HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Engineering Physics and Mathematics

**Ilmari Juva**

# Analysis of Quality of Service Routing Approaches and Algorithms

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

Espoo, 26.3.2003

Supervisor: Professor Samuli Aalto (pro tem)
Instructor: Professor Samuli Aalto (pro tem)

| | |
|---|---|
| Author: | Ilmari Juva |
| Department: | Engineering Physics and Mathematics |
| Major Subject: | Systems and Operations Research |
| Minor Subject: | Teletraffic Theory |

| | |
|---|---|
| Title: | Analysis of Quality of Service Routing Approaches and Algorithms |
| Title in Finnish: | Palvelunlaatureitityksen eri lähestymistavat ja algoritmit |
| Chair: | S-38 Networking Technology |
| Supervisor: | Professor Samuli Aalto (pro tem) |
| Instructor: | Professor Samuli Aalto (pro tem) |

Abstract:

So far the Internet has offered only best effort service. All traffic is processed as quickly as possible and no preferences are given to any type of traffic. Today there are more and more applications that need service guarantees in order to function properly. These kinds of applications include for instance IP telephony, video-conference applications or video on-demand services.

One important part of the QoS framework is the ability to find the paths that have sufficient resources to support the QoS requirements of a traffic flow. Current Internet routing protocols always forward packets to the shortest path based on hop count. This can cause problems for flows with a need for performance requirements, if the shortest path does not have the resources needed to meet these requirements. Quality of Service routing is a routing scheme that considers the quality of service requirements of a flow when making routing decisions. As opposed to traditional shortest path routing, which only considers the hop count, QoS routing is designed to find feasible paths that satisfy multiple constraints.

This thesis is a survey on QoS routing. It presents the most important problems in QoS routing concerning path selection algorithms, cost of QoS routing, and different approaches. Routing problems for cases with one or two metric are formalized as optimization problems, and solutions algorithms are presented. For the most complex problems heuristic approximation algorithms and their evaluations found in literature are discussed. Algorithms for the important special case where available bandwidth and hop count are used as metrics are considered in detail. The cost of QoS routing, and the factors contributing to the cost, are evaluated based on simulation and implementation study results found in literature. Different approaches concerning algorithm classes, link state information and timing of path computation are discussed.

While providing paths that satisfy the QoS guarantees of traffic flows, QoS routing can cause inter-class effects that may lead to congestion or even starvation of low priority traffic. The authors own contribution consists of a simple Markov-model to study the effects of a bandwidth reservation scheme that sets aside some portion of a link's bandwidth for low priority traffic only, in order to prevent the starvation.

| | |
|---|---|
| Number of pages: 88+9 | Keywords: QoS routing, routing algorithms, inter-class effects |

**Department fills:**

| | |
|---|---|
| Approved: | Library code: |

| | |
|---|---|
| Tekijä: | Ilmari Juva |
| Osasto: | Teknillinen Fysiikka ja Matematiikka |
| Pääaine: | Systeemi- ja operaatiotutkimus |
| Sivuaine: | Teleliikenneteoria |



| | |
|---|---|
| Työn nimi: | Palvelunlaatureitityksen eri lähestymistavat ja algoritmit |
| Title in English: | Analysis of Quality of Service Routing Approaches and Algorithms |
| Professuuri: | S-38 Tietoverkkotekniikka |
| Työn valvoja: | Professori Samuli Aalto (prof ma) |
| Työn ohjaaja: | Professori Samuli Aalto (prof ma) |

Tiivistelmä:

Tähän asti Internet on tarjonnut vain best effort -palvelua. Kaikki liikenne käsitellään niin nopeasti kuin mahdollista, eikä minkään tyyppiselle liikeneteelle anneta etusijaa muihin nähden. Nykyisin yhä useampi sovellus, kuten IP puhelin tai videoneuvottelu, tarvitsee kuitenkin takuita palvelutasosta toimiakseen kunnolla.

Sellaisten polkujen löytäminen, jotka pystyvät täyttämään vaaditut palvelunlaatu-ehdot liikennevoille, on tärkeä osa palvenlaatua. Nykyisessä Internetissä reititysprotokollat reitittävät liikenteen aina lyhimmälle polulle. Tämä saattaa aiheuttaa ongelmia voille, joilla on palvelunlaatuvaatimuksia, joita lyhin polku ei pysty tukemaan. Palvelunlaatureititys sen sijaan ottaa huomioon palvelunlaatuvaatimukset reitityspäätöksissään. Se pystyy myös löytämään useamman ehdon täyttäviä polkuja, toisin kuin perinteinen lyhimmän polun reititys.

Tämä dimplomityö on kirjallisuuskatsaus palvelunlaatureityksestä. Se esittelee alueen tärkeimmät ongelmat polun valintaan, kustannuksiin ja eri lähestymistapoihin liittyen. Yhden ja kahden metriikan reititysongelmat ja niiden ratkaisualgoritmit esitellään. Kompleksisimmille ongelmille käydään läpi kirjallisuudessa esitettyjä heuristisia algoritmeja sekä niiden arviointeja. Tarkemmin keskitytään tärkeään erikoistapaukseen, jossa metriikkana käytetään polun pituutta ja vapaata kaistanleveyttä. Palvelunlaatureitityksen kustannuksia, ja niihin vaikuttavia tekijöitä, arvioivia simulointi- ja implementaatiotutkimuksia tarkastellaan, kuten myös eri algortimiluokkia, linkkitila informaatiota ja polun laskennan ajoitusta.

Palvelunlaatuvaatimukset täyttävän polkujen valitseminen saattaa aiheuttaa liikenneluokkien välisiä vaikutuksia. Tällaiset inter-class effect -nimellä kutsutut vaikutukset saattavat johtaa alemman prioriteetin liikenteen ruuhkautumiseen tai täydelliseen estymiseen. Tekijän oma osuus käsittelee yksinkertaista Markov-mallia, jolla tutkitaan sellaisen kaistanvarausmallin vaikutusta, joka varaa tietyn osan kaistanleveydestä yksinomaan alemman luokan liikenteen käyttöön.



| | |
|---|---|
| Sivumäärä: 88+9 | Avainsanat: Palvelunlaatureititys, reititysalgoritmit, luokkien väliset vaikutukset |
| **Täytetään osastolla**: | |
| Hyväksytty: | Kirjasto: |

# Preface

This thesis was carried out at the Networking Laboratory at Helsinki University of Technology as part of the IRoNet project.

I would like to thank professor Samuli Aalto for his invaluable guidance and tireless work as both instructor and supervisor of the thesis, professor Jorma Virtamo for his valuable involvement and constructive suggestions, and Jouni Karvo for his insightful comments.

I would also like to thank everybody at the Networking Laboratory for a great working atmosphere.

Espoo, 26.3.2003,

Ilmari Juva

# Contents

# Acronyms

- **BFS** Breadth First Search
- **bsp** Bandwidth-inversion shortest path algorithm, page 31
- **dap** Dynamic-alternative path algorithm, page 32
- **DiffServ** Differentiated Services
- **ebsp** Enhanced bandwidth-inversion shortest path algorithm, page 31
- **H_MCOP** Heuristic Multi-Constrained Optimal Path routing algorithm, page 27
- **IETF** The Internet Engineering Task Force
- **IntServ** Integrated Services
- **IP** Internet Protocol
- **isp** Internet Service Provider
- **LSA** Link State Advertisement
- **LSP** Label Switched Path
- **MCOP** Multi-Constrained Optimal Path routing problem
- **MCP** Multi-Constrained Path routing problem
- **MIRA** Minimum Interference Routing Algorithm
- **MPLS** Multi-Path Label Switching
- **NP** Non-deterministic Polynomial complexity class

- **OPR** Optimal Premium class Routing
- **OSPF** Open Shortest Path First
- **P** Polynomial complexity class
- **PHB** Per-Hop-Behavior
- **QoS** Quality of Service
- **QOSPF** Quality of Service extensions to OSPF
- **RSVP** Resource ReSerVation Protocol
- **SAMCRA** Self-Adaptive Multiple Constraints Routing Algorithm, page 25
- **sdp** Shortest-distance path algorithm, page 31
- **swp** Shortest-widest path algorithm, page 30
- **TAMCRA** Tunable Accuracy Multiple Constraints Routing Algorithm, page 24
- **wsp** Widest-shortest path algorithm, page 29

# Chapter 1

# Introduction

## 1.1 Background

So far the Internet has offered only best effort service. All traffic is processed as quickly as possible and no preferences are given to any type of traffic. Today there are more and more applications that need service guarantees in order to function properly. These kinds of applications include for instance IP telephony, video-conference applications or video on-demand services. Even though there has been some debate whether bandwidth will come so cheap in the future that there would be no need to these kinds of guarantees, it is reasonable to assume that no matter how abundant bandwidth will come, new applications are going to emerge that will consume it [50]. So some kind of system is needed to provide these kinds of guarantees.

## 1.2 Quality of Service

Quality of Service, or QoS, covers several mechanisms that were designed to support flows that require some performance guarantees.

**Integrated Services,** or IntServ, was the first effort to provide QoS in the Internet. According to [22], the steps to end-to-end QoS support over the Internet are

as follows

1. Define the service class a packet should receive at each switch.
2. Allocate to each class a certain amount of resources.
3. Sort the incoming packets to their respective classes.
4. Control the amount of traffic admitted for each class.
5. Apply the four steps above to each and every switch, or at least all bottleneck routers.

This was the definition for the IntServ, which proposed two other service classes in addition to best effort service: Guaranteed Service and Controlled Load Service. A signalling protocol RSVP [11] is used for reserving resources. However, there were concerns about IntServ's complexity and scalability. Also for IntServ to work all the routers in a path must support it. Otherwise end-to-end guarantees cannot be provided. This meant that no transition period was possible but all the routers should have been switched to supporting IntServ at the same time. All these reasons made adopting IntServ difficult and it was never really adopted.

**Differentiated Services,** or DiffServ, dropped the last two items of the IntServ requirements, and concentrated on the first three. It separates the two sides of IntServ, providing forwarding on per-hop behavior (PHB) basis with queue management and queue service disciplines, but leaves the admission control and end-to-end concerns outside its scope. Other mechanisms working together with DiffServ can be used to provide them. DiffServ is actually more a building block than a complete solution for providing Quality of Service [22]. But it does solve the scalability concerns of IntServ.

**MPLS** or multiprotocol label switching [41] is a forwarding scheme where packets are routed based on a short label which makes forwarding faster than when dealing with IP addresses, and allows policy routing within a MPLS-capable domain. MPLS can be used together with differentiated services to provide QoS.

**QoS routing** is a routing scheme that takes into consideration the available bandwidth and other relevant information about each link, and based on that infor-

mation selects paths that satisfy the quality of service requirements of a traffic flow.

An overview of mechanisms used to provide Quality of Service in the Internet can be found in [50].

