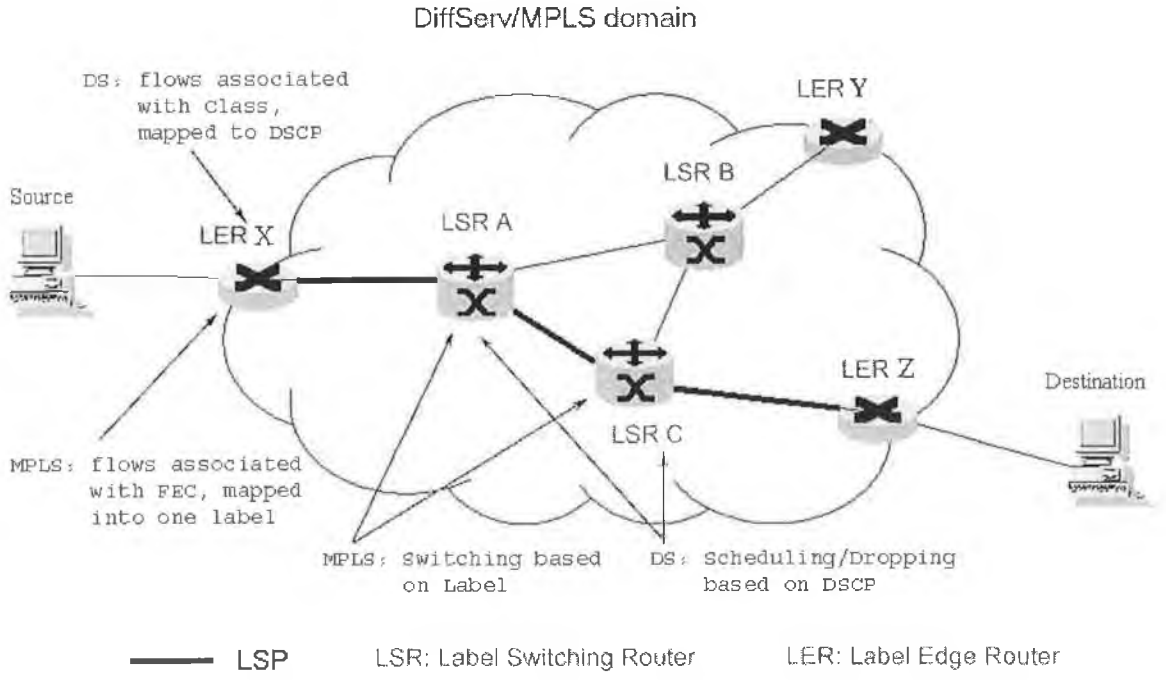


Figure 3-6: DiffServ/MPLS domain

A flexible solution for support of DiffServ over MPLS networks is defined in [RFC3270]. This solution allows the MPLS network administrator to select how DiffServ BAs are mapped onto LSPs so that he can best match the DiffServ and traffic engineering objectives within his particular network.

4 A Traffic Engineering Management System

4.1 Introduction

As was described in the previous section, the combination of DiffServ and MPLS techniques gives Internet Service Providers (ISPs) a flexible solution for support of QoS service in the network. A DiffServ/MPLS network provides to an ISP a means for delivering QoS services and useful mechanisms for traffic engineering. While performing traffic engineering of such a network, a number of problems arise. One of the major objectives of traffic engineering is avoiding network congestion. As congestion increases end-to-end delays, delay variation and packet loss, and reduces the predictability of network services, it is clearly a highly undesirable effect. To deliver QoS guarantees to the customer the network was typically over-provisioned to ensure that congestion never occurs during peak times. However, the ever-increasing demand for high quality bandwidth cannot always be met by over-provisioning, which is a very cost-ineffective approach. That is why during the past service providers have focused on deploying approaches to offering QoS guarantees while making more efficient use of the existing infrastructure. Balancing the load on the network reduces congestion and makes more efficient use of the available bandwidth. The goal of load balancing is to evenly distribute the load over the network and to avoid the use of highly loaded links, which makes the probability of congestion and therefore the probability of rejecting future connections considerably lower. The traffic engineering management system described in this thesis addresses the problems of load balancing of *QoS guaranteed LSPs*[1] in an MPLS domain. The system can set up QoS guaranteed LSPs between specified ingress-egress pairs. The path selection for LSPs is based on a developed routing algorithm aimed at making the best use of network resources while meeting QoS requirements. The algorithm is described in Chapter 5. In this chapter, we consider a network scenario typical for our traffic

[1] *A QoS guaranteed LSP* – an LSP set up for an aggregate of traffic with particular QoS requirements

engineering management system, specify the main objectives of the system and consider its architecture considerations and design choices.

4.2 Network Scenario

We consider a single administrative domain (Figure 4-1) where interior nodes and boundary nodes are grouped into Autonomous Systems (AS). An Autonomous System consists of a group of nodes administered by a single entity. We will consider a network where DiffServ and MPLS technologies are used to provide QoS guarantees. In DiffServ, packets are marked differently to create several packet classes. Packets in different classes receive different services. We assume that within an AS we have a single DiffServ domain that is a contiguous set of DiffServ (DS) nodes that operate with a common service provisioning policy and a set of PHB groups implemented in each node. PHB is the forwarding behaviour applied at a DS-compliant node to a *DS behaviour aggregate*[2]. To provide different levels of assurance, several PHB groups are defined.

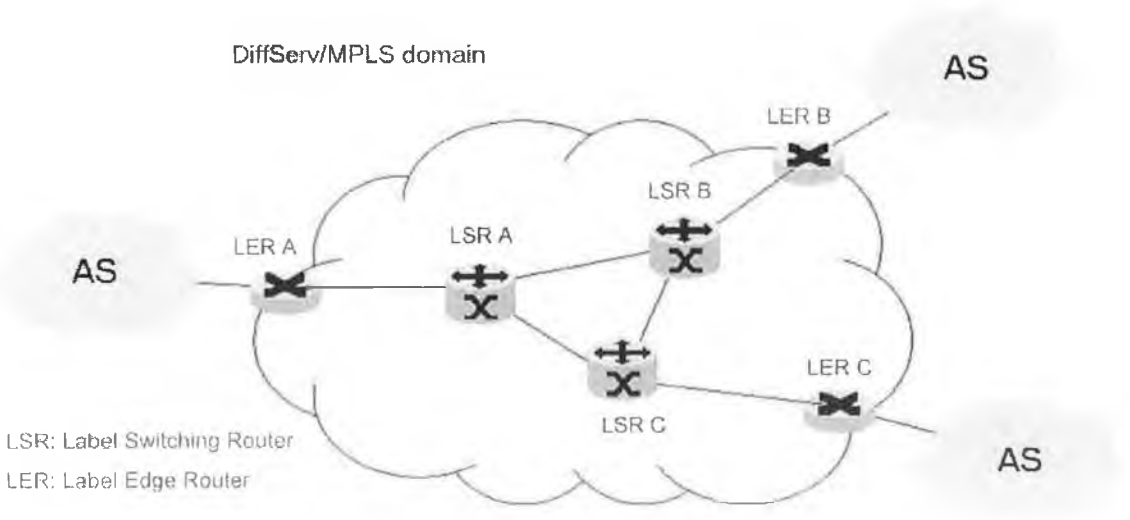


Figure 4-1: Autonomous system with DiffServ and MPLS technologies

MPLS is a forwarding scheme and it is used in combination with DiffServ for creation of LSP tunnels between a pair of nodes within a single AS. MPLS will specify a next hop and DiffServ will specify the treatment of a packet waiting to make that next hop.

[2] *DS behaviour aggregate* is a collection of packets with the same DS codepoint crossing a link in a particular direction.

Usually LSP tunnels are requested to be set up between an ingress and egress node. We assume that every node in the network can potentially be an ingress and egress point, and each node may have a number of customers connected to it. In order for a customer to receive differentiated services, it must contact an ISP for these services under Service Level Agreements (SLAs). In general, an SLA specifies the service classes supported and the amount of traffic allowed in each class. An SLA can be *static* or *dynamic*. Static SLAs are negotiated on a regular basis. In the case of dynamic SLAs, customers may request services on demand without subscribing to them. It is said that a system, dealing with such kind of requests, is for an *on-line* environment, where requests for establishing LSP tunnels can arrive any time one by one and where information about future requests is not available. A dynamic SLA can be requested by a customer through the automated services that provide dynamic creation of network services [CA03]. To our system, we can apply both static and dynamic SLA scenarios. The main point is that we consider all requests for SLAs are being received in an on-line fashion, that is when information about future requests is not available and when all routing decisions for LSPs should be based only on the current state of the network.

As an example, let us consider two distant machines A and B connected via an ISP's network (Figure 4-2). The ISP's network is a single AS with DiffServ/MPLS technologies deployed. The ISP may get the customers' request for a dynamic SLA to support traffic between *A* and *B* that will support 50 IP phone conversations. Our traffic engineering management system would be responsible for finding a QoS guaranteed path between the edge nodes *LER X and LER Z* and establishing an LSP between them. An explicit LSP can be set up in a MPLS network with the help of LDP or RSVP signalling protocols. Later we give an example of how an explicit route can be established using RSVP signalling protocol.

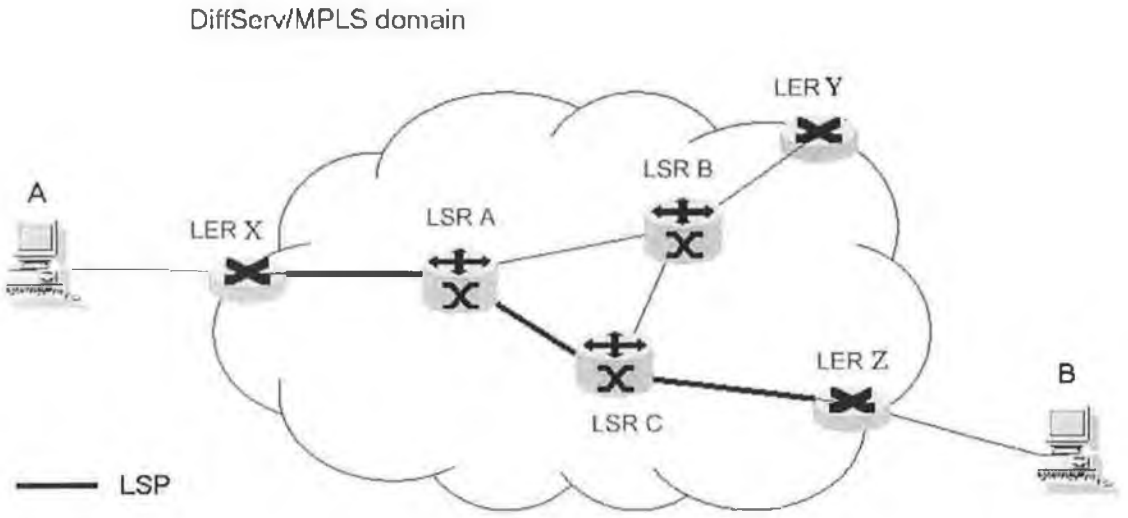


Figure 4-2: Two distant machines connected via an ISP's network

Let us summarize the main points of the network scenario:

- We have a DiffServ & MPLS network.
- Each node may be an ingress or egress node for prospective customers' requests.
- Each customer can send requests for a static or dynamic SLA.
- SLAs have QoS demands for latency, jitter and loss.
- When the traffic engineering system receives a request for a particular SLA, a decision about the optimal path and necessary resources (e.g. bandwidth) has to be made and establishment of an LSP has to be performed.

4.3 Objectives of the system

A key objective of the management system is to process customers' requests coming in on-line fashion while performing traffic engineering. The processing of customers' requests includes:

- finding QoS guaranteed paths

- establishing LSP tunnels
- performing resources allocation

Traffic engineering includes the ability of the system to route LSP tunnels around known points of congestion, thereby making more efficient use of the available resources.

The system's objective with respect to traffic engineering is performance optimisation of the network. The main goal of the optimisation task is to achieve the best resource utilisation. In the scope of this work, a good resource utilisation pattern is one in which the load is balanced. Having the load balanced allows network to avoid prospective future congestion states. With this in mind, the system is designed in a *pro-active* manner aimed at balancing the load on the network and minimising resource utilisation to avoid undesirable future network states.

4.4 Architecture considerations and design choices

This section discusses the issues pertaining to the general architecture of the presented system. The principal organisation of the network and its components is discussed. The main focus is on the Central Resource Manager responsible for monitoring and managing network resources, and on the routing strategy that can be applied to the network scenario outlined above.

4.4.1 Routing Strategy

In the following subsections, we consider constraint-based routing as a routing system that can assist in performance optimisation of our network. The goals of constraint-based routing can successfully be achieved by the explicit routing features of the MPLS architecture, which will be discussed in detail in the following subsections.

4.4.1.1 Constraint-Based Routing

Constraint-based routing is concerned with computing routes through a network subject to satisfying a set of constraints and requirements. The constraints and requirements may be specified by the network itself or by administrative policies. In a traffic engineering context, constraint-based routing may also seek to optimise overall network performance while minimising the costs (related to the constraints). Constraints may include bandwidth, delay, packet loss, hop count, and policy instruments such as resource class attributes.

Sometimes constraint-base routing is referred to as QoS routing [CH98], but, in fact, constraint-based routing is a generalisation of QoS routing. Unlike QoS routing which generally is concerned with routing individual traffic flows with QoS requirements, constraint-based routing is applicable to traffic aggregates as well as flows and may be subject to other constraints (besides QoS requirements), such as policy constraints.

The concept of constraint-based routing within the context of MPLS traffic engineering requirements in IP networks was first defined in [RFC2702]. Being a path-oriented technology, MPLS has made constraint-based routing feasible and attractive in public IP networks. In an MPLS context, a constraint-base routing system can use two methods for selecting LSP for a particular FEC: hop-by-hop and explicit routing. As we need the flexibility to have arbitrary routes, we use explicit routing, which is described in detail in the next section.

4.4.1.2 Explicit Routing

One of the main objectives of traffic engineering is to route traffic while balancing the load in the network. This is usually done by redirecting packets to other routers than the shortest path calculated by Interior Gateway Protocols (IGPs) that use hop-by-hop calculations (e.g. RIP, OSPF, ISIS). As the best path calculated by these protocols can become congested at peak times, the need for more sophisticated routing strategy is evident. In this section, we describe the explicit routing and discuss how it can be used to make the best out of the available resources, spreading the load over several paths. We consider as well how explicit routes are supported by MPLS technology.

In general, there are two options for route selection: *hop-by-hop* and *explicit routing*. Hop-by-hop routing is used in conventional IP routing. It allows each node to independently choose the next hop for a packet based on its destination address. As result, packets with the same destination follow the same path, which is usually the shortest path in the network. While it is sufficient to achieve connectivity, it does not always result in efficient use of network resources and is considered as being not efficient from the traffic engineering point of view. Explicit routing was introduced to address the shortcomings of current IP hop-by-hop routing schemes. With explicit routing, a path is *explicitly* specified for a packet (or group of packets) as a sequence of hops at one point in the network (possibly an ingress or egress node). With this technique, packets destined for the same destination may follow different paths; this enables much greater control over how the traffic is routed in the network, which in turn can be used to balance the load much more effectively.

Let us assume that Internet Service Provider (ISP) has the network topology presented in Figure 4-3. We then suppose that two ISP subscribers S_1 and S_2 are generating packets that are addressed to the destination S_3. In order to balance the load in the network, the ISP may decide that packets from S_1 should follow the route *A-B-E-F-D* and packets from S_2 should follow the route *B-C-D*. Since it requires that packets going through the node B, with the same destination, be sent on separate routes, the explicit routes in the network have to be defined.

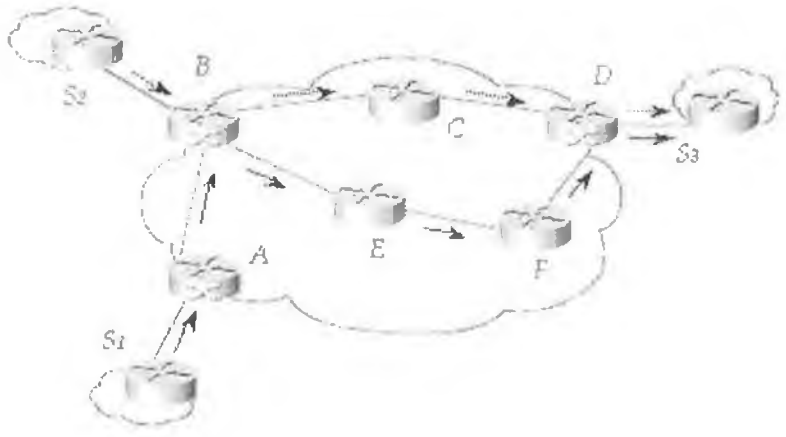


Figure 4-3: Explicit routing example

Setting up an explicit route in the network is supported in both conventional IP networks and MPLS networks. The corresponding techniques used for this purpose are known as *IP source routing* and *MPLS explicit routing* respectively. In this section, we will describe both of them and explain why MPLS explicit routing provides a more efficient way to establish paths for IP traffic.

4.4.1.2.1 IP Source Routing

The notion of *IP source routing* is usually referring to a technique whereby the source of an IP packet can supply routing information to be used by the routers in forwarding the packet to the destination. The IP protocol specification [RFC791] provides a means to specify in the IP packet header the route that a packet should take going through the network. This route data is attached to a packet in the "options" field of the IP header and is composed of a series of Internet addresses. As a packet travels through the network, each router will examine the route data and choose the next hop to forward the packet to.

There are two types of source routing defined: *strict source routing* and *loose source routing*. In strict source routing, the sender specifies the exact route the packet must follow. In loose source routing, the sender gives one or more hops that the packet must go through.

Source routing has not been widely adopted in IP routing and in general is seen as impractical. It is usually used more for debugging and diagnosis than for general routing purposes. The main disadvantage of IP source routing is that path must be contained in each IP header, which with lengthy paths considerably increases the size of IP header and system overhead. Moreover, often the host does not have knowledge about the network topology and hence is not in a position to suggest a good route.

4.4.1.2.2 MPLS Explicit Routing

MPLS architecture provides a more efficient way to define explicit paths for IP traffic than IP source routing. In IP source routing, defining an explicit path would require that addresses of all hops along the path from source to destination are included in each sent packet, which is not efficient it terms of packet overhead. In MPLS, establishing an LSP

between LSRs takes place only once at the setup time. After establishing an LSP, each packet carries only the label and subsequent communication will only require that LSRs switch the label. These operational features of MPLS technology provide a very efficient and flexible way to support explicit routes.

Explicit routing in MPLS provides control over the routing of LSPs, which is required for both policy and network efficiency reasons like load balancing. In MPLS, explicit routing is supported by both LDP and RSVP protocols. These are two signalling protocols that perform similar functions in MPLS networks. While either of these protocols can be used for setting up LSPs in the system described in this thesis, we give a more detailed description only for RSVP protocol. The mechanisms for support of explicit LSPs using LDP are given elsewhere [RFC3212].

Setting up Explicit Paths Using RSVP

All the necessary extensions for RSVP protocol to establish LSPs in MPLS are defined in [RFC3209]. The document contains all the necessary objects, packet formats and mechanisms required to establish and maintain explicit LSPs. The defined extensions to the original RSVP protocol give it a number of new capabilities that support operation of LSP-tunnels in an MPLS domain such as:

- establishment of explicit label switched paths
- allocation of network resources (e.g., bandwidth) to explicit LSPs

Example

Let us consider the establishment of an explicit LSP in an MPLS domain, Figure 4-4.

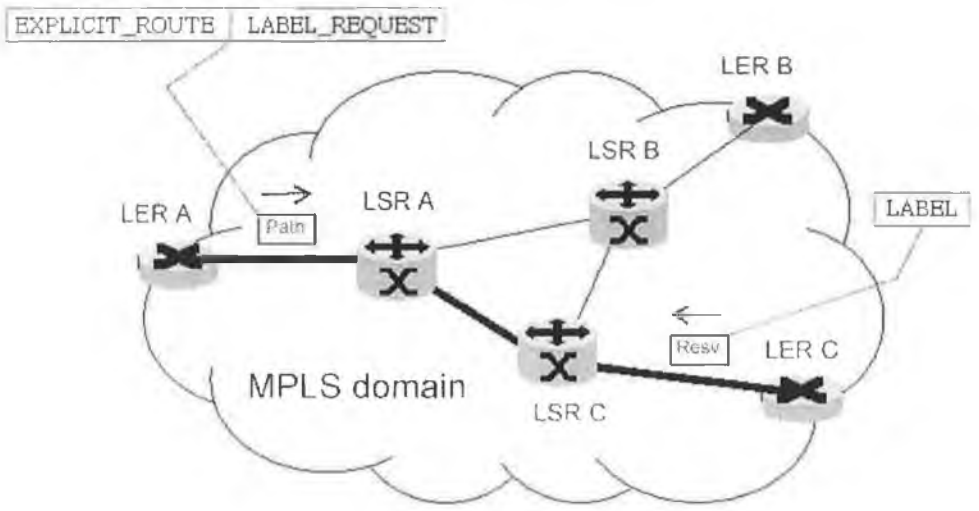


Figure 4-4: The establishment of an explicit LSP: *LER A – LSR A – LSR C – LER C*

A request for setting up an explicit route is initiated by the ingress router *LER A*. We suppose that the ingress router has knowledge of a route that meets the tunnel's QoS requirements and makes efficient use of network resources (an algorithm used to compute explicitly routed paths is described further in Chapter 5). To set up this route as an explicit LSP, the ingress node *LER A* creates an RSVP *Path* message and inserts an *EXPLICIT_ROUTE* object and *LABEL_REQUEST* object into it. The *EXPLICIT_ROUTE* object contains the route as a sequence of LSRs. The *LABEL_REQUEST* object carries the label binding information that allows the establishment of LSP along the explicit route. The *LER A* router sends then the *Path* message along the route specified by the *EXPLICIT_ROUTE* object. Each intermediate LSR along the path installs *Path* state. When the destination egress node *LER C* receives the *Path* message, it detects the *LABEL_REQUEST* object and initiates the setup of an LSP along the explicit route specified by the *EXPLICIT_ROUTE* object. *LER C* builds up an RSVP *Resv* message and inserts a *LABEL* object, specifying the label binding. Then *LER C* sends the *Resv* message upstream to *LER A*, using installed reverse routing state. While the *Resv* message is routed to *LER A* each intermediate router along the path inspects the *LABEL* object and updates its local label binding for the node upstream. As a result, an LSP is established along the explicit route. For resource reservation, the normal RSVP procedures may be used.

4.4.2 Offline Routing vs. Online Routing

A traffic engineering system can perform the computation of routing paths offline and online. Accordingly, there are two routing schemes that can be applied to a particular scenario - *offline* and *online* routing. In offline routing all LSP tunnels to be routed and their resource requests are known at the time routing is done. The objective of routing is obviously to route all these requests while making the best use of network resources. This objective can be met very efficiently since all requests are known at the time of routing. This is a great advantage of offline routing. However, in practice, it is more likely that paths for new requests have to be set up after the paths for initially expected requests have already been established. In this case, the routing paths have to be found and set up in real-time. The routing strategy that should be applied to this scenario is known in traffic engineering as online routing.

In this thesis, we consider a scenario where requests for establishing LSP tunnels arrive one by one and where information about future requests is not available; therefore, the main interest of this work with respect to the routing strategy is online routing.

4.4.3 Centralised Model vs. Distributed Model

From the point of view of how the computation of routing paths is organised in the system, we can distinguish two models: centralised and distributed. In the centralised model, there is a central route server, which performs the calculation of routing paths on behalf of each router. The central server collects periodically the network-state information from all routers, accepts their requests for establishing new paths and returns them routing decisions. In the distributed model, a routing decision is made by each router autonomously. The routing protocols usually used in this case are link-state or distance-vector protocols. When using link-state protocols (OSPF, IS-IS), each node within a network sends out information about its links to all other nodes. In the case of distance-vector protocols (RIP), each node informs its neighbours of its routing table. In both cases, each router has a means to get some knowledge about network topology to make local routing decisions.