## 1.3 Purpose and scope of the thesis

This thesis is a survey on QoS routing. It presents the most important problems in QoS routing concerning both path selection algorithms and cost of a QoS routing protocol. The problems and different strategies and approaches are presented in a systematic way to give an overview of the issues involved. The thesis discusses solutions and tools for solving the problems, reviews simulation and implementation study results, and draws conclusions from those results.

## 1.4 Structure of the thesis

The structure of the rest of the thesis is as follows: Chapter 2 introduces QoS routing, its objectives and the position of QoS routing compared to other quality of service concepts.

Chapter 3 discusses the QoS algorithms that select the paths for traffic requests based on their requirements, or constraints. First the metrics used are introduced. The chapter then presents the different routing problems with one or several metrics. For the most complex problems, heuristic algorithms are discussed. The most common situation, where available bandwidth and hop count are used as metrics, is considered in detail, and an important special case of end-to-end delay constraints is also discussed.

Chapter 4 concentrates on the different approaches, other than just the path selection algorithms, used in QoS routing. Routing algorithms are divided into classes, based on whether the entire path is explicitly computed in the source node or the computation is distributed so that each node computes the next hop. Also the computation

can be handled either by periodically pre-computing all paths, so that when a request arrives, the path is already computed, or the path can be computed on-demand for each request as they arrive. Finally the chapter introduces the extended role of link state information and its distribution in QoS routing.

Chapter 5 introduces the QoS routing protocol Quality of service extensions for OSPF protocol, or QOSPF. This protocol is being standardized by IETF, and is a likely candidate for first implementation of a QoS routing scheme.

While using QoS routing improves the quality of service in the network, it also introduces problems that are different from those in best effort routing. Chapter 6 discusses these problems. In best effort routing the link state information about the topology does not change rapidly, while in QoS routing quantities like available bandwidth can change dramatically between link state updates. This leads to a situation where routing choices are made under inaccurate information. The impact that using QoS routing to give guaranteed service to portion of traffic has on lower priority traffic is also studied.

Chapter 7 surveys various simulation results. The cost of QoS routing, and the factors contributing to the cost, are discussed. Also the performance of routing algorithms and heuristic algorithms presented in chapter 3 is evaluated.

Chapter 8 includes the own contributions. First, the impact of QoS guaranteed traffic on lower priority traffic is studied. One technique to prevent the starvation of lower priority traffic is resource reservation. Chapter 8 formulates a Markov-model to study the effects that a resource reservation scheme would have on blocking probability of guaranteed traffic and available bandwidth for lower priority traffic. Second, the *ebsp* routing algorithm performs well in simulation studies, but has some irregular properties. This is shown by an example.

Finally, chapter 9 summarizes the thesis and discusses possible directions of further work.

# Chapter 2

# QoS routing

## 2.1 Introduction

OSPF and other dynamic routing protocols always forward packets to the shortest path. This can cause problems for flows with a need for QoS guarantees if the shortest path does not have enough resources to meet the requirements. IntServ is supposed to reserve resources for the flow, but cannot make the reservation if there are not sufficient resources along the path to begin with. DiffServ is also better utilized if the path with the best chance to provide the required service is somehow found. The missing piece in the framework therefore seems to be a mechanism that can find a path, if one exists, which has the requested resources available. Only then it is possible to utilize DiffServ or IntServ techniques efficiently.

QoS routing is a routing scheme that considers the quality of service requirements of a flow when making routing decisions. As opposed to traditional shortest path routing, which only considers the hop count, QoS routing is designed to find feasible paths that satisfy multiple constraints. QoS routing is a routing scheme, under which

> ”paths for flows would be determined based on some knowledge of resource availability in the network as well as the QoS requirements of the flow.” *Crawley et al.* [17]

This chapter first introduces the concepts of traffic engineering and constraint-based

routing. Then the objectives of QoS routing are presented. The rest of the chapter discusses QoS routing's position in the QoS framework relative to other QoS related mechanisms.

## 2.2 Traffic engineering and constraint-based routing

Shortest path routing leads to uneven traffic distribution. This can cause congestion in some parts of the network even if traffic load is not particularly heavy. While QoS schemes try to provide better service under congestion for flows with QoS requirements, it would be even more desirable to avoid these situations altogether. Traffic engineering is the process of arranging how traffic flows through the network, so that congestion caused by uneven network utilization can be avoided [50].

Constraint-based routing evolves from QoS routing. Although the terms are sometimes used almost interchangeably, constraint-based routing is actually a more general term, which combines QoS routing and policy routing. It extends the QoS routing scheme by considering, in addition to QoS requirements, other constraints such as network policies and also utilization of the network to prevent situations of uneven load.

## 2.3 QoS routing objectives

QoS routing uses information about network state and resource availability as well as the QoS requirements of the flow to make routing decisions. The objectives are threefold [17]:

**Dynamic determination of feasible paths.** That is, to find a feasible path for the flow in question that can accommodate or at least has a good chance of accommodating the QoS requirements of the flow.

**Optimization of resource usage.** QoS-based routing can be used to help balancing the load of the network by efficient utilization of resources, and thus improving the total throughput of the network

**Graceful performance degradation.** In overload situations QoS routing should be able to provide better throughput in the network than best effort routing or any state-insensitive routing scheme, and more graceful performance degradation.

## 2.4 QoS routing's position in the QoS framework

This section discusses the relationships between the QoS routing and other QoS related mechanisms. The discussion follows similar sections in [50] and [43]. The relative position of the different components in QoS framework is shown in table 2.1.

<table>
<tr><td>Application Layer</td><td></td></tr>
<tr><td>Transport Layer</td><td>IntServ/RSVP , DiffServ</td></tr>
<tr><td rowspan="2">Network Layer</td><td>Constraint Based Routing</td></tr>
<tr><td rowspan="2">MPLS</td></tr>
<tr><td rowspan="2">Link Layer</td></tr>
<tr><td></td></tr>
</table>

Table 2.1: The relative position of the components in the QoS framework [50].

### 2.4.1 DiffServ and QoS routing

Originally, the DiffServ scheme is intentionally decoupled from IP routing, so all traffic between a source-destination pair may follow the same path no matter which service class it belongs to, and DiffServ itself has no effect on routing decisions [47]. This means the DiffServ domain is vulnerable to congestion.

For instance, aggregation of premium traffic in the core of the network could cause congestion. This is not a problem when traffic from boundary routers aggregate to

a core router, since the link from the core router to the next core router is faster than the links from the boundary routers. If, however, among the core routers traffic is routed so that it aggregates into one router, the link to which that router forwards the traffic may not be fast enough. This problem cannot be solved by DiffServ. QoS routing could be used to avoid this kind of a situation. Within a DiffServ domain QoS routing is used for finding paths that are able to accommodate the flows and prevent congestion.

Another possible problem situation happens if premium traffic is not routed optimally considering lower priority traffic. Due to high priority of premium traffic, this could lead to some problems for the low-priority traffic when the volume of premium class traffic is high [47]. QoS routing schemes could be used for balancing the load in the DiffServ domain, so that not just premium traffic but also lower class traffic gets better service. Specific routing algorithms have been proposed to route high priority traffic in a way that also considers the performance of other traffic classes.

### 2.4.2 MPLS and QoS routing

Since MPLS is a forwarding scheme and QoS routing is a routing scheme, they can be used together for traffic engineering purposes. In fact, MPLS can provide more accurate information about traffic loads in the domain than traditional IP routing, thus enabling QoS routing to compute better routes for setting up the label switched paths. Furthermore it is relatively easy to integrate a QoS routing framework with MPLS [10]. Constraint-based routing is among the three most significant problem areas in MPLS resource optimization, along with traffic partitioning and restoration [9].

An MPLS *traffic trunk* is an aggregate of flows that belong to the same class, for example all the traffic between specific ingress and egress routers. Traffic trunks are routable objects [10]. The aim of QoS routing in the MPLS network is to route the traffic trunks along the network in a way that satisfies the given constraints, and establish a more balanced traffic load distribution. It is also possible to reroute existing label switched paths to prevent congestion [9].

Based on information about the traffic trunks, network topology and resources, QoS routing computes explicit routes for each traffic trunk. The explicit route in this case is a specification of a label switched path, LSP, satisfying the requirements of the traffic trunk [10]. Given the routes, MPLS sets up the LSP's using its label distribution protocol. It makes no difference to MPLS whether the routes are computed by QoS routing or traditional dynamic routing, where paths are selected based on some dynamic criteria, available bandwidth perhaps, but QoS requirements of the flows are not considered.

The problem of routing the traffic trunks is generally NP-complete [9], so heuristic path selection algorithms have been proposed, such as the Minimum Interference Routing Algorithm (MIRA) by Kar et al. [26].

### 2.4.3 QoS routing with resource reservation

Resource reservation and QoS routing are independent mechanisms but complement each other well. QoS routing can find feasible paths for flows that need QoS guarantees but cannot ensure that the path will remain feasible for the duration of the flow. Resource reservation protocols can be used to allocate the required resources along the selected path.

#### RSVP

The protocol most often suggested in papers concerning QoS routing and resource reservation is RSVP. It is receiver oriented, which means that the receiver of the data flow is responsible for initiation of resource reservation. When the source node initiates a flow, it sends a PATH message to the destination node identifying the characteristics of the flow for which resources are requested. Intermediate nodes forward the PATH message according to routing protocol in question. After receiving the PATH message, the destination node sends back a RESV message to do the actual reservation. Intermediate nodes decide separately whether they can accommodate the request. If any of them rejects the reservation, an error message is sent to the receiver.

If the reservation is successful, necessary bandwidth and buffer space is allocated. After the connection and reservation is established the source periodically sends PATH messages to establish or update the path state, and the receiver periodically sends RESV message to establish or update the reservation state. Without update messages the reservation times out. This is called *Soft-State*.

When RSVP is used together with QoS routing, the PATH messages are routed using QoS routing. The RESV messages and the actual reservation on resources is not affected by the routing protocol. Due to the more dynamic nature of QoS path selection criteria, better routes can emerge more easily than in shortest path routing. That is, available bandwidth or other metrics can change rapidly, so that the current selected path is no longer the one with the best capabilities to accommodate the flow. Such sudden changes rarely happen in shortest path routing, since network topology usually stays more or less the same.

If QoS routing queries new routes, it may lead to a situation where the path is constantly changing and the reservation has to be made again and again for the new path. To avoid these kinds of oscillations between paths, it can be specified that the current path is not changed for a better one as long as it remains feasible. The path is called *pinned* if it is specified in the RSVP protocol that QoS routing need not to be queried anew, otherwise the path is called *unpinned*. In the latter case the path is to be abandoned in favor of a better path, should one emerge [19].

The path pinning uses the RSVP Soft-State mechanism, so a pinned path has to be established periodically. When a path is pinned the periodical PATH messages are routed along the pinned path. The pinning ensures that whenever RSVP queries QoS routing for the same flow, it returns the pinned path instead of QoS routing computing the current best path [21].

Paths get pinned during processing of PATH messages. They get unpinned when

- A time-out occurs or a PATHTEAR message is sent.
- The parameters of the PATH message change.
- A failure is detected or an error message is received.

Some modifications have to be made to the existing RSVP processing rules. These

are discussed in [19].

### 2.4.4 QoS routing with admission control

One of the goals for QoS routing is better network utilization. This is somewhat contradicted by the primary goal of finding alternate paths that can accommodate the QoS requirements of a flow. Under heavy traffic load the only paths able to provide the requested QoS guarantees may be so much longer than the shortest path, that when traffic is routed along the alternate path, a flow's contribution to the congestion of the network leads to more flows to be blocked in future. This calls for the use of *higher level admission control* [4] to ensure that the path selected does not use so much of the network's resources that the total throughput declines.

For each link the following fraction is calculated, when there is an attempt for reservation:

$$\frac{b_i^{\text{available}} - b_{\text{requested}}}{b_i^{\text{capacity}}}.$$

Here $b_i^{available}$ is the available bandwidth on the link $i$ where the resources are attempted to be reserved, $b_{requested}$ is the bandwidth requirement of the flow and $b_i^{capacity}$ is the total capacity of link $i$. The reservation is allowed only if this fraction, representing the available bandwidth on the link if the reservation is accepted, is larger than a predetermined trunk reservation, or bandwidth reservation, level [4]. The longer the path used, the higher the bandwidth reservation level. For example the authors in [4] use a value of 5% for paths one hop longer than the shortest path, 10% for paths two hops longer, and 20% for paths more than two hops longer.

# Chapter 3

# Routing algorithms

## 3.1 Introduction

This chapter discusses the QoS routing problem and algorithms used to solve them. Routing problems using one or two metrics are presented, and their complexity is discussed. The chapter presents solutions for simple problems and heuristic approaches for the more complex ones. The solutions are based on the shortest path algorithms, namely Dijkstra's algorithms and the Bellman-Ford algorithm, which are presented in appendix A. Section 3.6 discusses the most common situation, where bandwidth and hop count are the metrics. Finally, section 3.7 presents a special case of end-to-end delay constraint. It has a single metric, which is a complicated function of several elements.

## 3.2 Notation

The network can be modelled as a graph $G(N, A)$ where $N(G)$ is the set of nodes in the graph, and $A(G)$ is the set of arcs that represent the links of the network. Let $n$ and $m$ denote the number of nodes and links in the network respectively.

Link $a \in A$ from node $u$ to node $v$ is noted by $(u, v)$. Each link $a \in A$ has a weight $w_i(a)$ for all the metrics $i$. Let $w(u, v)$ be a weight corresponding to link $(u, v)$ on

path $P = (u_1, u_2, u_3, \ldots, u_l)$, and let $w(P)$ be the weight for the whole path.

## 3.3 Metrics

In order to find a feasible path that satisfies the quality of service requirements of a flow, there has to be some suitable metrics for measuring the requirements. The metrics have to be selected so that the requirements can be presented by one metric or a reasonable combination of them. As the metrics define the types of QoS guarantees the network is able to support, no requirement can be supported if it cannot be mapped onto a combination of the selected metrics [17]. The metrics commonly used on QoS routing and constraint-based routing are divided into three categories, also called the composition rules of the metrics [46].

The metric is

**Additive** if $w(P) = w(u_1, u_2) + w(u_2, u_3) + \ldots + w(u_{l-1}, u_l)$

**Multiplicative** if $w(P) = w(u_1, u_2) \cdot w(u_2, u_3) \cdot \ldots \cdot w(u_{l-1}, u_l)$

**Concave** if $w(P) = \min(w(u_1, u_2), w(u_2, u_3), \ldots, w(u_{l-1}, u_l))$.

Additive metrics include delay, delay jitter, cost and hop count. Reliability, defined as $(1-$loss rate$)$, is multiplicative while bandwidth, by far the most used metric, is concave. Multiplicative metrics can be handled as additive metrics by substituting the link weights $w_i$ and constraint $C$ by their logarithms $\log w_i$ and $\log C$.

## 3.4 Routing problems

### 3.4.1 Single metric routing problems

In the simplest case the QoS requirements of a flow are well presented by one of the metrics presented in section 3.3. The problem is either an optimization problem, or a constraint problem. The metrics are divided into path-constrained and link-constrained metrics. Concave metrics are link-constrained, because the metric for a

path depends on the bottleneck link's value. Additive and multiplicative metrics are path-constrained, because the metric for a path depends on all the values along the path. The four single metric problems are as follows:

**Problem 1 (link-optimization routing)** *Given a network $G(N, A)$ and a single concave metric $w(a)$ for each link $a \in A$, find the path $P$ from source node $s$ to destination node $t$ that maximizes $w(P)$.*

The link-optimization routing problem can be solved by a modified Dijkstra's algorithm or Bellman-Ford algorithm [16]. Dijkstra's algorithmth (see Appendix A) is modified by just changing the criteria which selects the next node added to set $M$, so that the next node to be added is the node that is connected to set $M$ with largest bandwidth, as in standard Dijkstra's algorithm the next node was selected based on the cumulative cost function.

For instance finding the path with most available bandwidth is a link-optimization problem. Bandwidth's concave nature as a metric makes it a bottleneck optimization, and thus link-optimization problem.

**Problem 2 (link-constrained routing)** *Given a network $G(N, A)$, a single concave metric $w(a)$ for each link $a \in A$ and a requested constraint $C$, find a path $P$ from source node $s$ to destination node $t$ such that $w(P) \geq C$.*

The link-constrained routing problem can be reduced to the link-optimization routing problem by finding the optimal path and checking whether the constraint is met. Another approach is to prune the topology by deleting links with bandwidth less than $C$, and then find the shortest path in the pruned topology. So, link-constrained routing finds a path that satisfies, but does not necessarily optimize, the required quality of service for a link constrained metric. An example is finding a path with required bandwidth.

**Problem 3 (path-optimization routing)** *Given a network $G(N, A)$ and a single additive metric $w(a)$ for each link $a \in A$, find the path $P$ from source node $s$ to destination node $t$ that minimizes $w(P)$.*

The path-optimization routing problem can be directly solved by Dijkstra's algorithm or Bellman-Ford algorithm [16]. Path-optimization routing finds the optimal path for a path-constrained metric. A typical problem could be finding the path with the least number of hops, the least total cost or the smallest delay, all of which are path-constrained metrics because they are additive.

**Problem 4 (path-constrained routing)** *Given a network $G(N, A)$, a single additive metric $w(a)$ for each link $a \in A$ and a requested constraint $C$, find a path $P$ from source node $s$ to destination node $t$ such that $w(P) \leq C$.*

The path-constraint routing problem can be directly solved by Dijkstra's algorithm or the Bellman-Ford algorithm [16]. Path-constrained routing finds a path with, for example, the delay or cost below a requested level.

All four of the above problems are of polynomial complexity. See Appendix B.

### 3.4.2 Routing problems with several metrics

Often some combination of metrics is needed to describe the required service. However, several of the combinations are computationally so complex that they are impractical to use. Any combination of two or more metrics that are either additive or multiplicative is NP-complete [18]. The only combinations that allow path computation with polynomial complexity are those that have a concave metric like bandwidth together with one other metric, most often the delay or hop count.

Some efforts have been made to simplify the situation. In [46], Wang and Crowcroft propose a single mixed metric that would combine all the desirable metrics to an expression used as the single metric. They conclude that it can be used only as an indicator, since it does not contain all the information needed to decide whether a path can meet the QoS requirements. However, bandwidth could be used as a single metric on some occasions. While a connection may have several QoS requirements, it turns out that these translate mainly into bandwidth requirements [40]. Guerin et al. propose an equation for mapping delay constraints onto bandwidth constraints [21].

Table 3.1 shows the composite problems with two metrics derived from the four single metric problems described. Double optimization problems are omitted as not reasonable. Of course, double optimizations could be done sequentially, but that would correspond to reducing the composite problems into two single metric problems, where the second optimization would be used only if there are more than one optimal path with regard to the first optimization. This approach is used in algorithms like $wsp$ and $swp$, which are presented in section 3.6.

| | Link-optimization | Link-constraint | Path-optimization | Path-constraint |
|---|---|---|---|---|
| Link-optimization | - | polynomial | - | polynomial |
| Link-constraint | | polynomial | polynomial | polynomial |
| Path-optimization | | | - | NP-complete |
| Path-constraint | | | | NP-complete |

Table 3.1: Computational complexity of metric combinations

### Pruning

An important technique in solving the composite problems is pruning the network. In the case of link constraints, the value of the metric on the path is always the same as on the link having the worst value. Hence, a link that does not have the requested resources, available bandwidth for example, is not feasible. These links are deleted from the topology. This will guarantee that any path found on the pruned topology satisfies the link constraint in question.

### Composite routing problems

**Composite Problem 5 (link-constrained link-optimization routing)** *Given a network $G(N, A)$, concave metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$ and a requested constraint $C_1$, find the path $P$ from source node $s$ to destination node $t$ that maximizes $w_2(P)$, while $w_1(P) \geq C_1$.*

The link-constrained link-optimization routing problem can be reduced to problem 1, the link-optimization routing problem, by first pruning the network, deleting all the links for which the constraint $w_1(P) \geq C_1$ does not hold. The pruning operation's complexity is proportional to $m$, the amount of links in the network, so the link-constrained link-optimization problem is of polynomial complexity.

**Composite Problem 6 (multi-link-constrained routing)** *Given a network $G(N, A)$, concave metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$ and requested constraints $C_i$, $i = 1, 2$, find a path $P$ from source node $s$ to destination node $t$ such that $w_i(P) \geq C_i$ for all $i$.*

As problem 5 was reduced to problem 1 by pruning the network, similarly the multi-link-constrained problem can be reduced to problem 2, the link-constrained routing problem.

**Composite Problem 7 (link-constrained path-optimization routing)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, of which $w_1$ is concave, for each link $a \in A$ and a requested constraint $C_1$, find the path $P$ from source node $s$ to destination node $t$ that minimizes $w_2(P)$, while $w_1(P) \geq C_1$.*

The link-constrained path optimization routing problem can be reduced by pruning to problem 3, the path-optimization routing problem.

**Composite Problem 8 (link-constrained path-constrained routing)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, of which $w_1$ is concave, for each link $a \in A$ and requested constraints $C_i$, $i = 1, 2$, find a path $P$ from source node $s$ to destination node $t$, such that $w_1(P) \geq C_1$ and $w_2(P) \leq C_2$.*

The link-constrained path constrained routing problem can be reduced by pruning to problem 4, the path-constrained routing problem.