Each of these models has its advantages and disadvantages. Having a central authority to make all routing decisions is a big advantage, which allows us to better optimise the traffic in the network. However, the centralised model needs high processing power to process all requests in the network and high bandwidth control channels to collect network-state information. From a robustness point of view, the centralised system represents a single point of failure, which cannot of course provide extensive fault tolerance. The centralised approach has also problems with scalability issues: as the number of routers on the network expands, the requirements of the central route server increase considerably. Conversely, the distributed model is scalable, but it does not provide such good possibilities to optimise routing as the centralised model does. Moreover, currently available routing protocols using distributed approach may not have all the required features to support QoS and may need some extensions.

For some deployments, a centralised approach is a reasonable approach. As the distributed approach is more complex and difficult to design and manage, we focus on the centralised solution here.

4.4.4 Central Resource Manager

We introduce the Central Resource Manager (CRM) as the central authority responsible for monitoring and managing resources within the network. The CRM makes all decisions to route traffic according to QoS requirements. An autonomous system with the CRM is depicted in Figure 4-5. The CRM makes all decisions about appropriate routes based on the measurements of the current network state, and thus the CRM needs access to information on the QoS resources currently available in the network. If there are no resources available within the network for a requested SLA then the request should be rejected or a negotiation process with the customer should be started.

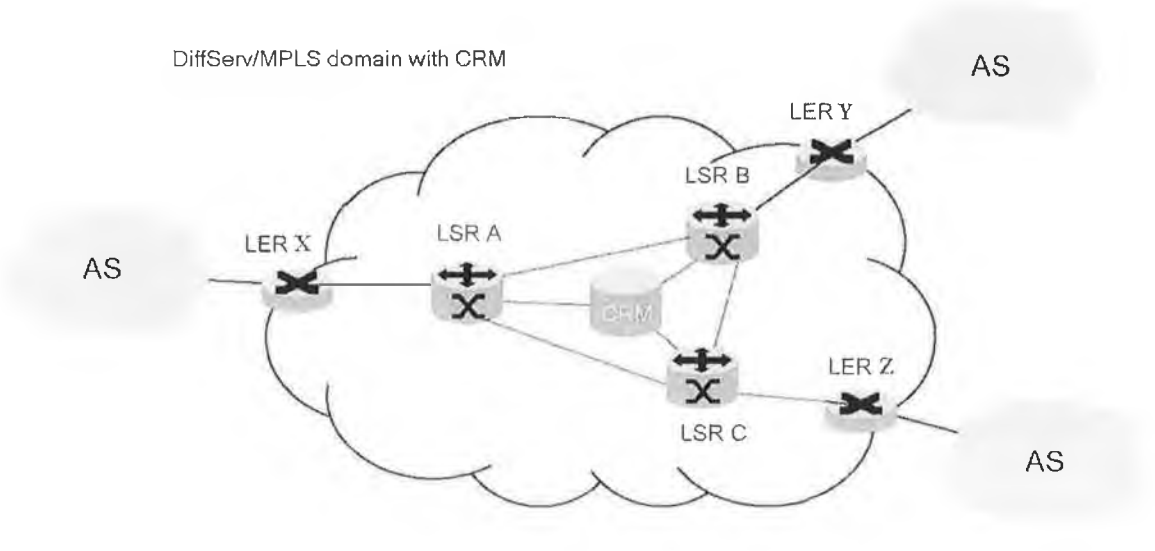


Figure 4-5: Autonomous system with the CRM

Requirements for the CRM

Let us specify the main requirements for the CRM. The CRM should be responsible for:

1. Maintaining a database containing a topological map of the network domain and information about the current state of the network resources. There are two approaches to obtain the network-state information. First approach is to use one of the standard network management protocols such as Simple Network Management Protocol (SNMP) [RFC1157] and the second approach is to use a link state protocol [RFC1247, RFC1142]. Anyone of these approaches can be used to maintain the database if it provides support for monitoring the input network-state variables of our traffic engineering system such as available bandwidth on links and buffers usage. The input network-state variables are discussed in more detail in Chapter 5. For example, when using SNMP protocol, the support for monitoring QoS parameters necessary for our system can be provided by the corresponding Management Information Bases (MIBs) [RFC1156]. When using a link-state protocol, the necessary traffic engineering extensions should be implemented as suggested in the documents [KY02] and [SL02].

   Ideally, the CRM should have the most current view of the bandwidth available on all links in the network, so that it can make the most accurate routing decisions.

Unfortunately, this then calls for very frequent updates, which can be not very practical. In general, there is always a trade-off between the protocol overhead of frequent updates and the accuracy of the network state information that the path selection algorithm depends on. Some possible link state update policies addressing this problem are outlined in [RFC2676].

2. Finding the routing paths for all incoming routing requests. When a new QoS guaranteed LSP is to be set up between a specified ingress-egress pair, an ingress node redirects a routing message to the CRM. Upon receiving a routing message, the CRM computes explicitly the routed path by running a routing algorithm aimed at making the best use of network resources while meeting the QoS requirements of the request. The routing algorithm is described in Chapter 5.

3. Setting up LSPs. Once a path that meets the QoS requirements of a flow has been found, the CRM is responsible for establishing an LSP between an ingress LSR and an egress LSR to make sure that the flow follows this path. The correct LSP setup and label distribution should be carried out at all the LSRs along the path. For this purpose, the LDP or RSVP protocols can be used. While establishing LSPs, the allocation of resources for the new flow is carried out as well. The extensions to the RSVP protocol necessary for setting up LSPs and allocating network resources are given in [RFC3209]. An end-to-end setup mechanism to establish LSPs and to provide means for reservation of resources using LDP is described in [RFC3212].

5 System Performance Optimisation

5.1 Introduction

The system introduced in the previous chapter provides general mechanisms for delivery of QoS services. In this chapter, we will define an optimisation problem for this system and propose a solution. The proposed solution seeks to optimise network performance at both the traffic and resource levels. This can be achieved by meeting traffic and resource oriented requirements that are described in the first sections of this chapter. Finally, at the end of the chapter, the proposed routing algorithm is presented as a solution to the defined optimisation problem.

5.2 Traffic Oriented Performance Measures

To perform network optimisation at the traffic level we consider some of the traffic oriented performance measures that are associated with the end-to-end QoS requirements. These include latency, packet loss and jitter. In the following subsections, we discuss these measures and address some issues of how they can be estimated.

5.2.1 Latency

Latency is the time it takes for a data packet to move across a network connection. In fact, it is end-to-end delay of transmitting a packet. Here, the terms *latency* and *end-to-end delay* are used interchangeably. Many kinds of network interactive applications, like VoIP and video teleconferencing are very sensitive to the latency requirement. Thus, providing end-to-end delay requirements is a very essential task of the QoS techniques.

The problem addressed in this thesis is how latency can be estimated and guaranteed. The next section describes some approaches to estimating latency, Section 5.3.2.2 explains how latency can be guaranteed with the help of reserving enough bandwidth and Section 5.5.4 shows how latency requirements are met in our algorithm.

Approaches to estimating latency

This section gives an overview of how end-to-end delay along a path within the network can be estimated.

We consider a computer network represented by a graph $G = (V; E)$ with n nodes and m edges or links. A message of size r must be transmitted from a source node s to a destination node d. A message transmitted on the network incurs three types of delays:

- *Link Propagation Delay:* is a delay related to the speed of transmission of an electrical signal in a transmission line. Propagation delay can be defined as the time required for a packet to propagate from one end of the link to the other. For each link $e = (v1; v2)$, there is a link-delay $d(e) \geq 0$ such that a message of unit length sent via e from node v_1 at time t will arrive at node v_2 at time $t + d(e)$.

- *Transmission Delay:* is a delay associated with putting a packet of a certain size onto a transmission system. Transmission delay for a particular packet depends on bandwidth availability on a link. Each link $e \in E$ has a bandwidth $b(e) \geq 0$. Once initiated, a message of r units can be sent into link e in $r/b(e)$ time.

- *Queuing Delay:* is the time $q_v(r)$ a packet of size r spends in the buffer waiting for packets that arrived ahead of it to leave a router or host.

Consider a path P, from source $s = v_0$ to destination $d = v_k$, given by $(v_0; v_1)$, $(v_1; v_2)$, ..., (v_{k-1}, v_k), where $(v_j, v_{j+1}) \in E$, for $j=0,1,...,(k-1)$, and $v_0, v_1, ..., v_k$ are distinct. Let $e_j = (v_j ; v_{j+1})$. Then, the end-to-end delay of path P in transmitting a message of size r is given by:

$$t(r, P) = \sum_{j=0}^{k-1} r / b(e_j) + \sum_{j=0}^{k-1} d(e_j) + \sum_{j=0}^{k-1} q_{vj}(r)$$

Transmission and propagation delays can be accurately determined since they depend only on links. Queuing delays $q_{vj}(r)$ are very hard to estimate. In general, methods for the estimation of queuing delays are based on either measurements or probabilistic approaches. To calculate the queuing delays requires an accurate model of the whole system's traffic, and then some approach to solving that model to obtain the queuing delays at each queue. This typically results in a very complex approach.

5.2.2 Packet loss

To deliver the packets over the network for real-time applications, the UDP protocol is mostly used (or more specifically the RTP protocol [RFC3550], which runs on top of UDP). The normal TCP retransmission schemes are not appropriate in this case due to high delay sensitivity of real-time applications. The disadvantage of the UDP protocol is that it cannot guarantee the delivery of all packets. Packets can be lost during the peak loads or periods of congestion. Packet loss is the amount of packets dropped during a network session. In other words, packet loss refers to how many packets never reach the final destination. For efficient use of an application, packet loss must be kept below a certain value. For example, some QoS applications for Voice over IP (VoIP) define the following QoS services [CA02]:

- < 0.2 % – GOLD service
- 5 % – SILVER service
- 10 % – BRONZE service

Packet losses greater than 10 % are usually intolerable [MU01].

In general, the estimation of packet losses in a network is a very complex problem. Approaches to computing packet loss are normally approximation techniques based on statistical methods. Approximation techniques may be based on a particular kind of buffer and traffic models:

- *Buffer model*

  There can be two principal models chosen for estimating packet loss: *bufferless* and *buffered*. These two models can be used to model different aspects of buffer behaviour and its affect on traffic. In the case of the buffered model a particular buffer implementation, including nodal traffic control functions implemented in the network (e.g queue management), should be taking into account.

- *Traffic model*

  Of course, packet loss estimations should be performed considering a certain type of traffic traversing a network. Approximation techniques are normally developed for a particular kind of traffic (e.g. data, voice, video), taking into account its statistical parameters. For example, in [NA91] authors describe an approach to computing packet loss for three different models of voice traffic. The authors analyse the accuracy of each of those models (renewal process, Markov Modulated Poisson Process, fluid flow approximation) and their applicability.

5.2.3 Jitter

Jitter is the variation in inter-packet arrival rate. In [RFC2598], authors define jitter as "the absolute value of the difference between the arrival times of two adjacent packets minus their departure times, $|(t_2-t_1) - (t_{02}-t_{01})|$". Jitter is caused by the data packets taking different lengths of time to reach their destination (Figure 5-1).

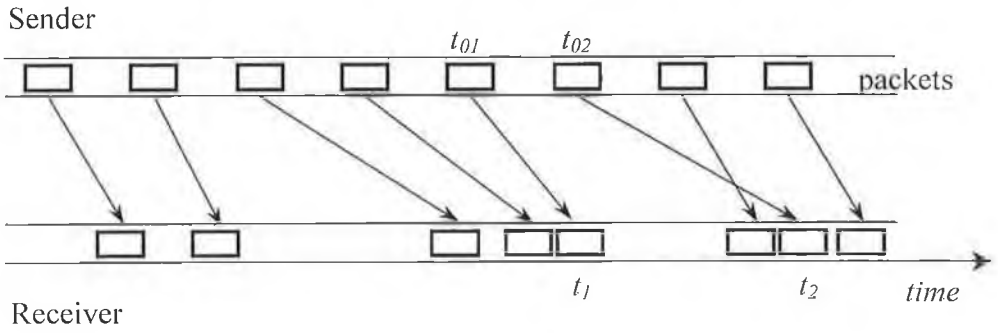


Figure 5-1: Variations of the interval between successive packets (jitter)

Variations of delays mostly happen due to the queuing effects, as queuing delays is the part that is most variable for a packet transiting a network. Jitter is usually measured as the variance of delay. For example, for VoIP service a variation between when a voice packet is expected for playout and when it actually is received causes a discontinuity in the real-time voice stream. That is why it is very important to ensure that jitter remains below some bound and to smooth out the data flow. In general, jitter is removed by buffering in the receiver that collects packets and stores them for some amount of time to permit the slowest packets to arrive in time to be "played" in the correct order. The implementation of removing the packet delay variation is usually known as *jitter buffer*. The jitter buffer is capable of sorting out-of-order packet payloads and discarding duplicate ones according to the provided timestamp information. Each jitter buffer adds to the overall delay increasing end-to-end latency.

5.3 Resource Oriented Performance Measures

For the optimisation of the network performance at the resource level, we consider such concepts as load balancing and effective bandwidth. By trying to find a good load-balanced solution and optimal amount of bandwidth to be reserved for a certain traffic aggregate, a good resource utilisation can be achieved. In the scope of this work, we consider some of the measures of load balancing (e.g. variation of link load) and of link utilisation (e.g. average link load) as *resource oriented performance measures*. That is, measures that can be used to optimise and thus achieve good network resource utilisation.

The concepts of load balancing and effective bandwidth are considered next.

5.3.1 Load Balancing

The purpose of load balancing is to distribute the load evenly across a network so as to ensure that some links are not heavily loaded while others are lightly loaded. Load balancing is especially important for networks where it is difficult to predict the number of requests to route traffic in the future. If a network deploys a routing algorithm seeking to balance the load, the probability of congestion and therefore the probability of rejecting future requests are considerably decreasing.

Currently commonly used shortest path algorithms select a path with as few hops as possible. Even though this approach is a natural way to limit resource consumption, it does not perform well from the load balancing point of view. Since shortest path algorithms are not designed to balance the load, they can be used in such a way that they result in situations where the load is not balanced on the network. For example:

- the shortest paths of different traffic streams may converge on specific links or router interfaces;
- traffic streams can be routed through the links or router interfaces which does not have enough bandwidth to accommodate it.

In the case of shortest path algorithms, the path computation is usually based on certain link metrics that are normally based on static quantities (e.g. cost, delay) and may be assigned administratively according to local criteria. However, static link metrics does not reflect the traffic load in the network, traffic attributes or capacity constraints. That is why shortest path algorithms result in traffic concentration being localised in subsets of the network infrastructure and potentially causing congestion.

Some of the recently developed QoS routing algorithms [CH98] address the problem of congestion by trying to avoid the overloaded links. In this case, link metrics used by the algorithms are normally based on dynamic quantities that may be functions of a network congestion measure such as unused link capacity, delay or packet loss. For example, the shortest-distance algorithm [MA98] uses the inverse residual bandwidths of links as a link

metric and selects a path with the smallest sum of the inverse residual bandwidths of all links along the path. Compared to the shortest path algorithm, this approach gives a much better load-balanced solution.

5.3.2 Effective bandwidth

Here, we discuss the notion of the effective bandwidth and our objectives of using the effective bandwidth in our approach.

5.3.2.1 Definition

The concept of effective bandwidth was originally proposed by Hui [HU88]. The concept was developed with regard to the admission control problem that focuses on how to decide whether or not a particular connection can be carried on the network. In the developed approach, the requirements of each connection are encapsulated in the notion of effective bandwidth. This makes the admission control decision easily made: if the effective bandwidth assigned to the requested type of the connection exceeds the residual capacity of the resource, then the new source is blocked. Even though this is not always a good model – it can result in inefficient use of the resource, it is a simple approach. For example, some QoS constraints (e.g. loss, jitter) can be incorporated in the notion of effective bandwidth [KL00]. Once the effective bandwidth is determined, efforts can be focussed on solving the routing problem.

The calculation of the effective bandwidth is in general a very complex process based on statistical methods. It is clear that the effective bandwidth of a connection should be some value between its mean rate and its peak rate. If the effective bandwidth is equal to the peak rate of a connection, then clearly there will be wasted bandwidth on the link, as the connection will likely not send bytes at the peak rate continuously. Conversely, if the effective bandwidth is equal to the mean rate of a connection, then there may be times when there will not be enough bandwidth to provide service, as the connection will occasionally send bytes at its peak rate. Thus, the value of the effective bandwidth should be between the mean and peak rate [KJ99]. The exact value of the effective bandwidth

assigned will depend on the QoS constraints (e.g. end-to-end delay, maximum allowable loss rate), on the number of flows aggregated together and on the stochastic characteristics of the individual traffic streams.

There can be different ways of calculating the effective bandwidth. For example, in Kelly's work [KL96] the effective bandwidth of the source is defined as:

$$B_{eff}(s,t) = \frac{1}{st}\log \mathrm{E}\left[e^{sX[0,t]}\right]$$

where s is the space-scale and t is the time-scale ($s>0$, $t<\infty$). $X[0,t]$ is the amount of workload produced by a source in a time interval of length t. Space scale s is a value that is specific to a particular link's operating point and in general is complex to calculate. It depends on the traffic source and on characteristics of the resource such as its capacity or buffer lengths, scheduling policy and required QoS. The effective bandwidth is calculated for several of the most common stochastic models of traffic sources in [KL96]. These include bufferless and finite buffer models for periodic, fluid, Gaussian and on-off input sources. The practical application of the effective bandwidth concept is analysed in [CO99] with examples on voice traffic and MPEG-1 compressed video traffic.

There could also be approximate methods for estimating effective bandwidth. For example when calculating effective bandwidth that guarantees that packet loss will not exceed 10^{-9}, the approximate formula can be used [JR03]:

$$B = 1.2m + 60\frac{\sigma^2}{c}$$

where m – mean source rate, σ – variance of the source rate and c – link rate. Using this formula, it is very easy to estimate the required effective bandwidth. In the case when variance of the source is not known it can be calculated as $\sigma^2=m(p-m)$, where p is peak rate.

In this thesis, we refer to the effective bandwidth as the minimum amount of bandwidth needed by the source to obtain the required QoS. The objective of this thesis with respect to effective bandwidth is to convert an SLA with QoS constraints into an effective

bandwidth requirement for the LSPs and to show how this can be used to balance the load on the network.

5.3.2.2 Bandwidth dependencies

As the effective bandwidth has to reflect the amount of resources necessary to guarantee the requested QoS for a source, it should take into account many parameters indicating particular properties of a connection, such as QoS demands and traffic flow characteristics. Below we describe what should be taken into account while calculating the effective bandwidth for an aggregate of flows. In practice, a particular method for calculating effective bandwidth usually does not take all these parameters into consideration. For example, VoIP applications are very sensitive to end-to-end delay and jitter requirements but quite tolerant to packet losses, therefore when calculating effective bandwidth for VoIP applications the main focus may be on providing only latency and jitter requirements.

Latency

The bandwidth reserved for a connection determines the rate with which packets traverse the path. Hence, effective bandwidth should be large enough to provide end-to-end delay (latency) requirements for packets. To meet this requirement, effective bandwidth must not be less than the minimum bandwidth providing the requested end-to-end delay.

How our approach guarantees latency is described further in Section 5.5.4.

Loss

Packet loss is another constraint that should be considered while calculating effective bandwidth. How can effective bandwidth guarantee packet loss? Effective bandwidth reflects the possible rate at which the buffer can be served and therefore it has an effect on the queue length and packet loss. The most commonly used methods for estimating effective bandwidth take into account the available buffer space in the nodes along the path and determine the required rate at which to serve the buffer such that the buffer loss is no more than some specified value. Also, there are some methods that use the notion of

effective bandwidth along with notion of effective buffer [YY01]. In these cases, providing packet loss is a matter of trade-off between the amount of bandwidth and buffer space.

Jitter

The particular amount of bandwidth can also guarantee jitter requirements. Usually this is performed by mapping jitter to latency requirements. This is possible because removing jitter is performed by buffering in the receiver, which takes some time and leads to the increasing of the overall delay. This contribution to the overall delay is then taken into account when considering latency requirements.

The number of nodes along the candidate path

The more hops a flow traverses, the more resources it consumes. For example, a 1-Mb/s flow that traverses two hops consumes twice as much bandwidth as one that traverses a single hop. Therefore, effective bandwidth should be a function of the number of hops along the path.

Traffic flow characteristics

Of course effective bandwidth depends on the specific features of traffic traversing the network. Methods used for estimating effective bandwidth use different models and statistic approaches to describing a particular flow or an aggregate of flows. Examples of calculating the effective bandwidth for some common types of traffic are given in [KL96]. In this work for our simulations, we use a simplified method for calculating the effective bandwidth (Section 6.3.1.2).

5.4 Performance Optimisation at the traffic and resource levels

The optimisation task of meeting both traffic and resource oriented requirements faces the problem of satisfying multiple QoS constraints. In general, this problem is known to be intractable for most realistic constraints. However, in practice, there are some approaches to finding compromise heuristic solutions. In this section, we address the problem of

satisfying multiple QoS constraints and define our approach to finding an optimal path in the network.

5.4.1 The problem of satisfying multiple QoS constraints

The optimisation of the network performance at both traffic and resource levels creates the problem of optimal path computation on two or more independent QoS-metrics. The objective is to find an optimal path that is able to satisfy multiple QoS constraints related to traffic and resource oriented measures.

Finding QoS-constrained routes is the subject of QoS routing [RFC2386]. There are a number of algorithms developed in QoS routing for finding constrained-based routes [CN98]. The complexity of computation algorithms for finding the optimal path depends on the metrics chosen for the routes. Usual route metrics are delay, jitter, bandwidth, hop-count, loss probability and monetary cost.

There are three basic classes of metrics:

additive : $d(P) = d(i,j) + d(j,k) + \ldots + d(l,m)$ (delay, jitter, cost, hop count)

multiplicative: $d(P) = d(i,j) \times d(j,k) \times \ldots \times d(l,m)$ (1- loss probability)

concave: $d(P) = min\{d(i,j), d(j,k), \ldots, d(l,m)\}$ (bandwidth)

where $d(i,j)$ is a metric for link (i,j) and P is a path $P=(i,j,k,\ldots l,m)$ between nodes i, m. Figure 5-2 gives an example of a network state with different classes of metrics.

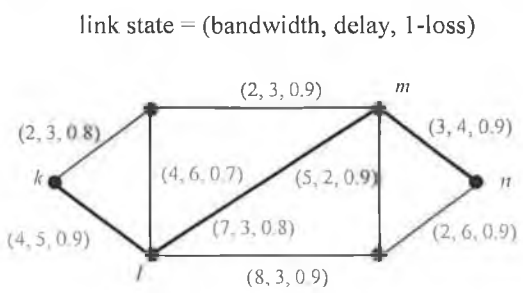


Figure 5-2: Example of the three classes of metrics

Generally, routing algorithms select routes that optimise one or more of these metrics. There is a theorem [WC96] that shows 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. It means that algorithms that use any two or more of delay, jitter, cost, hop count, and loss probability as metrics, and optimise them simultaneously can not be computed in polynomial time. Therefore, polynomial-time algorithms can be used only when combination of bandwidth and one of the other metrics, for example, bandwidth and end-to-end delay or bandwidth and cost. However, the proof of NP-completeness in [WC96] is based on the assumptions that all the metrics are independent. It was shown in [MA98] that in networks with rate dependent scheduling (e.g. Weighted Fair Queuing), the QoS metrics (e.g. delay, bandwidth, jitter) are not independent but correlated. Thus, polynomial-time algorithms can be used for computations. Some of these algorithms are described in [MA97]. As it is hard to find a path in a network which satisfies all requirements, these algorithms first find some candidate paths based on the combination of bandwidth and delay or hop count metric. Other requirements, for example, loss probability, jitter and cost can be considered later in the admission control. The deficiencies of these algorithms used in QoS routing are that they do not provide an optimal solution for all the QoS requirements [MA98] and the admission control is quite complex.

5.4.2 Optimal path concept

Ideally, we consider the optimal path as the one that satisfies all QoS constraints of the incoming request while trying to provide the best utilisation of network resources. The best utilisation of network resources is seen as a certain trade-off between balancing the load on the network and minimising the resource consumption.

We suppose that all QoS constraints can be converted into an effective bandwidth requirement for an LSP [KL00]. By guaranteeing the effective bandwidth for a connection throughout the network, the QoS requirements can be met. The resource utilisation objectives can be attained by balancing the aggregates of effective bandwidths on the network. The load balancing and minimisation of resource consumption can be achieved by choosing appropriate cost functions for the links in the network and then by selecting the route with the minimum cost.

However, the process of finding such a path can be very laborious and inappropriate in reality where a decision has to be made as quickly as possible. Thus, a certain compromise has to be found between the level of network optimisation and the time it takes to make a decision to find a path. An algorithm providing such a solution is presented later in the next section.

5.5 Optimisation of the Routing Functions

As a solution to the network performance optimisation problem of the traffic engineering system, we propose a new routing algorithm aimed at making the best use of network resources while meeting QoS requirements. Before describing the algorithm, we give a brief overview of the well-known QoS routing algorithms and provide some motivations for a new one. All the QoS algorithms considered here are based on Dijkstra's algorithm or its slight modification. Our proposed algorithm is also based on Dijkstra's algorithm. Therefore, a description of Dijkstra's algorithm is given in this section as well.

5.5.1 QoS routing algorithms

QoS routing algorithms solve the problem of finding the path to be used by the packets of a flow based on its QoS requirements, e.g., bandwidth or delay. The goal of such QoS routing algorithms is to satisfy the QoS requirements for every admitted connection while achieving global efficiency in resource utilisation. To achieve this goal, routing protocols and routing algorithms are developed. In this section we give a short overview of the most commonly used QoS routing algorithms.

The goal of achieving the efficiency in resource utilisation can be interpreted in different ways. For instance, the goal can be either to minimise the resources utilisation of selected paths or to distribute the load evenly through the network. In the first case, it is better to select the path with the minimum number of hops or the path requiring the minimum bandwidth. However, in the second case, the path with the minimum load (e.g. the minimum sum of the inverse bandwidths of all links along the path) provides better solution. Therefore, depending on the optimisation task it is possible to define several routing algorithms [MA98, ST97]:

Widest-shortest path - selects a path with the minimum hop count. If there is more than one path with the minimum hop count, the one with the maximum available bandwidth is selected.

Shortest-widest path - selects a path with the maximum available bandwidth. If there are several such paths, the one with the minimum hop count is selected.

Shortest-delay path - selects a path giving the minimal end-to-end delay if the maximal available bandwidth is reserved. That is, the algorithm checks from some list of paths what the delay would be if all the available capacity were reserved and uses the path that results in the minimum delay. If there are several such paths, the one with the minimum hop count is selected.

Shortest-distance path - selects a path with the shortest *distance*. In general, distance can be defined in any way (e.g. in terms of hop count, delay). With respect to QoS routing algorithms, it is usually defined as the sum of the inverse bandwidths of all links along the path.

All these algorithms represent a broad spectrum of different tradeoffs between resource utilisation and network load distribution. With respect to a particular network state, the performance of these algorithms can vary. For example, with regard to the *call-blocking rate*[3], the widest-shortest-path algorithm gives the best results for a network with heavy loads, while the shortest-delay-path algorithm performs better for light loads [MA98]. We analyse this later in Chapter 6, when we discuss network simulation results for some of these algorithms for both heavy and light loads.

5.5.2 Motivating the new algorithm

To find a path satisfying a number of QoS requirements, while achieving global efficiency in resource utilisation, can be in general a very complicated and resource consuming task. That is why all the QoS routing algorithms described here are designed with the intention that they could be relatively simple to use and at the same time provide good efficiency in resource utilisation. Of course, there are a number of disadvantages coming from their relative simplicity. For example, as was mentioned in the previous section, the performance of these algorithms can vary as the load of the network changes [MA98]. Therefore, development of a more sophisticated algorithm that would better accommodate to changes of the network load would be one challenging objective. Another objective, for example, would be to take into account possible future requests to be routed in the network.

In this work, we address one problem of the previously discussed QoS routing algorithms. They do not take into account the amount of resources necessary to be reserved for a currently routed request along a particular path in the network. This does not work well from the resource utilisation point of view. For example, the widest path chosen by the

shortest-widest algorithm can be a quite long one (in terms of hop count) and result in excessive amount of bandwidth reserved along it. There is yet another example of resource over-reservation. It is supposed that the amount of resources to be reserved along the prospective path (e.g. bandwidth) is to be known before running an algorithm. For example in this case the bandwidth can be calculated for some pre-determined number of hops [WJ00] (e.g. maximum possible number of hops the routing path may have), which leads eventually to over-reservation of bandwidth. Therefore, a routing algorithm has to be able to estimate the amount of resources for a particular path under consideration. A routing decision of such an algorithm would contain not only a path but also the optimal amount of resources to be reserved along it.

Thus, an algorithm that would address the specified above problems could improve the general resource utilisation of the network.

5.5.3 Dijkstra's Shortest-Path Algorithm

All QoS routing algorithms presented here can be directly solved by the modified variant of the Dijkstra's algorithm. Before we describe our modification of Dijkstra's algorithm, we give a detailed description of Dijkstra's algorithm in this section.

Dijkstra's algorithm solves the problem of finding the shortest paths from the source node s for a directed, nonnegative graph $G = (V, E)$. Before we describe the algorithm, let us list here some definitions of the shortest-paths problem.

In general, the shortest-paths problem is defined for a directed graph $G = (V, E)$, where V is a set of vertices v and E is a set of edges $e_i = (v_i, v_j)$, $v_i, v_j \in V$. An edge weight is a cost associated with the edge. The weight w of the path $p\langle v_0, v_1, \ldots, v_k \rangle$ is defined as a sum of the edges' weights comprising the path:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

[3] *Call blocking rate* is the number of rejected requests (due to the lack of resources) over the number of arrival requests

The weight of the shortest-path from u to v is defined as:

$$\delta(u,v)=\begin{cases}\min\{w(p): u \xrightarrow{p} v\} & \text{if there is a path from } u \text{ to } v,\\ \infty & \text{otherwise}\end{cases}$$

The shortest path from u to v is the path p for which $w(p)=\delta(u,v)$.

Let $S\subseteq V$ be a set of vertices v, for which $\delta(s,v)$ has been already found (i.e. $w(s,v)=\delta(s,v)$). The algorithm chooses a vertex $u\in V\setminus S$ (excluding the set S) with the smallest $w(s,u)$, adds u to the set S and makes *relaxation* of all edges coming from u (the term *relaxation* is explained later). Afterwards, this procedure is repeating. The vertices that are not from the set S, are stored in the queue Q. As it is necessary to find not only the shortest-path weight, but also the shortest path itself, the predecessor $\pi(v)$ is saved for each vertex $v\in V$. The predecessor $\pi(v)$ is the vertex preceding the vertex v along the path. For example, if we want to store the path $s\rightarrow u\rightarrow v$, then $\pi(v)=u$, $\pi(u)=s$ and $\pi(s)=NIL$. Then, at the end of the algorithm, the sequence of predecessors $\pi(v)$ starting from the vertex v will be the shortest path from s to v (in the reverse order).

The formal specification of the algorithm is given next.

```
        DIJKSTRA (G, w, s)
1       INITIALIZE-SINGLE-SOURCE (G,s) {
2       for all v∈V[G]
3           do w(s,v) ←∞
4               π(v)←NIL
5       w(s,s)←0
6       }
7       S←∅
8       Q←V[G]
9       while Q≠∅
10              do u←EXTRACT-MIN(Q)
11                  S←S∪{u}
12                  for all v∈Adj[u]
13                      do RELAX(u,v,w) {
14                      if w(s,v)>w(s,u)+w(u,v)
15                          then w(s,v) ←w(s,u)+w(u,v)
16                                  π(v)←u
17                      }
```

Figure 5-3: Pseudo-code for Dijkstra's algorithm

In lines *1-6*, the initialisation of w and π is performed. Lines *7* and *8* initialise the set S and the queue $Q=V\setminus S=V$. Then, by each iteration of lines *9-17*, a vertex u with the minimum value $w(s,u)$ is being chosen from the queue Q and added to the set S (in the first iteration we have $u=s$). In lines *12-16*, each edge (u,v) adjacent to u is processed by the procedure known as *relaxation*. During the relaxation the new values of weights for all vertices v (adjacent to u) are calculated and compared to the previous estimates; if any new value is less than the old one, then the new estimate replaces the old one (which can lead to changing of $\pi(v)$ as well). We can note that during the work of the algorithm, the new vertices are never being added to the queue Q and any vertex deleted from the queue Q is added to the set S only once. Hence, the number of iterations of the cycle **while** is $|V|$.

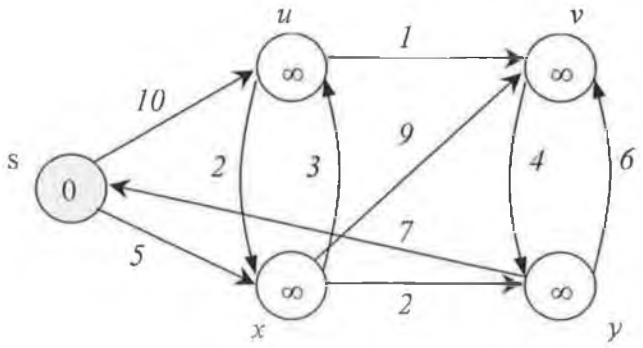
As an example, we show Dijkstra's algorithm step by step in Figure 5-4. The source node is the vertex s. In the cycles we use numbers to denote the total weight along the path from s to the current vertex. Bold black arrows denote the edges (p,q) for which $\pi(q)=p$. Black vertices are in the set S, all the rest vertices are in the queue $Q=V\setminus S$.

Step 1: before the first iteration of the cycle **while**. The grey vertex has the minimum value w and is chosen as vertex u in line *10*.

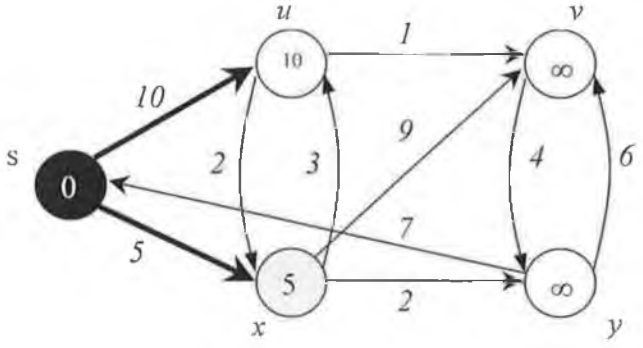
Step 2-6: the consecutive states after each iteration of the cycle **while**. A grey vertex is always chosen as a vertex u by the next iteration. The values w and π in Step 6 are final.

As Dijkstra's algorithm always chooses at each step the vertices with the minimum shortest-path estimate, it can be classified as a greedy algorithm. Even though greedy algorithms in general do not give the optimal solution, it can be shown that Dijkstra's algorithm provides right optimal decision [D59].

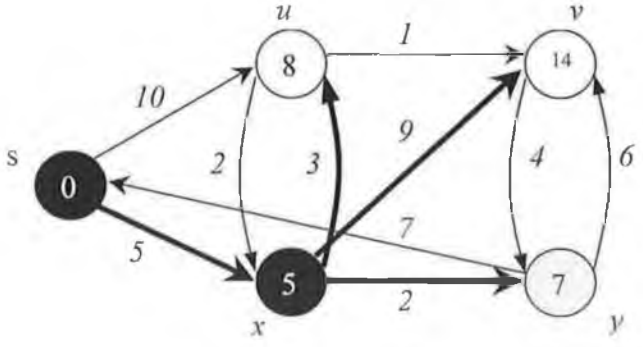
The Dijkstra's algorithm runs in time $O(n^2)$, where $n=|V|$. Some implementations of the Dijkstra's algorithm can run in much less time (such as R-heap implementation, Dial's implementation, Tarjan's implementation, etc [AM90]).



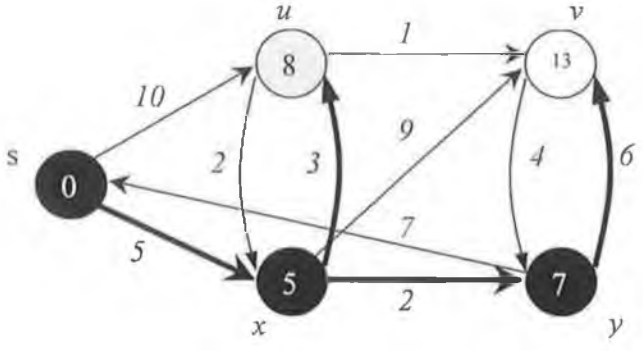
Step 1



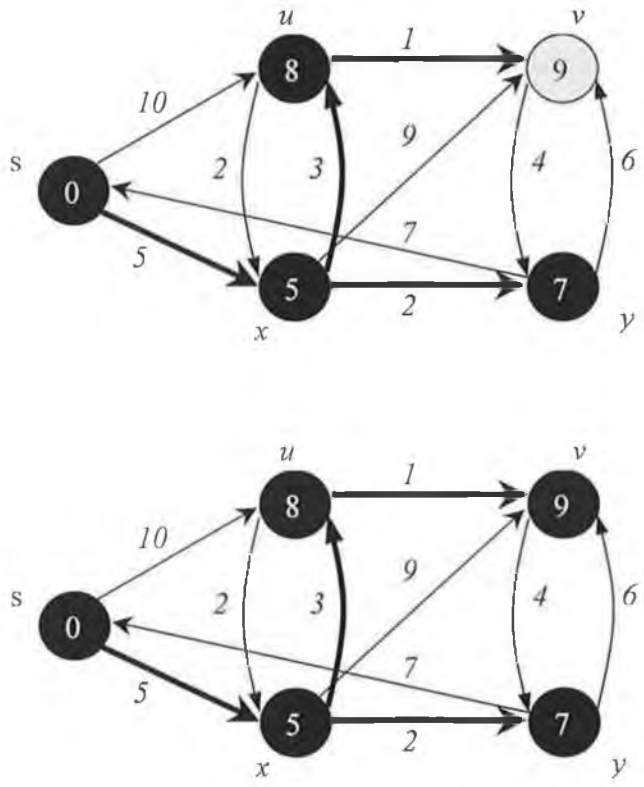
Step 2



Step 3



Step 4



Step 5

Step 6

Figure 5-4: Dijkstra's algorithm step by step

5.5.4 Modified Dijkstra's Algorithm

With respect to minimising utilisation of network resources, the objective of our algorithm is to take into account the amount of resources to be reserved along particular paths for currently routed requests. This can be achieved by calculating the network resources for each possible path under consideration at each step of the algorithm. With regard to balancing the load, the objective is to balance the aggregate effective bandwidths on the network. This can be achieved by choosing an appropriate cost function for the links and nodes and then by finding the route with the minimum cost and enough available resources in terms of effective bandwidth that would guarantee QoS requirements.

Here, an algorithm that determines such a route is proposed. The algorithm is the modification of the well known Dijkstra's shortest-path algorithm which has complexity $O(n^2)$. Each link of the network has its own cost (according to chosen cost function) and Dijkstra's algorithm is used to find the shortest path in terms of the costs that provide best

load balancing. The modification of Dijkstra's algorithm is related to the problem (explained in more detail later) that in our case the link costs are not fixed and they can change as the effective bandwidth is increasing over the number of hops along the path within the network.

Effective bandwidth and the number of hops

Here, we clarify how the effective bandwidth depends on the number of hops along the path.

If we want to transmit some amount of data from node N_1 to node N_2 (Figure 5-5-a), and then find how long it takes, we must calculate:

$$T = \frac{S}{B}$$

Where S is the size of data and B is available bandwidth to transmit these data.

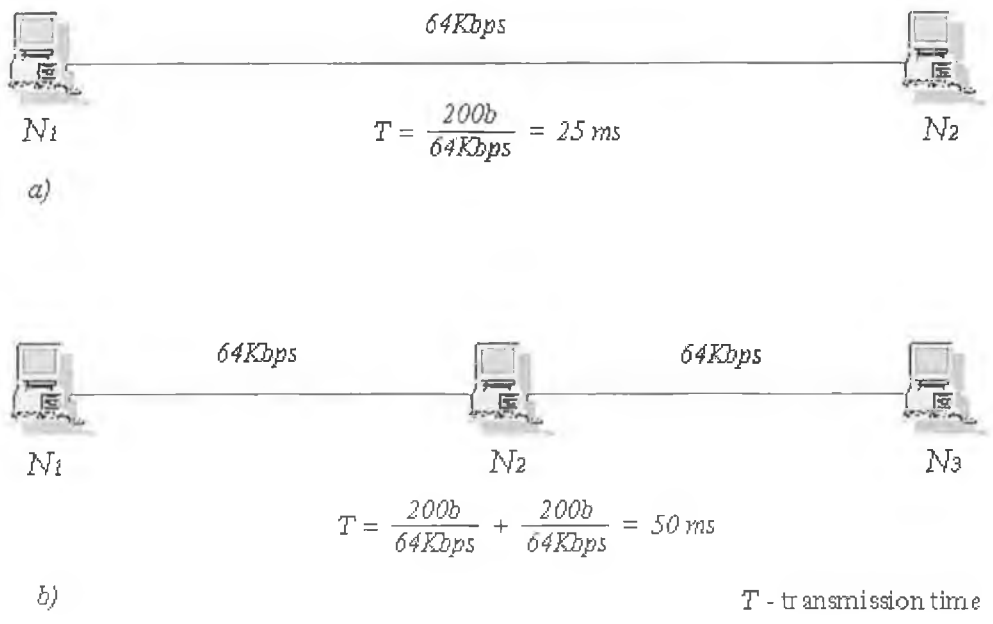


Figure 5-5: How bandwidth depends on the number of hops along the path

For example, if the available bandwidth is 64Kbps and we want to transmit a packet of 200 bytes, it will take: $200b / 64Kbps = 25\ ms$. Now, let us consider a case when we want to transmit the same packet from node N_1 to node N_3 (Figure 5-5-b). In this case, we have to

calculate the transmission time twice, as there are two links along the path. First, we calculate the transmission time between nodes N_1 and N_2, and then between N_2 and N_3. Let the available bandwidth of all lines be 64Kbps. Then, the transmission time between N_1 and N_3 will be: *200b / 64Kbps + 200b / 64Kbps = 50 ms*. As we see, the latency between end nodes increased. This means that if it is needed to meet the same bound of 25 ms for the transmission time (as it is in Figure 5-5-a), then the available bandwidth of all lines should be not less than 128Kbps. Indeed, in this case the transmission time between *N1* and *N3* would be: *200b / 128Kbps + 200b / 128Mbps = 25 ms*. So, it should be noted that the bandwidth necessary to provide the end-to-end delay depends on the number of hops along the path.

In general, the end-to-end transmission time is (Section 5.2.1):

$$T = \sum_{i=1}^{N} \frac{S}{B_i} = S \sum_{i=1}^{N} \frac{1}{B_i}$$

Where N is the number of nodes the data traverses along the path. If the same amount of bandwidth is available at each link along the path, the transmission time is:

$$T = N \frac{S}{B}$$

From this, it follows that when there is an end-to-end transmission delay requirement, the bandwidth necessary to be reserved at each link along the path to guarantee this delay requirement is:

$$B = N \frac{S}{T} \qquad (5\text{-}1)$$

Referring back to Figure 5-5-b, the bandwidth to be reserved at each link to meet the bound of 25 ms for a transmission delay would be:

$$B = N \frac{S}{T} = 2 \frac{200b}{25ms} = 128Kbps$$

Cost Functions

The cost of a link can be defined as a monetary price or as a function of some network parameters that are to be optimised. For our system, we define the cost of a link as a function of the bandwidth utilisation. The overall cost of a path is the sum of all individual links' costs on the path. The optimisation problem is to find the lowest-cost path. The solution of the problem can vary with different chosen cost functions. We do not discuss here how to choose a better cost function for a particular network scenario. Rather, we give an example of how different cost functions can be applied to our approach. In the following subsections, we consider two types of cost functions:

- linear cost function – $Cost = \frac{B}{C}$
- exponential cost function – $Cost = e^{\alpha(B-C)}$

where B is bandwidth allocated in the link; C is the total capacity of the link, and α is a parameter that can be varied [SM00].

How link costs depend on the number of hops

Here, we show how the cost of a link may change as the number of links along the path grows.

Let us consider two alternative paths connecting remote nodes *1* and *3* (Figure 5-6). The first path is the two-hop path *1-2-3* (Figure 5-6-a) and the second path is the three-hop path *1-2-4-3* (Figure 5-6-b). The effective bandwidth B_1 is the bandwidth that is to be reserved along the first path and the effective bandwidth B_2 is the bandwidth that is to be reserved along the second alternative path. As was discussed in the previous subsection, the effective bandwidth depends on the number of hops along the path. That is why, to provide the same QoS requirements between nodes *1* and *3*, we would have to reserve more bandwidth along the three-hop path than along the two-hop path ($B_2 > B_1$). Let C_1 be the capacity of the link *1-2*. Then in the case of the two-hop path, the cost of the link would be B_1/C_1 and, in the case of the three-hop path, the cost of the same link would be B_2/C_1 (we

suppose that the cost function is linear). As $B_2 > B_1$ then the cost of the link *1-2* along the two-hop path is less than along the three-hop path.

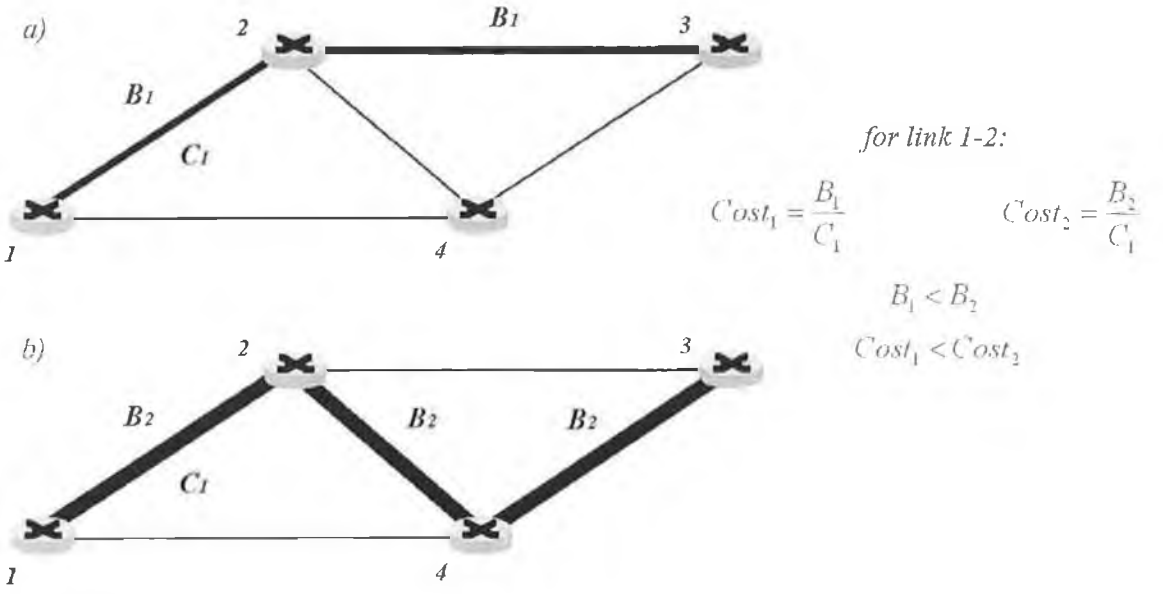


Figure 5-6: How link cost depends on the number of hops along the path

From this follows that we cannot estimate a cost of a particular link unless we know what path this link is a part of. Once we know the path, theprocedure for calculating the costs of the links along the path will be the following:

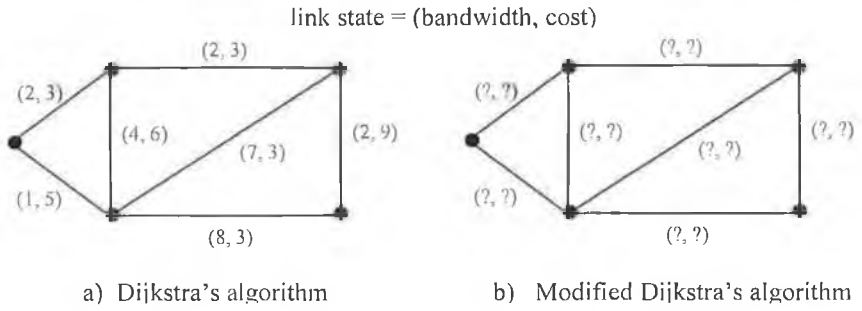
1. to calculate the effective bandwidth necessary to provide QoS requirements along this path. The calculation of the effective bandwidth may take into account some parameters of this particular path (e.g. the number of hops along the path, propagation delays), which is explained later.