The four composite problems above are easily solved because one of the metrics is link-constrained. Pruning the network by disregarding all the links with insufficient values takes care of the constraint, and reduces the composite problem to a single

metric problem on a subset graph. As shown in table 3.1, besides the four link-constraint based composite problems there is one additional composite problem solvable in polynomial time.

**Composite Problem 9 (link-optimization path-constrained routing)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, of which $w_1$ is concave, for each link $a \in A$, and a requested constraint $C_2$, find the path $P$ from source node $s$ to destination node $t$ that maximizes $w_1(P)$, while $w_2(P) \leq C_2$.*

This problem is solvable in polynomial time by a modified shortest path algorithm [16]. In the network, there are $m$ links, which all have some value for the link-optimization metric. Let $K$ denote the number of different values for the metric. Clearly $K \leq m$, because some links may have identical values. These values can be used as artificial lower limits $C_1^1, C_1^2, \ldots, C_1^K$ for the link constrained metric to be optimized. This reduces the problem to several link-constrained path-constrained routing problems, problem 8 above. Starting from the largest value $C_1^K$ and pruning the network by deleting all links $a$ for which $w_1(a) < C_1^K$, the optimal path $P$ is the first feasible path found such that $w_2(P) \leq C_2$.

If the link constrained metric $w_1$ is bandwidth, this means that first all links but those with the highest available bandwidth $C_1^K$ are deleted. If a path satisfying the path constraint can be found in this pruned topology, it is the one with maximal bandwidth. If such path is not found, the links with bandwidth $C_1^{K-1}$ are added to the topology, and the search is repeated. If no feasible path is found, the links with bandwidth $C_1^{K-2}$ are added to the topology. This is continued until a feasible path is found, or all the links are added to the topology and still there is no feasible path, in which case one does not exist.

This problem is more complex than composition problems 5 through 8, but still solvable in polynomial time.

The remaining two composite problems are multi-path-constrained routing problem (MCP), and path-constrained path-optimization routing problem or multi-constrained optimization problem (MCOP). The metrics in these problems are not concave, so the problems are NP-complete. They cannot be solved in polynomial time, and thus heuristic approaches are needed if these metric combinations are to

be used.

**Composite Problem 10 (multi-path-constrained routing)** *Given a network $G(N, A)$, additive metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, and requested constraints $C_i$, $i = 1, 2$, find a path $P$ from source node $s$ to destination node $t$ such that $w_i(P) \leq C_i$ for all $i$.*

The multi-path-constrained routing problem (MCP) is NP-complete. See [46] for proof.

The proof given in [46] starts from the fact that *Partition* is a well known NP-complete problem, and shows that Partition $\propto$ Additive Metrics Problem, to prove the NP-completeness of the latter. More intuitively, it can be seen that all the polynomial problems above were, one way or another, reduced to single metric problems in order to solve them by shortest path algorithms. This was achieved by, for example, pruning the network of links with insufficient capacity, to account for a link-constraint. Here both metrics are path constrained, so that kind of simple pruning is not possible. Thus the problem is not solvable in polynomial time.

**Composite Problem 11 (path-constrained path-optimization routing)** *Given a network $G(N, A)$, additive metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$ and a requested constraint $C_1$ for the metric $w_1$, find the path $P$ from source node $s$ to destination node $t$, that minimizes $w_2(P)$ while $w_1(P) \leq C_1$.*

The path-constrained path-optimization problem is NP-complete [46]. The Multi-constrained optimization problem, MCOP, is sometimes used as a synonym for the path-constrained path-optimization routing, but could also be understood as a multi-constrained problem with a cost function, that may, or may not be one of the path-constrained metrics.

The above reasoning is based on the assumption that the metrics are independent of each other. This is not, however, necessarily true. In certain networks using rate-proportional scheduling algorithms, specifically WFQ-scheduling, the delay/delay-jitter problem can be solved in polynomial time by the Bellman-Ford algorithm [34]. This is based on the fact that in such an environment hop count is the only

parameter that determines if the delay-jitter constraint can be met. So the problem can be solved by solving for the delay constraint and restricting the hop count so that the jitter requirement is also met. Also, if all the metrics except one are allowed to take only bounded integer values, then the problem is solvable in polynomial time [16]. For instance hop count by nature is a metric that is integer-valued and bounded by the diameter of the graph.

## 3.5 Heuristic approaches for the NP-complete composite problems

As said in section 3.4.2, the computational complexity for composite problems MCP and MCOP, problems 10 and 11, is NP-complete. To get solutions for these problems, polynomial time heuristic approximations are needed. The problem is simplified to a problem that is solvable by a shortest path algorithm.

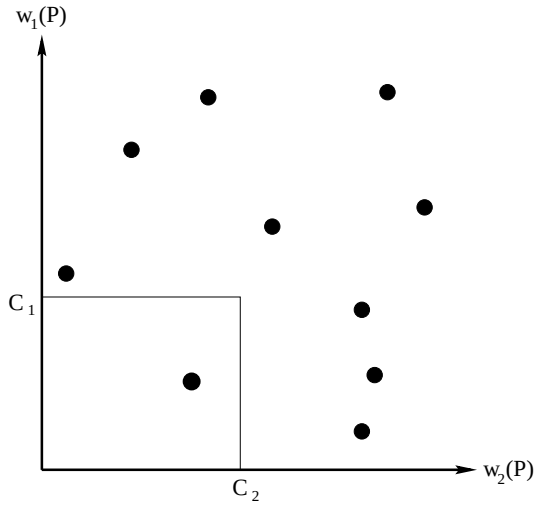



Figure 3.1: Multi-path-constrained routing problem. The feasible area, $w_i(P) \leq C_i$ for each $i$, is represented by the rectangle in the bottom left corner. The black dots represent the paths.

To illustrate the situation, consider figure 3.1. Each path $P$ has some values for every metric $w_i(P)$. The figure shows the situation with two constraints. The paths are plotted on the graph according to the values of metrics.

The following sections review some of the proposed polynomial time heuristic approximations, following similar presentation in [29]. A simulations study of these algorithms is reviewed in chapter 7. The algorithms that are for the optimization problem MCOP can easily solve also the MCP problem by just checking if the optimal path is feasible with regard to the given constraints.

## Chen and Nahrstedt's algorithm

In [15] Chen and Nahrstedt propose the following technique for the MCP problem, where the values of the second metric are limited to bounded integer values, and the problem is thus solvable in polynomial time. Replace the link weight $w_2$ with a new weight function

$$w_2'(u,v) = \left\lceil \frac{w_2(u,v) \cdot x}{C_2} \right\rceil \tag{3.1}$$

and reduce the original problem $MCP(G, s, t, w_1, w_2, C_1, C_2)$ to a simpler problem $MCP(G, s, t, w_1, w_2', C_1, x)$, where $x$ is some predetermined integer. Problem 10 is then

**Heuristic approach 1 (Chen and Nahrstedt approximation)** *Given a network $G(N, A)$, metrics $w_1(a)$ and $w_2'(a)$ for each link $a \in A$, and requested constraints $C_1$ and $x$, find a path $P$ from source node $s$ to destination node $t$, so that $w_1(P) \leq C_1$ and $w_2'(P) \leq x$.*

Chen and Nahrstedt propose extensions to Dijkstra's algorithm and the Bellman-Ford algorithm to solve the problem.

**Proposition 1** *A solution for the heuristic approximation problem is also a solution for the original problem.*

Proof: From equation 3.1 it follows that

$$w_2'(u,v) \geq \frac{w_2(u,v) \cdot x}{C_2}.$$

Thus,

$$w_2(u,v) \leq \frac{w_2'(u,v) \cdot C_2}{x}.$$

Using that and the fact that since path $P$ is a solution for the simplified problem, the equation

$$w_2'(P) \leq x \tag{3.2}$$

must hold, it is easy to show that $P$ is also a solution for the original problem.

$$w_2(P) = \sum_{(u,v)\in P} w_2(u,v) \leq \sum_{(u,v)\in P} \frac{w_2'(u,v) \cdot C_2}{x}$$

$$= \frac{C_2}{x} \cdot \sum_{(u,v)\in P} w_2'(u,v) = \frac{C_2}{x} \cdot w_2'(P) \leq \ = \frac{C_2}{x} \cdot x = C_2$$

So $w_2(P) \leq C_2$. □

A solution for the original problem is not necessary a solution for the simplified problem. The rounding in (3.1) makes (3.2) stricter than the original condition. For example, if a path $P$ with three hops includes links with weights $4$, $5$ and $6$, it is a solution for the original problem when $C_2 = 15$. Selecting $x = 5$, (3.1) yields the new weights $2$, $2$ and $2$. Summing these we get the path weight $w'(P) = 6$ and $w'(P) \leq x$ does not hold. If path $P$, with length $l(P)$ is a solution for the original problem, a sufficient condition for it to also be a solution for the simplified problem is

$$w_i(P) \leq (1 - \frac{l(P) - 1}{x}) \cdot C_i. \tag{3.3}$$

This means that the path needs to be overqualified by a coefficient depending on its length $l(P)$ and the value of $x$. Condition 3.3 is sufficient but not necessary. Had the weights in the example above been $6$, $6$ and $3$, the new weights would be $2$, $2$ and $1$ respectively, and the condition $w'(P) \leq x$ holds, although (3.3) does not hold.

The heuristic approach can be applied for either metric, so if there is no solution for the simplified problem $MCP(G, s, t, w_1, w_2', C_1, x)$, the other problem $MCP(G, s, t, w_1', w_2, x, C_2)$ should be tried next. So if an solution for the original problem exists and (3.3) holds for one of the metrics, the solution is found by the heuristic approach.

## Jaffe's algorithm

Jaffe [25] proposes a non-linear path length function

$$f(P) = \max\{w_1(P), C_1\} + \max\{w_2(P), C_2\}, \tag{3.4}$$

whose minimization guarantees to solve the MCP problem by finding a feasible path, if one exists. This is not, however, solvable by a shortest path algorithm, since the length function is non-linear [29]. So he presents an approximation algorithm, that uses a linear combination of the weights:

$$w(u, v) = d_1 \cdot w_1(u, v) + d_2 \cdot w_2(u, v). \tag{3.5}$$

The problem with the combined weight as a metric is solvable by a shortest path algorithm. Then the solution is checked for the constraints.