2. given the amount of bandwidth to be reserved at the links and the values of the capacity at each link, we can calculate the costs of all links along the path.

Why to modify Dijkstra's algorithm?

The modification of Dijkstra's algorithm is related to the following fact. All links in the network have their estimated costs that are fixed for Dijkstra's algorithm. An input for the algorithm is a graph with predefined edge costs (Figure 5-7-a). As Dijkstra's algorithm runs looking for the shortest-path, the values of costs do not change. In our case, it is

different (Figure 5-7-b); the link costs are not known in the beginning, as they depend on the amount of bandwidth to be reserved at links for currently routed requests. Also, the amount of bandwidth is not known in the beginning, as it depends on the number of hops along the finally chosen path. However, it is not known in advance how many hops will be along the prospective path. One of the possible solutions is to calculate effective bandwidth for some maximum possible number of hops along the path, but this approach may lead to overestimation of effective bandwidth. That is why the link costs have to be estimated at each step of the algorithm for a particular path under consideration, which requires some changes in Dijkstra's algorithm.



a) Dijkstra's algorithm b) Modified Dijkstra's algorithm

Figure 5-7: The initial state of the graphs for the algorithms

If we look at the formal specification of Dijkstra's algorithm (Figure 5-3), we can see that the only procedure we have to change is *EXTRACT-MIN(Q)* (line 10). This function performs the search of the vertex with the minimal cost. As in our case, link costs are not fixed, we have to estimate new link costs every time before we run *EXTRACT-MIN(Q)*. How new link costs can be calculated at each step of the algorithm is explained in the subsequent subsections.

Calculation of costs

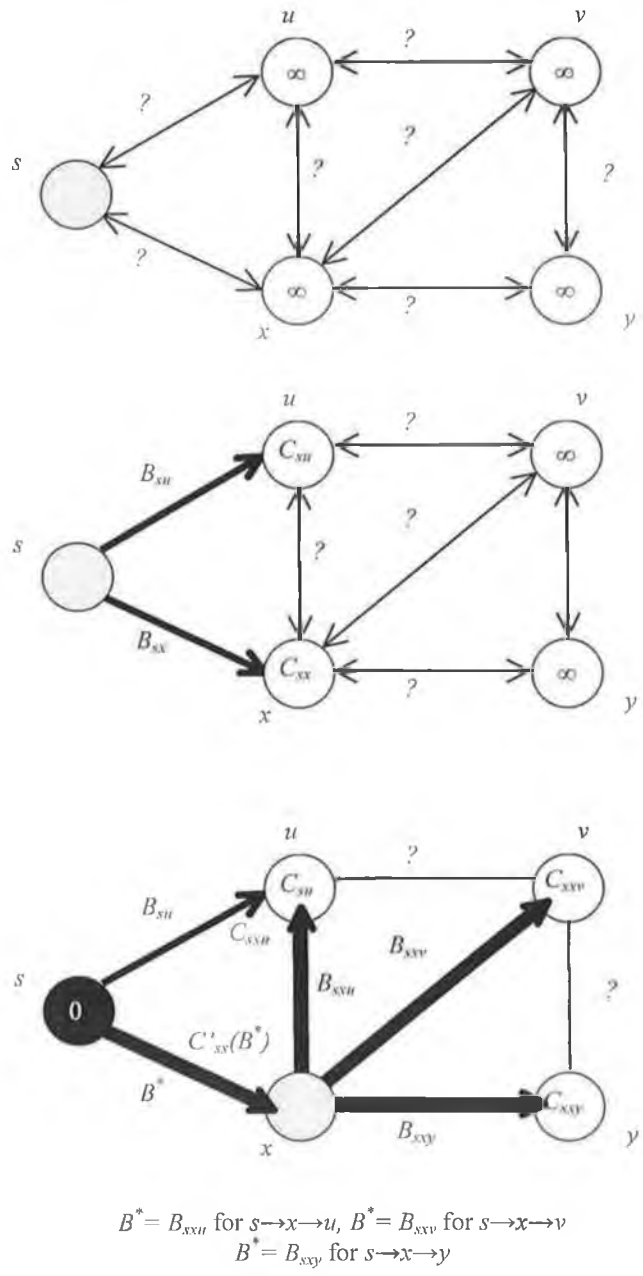
Calculation of link costs at each step of the algorithm is our modification of Dijkstra's algorithm. To clarify the necessity of doing this, let us consider a few steps of the algorithm in Figure 5-8, where node *s* is a source.

Step 1: Unlike in the case of Dijkstra's algorithm, the link costs are not known before the first iteration in our case.

Step 2: According to Dijkstra's algorithm, we have to choose one of the adjacent nodes to s with the smallest cost. As link costs are not known, we have to calculate them. In doing so, we calculate first bandwidths B_{su} and B_{sx} for the adjacent links s-u and s-x (as if u and x were end-nodes), and then given the bandwidth and capacity of these links, we calculate their costs C_{su} and C_{sx}. Now we let Dijkstra's algorithm find a node with the smallest cost. Let us assume that $C_{sx} < C_{su}$ and we move onto the node x for the next step.

Step 3: At this step, Dijkstra's algorithm estimates costs for the nodes adjacent to node x and then chooses the node with the smallest cost. First, we have to again perform the calculation of link costs. This time we consider three two-hop paths: s-x-u, s-x-v and s-x-y. For each path, we calculate the necessary bandwidths B_{sxu}, B_{sxv}, B_{sxy} and then given the residual capacity of the links, we calculate the costs C_{sxu}, C_{sxv} and C_{sxy}. Now, when the costs are known, we can let Dijkstra's algorithm run further.

We do not consider the next steps of the algorithm here, as we wanted only to show when the problem of calculation of link costs arises.



Step 1

Costs and bandwidth are unknown

Step 2

Considering paths:
$s \rightarrow u$, $s \rightarrow x$
1) Calculate bandwidth:
B_{su}, B_{sx}
2) Calculate costs:
$C_{su}=f(B_{su})$, $C_{sx}=f(B_{sx})$
let $C_{sx} < C_{su}$

Step 3

Considering paths:
$s \rightarrow x \rightarrow u$, $s \rightarrow x \rightarrow v$, $s \rightarrow x \rightarrow y$
1) Calculate bandwidth:
B_{sxu}, B_{sxv}, B_{sxy}
2) Calculate link costs:
$C'_{sx}=f(B^*)$,
$C_{xu}=f(B_{sxu})$, $C_{xv}=f(B_{sxv})$,
$C_{xy}=f(B_{sxy})$
3) Calculate total costs:
$C_{sxu} = C'_{sx}(B_{sxu}) + C_{xu}$
$C_{sxv} = C'_{sx}(B_{sxv}) + C_{xv}$
$C_{sxy} = C'_{sx}(B_{sxy}) + C_{xy}$

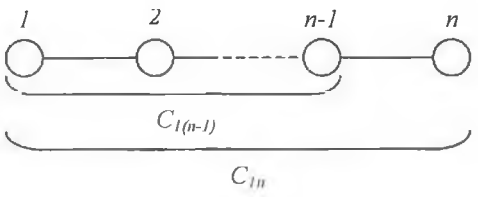
$B^* = B_{sxu}$ for $s \rightarrow x \rightarrow u$, $B^* = B_{sxv}$ for $s \rightarrow x \rightarrow v$
$B^* = B_{sxy}$ for $s \rightarrow x \rightarrow y$

Figure 5-8: Why calculate costs at each step

How to perform the calculation of costs?

From the example of the previous subsection it is seen that calculation of link costs at each step of the algorithm can be a very laborious process. In the current subsection, we show how this process can be simplified.

Let us assume that we are at some step of Dijkstra's algorithm considering adjacent links of the node *(n-1)* (Figure 5-9). The cost of the path $1 \rightarrow (n-1)$ is $C_{1(n-1)}$ and the task is to determine the cost of the path $1 \rightarrow n$.



Dijkstra's algorithm: $C_{1n} = C_{1(n-1)} + C_{(n-1)n}$

New algorithm: $C_{1n} = f(C_{1(n-1)}, C_{(n-1)n})$?

C_{ij} - cost of the path *i-j*

Figure 5-9: Calculation of costs

For Dijkstra's algorithm it is very simple:

$$C_{1n} = C_{1(n-1)} + C_{(n-1)n}$$

In our case, it is more complicated. The cost $C_{1(n-1)}$ within the path $1 \rightarrow n$ is now not the same as it was within the path $1 \rightarrow (n-1)$. The new cost $C_{1(n-1)}$ is increasing its value because the bandwidth to be reserved along the path $1 \rightarrow n$ will be larger than it was for the path $1 \rightarrow (n-1)$ (to meet the same QoS requirements). Therefore, the new values of costs should be calculated for all links along the path $1 \rightarrow (n-1)$.

The total cost C_{1n} depends on the amount of effective bandwidth necessary to allocate for the path $1 \rightarrow n$ and it depends as well on the individual links' characteristics along the whole path $1 \rightarrow n$ (e.g. propagation delays and link residual capacities). It would be very laborious to keep all of these links' characteristics in memory and each time to calculate the costs for all links along the new path from the very first node:

$$C_{1n} = C_{12} + C_{23} + ... + C_{(n-1)n} = \sum_{i=1}^{n-1} C_{i(i+1)}$$

where C_{ij} is the cost of the link $i \rightarrow j$ within the path $1 \rightarrow n$.

Instead, the objective is to find such function so that:

$$C_{1n} = f(C_{1(n-1)}, C_{(n-1)n})$$

This function depends on the cost functions used in the network. As an example, we show how this function can be derived for two different types of cost functions: linear cost functions and exponential cost functions.

Linear cost function

In this case, cost of a link is calculated as:

$$C = \frac{B}{L}$$

Where L is link's residual capacity and B is the bandwidth requested for a new connection. The cost of a link is calculated only if $L \geq B$, otherwise the link is not considered for the prospective path because of the lack of bandwidth.

The cost of a link will depend not only on the link's available capacity but also on the amount of bandwidth necessary to be reserved for a current request. If B_{1n} is bandwidth to be reserved along the path $1 \rightarrow n$ (Figure 5-9), then the overall cost for the whole path is a sum of all individual links' costs on the path:

$$C_{1n} = \sum_{i=1}^{n-1} \frac{B_{1n}}{L_{i(i+1)}} \qquad (5\text{-}2)$$

Where $L_{i(i+1)}$ is capacity of the link $i \rightarrow (i+1)$.

As bandwidth B_{1n} is the same for all links, then:

$$C_{1n} = \sum_{i=1}^{n-1} \frac{B_{1n}}{L_{i(i+1)}} = B_{1n} \sum_{i=1}^{n-1} \frac{1}{L_{i(i+1)}} = B_{1n} \left(\sum_{i=1}^{n-2} \frac{1}{L_{i(i+1)}} + \frac{1}{L_{(n-1)n}} \right) = B_{1n} \sum_{i=1}^{n-2} \frac{1}{L_{i(i+1)}} + \frac{B_{1n}}{L_{(n-1)n}} =$$
$$= B_{1n} \frac{B_{1(n-1)}}{B_{1(n-1)}} \sum_{i=1}^{n-2} \frac{1}{L_{i(i+1)}} + \frac{B_{1n}}{L_{(n-1)n}} = \frac{B_{1n}}{B_{1(n-1)}} \sum_{i=1}^{n-2} \frac{B_{1(n-1)}}{L_{i(i+1)}} + \frac{B_{1n}}{L_{(n-1)n}} \qquad (5\text{-}3)$$

Where $B_{1(n-1)}$ is the bandwidth to be reserved for the path $1 \rightarrow (n-1)$.

It can be seen here that:

$\sum_{i=1}^{n-2} \frac{B_{1(n-1)}}{L_{i(i+1)}} = C_{1(n-1)}$ - is the total cost for the path $1 \rightarrow (n-1)$

$\frac{B_{1n}}{L_{(n-1)n}} = C_{(n-1)n}$ - is the cost of the link $(n-1) \rightarrow n$ (when it is used for the path $1 \rightarrow n$ with the bandwidth B_{1n})

Therefore, the equation *(5-3)* for the overall cost C_{1n} can be rewritten as:

$$C_{1n} = C_{(n-1)n} + \frac{B_{1n}}{B_{1(n-1)}} C_{1(n-1)} \qquad (5\text{-}4)$$

The values of $C_{1(n-1)}$ and $B_{1(n-1)}$ are always known at the *n-th* step of the algorithm, as they are to be calculated at the previous step. Hence, at the *n-th* step, only values of B_{1n} and $C_{(n-1)n}$ have to be calculated before calculating the total cost C_{1n}. This makes the equation *(5-4)* much easier to use than the equation *(5-2)*.

Exponential cost function

In this case, the cost of a link is calculated as:

$$C = e^{\alpha(B-L)}$$

Where L is the link's residual capacity, B is its available bandwidth and α is a parameter that can be varied.

The total cost C_{1n} for the path $1 \rightarrow n$ (Figure 5-9):

$$C_{1n} = \sum_{i=1}^{n-1} e^{\alpha(B_{i(i+1)} - L_{i(i+1)})} \qquad (5\text{-}5)$$

This equation can be rewritten as:

$$C_{1n} = \sum_{i=1}^{n-1} e^{\alpha(B_{1n} - L_{i(i+1)})} = \sum_{i=1}^{n-1} \frac{e^{\alpha B_{1n}}}{e^{\alpha L_{i(i+1)}}} = e^{\alpha B_{1n}} \sum_{i=1}^{n-1} \frac{1}{e^{\alpha L_{i(i+1)}}} = e^{\alpha B_{1n}} \left(\sum_{i=1}^{n-2} \frac{1}{e^{\alpha L_{i(i+1)}}} + \frac{1}{e^{\alpha L_{(n-1)n}}} \right) =$$

$$= e^{\alpha B_{1n}} \sum_{i=1}^{n-2} \frac{1}{e^{\alpha L_{i(i+1)}}} + \frac{e^{\alpha B_{1n}}}{e^{\alpha L_{(n-1)n}}} = e^{\alpha B_{1n}} \frac{e^{\alpha B_{1(n-1)}}}{e^{\alpha B_{1(n-1)}}} \sum_{i=1}^{n-2} \frac{1}{e^{\alpha L_{i(i+1)}}} + \frac{e^{\alpha B_{1n}}}{e^{\alpha L_{(n-1)n}}} = \qquad (5\text{-}6)$$

$$= \frac{e^{\alpha B_{1n}}}{e^{\alpha B_{1(n-1)}}} \sum_{i=1}^{n-2} \frac{e^{\alpha B_{1(n-1)}}}{e^{\alpha L_{i(i+1)}}} + \frac{e^{\alpha B_{1n}}}{e^{\alpha L_{(n-1)n}}} = e^{\alpha(B_{1n} - B_{1(n-1)})} \sum_{i=1}^{n-2} e^{\alpha(B_{1(n-1)} - L_{i(i+1)})} + e^{\alpha(B_{1n} - L_{(n-1)n})}$$

It can be seen here that:

$\sum_{i=1}^{n-2} e^{\alpha(B_{1(n-1)} - L_{i(i+1)})} = C_{1(n-1)}$ - is the total cost for the path $1 \rightarrow (n\text{-}1)$

$e^{\alpha(B_{1n} - L_{(n-1)n})} = C_{(n-1)n}$ - is the cost of the link $(n\text{-}1) \rightarrow n$ (when it is used for the path $1 \rightarrow n$ with the bandwidth B_{1n})

Hence the equation *(5-6)* for the overall cost C_{1n} can be written as:

$$C_{1n} = C_{(n-1)n} + e^{\alpha(B_{1n} - B_{1(n-1)})} C_{1(n-1)} \qquad (5\text{-}7)$$

As was said before, the values of $C_{1(n-1)}$ and $B_{1(n-1)}$ are always known at the *n-th* step of the algorithm, as they are to be calculated at the previous step. Therefore, at the *n-th* step, only values of B_{1n} and $C_{(n-1)n}$ have to be calculated before calculating the total cost C_{1n}. This makes the equation *(5-7)* much easier to use than the equation *(5-5)*.

Hence, in spite of the non-static nature of links' costs in our case, Dijkstra's algorithm can be used with the costs at each step to be calculated according to the equations *(5-4)* or *(5-7)*. In our simulations described in the next chapter we will consider the linear cost function and use the equation *(5-4)*.

How to calculate effective bandwidth?

In this thesis, we do not propose a new approach to calculating the effective bandwidth. Rather, we discuss how the notion of the effective bandwidth can be applied to our routing algorithm.

In general, the effective bandwidth reflects both a particular kind of traffic and specific network configuration. It is supposed that the effective bandwidth has to be provided "end-to-end". Because of specific network parameters along the prospective path (e.g. propagation delays, queuing delays), actual bandwidth reserved at the links along the path can be much greater than the bandwidth necessary to satisfy a transmission delay requirement only. That is why a method to be used in our system for calculating effective bandwidth should take into account end-to-end delay estimation.

If we have a path in the network with n nodes and links then the end-to-end delay d in transmitting a packet is given by (Section 5.2.1):

$$d = \sum_{i=1}^{n} \left(t_{li} + t_{qi} + t_{pi} \right)$$

where t_p is the link propagation delay (includes the propagation time of the link), t_l is the transmission delay (captures link capacity), and t_q is the queuing delay.

Transmission and queuing delays are related to amount of resources reserved into network and propagation delays are related to individual links' parameters. As transmission and queuing delays depend on the available bandwidth, we suppose that they can be guaranteed by reservation of enough bandwidth in the network. Such bandwidth reserved between two remote nodes could guarantee that the total transmission and queuing time $\sum_{i=1}^{n} \left(t_{li} + t_{qi} \right)$ along the path never exceeds some required value t_0. This time t_0 plus the total propagation time along the path should not be longer than the end-to-end delay requirement. Indeed, if T is an end-to-end delay (latency) requirement, then:

$$d = \sum_{i=1}^{n} \left(t_{li} + t_{qi} + t_{pi} \right) \leq T$$

$$\Rightarrow \sum_{i=1}^{n} \left(t_{li} + t_{qi} \right) \leq T - \sum_{i=1}^{n} t_{pi}$$

From the last equation it follows that first we have to find the total propagation delay $\sum_{i=1}^{n} t_{pi}$ along the path and, then, we need to calculate a requirement for transmission and queuing delays: $t_0 = T - \sum_{i=1}^{n} t_{pi}$, from which we can then determine the bandwidth necessary to be reserved to meet the requested end-to-end delay requirement:

$$B_{eff} = f(t_0)$$

Here, the function *f(t0)* reflects the method chosen for calculating the effective bandwidth given the end-to-end requirements *t0* for transmission and queuing delays. For example, in

$$B_{eff} = \frac{N_{hops} \times S_{packet}}{t_0}$$

the simple case when we do not consider queuing delays, the function $f(t_0)$ for the effective bandwidth which guarantees only end-to-end transmission delay for a single source would take the form:

Where N_{hops} is the number of hops along the path and S_{packet} is the size of a packet.

In general, when there are queuing, loss and jitter end-to-end requirements and a number of sources in the aggregate, the calculation of the effective bandwidth is much more complex. For our simulation, we consider a simplified method for calculating the effective bandwidth (which is explained later in Chapter 6), but any other method can also be applied.

Finally, having determined the effective bandwidth and given the links' residual capacities, we can evaluate the cost of the path using equations (5-4) and (5-7).

Algorithm's specification

We now give the brief description of our algorithm and then we give the formal specification of it.

At each step of Dijkstra's algorithm we should:

- compute the propagation delay for the path from the source node to the current node;
- subtract the propagation delay from the latency requirements;
- for this time compute the effective bandwidth for the path from the source node to the current node;
- check if there is enough available bandwidth along the path from the source node to the current node;
- compute the cost for the path from the source node to the current node.

For the formal specification we use:

s - source node

n_i - the number of hops along the path from the source node s to the node i

e_{ij} - the link between nodes i and j

B'_{min} - minimum available bandwidth along the path from the source node s to the node i

B_{ij} - residual bandwidth for the link e_{ij}

C_i - the overall cost for the path from the source node s to the node i (c_i - similar temporary array)

b_i - effective bandwidth for the path from the source node s to the node i

T'_{prop} - link propagation delay along the path from the source node s to the node i

t^{ij}_{prop} - link propagation delay for the link e_{ij}.

P_i - array which contains the shortest path from the source node s to the node i (actually, it contains the list of the nodes along that path).

The pseudo-code for the algorithm is given next.

Initialization

$i=s$;

$n_s=0$;

$B^s_{min}=\infty$;

$C_s=0$; $b_s=\infty$;

$T^s_{prop}=0$;

$P_j \leftarrow 0$; (for all j)

$P_s \leftarrow s$;

$i=s$;

Main Loop

for each j when $e_{ij}\neq 0$ **do**

if $B_{ij} < B^i_{min}$ **then** $B^j_{min}=B_{ij}$

$T^j_{prop} = T^i_{prop} + t^0_{prop}$;

$n_j = n_i + 1$;

$b_j = f(n_j, T^j_{prop})$; // find effective bandwidth

if $b_j > B^j_{min}$ **then** skip this link

else $c_j = b_j / B_{ij} + C_i b_j / b_i$ //compute the cost when the cost function is linear

or $c_j = exp[\alpha(b_j - B_{ij})] + C_i exp[\alpha(b_j - b_j)]$ //when the cost function is exponential

if $c_j < C_j$ **then** $C_j = c_j$; relaxation

for each j: $e_{ij} = 0$;

for all j when $P_j=0$ find j_{min} so that $C_{jmin} = min(C_j)$ $\forall j$;

$P_{jmin} \leftarrow P_i + j_{min}$;

$i=j_{min}$;

while for at least one j: $P_j=0$;

Figure 5-10: Pseudo-code for the modified Dijkstra's algorithm

6 Simulation Model

6.1 Introduction

The objective of the simulation model is to validate the approach of the presented traffic engineering system. The simulation model is designed for a network scenario running a Voice over IP (VoIP) service across a DiffServ/MPLS network. On the example of voice traffic, it is demonstrated how the routing can be performed in the network by applying the modified Dijkstra's algorithm described in the previous chapter. For estimation of the performance of our algorithm, the simulations are carried out for the system based on the developed routing algorithm and for the system running shortest path algorithms. Comparison results are presented and discussed in the last section of this chapter.

6.2 Network simulation scenario

As an example of the operation of our traffic engineering system, we consider a Voice over IP (VoIP) service across a DiffServ/MPLS network. The implementation of our system can satisfy the strict service requirements of voice traffic by providing strong QoS guarantees while providing the good use of network resources.

The implementation of voice trunks, across an MPLS network, with strong QoS guarantees for bandwidth, delay, jitter and packet loss is one of the applications that is of main interest for today's network providers.

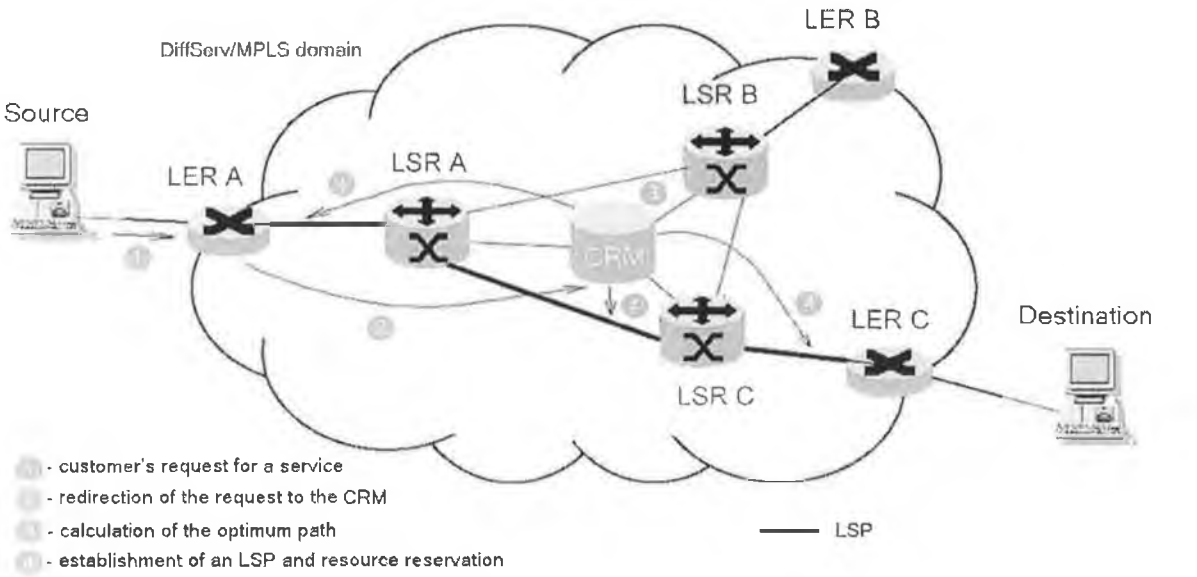


Figure 6-1: Network simulation scenario

In our simulation model, we consider a network depicted in Figure 6-1. We suppose that an ISP has a single AS with DiffServ/MPLS technologies deployed to provide VoIP services to its customers. We assume that every router within the network can potentially be an ingress and egress point. Therefore, every router can be a source of a request for an SLA to support voice traffic between this router and any other one in the network. As an example, let us consider router *LER A* as a source point of a request and router *LER C* as a destination point (Figure 6-1). For this request, all the other routers are considered as core routers (e.g. LSR A, LSR B, LSR C). In general, customers may request an ISP to support any number of IP phone conversations with particular QoS requirements for delay, jitter or packet loss. In our simulation model, we take into account QoS requirements for end-to-end delay only and for simplicity, we assume that there are no packet losses in the network. So, router *LER A* may request the ISP to support *N* IP phone conversations between *LER A* and *LER C* with an end-to-end delay requirement of not longer than *T* ms. The request goes to the CRM that is responsible for finding a path between *LER A* and *LER C* with enough amount of bandwidth to meet the end-to-end delay requirement. For doing this, the CRM runs our developed routing algorithm. The CRM has all knowledge about the current network-state information, as we assume that the ISP has all the means, as defined in Chapter 4, for maintaining a database containing a topological map of the network domain and information about the current state of the network resources. After finding a path, the CRM is involved in setting up an LSP along an explicit route between *LER A* and *LER C*,

and reserving the necessary bandwidth, as it is described in Chapter 4. After establishment of the LSP and resource reservation, the CRM is ready to process new incoming requests from other routers and the network is ready to transfer customers' data between *LER A* and *LER C*.

All reservations are to be set up in the network for a particular duration of time. After the time for a certain reservation is over, the CRM frees its resources with the help of a signalling protocol (e.g. RSVP, LDP). If the path between the requested source and destination points has not been found due to the lack of resources, customer's request is rejected.

We carried out simulations for the described network scenario with different routing algorithms: shortest-path algorithms and our developed routing algorithm. During the simulations, we performed observations of the way resources were used in the network. A comparative analysis of the performance of the routing algorithms is presented in Section 6.6.

6.3 Simulation model

We performed network simulations of the proposed management system. In our simulations, the CRM receives the requests for dynamic SLAs to be installed in the network (Figure 6-2). The requests are randomly generated by a Poisson process. Each request for an SLA contains a source node, destination node and latency constraint. The CRM is responsible for finding the optimal path between the source and destination, determining the amount of resources necessary to meet the latency requirement (converted into effective bandwidth) and reserving the required resources in the network. Each request is an SLA aggregate of multiple voice sources. As the CRM satisfies the requests for SLAs by running our modified Dijkstra's algorithm, we try to analyse how well the load is balanced on the network.

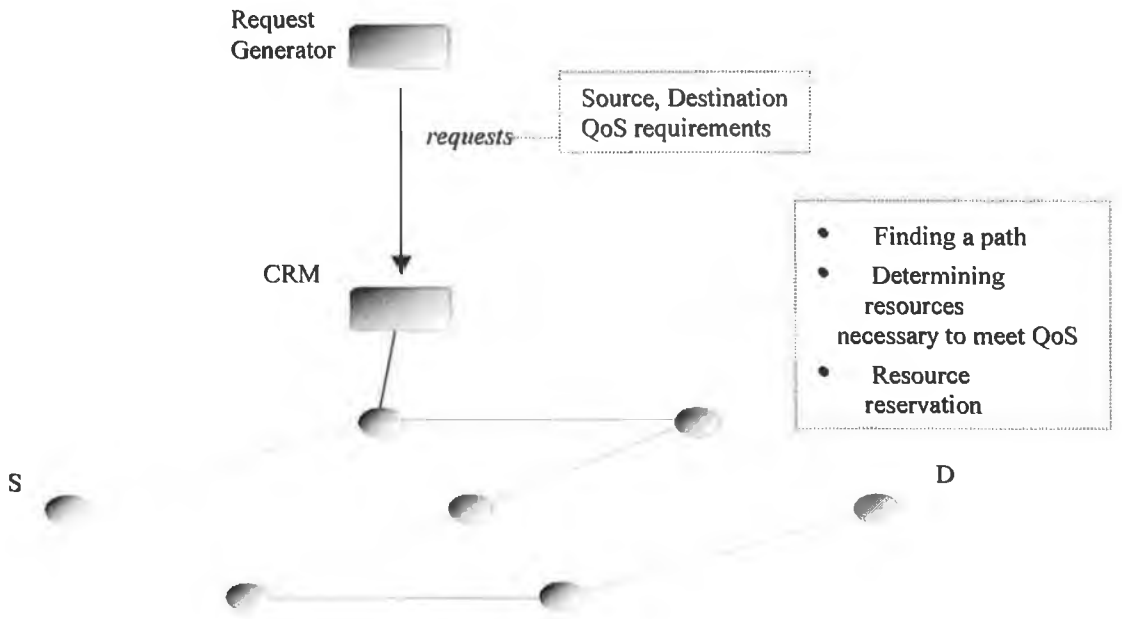


Figure 6-2: Simulation model

In order to estimate the performance of the traffic engineering system based on our routing algorithm, we also performed simulations for the CRM running some of the well known and broadly used routing algorithms. We estimate the performance of our system by comparing the resource utilisation results of the system based on our algorithm and systems based on one of the commonly cited QoS routing algorithms.

The main components of the simulation model are:

- Traffic model. The traffic model specifies some parameters of voice traffic that we have to know for calculating effective bandwidth for a request.

- Request generation model. The request generation model is implemented for generating customers' requests for services. The model is for modelling the arrival process of the requests and for generating random contents of the requests (e.g. source, destination, latency value).

- Modified Dijkstra's algorithm implementation. This is an implementation of our developed routing algorithm.

- Shortest-path algorithm implementations. This includes implementation of four different QoS routing algorithms:
  - shortest-distance algorithm implementation
  - shortest-delay algorithm implementation
  - shortest-hop algorithm implementation
  - widest-shortest algorithm implementation

  An implementation of each of these algorithms is either a direct implementation of Dijkstra's algorithm or its slight modification.

These main components of our simulation model are considered in more detail in the subsequent sections.

6.3.1 Traffic Model

6.3.1.1 ON/OFF traffic model for voice source

We need to analyse the behaviour of the traffic within the network; namely, we need to calculate the transmission delays into the links and effective bandwidth of the voice source. Therefore it is necessary to describe an analytic model of the voice traffic that we going to model into our network.

Voice traffic is modelled as alternative bursts and silences of variable length. In Figure 6-3 you can see sample traffic for a single voice source.

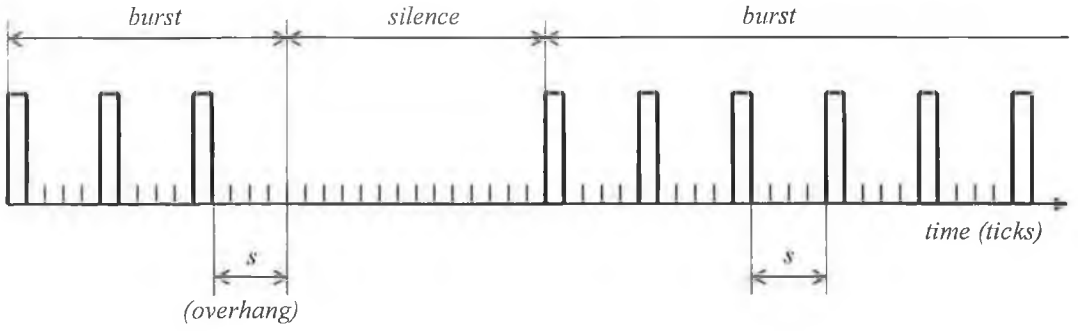


Figure 6-3: Modelling voice traffic as on/off source

- **Burst.** A burst is packetised into a series of fixed length packets and continues until a silence longer than the overhang time.
- **Overhang.** The overhang is a deterministic period of time after the burst has ended; it is a waiting period to see if another cell of information arrives or if silence has begun.
- **Silence.** This is a period of time during which there is no speech activity; it represents the time where a caller is listening and not talking; it continues until the next burst starts.

A probabilistic approach is used to describe the behaviour of voice traffic. When modelling voice traffic, the usually used parameters are: probability distribution of the times between cells (it can never be less than s ticks), probability that a line source is active or the probability of having a particular number of cells during burst periods or ticks during silence periods. Generally, the voice traffic is considered as the Markov process and a lot of interesting conclusions are derived from this model, which are useful for developing an analytical model for transmission, queuing delays or buffer lengths [NA91, HL86]. In [TC94], the author shows that some of the stochastic parameters of voice traffic can be easily derived also from the on/off fluid model. As the Markov model of voice traffic is more complex, we consider the on/off fluid model in this work (Figure 6-4).

Since there are s *ticks*[4] between cells, a burst can be described as a grouping of s ticks, where the last s ticks are the overhang. And we can represent voice traffic as a fluid source

[4] *"tick"* is one unit of time, the real value of which in practice depends on the encoding scheme used for processing voice data

which has two states - "on" and "off" corresponding to activity of speech and to silence, which are characterised by the transition rate from "off" state to "on" state and the transition rate from "on" state to "off" state. In figure below you can see voice traffic represented as a fluid source. For instance, when calculating the effective bandwidth we are going to use this form of representation for voice traffic.

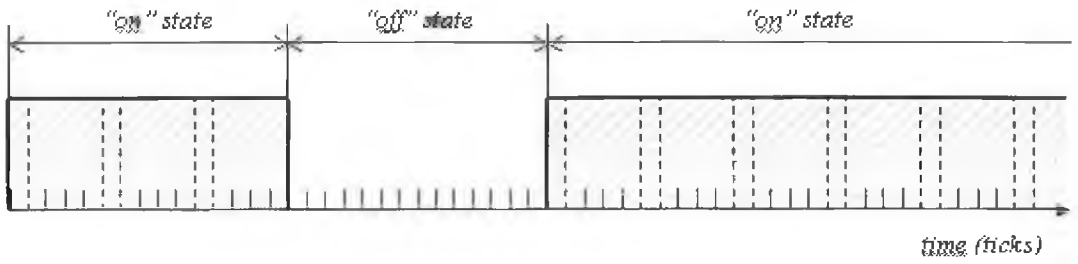


Figure 6-4: Voice traffic as on/off fluid model

We will need the activity rate of the source, in other words, the probability that a line source is active. Let T_B and T_S be the mean burst and mean silence lengths respectively. Then, the activity rate P_A can be defined as:

$$P_A = \frac{T_B}{T_B + T_S} \qquad (6\text{-}1)$$

6.3.1.2 Implementation issues for voice traffic

Here, some parameters for voice traffic used in the simulation model are considered.

Voice data rate

For our simulations, we suppose that each VoIP connection has peak rate of 64 Kbps. Let us explain where this number comes from.

According to the procedure made decades ago by the original engineers of telephony, the digitisation process of voice consists of four steps:

- Sampling

- Quantization
- Encoding
- Compression

First, speech is sampled at 8000 samples/sec. It means that the every second of speech is divided to 8000 segments for further processing. Then, each segment (or sample) is scaled to the number between 0 and 255 (depending on the analog value of the sample). The 256 levels of quantization provide sufficient resolution to encode the samples. This requires each sample be quantized at 8 bits/sample. With the sampling rate 8000 samples/sec it makes the bit rate of speech as much as 64000 bits/sec:

$$8000\ samples/sec * 8\ bits/sample = 64000\ bits/sec$$

In other words, it takes 64Kb to carry one second of uncompressed sound or speech. There are several generally used methods of compressing voice that allow bit rates to be reduced to as little as 8 Kbps. In our simulation, we use one of the most common encoding schemes - G.711 [G711]. This standard does not compress voice data and operates at 64 Kbps.

Latency

Here, latency is end-to-end delay of voice packet delivered between two parties. Voice traffic is real-time traffic. If latency is too long, interactive communication can be difficult. The lower the latency, the more natural and interactive the conversation becomes. That is why providing low latency is a crucial task for VoIP implementation.

Studies suggest that delays less than 200 ms would be acceptable for users. The 1996 ITU Recommendation G.114 [G714] for one-way end-to-end delay is:

- under 150 ms: acceptable for most user applications;
- 150 to 400 ms: acceptable provided that administrators are aware of the transmission time impact on the transmission quality of user applications;
- over 400 ms: unacceptable for general network planning purposes.

For example, some QoS applications for VoIP define the following services for latency [CA02]:

- 100 ms – GOLD service
- 150 ms – SILVER service
- 200 ms – BRONZE service

For our simulations we suppose that users can request latency from 100 to 150 ms.

Packet size

IP packets carrying the voice frames cannot be too large, otherwise it would take too much time to create and then to transmit each packet across the wire. For example, for a 500-byte packet it would take 62.5 ms to transmit the packet over a 64 Kbps link. For the desired latency of no more than 100 ms, it would take 62.5 percent of the whole delay budget!

Each voice packet comprises an uncompressed layer 2 and layer 3 headers (typical numbers 14 and 40 bytes respectively) and a voice payload (differs in size depending on the compression method). Layer 3 header consists of 20 bytes for the IP header, 8 bytes for the UDP header, and 12 bytes for the Real-Time Transport Protocol (RTP) header. As was said, a voice payload depends on the encoding scheme (or compression method). For example, G.711 standard (operates at 64 Kbps) does not compress voice data and implies a voice payload of 160 bytes per packet, whereas G.729 compression method (operates at 8 Kbps) has a voice payload of 20 bytes. As we chose the G.711 standard for our simulations, a voice payload of 160 bytes will be used henceforward.

The total bandwidth occupied by a single VoIP call

The total bandwidth used for a single VoIP connection depends on the compression type, and the total packet size. The equation to calculate the bandwidth is:

$$B = \left(S_{v} + S_{l2} + S_{l3}\right) \times \frac{R_{v}}{S_{v}}$$

where:

B – the total bandwidth needed for a single VoIP connection;

S_v – voice payload (the size of voice data per packet);

S_{l2} – layer 2 header size;

S_{l3} – layer 3 header size;

R_v – voice data rate;

The G.711 encoding scheme chosen for our simulations has the following characteristics:

- voice payload – 160 bytes per packet, S_v = 160 bytes
- voice data rate – 64 Kbps, R_v = 64000 bps
- layer 3 header size – 40 bytes, S_{l3} = 40 bytes
- layer 2 header size – 14 bytes, S_{l2} = 14 bytes

Total bandwidth for a single connection:

$$B = (160 + 14 + 40) \times \frac{64000}{160} = 85600bps = 85.6Kbps$$

In other words, this value is the peak-rate bandwidth for a single voice source.

Mean burst and mean silence length

For the on/off fluid model that we use for voice traffic the following parameters are suggested [IH92]:

mean "on" period: 352 ms

mean "off" period: 650 ms

During "on" periods the voice source is active and operates at 64 Kbps resulting in output data stream of 85.6 Kbps. During "off" periods the source is idle and produces no cells.

Mean rate of a single connection

The resultant effective bandwidth is always a value greater than the mean rate of the connection. The mean rate is the amount of data transferred divided by the time taken to transfer it. To find the mean rate of a single voice source we have to take into account both silence and burst periods. If the available bandwidth along the path is less than the mean rate, then the transmission of packets becomes impossible.

The activity rate P_A of a single voice source *(Equation 6-1)*:

$$P_A = \frac{T_B}{T_B + T_S} = \frac{352}{352 + 650} \approx 0.35$$

where T_B =352ms and T_S=650ms are the average times spent in "on" and "off" states respectively [IH92].

It means that as much as 35 percent of the whole time the source operates at 85.6 Kbps (peak-rate) and 65 percent of the time the source is idle. It makes the mean rate of a single connection be:

$$\textit{(mean rate)} = \textit{(activity rate)} \times \textit{(peak-rate)} = 0.35 \times 85{,}6\ \textit{Kbps} \approx 30\ \textit{Kbps}$$

So, if the available bandwidth is less than 30 Kbps, then the request for connection must be rejected.

Effective bandwidth

In our algorithm, effective bandwidth is calculated individually for each requested connection. How this is to be done in general is described in Chapter 5, but for our simulations we use simplified method for calculating effective bandwidth. We are not considering jitter and packet loss constraints, and we suppose that effective bandwidth

guarantees only latency requirements. Following the equation *(5-1)* from Chapter 5, the basic formula we use for calculations is:

$$B_{eff} = \frac{N_{hops} \times S_{packet}}{latency}$$

Where N_{hops} is the number of hops along the path and S_{packet}=*160 bytes* is the size of a voice packet.

The connection can be established not just for a single source of voice traffic but for an aggregate of voice sources as well. In this case the resulting effective bandwidth is:

$$B_{eff} = N_{sources} \frac{N_{hops} \times S_{packet}}{latency} \qquad (6\text{-}2)$$

Where $N_{sources}$ is the number of sources in the aggregate.

Propagation delays

Propagation delay depends on the physical characteristics of links and their lengths. Typical delay for cables of category 5e UTP, commonly used within the network, is slightly less than 5 ns per meter. For our simulations, we suppose that distance among the nodes varies from 1 to 10 km. This corresponds to variation of propagation delays from 0.005 to 0.05 ms.

Link capacities

In our simulated network, all links have capacities of 10Mbps.

6.3.2 Requests Generation Model

The request generation model is responsible for generating customers' requests for services. The model consists of two parts. The first part is responsible for modelling the arrival process of the requests to the CRM. The second part is responsible for constructing the contents of the requests (e.g. QoS requirements).

Modelling an arrival process

For our simulation model we suppose that customers' requests are mutually independent, identically distributed and with a small probability of happening simultaneously. Accepting these assumptions we can model a rate of arriving requests as a Poisson process.

If $X=(X_k : k\geq 1)$ denotes the number of requests arriving in successive, non-overlapping time intervals of length $\Delta t>0$, then X is the increment process of a Poisson process with parameter λ if and only if the random variables X_k are independent and identically distributed with:

$$P[X_k = n] = e^{-\lambda\Delta t}\frac{(\lambda\Delta t)^n}{n!}, n \geq 0$$

where λ is an expected rate of arriving requests.

To simulate a Poisson process, we perform the following algorithm:

1. Set $n=0$, $T_n=0$
2. Generate ξ from an $exp(\lambda)$ distribution
3. Set $n=n+1$, $T_n=T_{n+1}+\xi$
4. Return to step 2

Where T_n is the time at which the *nth* customer arrives and ξ is a random value representing interarrival times, which are exponentially distributed in the case of the Poisson process.

Constructing a request

A request coming to the Central Resource Manager contains the following parameters:

Source – randomly generated source node, which is an initiator of the connection.

Destination – randomly generated destination node of the connection.

Latency – a requirement for end-to-end delay of the connection. This is a random value from 100 to 150 ms (chosen according to Section 6.3.1.2).

Session Time – duration of a connection. This is a random value from 1 to 20 minutes. We suppose that each reservation in the network can be requested for the duration of time from 1 to 20 minutes.

Number of sources – the number of voice sources in the aggregate. Following the work performed by [KC01], we choose this value to be random in the range from 1 to 10.

6.3.3 Shortest path algorithm implementations

Our shortest-path algorithm implementations comprise four different routing algorithms that are implemented on the base of Dijkstra's algorithm. The implementation of Dijkstra's algorithm follows the main steps of the pseudo-code represented in Figure 5-3.

The implemented algorithms are the following:

1. *Shortest-distance algorithm* – an implementation of the algorithm that selects the path with the shortest distance. The distance function is defined by [ST97]:

$$dist(P) = \sum_{j=1}^{k} \frac{1}{B_{ij}}$$

   where B_{ij} is the free bandwidth available on link $i \rightarrow j$.

   The implementation of the algorithm is an implementation of Dijkstra's algorithm with link costs equal to the inverse bandwidths *1/B* of the links.

2. *Shortest-delay algorithm* – an implementation of the algorithm that selects the path with the minimal end-to-end delay. This is an implementation of Dijkstra's algorithm with link costs equal to the sum of the transmission and propagation delays on the links (we do not consider queuing delays in our simulations).

3. *Shortest-hop algorithm* – an implementation of the algorithm that selects the path with the minimum hop count. This implementation is that of Dijkstra's algorithm with all link costs equal to one unit.

4. *Widest-shortest algorithm* – an implementation of the algorithm that selects the path with the minimum hop count. If there is more than one path with the minimum hop count, the one with the maximum available bandwidth is selected. This is slightly a modified shortest-hop algorithm implementation. Unlike the shortest-hop algorithm, the modification takes into account available bandwidths along prospective paths.

6.3.4 Modified Dijkstra's algorithm implementation

We implemented our algorithm as it is presented in Figure 5-10. Some features related to our implementation are the following:

- we use linear cost functions $Cost = \frac{B}{C}$, where B is the bandwidth to be reserved for a current request and C is the residual capacity of the link;
- we simplified the calculation of the effective bandwidth; the calculation is now performed as it is described in Section 6.3.1.2 using the equation 6.3.

6.4 Simulation software

For our simulation needs we developed our own software using the development tool Rational Rose. This tool was used in the beginning of the development stage for designing the general structure of our simulation model. The program code was written in the Java language.

Rational Rose is a powerful visual modelling tool for object-oriented analysis and design. It allows users to visualize and understand the software architecture before writing any code, eliminating wasted effort in the development cycle. Based on the industry standard Unified Modelling Language (UML), Rational Rose (using UML notation) provides static

and dynamic views of a logical model and enables users to create and refine these views within an overall model representing the whole software system. The overall model contains classes, objects, use cases, packages, processors, devices and the relationships between them. The notation provides graphical icons to represent each kind of model element and relationship. A model also contains diagrams and specifications, which provides a means of visualising and manipulating the model's elements and their model properties.

In addition, Rational Rose provides the Interface Design Language (IDL) Code Generator to produce IDL source code from the information contained in a model. The large variety of supported languages allows users to handle the needs of all modelling environments (Web development, Data Modelling, Java, Visual Studio, and C++). The code generated for each selected model element is a function of that element's specification, the model's properties, and the model's project properties. These properties provide the language-specific information required to map the model into IDL.

6.5 Description of the simulation program

The flowchart of the simulation program is shown in Figure 6-5. The first part of the program is generation of the network. There were four different networks generated for our simulations. These were 25, 50, 75 and 100 node networks. The generated networks have randomly connected links with the average *node connectivity*[5] of 3.0. The values for propagation delays are randomly generated for each link given the propagation delay range 0.005-0.05 ms. The value of link capacity is 10Mbps for all links. After generating one of the networks, the main part of the program is a loop consisting of N cycles. At the beginning of each cycle of the loop, a Poisson process generates an SLA request. Parameters of the Poisson process are adjusted in such a way that each cycle of the loop corresponds to 1 second of real time. For example to simulate this, if the average rate of incoming requests in real life is expected to be 20 requests per minute, the expected rate of arriving requests is adjusted in the Poisson process in such a way so that 20 requests are generated on average for every 60 cycles of the loop. Depending on the average rate of

[5] *Node connectivity* is the average number of links for each node in the network

arriving requests, an SLA request may or may not be generated at a certain cycle of the loop. If it is generated, its random parameters are: source, destination, number of voice sources (1-10), latency (100-150ms) and duration of session (1-20min).