**Heuristic approach 2 (Jaffe's approximation)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, and requested constraints $C_i$ and positive integers $d_i$, $i = 1, 2$, calculate a new weight function $w(u, v) = d_1 \cdot w_1(u, v) + d_2 \cdot w_2(u, v)$ and find the path $P$ from source node $s$ to destination node $t$ that minimizes $w(P)$. Check if the solution path $P$ satisfies the constraints $w_i \leq C_i$ for each $i$.*

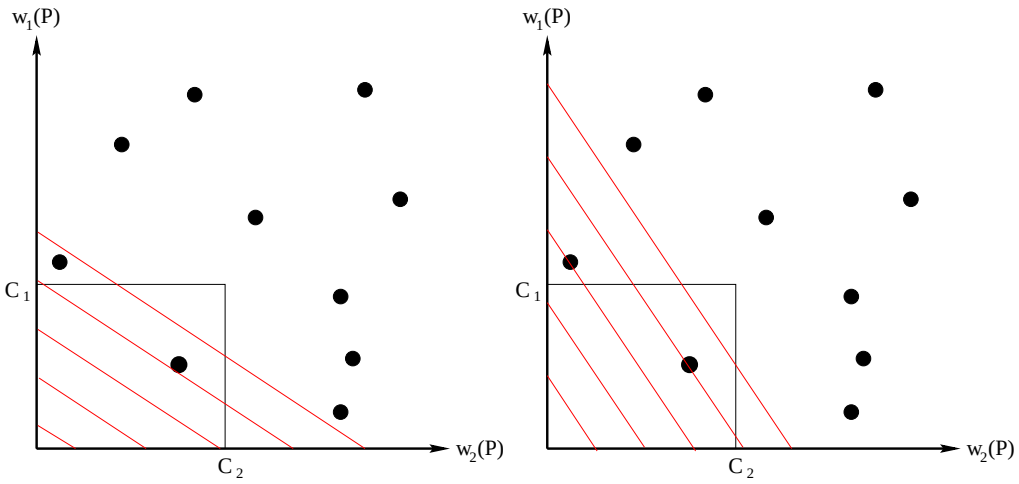



Figure 3.2: Jaffe's algorithms's equivalence lines. The feasible solution could be found with only specific values of $d_i$.

Figure 3.2 shows the equivalence curves of the combined weight function. Even if a feasible solution exists, the algorithm might not found it, as is the case in the

right. If the solution that the algorithm finds is not feasible, it may be possible, by changing the values of $d_i$, to find the feasible solution. But because of the linear equivalence curves, a situation where a feasible solution exists, but is not found with any values of $d_i$, can occur.

## Tunable Accuracy Multiple Constraints Routing Algorithm

The TAMCRA algorithm [38], for the MCP problem, uses, instead of the linear combination weight, a non-linear function that leads to curved equivalence lines. The combined path weight is

$$w(P) = \left( \left[ \frac{w_1(P)}{C_1} \right]^q + \left[ \frac{w_2(P)}{C_2} \right]^q \right)^{\frac{1}{q}}. \tag{3.6}$$

When $q \to \infty$, equation 3.6 becomes

$$w(P) = \max_i \left( \frac{w_i(P)}{C_i} \right), \tag{3.7}$$

which guarantees that all the constraints are met if $w(P) \leq 1$. Even finite size

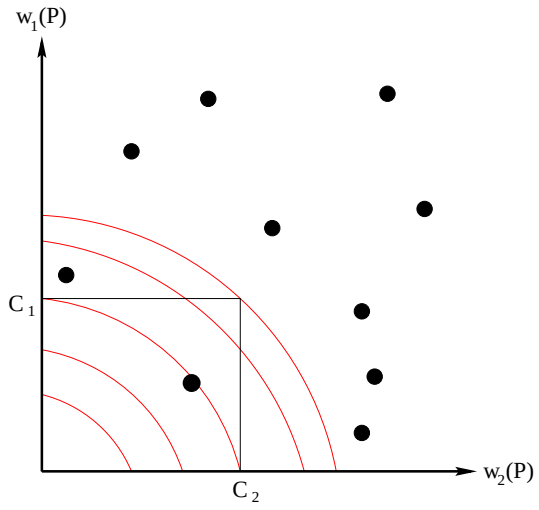



Figure 3.3: Non-linear equivalence curves are more effective.

$q$ values improve the performance of the algorithm. Figure 3.3 shows the same situation as figure 3.2. The non-linear combined metric correctly finds the feasible path.

The problem of the metric in (3.6) is that the metric is not additive, that is, when link weights are calculated, the sum of the link weights is not the weight of the path. Consequently, the TAMCRA uses the $k$-shortest path algorithm. It is essentially a version of Dijkstra's algorithm that does not stop after finding a feasible path to the destination, but goes on to find $k$ different paths [29]. Then for each of these paths, the path weight of equation (3.6) is calculated.

In TAMCRA, $k$ is pre-selected, while another version of the algorithm, Self-adaptive multiple constraints routing algorithm, SAMCRA [45], controls the value of $k$ self-adaptively. This means that TAMCRA is of polynomial complexity, while SAMCRA is an exact algorithm and its complexity is exponential. The selection of $k$ in TAMCRA is a trade-off between performance and complexity. SAMCRA, on the other hand, guarantees to find a feasible path, if one exists.

To reduce the complexity of the algorithm, it considers only non-dominated paths. A path $Q$ is dominated by a path $P$ if $w_i(Q) \leq w_i(P)$, for all $i$, with an inequality for at least one $i$ [29]. This limits the search-space.

**Heuristic approach 3 (TAMCRA)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, requested constraints $C_i$ and positive integers $d_i$, $i = 1, 2$, and $k$, calculate a new weight function defined in equation 3.6, and find $k$ shortest paths from source node $s$ to destination node $t$. For those paths calculate the path weight, and select the path $P$ with minimum $w(P)$. Check if the solution path $P$ satisfies the constraints $w_i \leq C_i$ for each $i$.*

## Iwata's algorithm

Iwata et al. [24] propose a straightforward approach for the MCP problem, where the algorithm finds a shortest path, or paths, based on one metric and then checks if all the constraints are met. If not, it finds a shortest path for another metric and again checks if the other constraints are met. The problem is that the feasible paths are not necessarily shortest for any metric. In figure 3.4, the feasible path is the second shortest regarding weight $1$, and third the third shortest regarfing weight $2$.

**Heuristic approach 4 (Iwata's algorithm)** *Given a network $G(N, A)$, metrics*

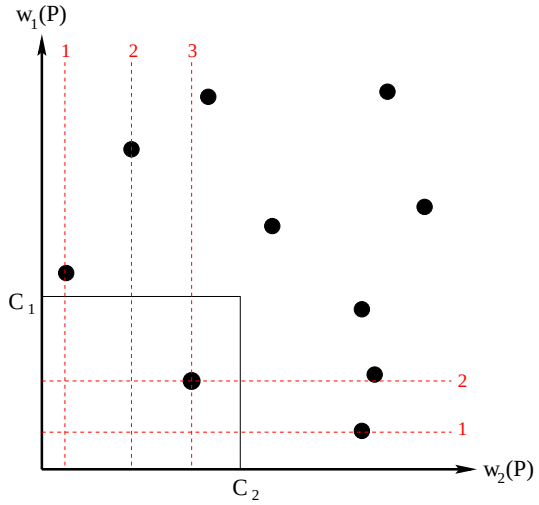



Figure 3.4: Iwata's algorithm's search process

$w_i(a)$, $i = 1, 2$, *for each link* $a \in A$, *and requested constraints* $C_i$, $i = 1, 2$, *find a path* $P$ *that minimizes* $w_i(P)$ *for a specific* $i$, *or several lightest paths for* $w_i(OP)$. *Check if the solution path* $P$ *satisfies the constraints* $w_i \leq C_i$ *for each* $i$. *If not, select the next* $i$ *and start over.*

## Randomized algorithm

The randomized algorithm, proposed by Korkmaz and Krunz [28], is a heuristic algorithm for the MCP problem. It consists of two phases: initialization and randomized search. In the initialization phase, the algorithm computes optimal paths from every node $u$ to destination node $t$ with regard to each metric $w_i$, and with regard to a linear combination metric (3.5).

The information from the initialization is used to determine, whether a feasible path can be found.

In the second phase the algorithm uses a randomized breadth-first search. While the conventional breadth-first search algorithm (BFS) systematically discovers all the nodes, the randomized algorithm selects randomly nodes to be discovered, until the destination is reached. The randomized algorithm also uses the look-ahead property, which means that before discovering a node, it uses the initialization phase

information to see if there is a chance of reaching the destination node $t$ through that node. If the optimal paths from that node to the destination node, with regard to the different metrics, are not within the required constraints separately, of course there is not a path that satisfies them simultaneously. If there is not a good chance of reaching $t$, such node is not included. This method reduces the search-space.

**Heuristic approach 5 (Randomized algorithm)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, requested constraints $C_i$ and weights $d_i$, $i = 1, 2$, first initialize by computing the shortest path from each node $u$ to destination node $t$, with regard to each metric $w_i$ and the linear combination metric (3.5). Using the randomized breadth-first search find a path $P$, from source node $s$ to destination node $t$, such that $w_i(P) \leq C_i$ for all $i$. To reduce the search space, at each node to be discovered, use the initialization information to see if there is a chance of finding a feasible path through that node. If not, exclude the node.*

## H_MCOP

The H_MCOP heuristic algorithm [27] is another approach by Korkmaz and Krunz for the MCOP problem. It returns a feasible path that also minimizes a selected cost function, based on a link cost $c(u, v)$ assigned to each link, which could be one of the QoS constraints $w_i(u, v)$ or some other appropriate cost function. In the first phase the algorithm computes the optimal path from every node $u$ to the destination node $t$ with regard to the linear combination metric of (3.5), setting $d_i = \frac{1}{C_i}$. This first execution returns the optimal path for the linear combination metric.

Then the algorithm uses the non-linear path length function of equation (3.6) to compute paths starting from source node $s$. It discovers each candidate node $u$ based on the minimization of the non-linear length function. For each node $u$, the algorithm selects the shortest path from $s$ to $t$ via $u$, by concatenating the non-linear path length from $s$ to $u$ and the linear path length form $u$ to $t$ to approximate the length of the path from $s$ to $t$. If the paths passing through the candidate nodes $u$, are feasible, the algorithm selects the node that has the path which minimizes the primary cost function. If none of them seem feasible, the algorithm selects the node that has the path which minimizes the non-linear metric, since this path has the best chance of being feasible. Then the algorithm continues from the selected node $u$,

considering the next node $v$ to be discovered in the same way. Eventually it returns the optimal path.

**Heuristic approach 6 (H_MCOP)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, and $c(a)$ for each link $a \in A$, and requested constraints $C_i$, $i = 1, 2$, for the metrics $w_i$, first initialize by finding the shortest path from each node $u$ to destination node $t$ with regard to the linear combination metric*

$$w(u, v) = \frac{w_1(u, v)}{C_1} + \frac{w_2(u, v)}{C_2}.$$

*Then starting from node $s$, discover each node based on the weight of the path $P$ from $s$ to $t$ via $u$. This weight is approximated by concatenating non-linear weight $w(s, u)$ and linear weight $w(u, t)$. If some of the foreseen paths are feasible, select the one minimizing the linear combination weight, otherwise select the one minimizing the non-linear weight. Continue until destination node $t$ is reached, finding the path $P$ that minimizes $c(P)$, while $w_i(P) \leq C_i$ for all $i$.*

## A*Prune

Liu and Ramakrishnan propose the A*Prune algorithm [31] for the MCOP problem, which can find multiple shortest paths. This is an exact algorithm, but uses techniques similar to heuristic algorithms.