After generating a request, an implementation of a routing algorithm tries to find a path satisfying the latency requirement. This part of the program is different for each routing algorithm used in the simulations. If the path is found, the reservation of the necessary bandwidth is performed along the path. If the path is not found, the request is rejected. The total number of the rejected requests is used at the end of the program for calculating the call-blocking rate, which is explained later.

This is followed by removing all expired reservations in the network. That is, the bandwidth in a link is freed if its reservation is expired by the current time and after that the program runs the next cycle of the loop.

For each of the simulated routing algorithms, the loop runs for 300000 cycles, which corresponds to 3.5 days of real time. At the end of each simulation, the information about links' utilisation is collected. This information is used later for analysing the utilisation of network resources.

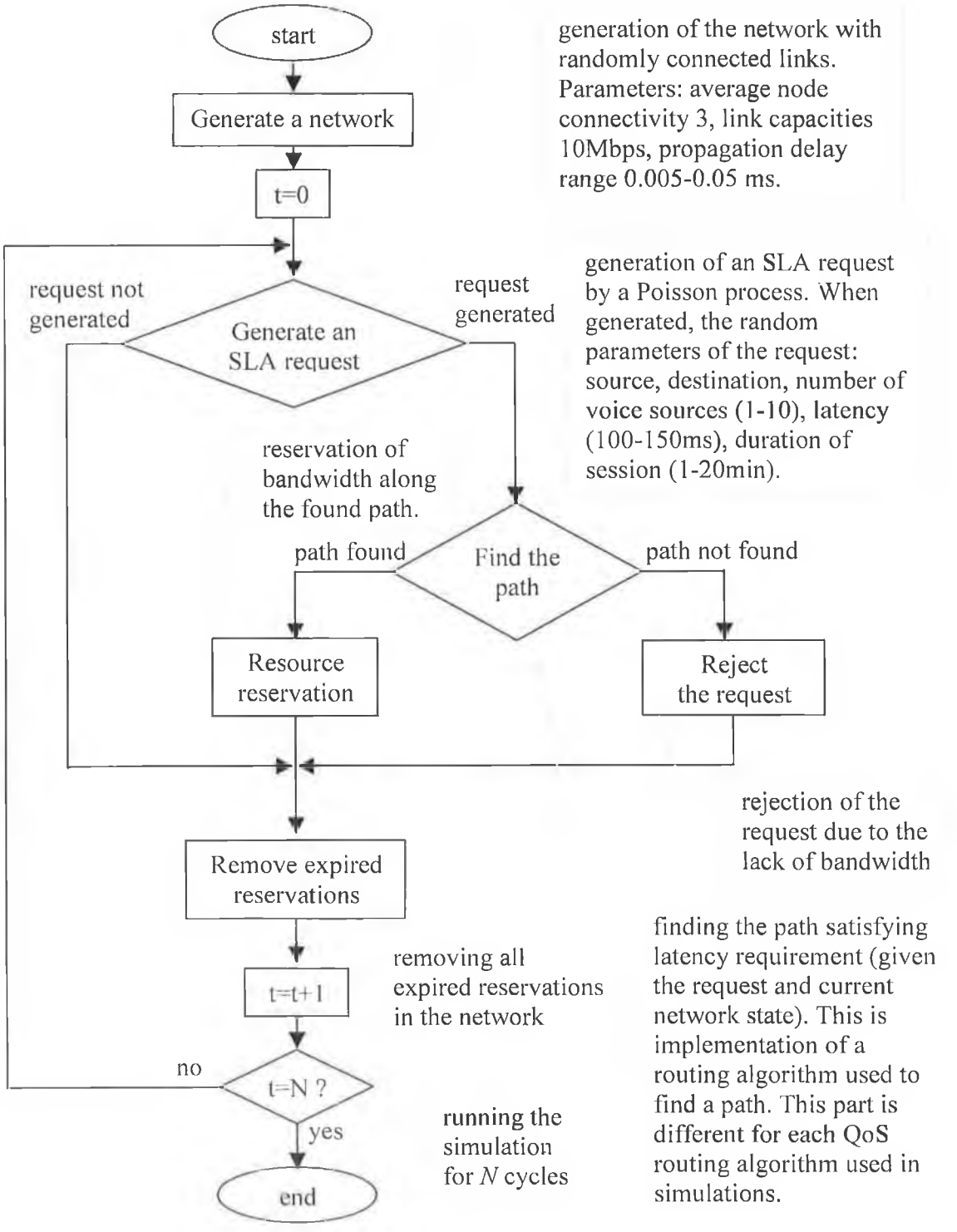


Figure 6-5: Flowchart of the network simulation program

The class diagram of our simulation model is represented in Figure 6-6. Below we give a description of main classes of the software model and their main responsibilities.

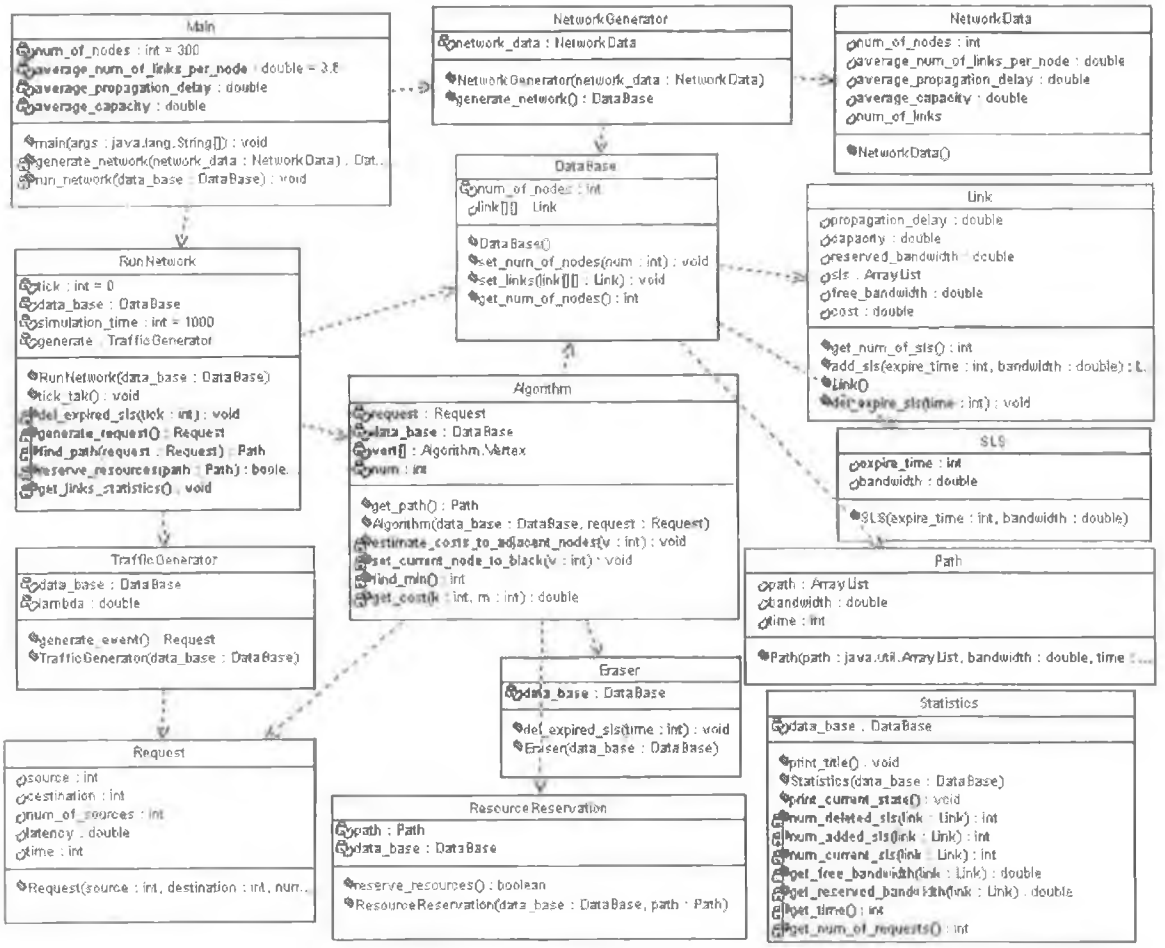


Figure 6-6: Class diagram of the simulation model

Main – the main class of the model responsible for starting up the simulation.

NetworkGenerator – given the total number of nodes in the network and the average number of links per each node, this class generates the whole network topology. First, it generates for each node the random number of links and, second, it generates the random nodes to which these links are connected.

RunNetwork – this class is responsible for managing the whole simulation process. It contains the main cycle of the simulation program including generating a request, finding a path and resource reservation.

TrafficGenerator – this class is responsible for generating customers' requests for services. It generates requests according to the Poisson arrival process and it generates random values for requests' parameters (e.g. source, destination, end-to-end delay requirements).

AlgorithmModified – this is the implementation of our algorithm that finds a path taking into account particular QoS requirements (end-to-end delay) and resource load.

AlgorithmShortestDistance – this class contains the implementation of the shortest-distance algorithm. It finds a path complying to the customer's request with the shortest distance.

AlgorithmShortestDelay – this class contains the implementation of the shortest-delay algorithm. It finds a path with the minimum end-to-end transmission and propagation delays.

AlgorithmShortestHop – this class contains the implementation of the shortest-hop algorithm. It finds a path with the minimal number of hops.

AlgorithmWidestShortest – this class contains the implementation of the widest-shortest algorithm. It finds a path with the minimum hop count. If there is more than one path with the minimum hop count, the one with the maximum available bandwidth is selected.

ResourceReservation – this class is responsible for resource reservation along the given path (e.g. bandwidth reservation).

Path – this is the class for storing information about found paths. It contains the list of nodes comprising the path, the bandwidth necessary to reserve along this path and the duration of time the bandwidth should be reserved for.

SLS – this class is for storing information about currently reserved resources in the network, or in other words currently installed SLAs. It contains the amount of bandwidth reserved in a particular link and the time when this reservation is expired and the bandwidth should be freed.

Link – this class keeps all information about a particular link and provides functions for operating with links (e.g. bandwidth reservation)

NetworkData – this class contains some network information (e.g. the number of nodes, the number of links)

DataBase – this is the data base of the whole network topology. It contains information about all nodes and links in the network and their interconnections.

Request – this class represents customer's request and contains the source node (where request was generated), the destination node (with which the connection is to be established), the QoS requirements (only latency in this simulation), the duration of session, and the number of voice sources in the aggregate.

Eraser – this class is responsible for removing the expired SLSs in the network. In other words it frees the bandwidth in links when the session time of a particular connection is over.

Statistics – this class provides functions for collecting statistics information about the network. The main information that it provides is related to the distribution of resources in the network. It collects data concerning the amount of free resources and reserved resources in the network and calculates the variance of resource load among the links. This is used further for analysing the load balancing in the network and comparing the performance of the shortest path algorithms and our algorithm.

6.6 Simulation Results and Discussions

In order to estimate the performance of our traffic engineering system, we performed simulations for the system running several different routing algorithms. We estimate the performance of our system by comparing the resource utilisation results of the system based on our algorithm and systems based on other well known QoS routing algorithms. As the performance of a particular QoS routing algorithm can depend on the network load [MA98], we performed network simulations for different traffic loads. Also, we analysed how the performance of the algorithms depends on the size of the network. With this in mind the simulations were carried out on 25, 50, 75 and 100 node networks.

The experimental results presented in the following sections are the results for the network resource utilisation after a certain amount of simulation time corresponding to 3.5 days of real time (Section 6.5). All simulations of the network scenario were performed for the network carrying guaranteed traffic only. The light and heavy traffic loads were simulated

by adjusting the rate of customers' requests in the request generation model. The light load of traffic was simulated with an average rate of 5 requests arriving each minute. The average rate of arriving requests in the case of heavy loads was 60 requests per minute.

6.6.1 Simulation results for light loads

In the first set of experiments that were performed, we analysed the performance of routing algorithms for light loads. The experimental results for light loads for the four different networks are presented in Figures 6-9, 6-10, 6-11 and 6-12, and summarised in Figure 6-7 and Figure 6-8.

On each graph (Figures 6-9, 6-10, 6-11 and 6-12) the X axis represents the resource load in percents and the Y axis denotes the number of links having the particular load of resources. The resource load is related to the reserved bandwidth in the network and also reflects how much of the whole available bandwidth is free for future requests. The load of a link is calculated as:

$$L = \frac{B_{reserved}}{C}$$

where $B_{reserved}$ is the reserved bandwidth on the link, and C is its total capacity.

Thus, the graphs show how many links in the network have a particular amount of reserved bandwidth. The variation of link load shows the general resources utilisation and how balanced the load on the network is. The average load on the graphs shows us how much of the resources are utilised in the network and the standard deviation shows how balanced the load is.

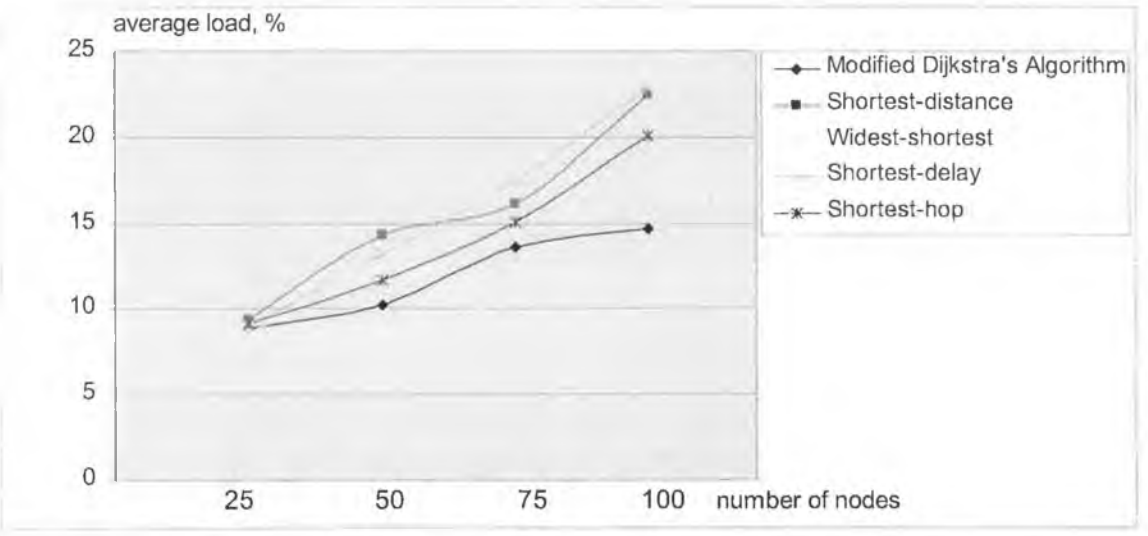


Figure 6-7: Resource utilisation for 25, 50, 75 and 100 node networks (light loads)

As was mentioned before, there could be two goals of achieving efficiency in resource utilisation. These goals are to minimise the resource utilisation and to balance the load on the network. Figure 6-7 shows the average link load and thereby illustrates how well the algorithms achieve the first goal of minimising the resource utilisation. Figure 6-8 demonstrates the variation of link load for each algorithm and shows how well the load on the network is balanced.

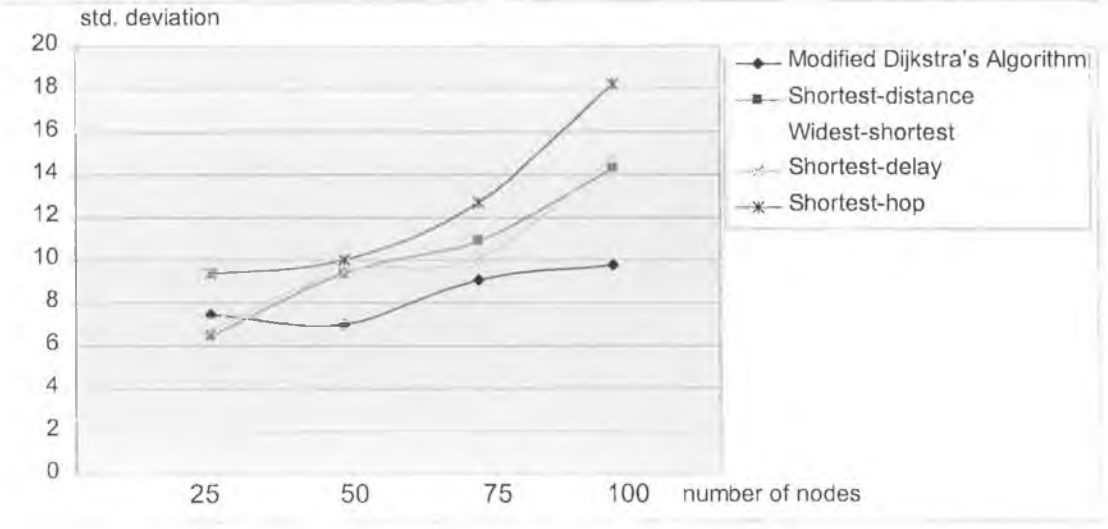


Figure 6-8: Variation of link load for 25, 50, 75 and 100 node networks (light loads)

Even though the algorithms give quite similar performance to each other for 25 node network, they result in varying performance levels for larger networks. It can be seen that

even though our modified Dijkstra's algorithm does not provide a very good solution for the 25 node network, it gives the best results for the other networks with respect to both resource utilisation and load balancing. The next best results for load balancing are those of the shortest-distance and shortest-delay algorithms. The shortest-distance algorithm selects a path with the minimum sum of the inverse residual bandwidths along it, and the shortest-delay algorithm selects the path with the minimal end-to-end delay (considering transmission and propagation delays in our simulations). All this is aimed at balancing the load on the network and gives good results. However, with respect to resource utilisation, these algorithms perform slightly worse than the widest-shortest and shortest-hop algorithms. The shortest-hop algorithm selects the paths with as few hops as possible to conserve resources. That is why it results in a very good resource utilisation solution. However, it is not designed to distribute the load and, thus, gives the worst load-balanced solution. The widest-shortest algorithm tries to distribute the load and provides slightly better results with respect to load balancing, but gives slightly worse results with respect to resource load.

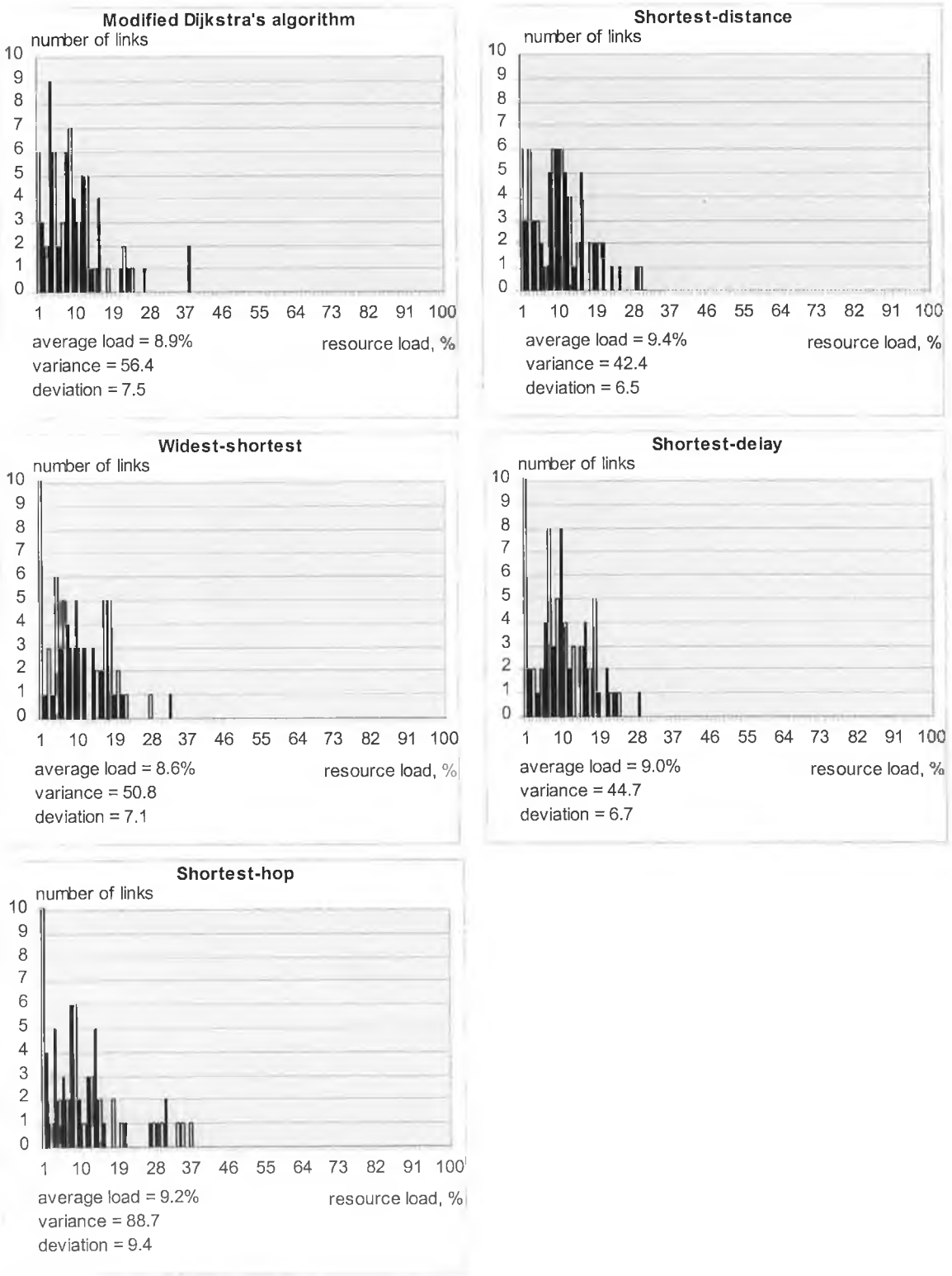


Figure 6-9: Resource utilisation for 25 node network (light loads)

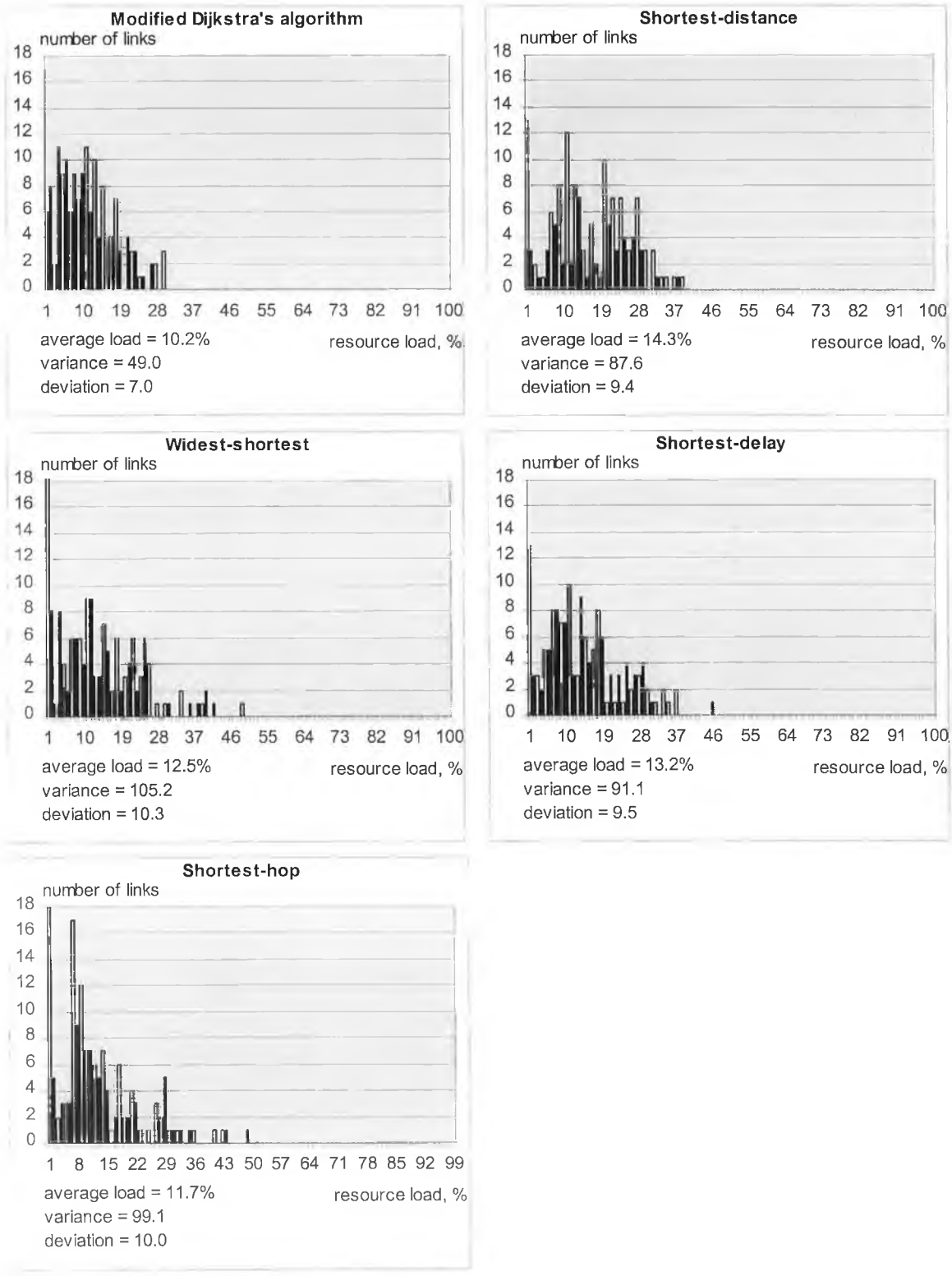


Figure 6-10: Resource utilisation for 50 node network (light loads)