It uses a similar initialization phase as the randomized algorithm. The algorithm computes shortest paths for each metric, from every node $u$ to destination node $t$, as well as from source node $s$ to all nodes $u$. After this it proceeds like Dijkstra's algorithm, except the nodes are selected by the predicted end-to-end length of the path, with regard to linear combination metric. All neighbor nodes are considered, and those that cause a loop, or would not meet the constraint, are pruned. The algorithm continues until a pre-determined number of shortest paths ($K$) are found, or there are no more nodes to be extracted from the heap. The algorithm always founds the $K$ shortest paths, or all the feasible paths if there are less than $K$. This suggests that it can not run in polynomial time, and indeed this is the case. A bounded version of the algorithm would run in polynomial time, but would not necessarily find a feasible solution.

**Heuristic approach 7 (A*Prune)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, and requested constraints $C_i$, $i = 1, 2$, first initialize by finding the shortest path from each node $u$ to destination node $t$ and form source node $s$ to each node $u$, with regard to each metric and the linear combination metric (3.5). Then starting from node $s$, discover each node based on the weight of the path $P$ from $s$ to $t$ via $u$ based on the linear weight function. At each step prune all the nodes through which a feasible path can not be found based on the initialization information. Continue until $K$ shortest paths are found, or the heap is empty.*

## 3.6 Bandwidth constraint

In many cases bandwidth is the most important metric, perhaps the only metric used. Even if there are QoS requirements concerning other metrics, they can usually be mapped to bandwidth requirements. Subsequently, many QoS routing schemes consider only bandwidth and hop count. Hop count is additive, but bandwidth is concave, so these routing problems fall into the category of polynomial time composite problems.

The choice of an algorithm is made based on the selection between resource conservation and load balancing in the network. The optimal path with regard to bottleneck bandwidth may not be the best choice, if another path satisfies the requirements and consumes less network resources. Using alternate routes around congested links balances the network utilization, but consumes more resources because longer paths use more links. And the other way round, using the shortest path conserves network resources but may lead to congestion.

### 3.6.1 Basic path selection algorithms

#### wsp

Widest-shortest path [7] selects the minimum hop count path among those that satisfy the bandwidth requirements. If there are several paths with the same hop count, the widest, that is the one with most available bandwidth is selected. Basically it

finds the shortest feasible path, so it is a link constrained path-optimization routing problem (problem 7) and is solvable in polynomial time. The widest path criteria is used only to choose between several paths with the same length as the shortest paths.

**Bandwidth guaranteed algorithm 1 (widest-shortest path)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, where $w_1$ is bandwidth and $w_2$ is hop count, and requested minimum bandwidth $C_1$, prune the network by deleting all the links for which $w_1 < C_1$, find the path $P$ from source node $s$ to destination node $t$ that minimizes hop count $w_2(P)$. If there is multiple paths with the same hop count, select the one that maximizes bandwidth $w_1(P)$.*

After the pruning the first path selection is similar to problem 3 in section 3.4.1. The second phase is used only as a tie-breaker if there are several paths with minimal hop count.

This approach preserves network resources.

### swp

Shortest-widest path [46] selects the path with the largest available bandwidth. If several paths exist with as large bandwidth, the one with the smallest hop count is selected. This approach emphasizes balancing the load in the network. The shortest-widest path algorithm applies Dijkstra's algorithm twice. First for finding the widest path (problem 1). Let us denote the bandwidth of the widest path with $B$. Then after pruning all the links with less bandwidth than $B$, it runs the algorithm for the second time for finding the shortest path among the widest paths (problem 3).

**Bandwidth guaranteed algorithm 2 (shortest-widest path)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, where $w_1$ is bandwidth and $w_2$ is hop count, and requested minimum bandwidth $C_1$, find the path $Q$ from source node $s$ to destination node $t$ that maximizes $w_1$. Let $B = w_1(Q)$ and prune the network by deleting all the links for which $w_1(a) < B$. Then find the path $P$ from source node $s$ to destination node $t$ that minimizes hop count $w_2(P)$.*

## sdp

Shortest-distance path [33], sometimes called the bandwidth-inversion shortest path algorithm *bsp*, selects the path with the shortest distance. The distance of a link is defined as the inverse of the available bandwidth on the link, and the distance of a path is the sum of distances over all the links along the path,

$$\mathrm{dist}(P) = \sum_{a \in P} \frac{1}{w_1(a)}.$$

This is a compromise between the two approaches above. The distance is defined so that it favors shortest paths when load in the network is heavy, and widest paths when load is medium [50]. The shortest-distance path can be found by shortest path algorithms, using the distance as a cost function.

In [36] the *sdp* is extended to the form

$$\mathrm{dist}(P, n) = \sum_{a \in P} \frac{1}{(w_1(a))^n}.$$

**Bandwidth guaranteed algorithm 3 (shortest-distance path)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, where $w_1$ is bandwidth and $w_2$ is hop count, and requested minimum bandwidth $C_1$, let*

$$\mathrm{dist}(a) = \frac{1}{w_1(a)},$$

*and find the path $P$ from source node $s$ to destination node $t$ that minimizes*

$$\mathrm{dist}(P) = \sum_{a \in P} \mathrm{dist}(a).$$

## ebsp

Enhanced bandwidth-inversion shortest path algorithm is an enhancement proposed to the *sdp* algorithms by Wang and Nahrstedt [47]. It adds a penalty term to the weight function of the *sdp* that prevents the paths from becoming excessively long by penalizing for large hop counts,

$$\mathrm{dist}(P) = \sum_{j=1}^{k} \frac{2^{j-1}}{w_1(a_j)}.$$

**Bandwidth guaranteed algorithm 4 (ebsp)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, where $w_1$ is bandwidth and $w_2$ is hop count, and requested minimum bandwidth $C_1$, find the path $P$ from source node $s$ to destination node $t$ that minimizes*

$$\mathrm{dist}(P) = \sum_{j=1}^{k} \frac{2^{j-1}}{w_1(a_j)},$$

*where the links $a_1, a_2, \ldots, a_k$ are the links along the path $P$ from the source node $s$ to the destination node $t$, such that $a_1$ is the first link, and $a_k$ is the last link.*

## dap

Dynamic-alternative path uses the widest-shortest path algorithm but limits the hop count to $n + 1$, where $n$ is the minimum hop count in an unpruned network [34]. If no feasible minimal hop path is found, it selects the widest path that is no more than one hop longer. If no such feasible path exists, the flow's request for a path with QoS guarantees is rejected.

**Bandwidth guaranteed algorithm 5 (dynamic-alternative path)** *Given a network $G(N, A)$, metrics $w_i(a)$, $i = 1, 2$, for each link $a \in A$, where $w_1$ is bandwidth and $w_2$ is hop count, and requested minimum bandwidth $C_1$, find the path $Q$ from source node $s$ to destination node $t$ that minimizes hop count $w_2$ and let $n = w_2(Q)$. Prune the network by deleting all the links for which $w_1 < C_1$, find the path $P$ from source node $s$ to destination node $t$ that minimizes hop count $w_2(P)$ while $w_2(P) \leq n + 1$. If there are multiple paths with the same hop count, select the one that maximizes bandwidth $w_1(P)$.*

### 3.6.2 Summary of algorithms with bandwidth constraint

Figure 3.5 shows the relations of the algorithms. The term *shortest path algorithm* here means an algorithm that uses only hop count as a metric. This corresponds to problem 3, path-optimization, although that allows any additive metric, not just ones where all link weights are one. Changing the weights to available bandwidth and solving for the widest path corresponds to problem 1, link optimization problem.

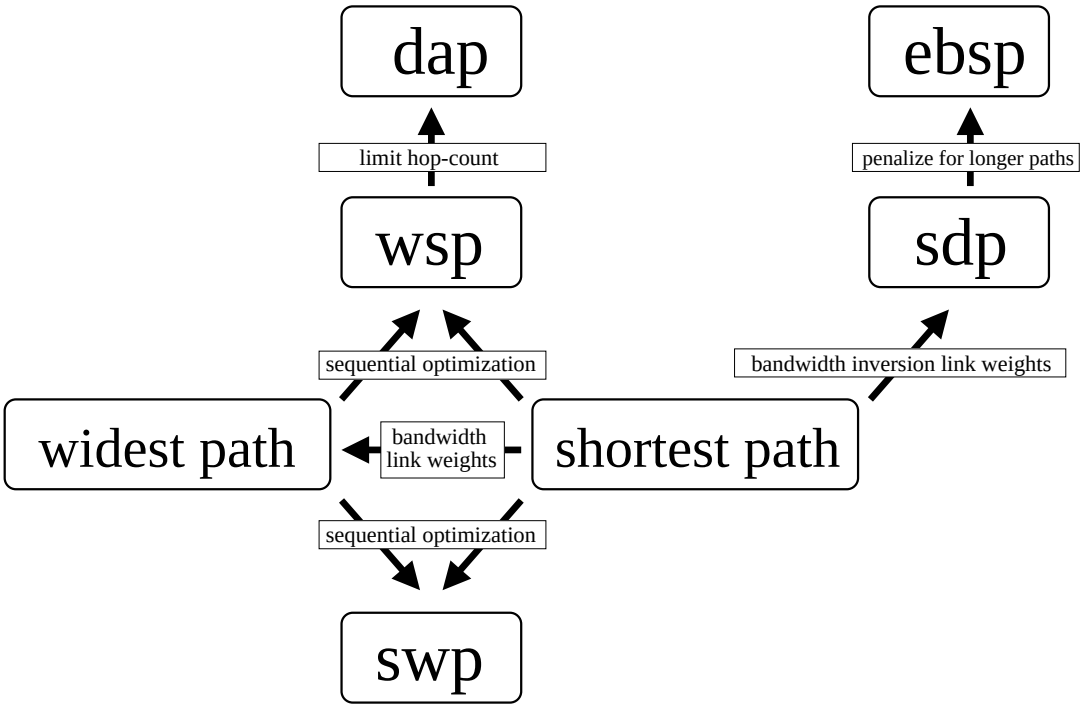



Figure 3.5: Bandwidth constraint algorithms

Combining these two sequentially lead to shortest-widest path and widest-shortest path algorithms. The primary metric is optimized first, and if several paths have the optimal value, then the secondary metric is used. The *dap* algorithm adds an admission control element to the *wsp* algorithm. It rejects the flow if the selected path is more than one hop longer than shortest path.

Shortest-distance path algorithm uses a shortest path algorithm with link weights that are inverse of available bandwidth. Enhanced bandwidth-inversion shortest path algorithm adds a penalty term that grows larger as a function of hop count thus preventing excessively long paths. The higher the algorithm is in the diagram, the more it emphasizes limiting hop count.

## 3.7 End-to-end delay constraint

Although bandwidth guaranteed QoS routing has got most interest in research, providing end-to-end delay guarantees is another important area. The problem is to

compute a path for a request with an end-to-end delay constraint $D$.

The total end-to-end delay consists of the propagation delay on each link along the path, and the delay on the routers. The former one is a typical additive cost function. The latter one depends on the rate reserved for the connection. This is essentially the same as bandwidth, so it is a link constraint. The routing problem is not, however, any of the presented composite problems, because the object of minimization is (3.8), which depends on two additive constraints, propagation delay $d(a)$ and hop count $n$, and one path constraint, the reserved rate $r$. So this is not a multi-constrained routing problem, but rather we have only one object function. Hop count can take only integer values and can be limited to be under some specific limit, if nothing else, at least the total number of links in the network. Also the values for $r$ can be limited as in routing problem 9. This enables the solution in polynomial time.

## End-to-end delay as the metric

In [39] an approach for rate-based schedulers is presented. The upper bound for the end-to-end delay is

$$D(P, r) = \frac{\sigma + n(P)c}{r} + \sum_{a \in P} d(a), \tag{3.8}$$

where $P$ is the path, $r$ is the reserved rate, $n(P)$ is hop count of path $P$, $c$ is the maximal packet size, $d(a)$ is the propagation delay of link $a$, and $\sigma$ is a bias value related to the connection's maximal burst. Let $D(P, r(P)) = D(P)$ denote the minimal possible value of $D(P, r)$, where $r(P)$ is the maximal available rate on the path. So

$$r(P) = \min_{a \in P} r(a),$$

where $r(a)$ refers to the available rate of link $a$.

## The exact algorithm

There are $m$ links in the topology, but some links may have the same available rates, so the number of different available rate values in the network is $K \leq m$. The rates are denoted by $r^k$. There are now $K$ different available rate values: $r^1 < r^2 <$

$\ldots < r^K$. This is similar to the situation of problem 9 in section 3.4.2. A feasible path is found as follows.

- For each $k$, $1 \leq k \leq K$
  - delete all links $a$ whose available rate $r(a)$ is less than $r^k$
  - execute the Bellman-Ford shortest path algorithm $H$ times, using the propagation delays $d(a)$ as the cost function, and solving problem 3, the path-optimization problem, to identify the paths with smallest delay for all hop counts n, $1 \leq n \leq H$.
- Compute the delays of the $K \cdot H$ identified paths to find
  a) the path with the smallest guaranteed delay, or
  b) the feasible path with the smallest reservable rate.

The complexity of the algorithm is $O(MHK)$ and since $K$ is likely to be close to $M$ it is almost $O(HM^2)$, which could be too high. So quantization schemes are needed to lower the complexity.

## The quantized algorithm

The quantization can be done for rate or hop count. In the case of quantizing the rate values, the link rates are grouped into classes such that rate-class $j$ contains the rates between $r^1 \cdot (1+\epsilon)^j \ldots r^1 \cdot (1+\epsilon)^{j+1}$. Then the pruning is done for each class, rather than each possible rate as in the above algorithm. The guaranteed delay of the algorithm is at most $1+\epsilon$ times larger than the minimal value. Choosing the value of $\epsilon$ is a trade-off between complexity and accuracy.

## The Minimum Cost algorithm

Define the cost function $w_{\text{cost}}$ to minimize. The cost function is defined in equation (3.9). It may depend on rate $r$, the number of hops $n(P)$ on the selected path $P$ and the delay bound $D$.

$$w_{\text{cost}}(r, P) = C(r, n(P), D(P, r)) \tag{3.9}$$

Now it can be selected which criteria is wished to minimize. For example cost function $w_{\text{cost}} = r$ minimizes the consumed rate, while $w_{\text{cost}} = n(p) \cdot r$ minimizes the overall rate. Another possible approach is the load-balancing criterion. The paper [39] describes Minimal Relative Rate algorithm which minimizes the maximal percentage that any link consumes the available rate.

# Chapter 4

# Routing strategies and approaches

## 4.1 Introduction

The previous chapter discussed the various path selection algorithms. There is, however, more to QoS routing than just the path selection process. Different strategic approaches, like the choices between source routing and distributed routing, or between pre-computation of paths and on-demand computation, need to be addressed. The routers need to get information about the current link states in the network in order to compute the paths.

In this chapter, the first section presents the three algorithm classes: source routing, distributed routing and hierarchical routing. The following section introduce pre-computation and on-demand computation, and finally the distribution of link state information is discussed.

## 4.2 Routing algorithm classes

The main tasks of QoS routing are collecting state information, keeping the state information up-to-date, and searching the routing tables for the information to find feasible paths for flows. Various QoS routing algorithms are divided into three broad classes based on the way they carry out these tasks [16].

- Source routing
- Distributed routing
- Hierarchical routing

### 4.2.1 Source routing

In source routing, feasible paths are computed by the source node. Then a control message is sent to inform intermediate nodes along the selected path of their precedent and successive nodes, or the path is included in the header of each packet. Source routing's local path computation strategy means that nodes need to know the global state of the network. So the global state information has to be updated frequently at every node, which makes the communication overhead very large. Also, since the source node computes the whole path on its own, the computational overhead in the source node could be very high if the network is large. All in all, source routing does not scale very well. On the other hand, it is very simple and flexible. As the routing is done in the source node, any algorithm can be used, or even different algorithms for different purposes. The centralized nature of source routing algorithms makes them easy to implement and upgrade.

Source routing has not been widely adopted in the Internet. It is seen impractical as the explicit path would have to be included in the IP header. Source routing is used in today's Internet for special cases only, such as mapping the network with traceroute, troubleshooting, or forcing an alternate link to some traffic flow for example to avoid congestion.

Many QoS routing algorithms, however, are source routing algorithms [46, 34, 20, 15], see [16] for an overview. The nature of QoS routing is more suitable for source routing than traditional shortest path routing is. For best effort shortest path distributed routing, where a node just forwards the traffic to the next node, is more practical. For QoS routing there are issues like admission control and resource reservation. So the source node has to take the whole path into consideration anyway, implying that the advantages of distributed routing are not so clear. In a MPLS framework, however, source routing could be practical, as the labels would represent the explicit paths.

### 4.2.2 Distributed routing

In distributed routing, also called hop-by-hop routing, the paths are computed by distributed computation. Most distributed routing algorithms require each node to maintain a global network state, based on which the decisions of the next hop are made. There are, however, some flooding-based algorithms that do not require any global state information at all [13], but they tend to have larger overhead from sending update messages. Most often though, link state protocol is used to maintain a complete global state at every node.

Another approach is to extend a distance vector protocol to include information about available bandwidth and other metrics used, as well as the information about the next hop. Based on the distance vectors, the routing is done hop-by-hop. The computational burden in a single node is much smaller than in source routing, since it only needs to find the next hop. That is, it can use fully distributed computation algorithms.

Distributed routing is more scalable than source routing. Problems arise if information in the nodes along the path is not consistent. This can cause routing loops and the path computation to fail. Distributed routing is the common strategy for routing in the Internet today.

### 4.2.3 Hierarchical routing

In hierarchical routing, a hierarchical topology is created by clustering the nodes into groups that form a logical node. These logical nodes are clustered to new groups to form a higher level logical node and so on. Figure 4.1 shows the actual physical network. The nodes, represented by the black dots, are clustered into groups of three or four as indicated by the red circles. Figure 4.2 shows the resulting aggregated topology, along with the second aggregation, in which the groups are clustered into new groups, in this case consisting of three first level groups, indicated by the blue boxes.

Hierarchical routing proposed for QoS routing [16, 23] is somewhat similar to source routing, but each node has detailed state information only about the nodes in

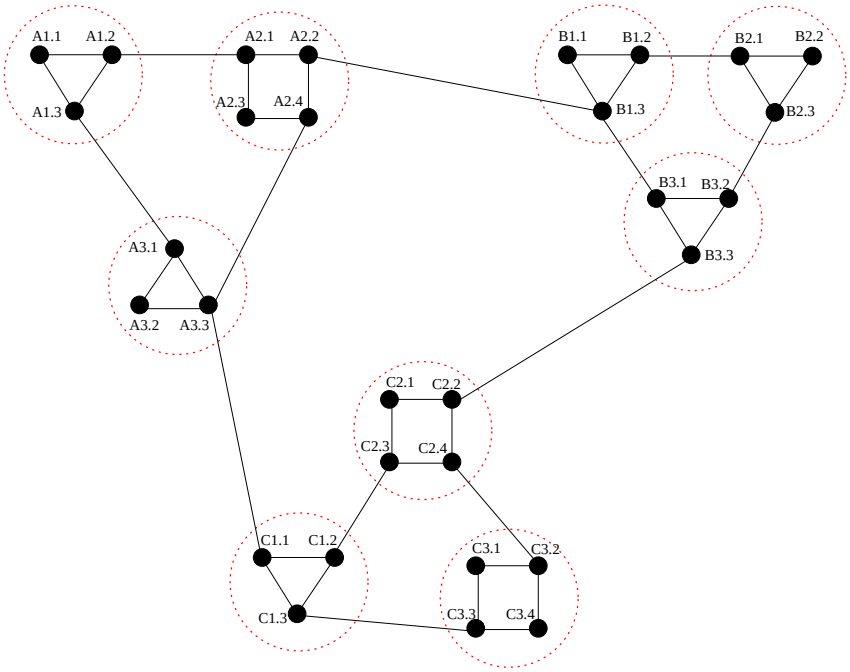



Figure 4.1: Clustering in hierarchical topology

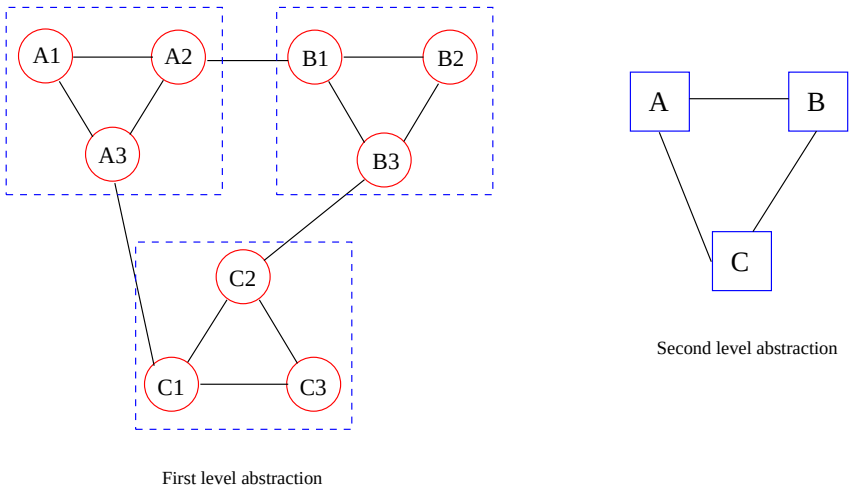



Figure 4.2: The first and second level aggregation of the topology

the same group, and aggregated information about nodes in the other groups. Figure 4.3 shows the topology as seen by the node A2.1. Source node computes a path using the aggregated topology. When a node starts to transmit, first a control message is sent. When it reaches a border node which is a part of a group presented as a logical node in the path, it uses source routing to expand the path through the group. In

the proposed hierarchical QoS routing schemes, source routing algorithms are used at each hierarchical level.