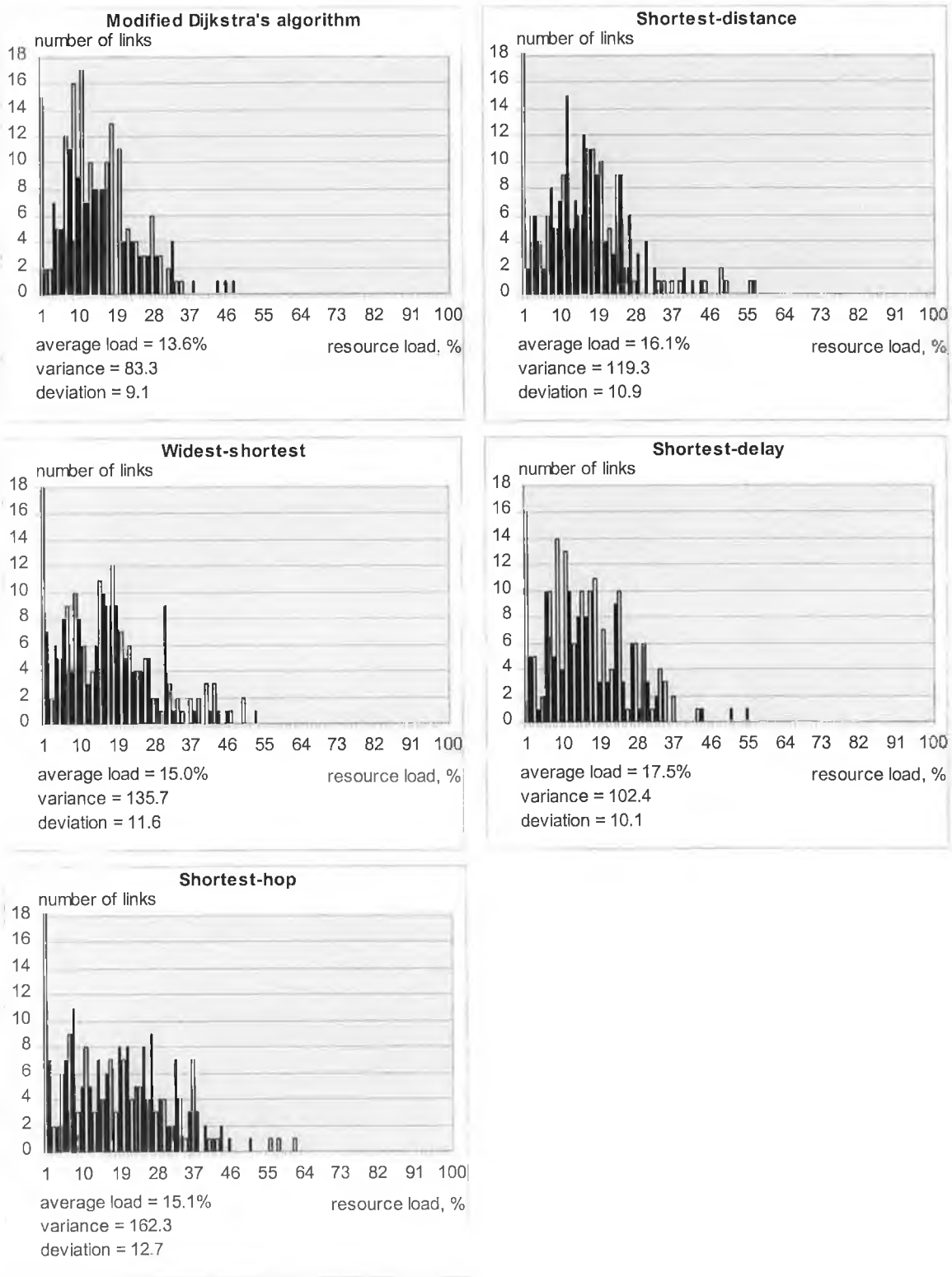


Figure 6-11: Resource utilisation for 75 node network (light loads)

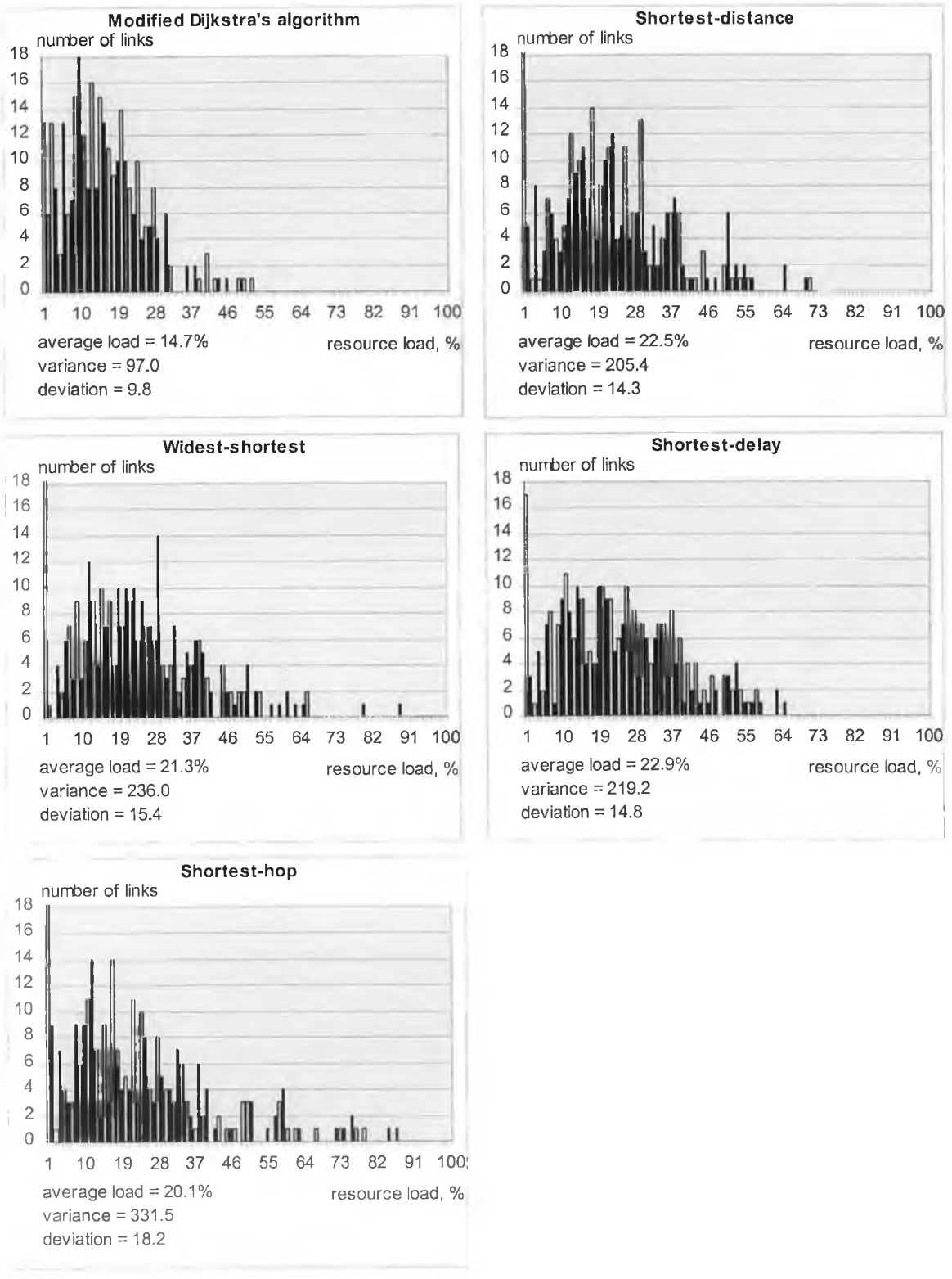


Figure 6-12: Resource utilisation for 100 node network (light loads)

6.6.2 Simulation results for heavy loads

In the second set of our experiments, we analysed the performance of routing algorithms for heavy loads. The experimental results for heavy loads for the four different networks are presented in Figure 6-16, 6-17, 6-18 and 6-19, and summarised in Figure 6-13 and Figure 6-14.

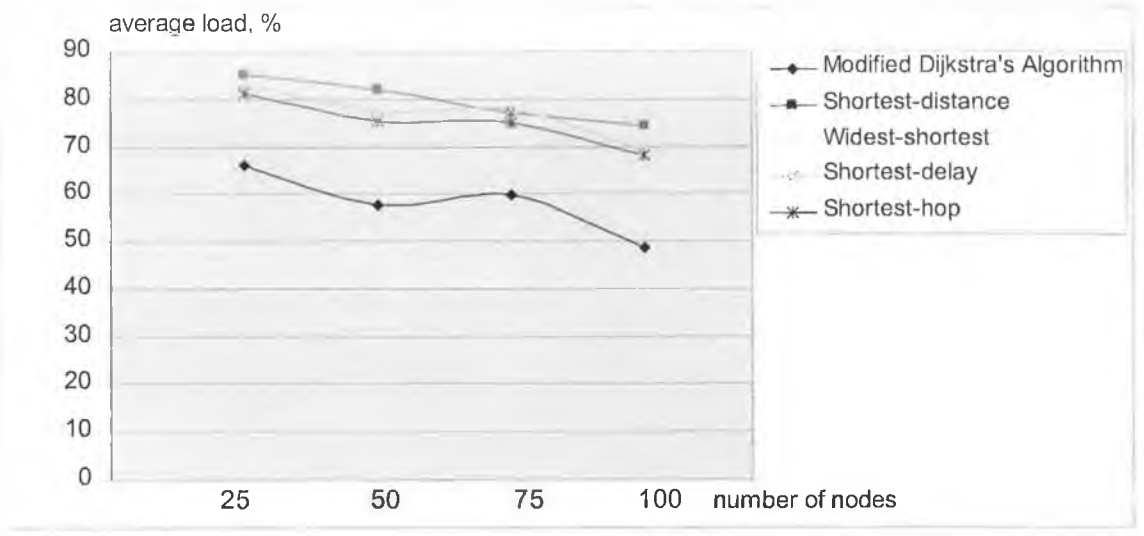


Figure 6-13: Resource utilisation for 25, 50, 75 and 100 node networks (heavy loads)

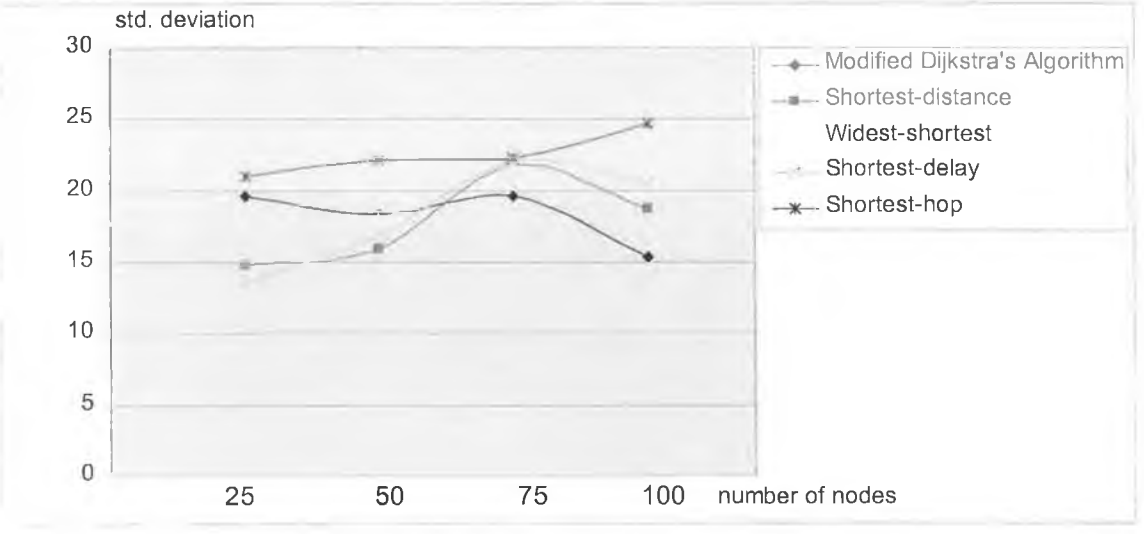


Figure 6-14: Variation of link load for 25, 50, 75 and 100 node networks (heavy loads)

Unlike in the case of light loads, in this case the load-balanced solution of our modified Dijkstra's algorithm is better than that of the others only for the big networks of 75 and

100 nodes (Figure 6-14). However, it is seen that it results in the lowest link utilisation for all simulated networks Figure 6-13). This happens because of the different approaches to calculating the effective bandwidth for a particular connection. While our algorithm calculates the effective bandwidth at each step, whereby adjusting the value of the bandwidth to individual path's parameters (e.g. number of hops), the shortest-path algorithms calculate the bandwidth for some maximum possible number of hops that the routing path can traverse [WJ00]. In case when the real number of hops along the path is less than that value, an excessive amount of bandwidth is reserved along the path.

For heavy loads, we also analysed the performance of the routing algorithms with respect to the *call blocking rate*. The call blocking rate is the percentage of requests being rejected by the network over the total number of arrival requests:

$$\text{call blocking rate} = \frac{\text{number of rejected requests}}{\text{number of arrival requests}}$$

The *number of rejected requests* shows for how many requests the routing algorithm could not find a path satisfying the requested QoS service. This happens in the case when there are no sufficient resources available in the network. Thus, the call blocking rate is a good performance metric showing how efficiently the routing algorithm distributes the load on the network.

Figure 6-15 shows the call blocking rate results for all simulated networks.

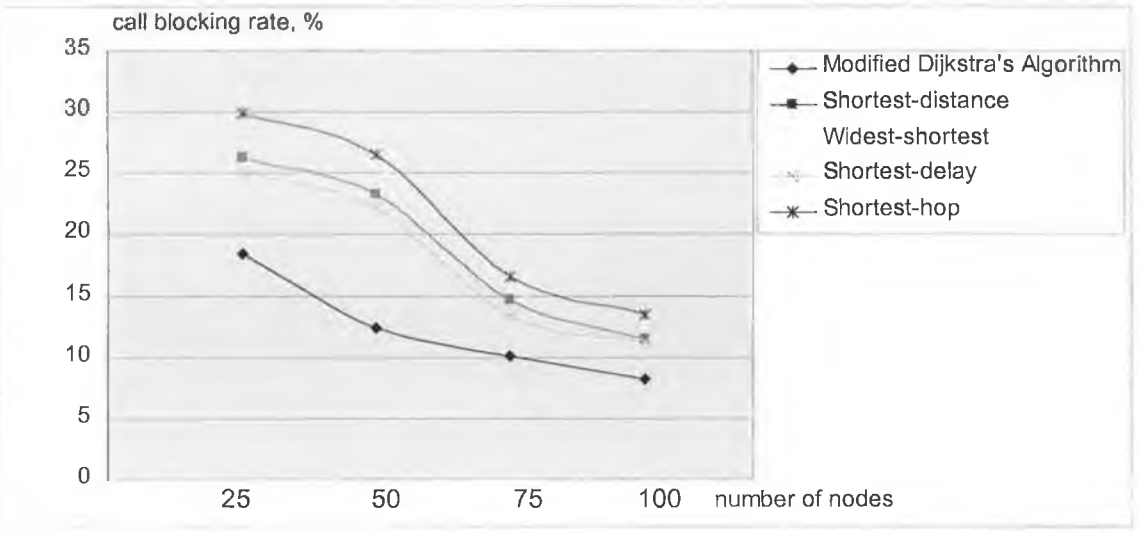


Figure 6-15: Call blocking rate results

The modified Dijkstra's algorithm results in the lowest call blocking rates for all simulated networks. This shows that our approach provides better solution for distribution of resources in the network resulting in the more effective way of serving the arrival requests.

The graph also shows that the shortest-hop algorithm gives the worst results for call blocking rates. This suggests that, when the load is heavy, choosing the path with the minimum number of hops does not perform well, because in this case conserving resources by selecting the shortest path is not so important as balancing the load. In case of heavy loads, selecting the shortest-path may result in blocking future arrivals, as the selected path may be heavily loaded.

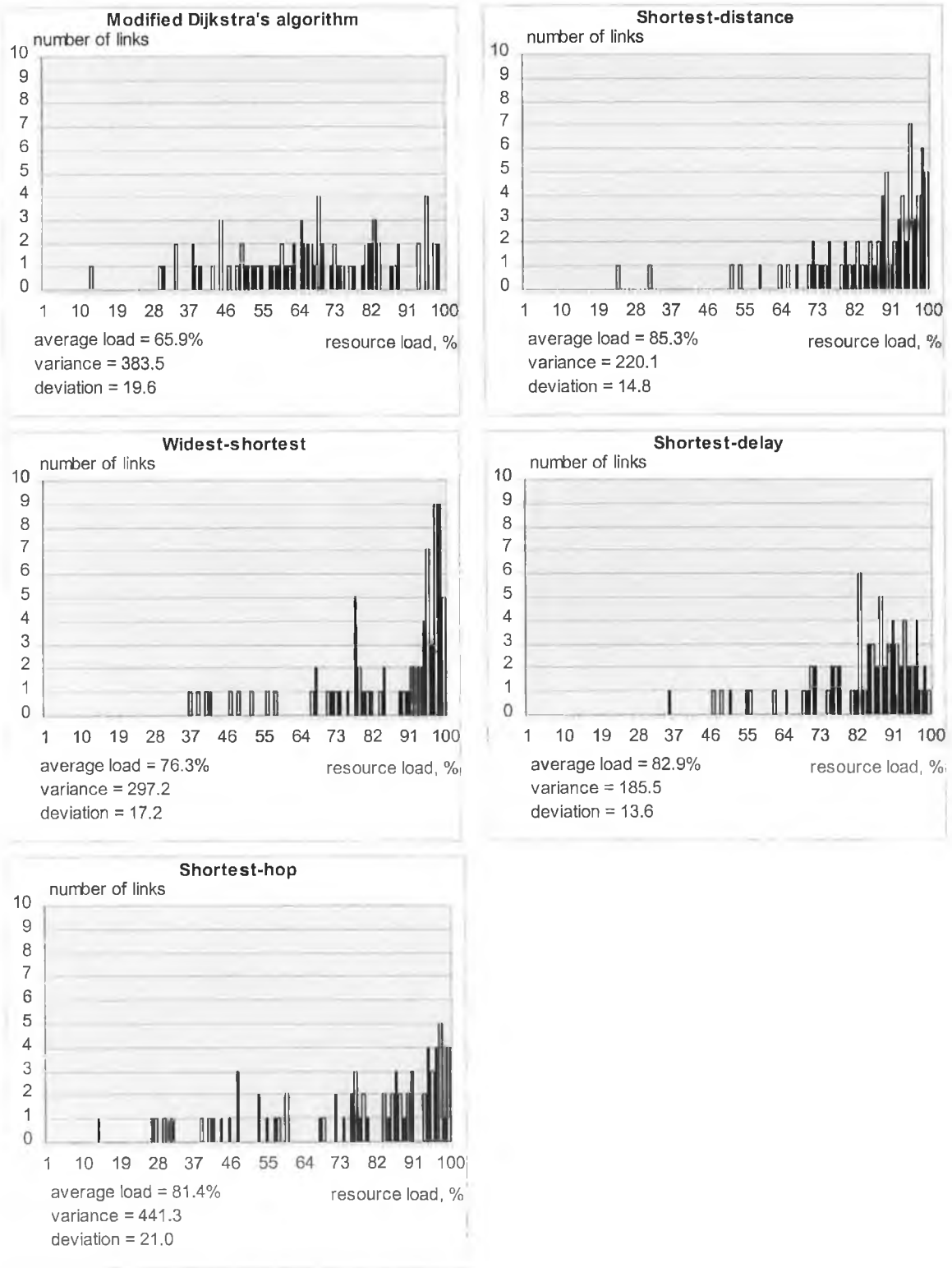


Figure 6-16: Resource utilisation for 25 node network (heavy loads)

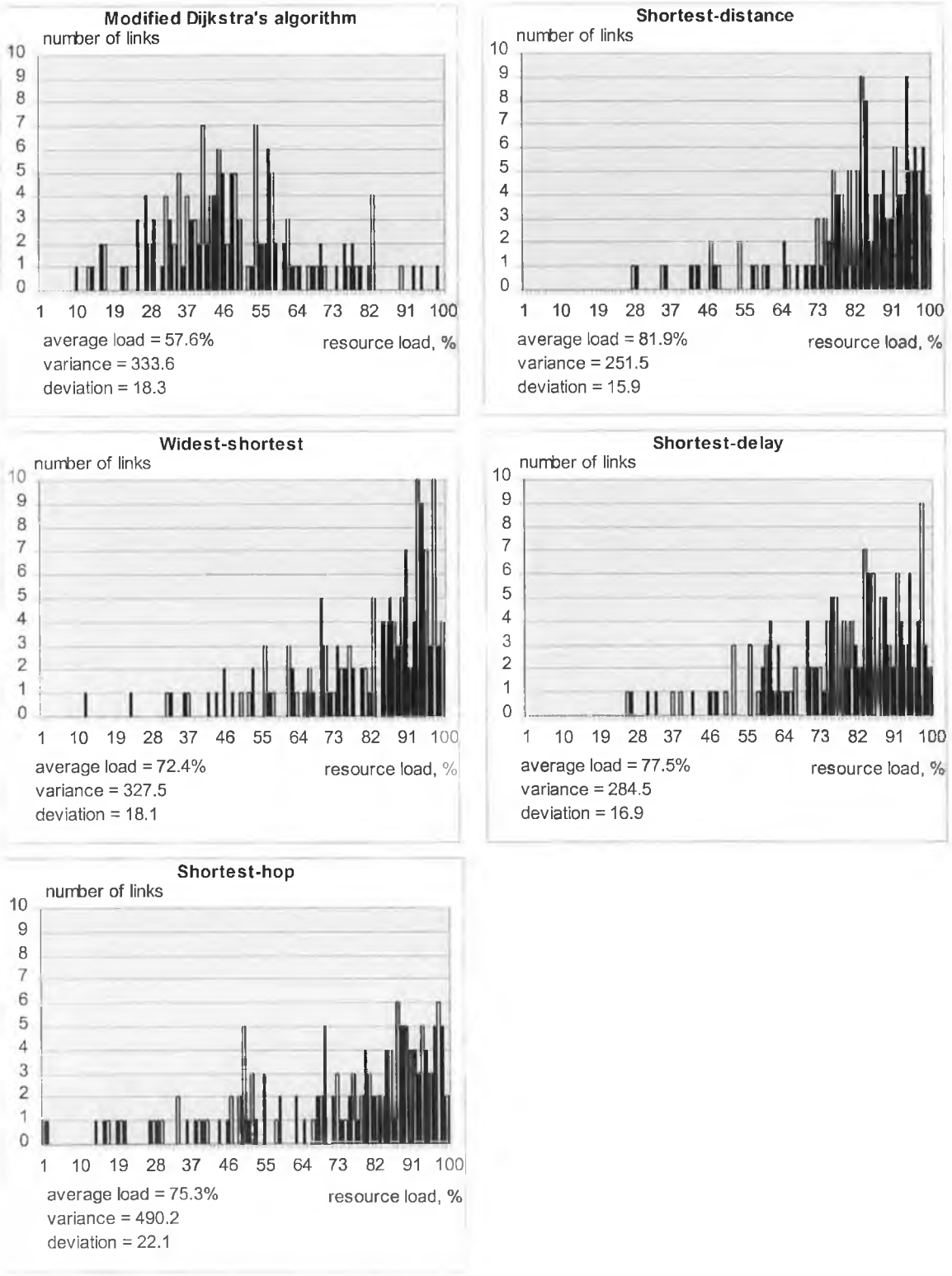


Figure 6-17: Resource utilisation for 50 node network (heavy loads)

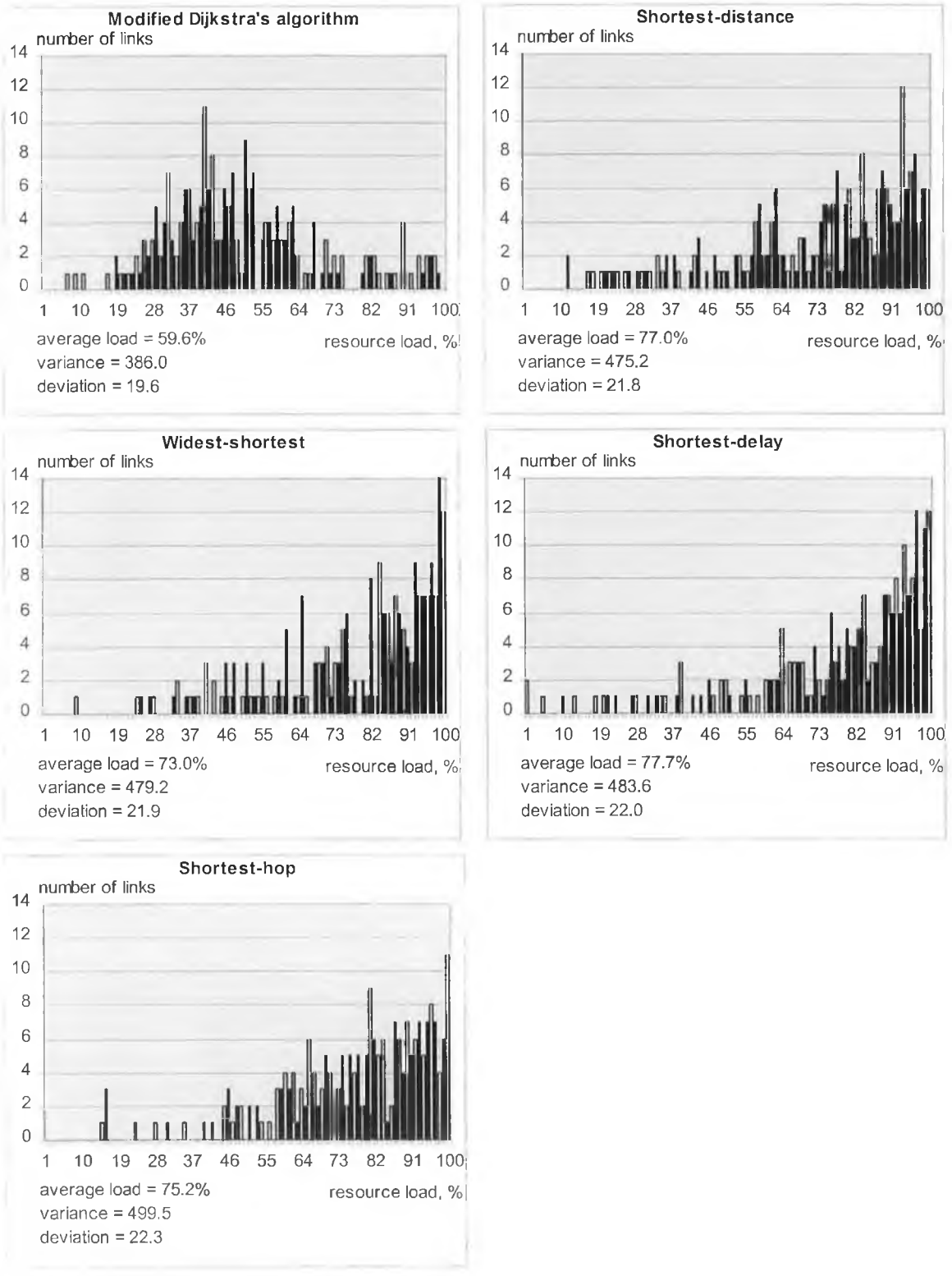


Figure 6-18: Resource utilisation for 75 node network (heavy loads)

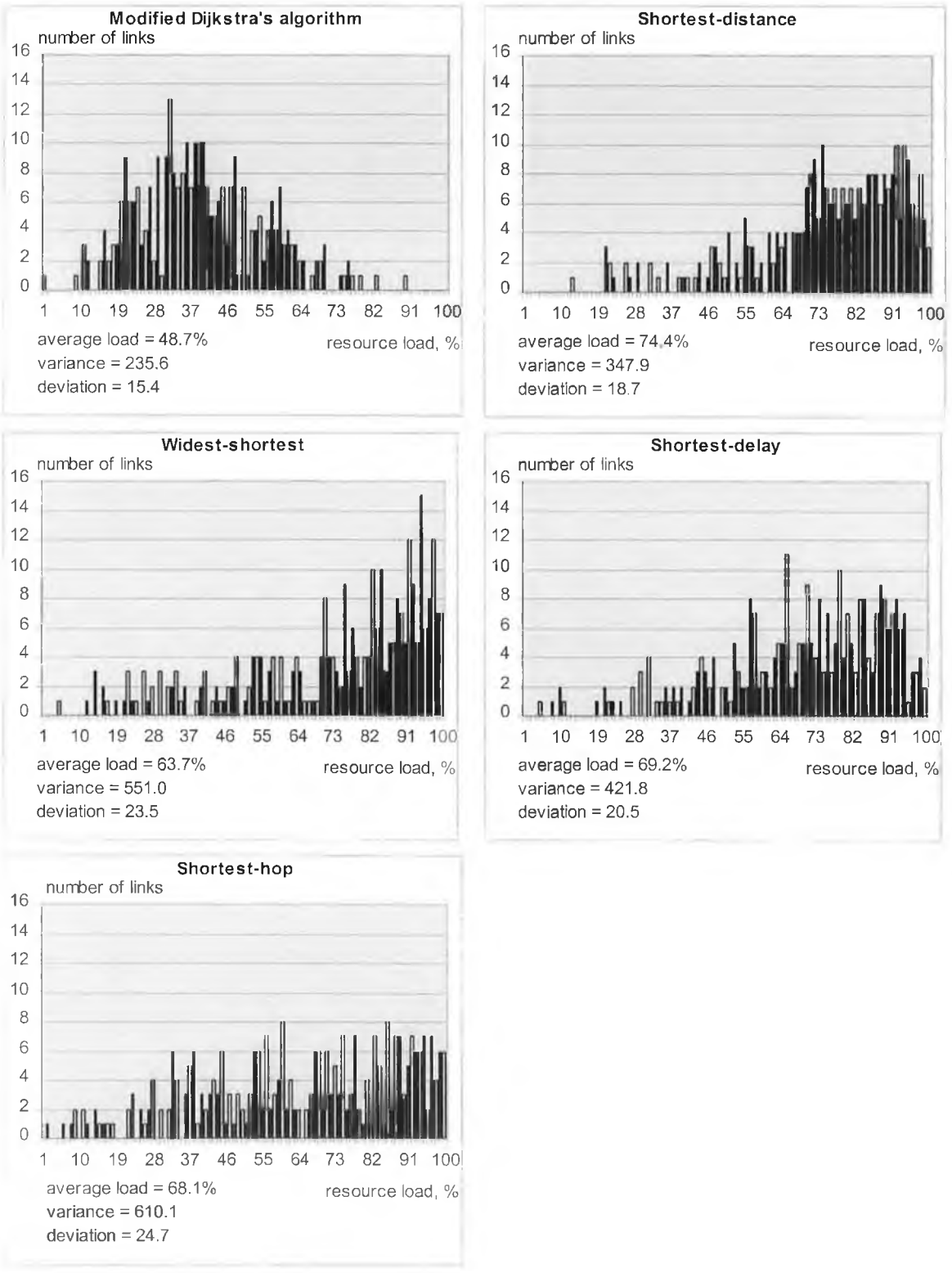


Figure 6-19: Resource utilisation for 100 node network (heavy loads)

6.6.3 Summarising simulation results

All sets of experiments show that the traffic engineering system running our modified Dijkstra's algorithm results in a very good balanced load solution while providing also good link utilisation.

Let us summarise the reasons why our modified Dijkstra's algorithm results in the best solution. The main reason is that, unlike the QoS routing algorithms described here, our algorithm is aimed at not only selecting the path satisfying the QoS requirements but also determining the optimal amount of resources that has to be reserved along that path.

The input parameters for a QoS routing algorithm are normally QoS requirements (e.g. end-to-end delay) and the amount of resources (e.g. bandwidth) necessary for a connection. The output of a QoS routing algorithm is the path that satisfies the QoS requirements and has enough of the requested resources. The disadvantage of this approach is that the initially requested amount of resources may be overestimated, which results in resource over-reservation. Resource overestimation may happen due to the fact that all potential paths in the network have individual characteristics (e.g. propagation delays, number of hops along the path), which are not taken into account. The amount of resources for a connection is calculated before running the algorithm and is irrespective of the prospective path. To be sure that the QoS requirements will be satisfied, the calculation is performed for the worst expected path (in terms of delay or packet loss). For example, the bandwidth for a connection can be calculated for the maximum possible number of hops along the path and for the longest possible propagation delay. It is evident that this leads to resource over-reservation when the number of hops along the path is not large or the total propagation delay is not long.

The advantage of our algorithm is its ability to take into account the individual path's parameters. The amount of resources necessary for a connection is not known in the beginning of the algorithm, as it depends on the finally chosen path of the connection. When the solution is found, the amount of resources reflects individual characteristics of the selected path (e.g. number of hops along the path, propagation delay).

Another advantage of the presented algorithm is that it calculates the link costs taking into account the bandwidth necessary to be reserved for the currently routed path. Whereas, for

example, the shortest-distance algorithm calculates the costs considering only available bandwidth in links without taking into account the bandwidth of the currently routed path. This results in a different load balanced solution.

Altogether, this gives a better load balanced solution while providing good link utilisation.

7 Conclusion

7.1 Summary and Conclusions

The prime research objective of the thesis included developing an approach to traffic engineering that uses DiffServ and MPLS technologies to provide QoS guarantees over an IP network. The main focus of the work described here was on the problem of how best to route traffic within the DiffServ/MPLS network so that the demand can be carried with the requisite QoS while making the best use of network resources. The problem is motivated by the needs of network service providers to quickly setup QoS guaranteed paths for the real-time applications (e.g. VoIP, VoD) while optimising resource utilisation.

We considered a network scenario where customers contact an Internet Service Provider (ISP) in an *online* fashion. That is when requests for establishing QoS guaranteed paths can arrive any time one by one and when information about future requests is not available.

In this thesis, we presented a traffic engineering system that can set up QoS guaranteed label-switched paths (LSPs) between specified ingress-egress pairs in the DiffServ/MPLS domain. The main architecture components of the system were defined. The key component of the system is a central resource manager (CRM) responsible for monitoring and managing resources within the network and making all decisions to route traffic according to QoS requirements. Applying the central resource manager we removed the complexity of finding QoS routes at the core of the network.

With the help of the Simple Network Management Protocol (SNMP) or one of the link state protocols, the CRM maintains a database containing a topological map of the network domain and information about the current state of the network resources. Upon receiving a routing message for setting up a QoS guaranteed LSP, the CRM computes the explicit path by running a routing algorithm aimed at making the best use of network resources. Once a

path has been found, the CRM uses the RSVP protocol to establish an explicit LSP between the specified ingress and egress label-switched router (LSR).

The routing algorithm, which is used by the CRM for finding QoS guaranteed paths, was developed. The algorithm generates a solution for the QoS routing problem of finding a path with a number of constraints (delay, jitter, loss). The primary objective of the proposed algorithm is to address the shortcomings of the currently used shortest-path algorithms. The main problem when using the shortest path algorithms for finding QoS routes is that some links between the ingress-egress pairs may get congested while links along possible alternative paths remain free. The introduced algorithm seeks to balance the load on the network and to achieve better resource utilisation. The algorithm is based on the notion of effective bandwidth and cost functions for load balancing. The effective bandwidth reflects how much of the resource is needed by the source to obtain the required QoS. It was assumed here that the QoS constraints (e.g. latency, loss, jitter) can be incorporated into an effective bandwidth requirement for the LSPs. The notion of effective bandwidth is then used to balance the load on the network. The load is balanced if the effective bandwidths on the links are balanced, i.e. if the difference between the effective bandwidths of the traffic carried on each link is minimised.

The algorithm is the modification of the well-known Dijkstra's shortest-path algorithm. The modification of Dijkstra's algorithm is related to the calculation of link costs, which in our case is realised in a different way. When calculating the link costs, the modified algorithm uses the appropriate cost functions for the links and takes into account the individual paths' characteristics such as the number of hops along the paths, queuing and propagation delays. This is aimed at the more precise estimation of the effective bandwidth for a connection and minimising the overall resource utilisation.

For validating the approach of the presented traffic engineering system the network simulations were performed. The simulation model for a network scenario running a Voice over IP (VoIP) service across a DiffServ/MPLS network was designed and implemented. The general structure of the simulation model and the software code in the Java language were developed with the help of the visual modelling tool for object-oriented analysis and design Rational Rose.

The major objective of the network simulations was to demonstrate on the example of voice traffic how the routing can be performed in the network by applying the proposed algorithm. For estimation of the performance of our algorithm, the simulations were carried out for the system under different routing algorithms. These were our modified Dijkstra's algorithm and some of the most commonly used QoS routing algorithms: shortest-distance algorithm, shortest-delay algorithm, shortest-hop algorithm and widest-shortest algorithm. We estimated the performance of our approach by analysing the resource utilisation results of the system running all these algorithms. As the performance of a particular QoS routing algorithm can depend on the network load, we performed the network simulations for different traffic loads. We also performed our simulations for the four different networks of 25, 50, 75 and 100 nodes.

The network simulation results included the graphs with distribution of the traffic load in the network after a certain amount of running time of the VoIP scenario. The average link load and variation of link load for each algorithm were analysed. Both sets of experiments for light and heavy loads showed that the traffic engineering system running our modified Dijkstra's algorithm results in a very good balanced load solution while providing also good link utilisation. Compared to other routing algorithms, the proposed algorithm gives much better performance with respect to resource utilisation and call blocking rate.

7.2 Directions for Future Work

The main objective of future work could be the validation of the described approach in the real network scenario. From this point of view, the directions for future work can be considered as architecture related and algorithm related. With respect to the architecture of the presented system, future work could be on consolidation of the system components and mechanisms necessary for the implementation of the system. The finally developed architecture would be a more defined structure of the described here techniques for a particular network scenario.

With respect to the algorithm, the work could be extended in the several ways. A more realistic approach to calculating the effective bandwidth for an aggregate of traffic could be applied. Apart from latency, the approach would take into account queuing delays and

packet loss. The objective of this approach would be as well to calculate the effective bandwidth considering specific network parameters (scheduling, queue management). A study on different cost functions that could be applied to the algorithm would also be of interest. This work would explore what cost functions result in a better resource utilisation solution in the network with particular traffic load.

References

[AW98] D. Awduche et al., "Extension to RSVP for Traffic Engineering," Internetdraft, draft-swallow-mpls-RSVP-trafeng-00.txt, Aug. 1998.

[AM90] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, R. E. Tarjan, "Faster algorithms for the shortest path problem", Journal ACM 37 (2), 213-223, 1990.

[CA02] "The CADENUS video on demand service", Workshop on 'Future Prospects for Quality of Service", May 2002, Maastricht. http://www.ist-tequila.org/workshop2002/cadenus-demo.pdf

[CA03] CADENUS - Creation and Deployment of End-UserServices in Premium IP Networks, EU Project, 2003. http://www.cadenus.org/papers/; http://www.cadenus.org/deliverables/

[CH98] S. Chen, K. Nahrstedt, "An Overview of Quality of Service Routing for Next-Generation High-Speed Networks: Problems and Solutions", IEEE Network, November/December 1998.

[CN98] S. Chen, K. Nahrstedt, "On Finding Multi-Constrained Paths," IEEE ICC '98, June 1998.

[CO99] Costas Courcoubetis, Vasilios A. Siris, George D. Stamoulis, "Application of the Many Sources Asymptotic and Effective Bandwidths to Traffic Engineering", 1999.

[D59] E. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, vol. 1, 1959, pp. 269–71.

[FR00] Fulvio Risso, "Quality of Service on Packet Switched Networks", PhD Thesis, University of Torino, 2000.

[FR01] A. Feldmann and J. Rexford, "IP Network Configuration for Intradomain Traffic Engineering", IEEE Network Magazine, vol. 15, no. 5, pp. 46-57, September 2001.

[G711] ITU-T Recommendation G.711, "Pulse Code Modulation (PCM) of voice frequencies", 1988.

[G714] ITU-T Recommendation G.714, "Separate performance characteristics for the encoding and decoding sides of PCM channels applicable to 4-wire voice-frequency interfaces", 1988.

[GU97] R. Guerin, S. Blake, and S. Herzog, "Aggregating RSVP-based QoSRequests," Internet draft, draft-guerin-aggreg-RSVP-00.txt, Nov. 1997.

[HL86] H. Heffes, D. Lucantoni, "A markov modulated characterization of voice and data traffic and related statistical multiplexer performance", IEEE J.Select.Areas Commun., SAC-4:856-867, September 1986.

[HU88] J. Y. Hui, "Resource Allocation for Broadband Networks," *IEEE Journal on Selected Areas in Communications*, December 1988.

[JR03] Jim Roberts, "Telecommunication Network Design", COST 242 project, 2003.

[KC01] DaeHo Kim, SeongGon Choi, Jun Kyun Choi, "Performance Analysis of Differentiated Service for Voice over IP", IEICE Trans. Commun., Vol. E84-B, No.11 November 2001.

[KJ99] Kaj, Ingemar, "Stochastic Modelling in Broadband Communications Systems", 1999.

[KL00] Murali Kodialam, T. V. Lakshman, „Minimum Interference Routing with Applications to MPLS Traffic Engineering", 2000.

[KL96] F. P. Kelly, *"Notes on effective bandwidths,"* in: *Stochastic Networks: Theory and Applications*, Eds. F. P. Kelly, S. Zachary and I. Ziedens, Royal Statistical Society Lecture Note Series vol. 4, 1996.

[KY02] D. Katz, D. Yeung, K. Kompella, "Traffic Engineering Extensions to OSPF", IETF Internet Draft, October 2002.

[MA97] Q. Ma, P. Steenkiste, "Quality-of-Service Routing for Traffic with Performance Guarantees", IFIP 5th Int. Workshop on Quality of Service, Columbia University, New York, 1997.

[MA98] Q. Ma, P. Steenkiste, "Routing Traffic with Quality-of-Service Guarantees in Integrated Services Networks", 1998.

[MU01] P. Mehta and S. Udani, "VoIP: Sounding Good on the Internet", IEEE Potentials Magazine, pp. 36-40, October/November 2001.

[NA91] Ramesh Nagarajan, James F. Kurose, Don Towsley, "Approximation Techniques for Computing Packet Loss in Finite-Buffered Voice Multiplexers", IEEE Journal on Selected Areas in Communications, 1991.

[PA98] T. Li and Y. Rekhter, "Provider Architecture for Differentiated Services andTraffic Engineering (PASTE)," RFC 2430, Oct. 1998.

[RA00] P. Aukia, M. Kodialam, P. Koppol, T. Lakshman, H. Sarin, B. Suter, "RATES: A Server for MPLS Traffic Engineering", IEEE Network Magazine, vol. 14, no. 2, pp. 34-41, March 2000.

[RFC1142] D. Oran, "OSI IS-IS Intra-domain Routing Protocol", Feb-01-1990.

[RFC1156] K. McCloghrie, M.T. Rose, "Management Information Base for network management of TCP/IP-based internets", May 1990.

[RFC1157] J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin "Simple Network Management Protocol (SNMP)", May-01-1990.

[RFC1247] J. Moy, "OSPF Version 2", Jul-01-1991.

[RFC1267] K. Lougheed and Y. Rekhter, "Border Gateway Protocol 3 (BGP-3)," RFC 1267, SRI Int'l., Menlo Park, CA, Oct. 1991.

[RFC2205] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP)", RFC 2205, September 1997.

[RFC2210] Wroclawski, J., "The Use of RSVP with Integrated Services", RFC 2210, September 1997.

[RFC 2211] Wroclawski, J., "Specification of the Controlled-Load Network Element Service", RFC 2211, September 1997.

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

[RFC2475] S. Blake et al., "An Architecture for Differentiated Services," RFC 2475, Dec. 1998.

[RFC2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, June 1999.

[RFC2598] V. Jacobson, K. Nichols, K. Poduri, "An Expedited Forwarding PHB", RFC 2598, June 1999.

[RFC2638] K. Nichols, V. Jacobson, L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", RFC2638, July 1999.

[RFC2676] G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A. Orda, T. Przygienda, "QoS Routing Mechanisms and OSPF Extensions", RFC2676, August 1999.

[RFC3031] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, Jan. 2001.

[RFC3036] Andersson, L., Doolan, P., Feldman, N., Fredette, A. and B. Thomas, "LDP Specification", RFC 3036, January 2001.

[RFC3107] Y. Rekhter and E. Rosen, "Carrying Label Information in BGP-4", RFC 3107, May 2001.

[RFC3209] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

[RFC3212] B. Jamoussi, L. Andersson, R. Callon, R. Dantu, L. Wu, P. Doolan, T. Worster, N. Feldman, A. Fredette, M. Girish, E. Gray, J. Heinanen, T. Kilty, A. Malis, "Constraint-Based LSP Setup using LDP", RFC 3212, January 2002.

[RFC3272] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, X. Xiao 'Overview and Principles of Internet Traffic Engineering", RFC 3272, May 2002.

[RFC3550] Schulzrinne H., Casner S., Jacobson V. and R. Frederick, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, June 2003.

[SL02] Henk Smit, Tony Li, "IS-IS extensions for Traffic Engineering", IETF Internet Draft, December 2002.

[SM00] S. Murphy, D. Botvich, T. Curran, "On design of diffserv/MPLS networks to support VPNs", 16th UK Teletraffic Symposium, May 2000.

[ST97] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees". *In Proceedings of IEEE Interna-tional Conference on Network Protocols*,October 1997.

[TC94] T. Corcoran, *"Prediction of ATM multiplexer performance by simulation and analysis of a model of packetized voice traffic"*, M.Sc. Thesis, Dublin City University, February 1994.

[TF02] P. Trimintzios, P. Flegkas, G. Pavlou, 'Policy-driven Traffic Engineering for Intra-domain Quality of Service Provisioning", 2002.

[WC96] Z. Wang and J. Crowcroft, "QoS Routing for Supporting Resource Reservation," IEEE JSAC, Sept. 1996. pp. 605–14.

[WJ00] Wang Jianxin, Wang Weiping, Chen Jianer, Chen Songqiao, "A Randomized QoS Routing Algorithm On Networks with Inaccurate Link-State Information", ICCT 2000.

[XL99] X. Xiao, L. M. Ni, "Internet QoS: A big Picture", IEEE Network, March/April 1999.

[YY01] Yong Jin Kim, Yoshiaki Nemoto, "Improved Connection Admission Control by Effective Bandwidthe and Effective Buffer with Multi-class Traffic", IPSJ SIGNotes Distributed Processing System, 2001.