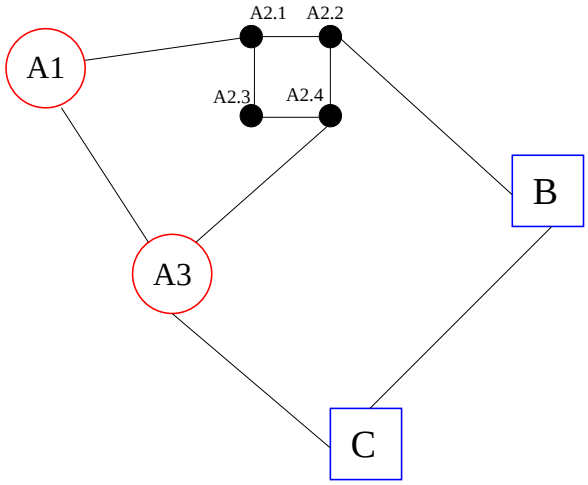



Figure 4.3: The topology as seen by the node $\mathrm{A2.1}$

The size of the aggregated state information is logarithmic in the size of complete global information. Hierarchical routing solves the scalability issues of source routing and is regarded as the most promising scalable QoS routing approach [23]. Negative effects of the aggregation include additional imprecision. For instance, there might be multiple paths through a logical node, one of which has lots of available bandwidth but larger hop count or longer delay, while another may have shorter delay, but does not have similar bandwidth. Requests with either requirements for a small delay or large bandwidth can be routed through the logical node, but both requirements cannot be satisfied at the same time.

Figure 4.4 shows an example of this kind of a situation. What should node A use as the values of delay and bandwidth on link BC? The information in the figure includes maximum bandwidth and minimum delay, but as mentioned they cannot be attained at the same time. The maximum bandwidth available from B to C is $3$ and the minimum delay is $1+1+1=3$. But these are through different paths inside logical node B. Along the path through B2 the values of bandwidth and delay are $(3,7)$ and through B3 they are $(1,3)$. Which one should be used? If a connection from node A to node C has both bandwidth and delay constraints, the aggregated values in the figure are misleading. Aggregation of this kind of information remains a problem [16].

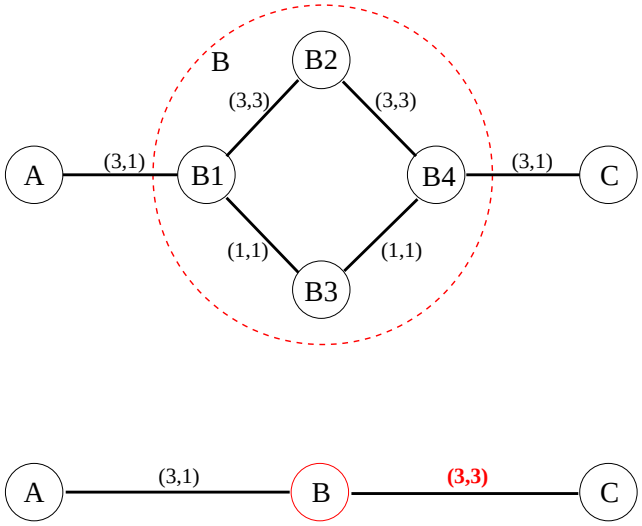



Figure 4.4: Difficulties in aggregating link information. The link state information in the figure is in the form (bandwidth, delay)

A typical example of hierarchical routing protocol is Private Network-Network Interface, PNNI [8], that has been used for ATM networks.

## 4.3 Pre-computation versus on-demand computation

QoS routing can compute paths either on-demand when a flow with QoS requirements arrives, or periodically pre-compute paths for all destinations. The on-demand computation has the benefit of always being able to use the most recent information about the network and it is simpler than pre-computation. If requests arrive too frequently, it is not efficient to use on-demand computation. Path-caching [5, 30] is proposed as a way to lower the computational cost by using previously computed paths, and is shown to effectively reduce the processing cost of on-demand path computation [6].

The pre-computation scheme periodically computes paths for every destination. In section 5 the pre-computation scheme described in [2] is presented. In [1] the authors show by a real implementation of the QOSPF that the path pre-computation does not consume time excessively, so it should not be considered a major cost issue. Simulations in [3] got similar results. The choice between pre-computation

and on-demand computation is a cost-performance trade-off. See section 7.2.

## 4.4 Link state information

In addition to topology information about the network, a router needs information about the availability of resources in the network in order to compute routes supporting the QoS requirements of a flow. This calls for extensions on the link state advertisements of current Internet routing protocols such as OSPF, to include information about the metrics. Section 5 discusses the Quality of Service extensions for OSPF protocol, or QOSPF.

Since the resources such as available bandwidth changes considerably more often than the topology information, more link state advertisement messages are needed. To prevent them from becoming too frequent, various techniques have been proposed. Reducing the frequency of the link state advertisements means that routers do not have the most recent accurate information about the situation. This can lead to performance degradation. A trade-off has to be made between the accuracy and the frequency of flooding of link state advertisements. This problem has got a lot of attention in recent research, and is discussed more in sections 7.2.1 and 7.2.3.

# Chapter 5

# QOSPF

## 5.1 Introduction

The QoS routing scheme receiving most attention currently is the QoS extended OSPF protocol, QOSPF, which is being standardized by the IETF [23]. This section summarizes the QOSPF described in RFC2676 "QoS routing mechanisms and OSPF extensions" [7].

## 5.2 Link state information

The QOSPF protocol uses three metrics:

**Available bandwidth.** Bandwidth is probably the most important metric in QoS requirements, and as mentioned in section 3.4 some other requirements can be mapped to bandwidth requirements. So it is not surprising that one of the metrics used by the QOSPF protocol is the available bandwidth on a link. The maximum value of the available bandwidth can be the link's physical bandwidth itself, or a fraction of that bandwidth dedicated for QoS flows.

**Propagation delay** is also used, but only to avoid the use of high latency links, like satellite links, unsuitable for certain flows, such as interactive real-time

applications. Links with unsuitably high propagation delay can be pruned from the network before running routing algorithms.

**Hop count.** This is not a QoS requirement metric. It is used as a measure of the path's cost to the network. The more links a path uses, the more it consumes resources in the network.

Hop count is calculated while running the path selection algorithm, but the other two metrics have to be advertised to nodes in the network. Each router has to maintain a database of the network topology and the current state of each link. This is achieved by extending the OSPF's link state advertisements to include information about the current values of metrics in the links.

Available bandwidth can change very rapidly with traffic fluctuations, so the link state advertisements have to be sent much more frequently than in the original OSPF protocol. A trade-off has to be made between the accuracy of information and the protocol overhead, when considering how frequently the link state updates should be flooded to the network. See section 4.4 and *Triggers for network state updates* in section 7.2.1 for a discussion of this issue. In RFC2676 [7] no trigger policy is specified, but a lot of latitude is given for experimenting with different approaches.

## 5.3 Path selection

The path selection algorithms of QOSPF use hop count and bandwidth as metrics, as mentioned earlier. They use the *wsp* algorithm described in section 3.6.

The reference implementation focuses on a pre-computed hop-by-hop routing scheme using the Bellman-Ford algorithm. However, two Dijkstra-based algorithms are also presented, with the other one using on-demand path-computation. Pseudo codes for the algorithms are given in [7].

### 5.3.1 Bellman-Ford pre-computation

The Bellman-Ford shortest path algorithm is adapted to compute the maximum bandwidth paths for all hop counts. Routers record the results into a QoS routing table. The table is a $K * H$ matrix, where $K$ is the number of destination nodes and $H$ is the largest allowed hop count. The entries in the table, indexed by $(n, h)$, consists of two fields, namely $bw$, the maximal bandwidth available on a path from the source node to node $n$ with at most $h$ hops, and the information about the next-hop node on the path.

When computing the table, the algorithm starts by modifying the first column so that for nodes reachable by one hop the bandwidth is set to the available bandwidth on that link, and the neighbor field is set to the destination node itself. Then in iteration $h$, column $h-1$ is copied to column $h$. Since the destination was reachable with at most $h-1$ hops with certain bandwidth, it obviously is reachable through the same path with at most $h$ hops. Then it is checked whether a wider path can be found with hop count $h$. So the comparison is made between the bandwidth available when using the previous column's path to reach node $m$, and the bandwidth available using the path through a node $n$ that can be reached with $h-1$ hops and is a neighbor to node $m$. Let $bw(n, h)$ denote the widest bandwidth available on a path from source node to node $n$ by at most $h$ hops, and $b(n, m)$ denote the bandwidth available on the link $(n, m)$ from node $n$ to node $m$. Then, if

$$\min\left(bw(n, h-1), b(n, m)\right) > bw(m, h)$$

then the entry for $(m, h)$ is modified so that the path through node $n$ is used. This check is done for all possible $n$.

### 5.3.2 Dijkstra on-demand computation

The benefit of on-demand computation is that both the bandwidth requirement and the destination is known, so only one path has to be computed instead of a whole table. This makes the computation straightforward. The links with less than the required bandwidth are pruned from the network, either while running the algorithm or beforehand. Then a standard Dijkstra minimum hop path computation is performed on the pruned network.

### 5.3.3 Dijkstra pre-computation

As opposed to the Bellman-Ford, the Dijkstra algorithm uses quantized bandwidth values. The bandwidth capacity is divided into quantized classes: $(0, B)$, $(B, 2B)$, ..., $((Q-1)B, QB)$ such that $QB = C$, where $C$ is the maximum capacity of the link, and the class size $B$ depends on the number of classes $Q$. The difference is that now the routing table is a $K * Q$ matrix, where $Q$ is the number of the quantized bandwidth values, and the hop count is a field in the table entry instead of bandwidth. So instead of finding maximal bandwidth for a given hop count as in the Bellman-Ford algorithm, it finds the minimal hop count for a given bandwidth.

The algorithm starts from the highest quantized bandwidth value. It prunes the original network topology by deleting all the links with insufficient bandwidth. Then it runs Dijkstra's minimum hop algorithm between the source node and all the possible destination nodes. This updates the $Q$th column of the QoS routing table. The quantized bandwidth values are gone through in the decreasing order until the table is finished.

## 5.4 Forwarding

### 5.4.1 Hop-by-hop routing

In the pre-computation schemes the routing information is stored in separate QoS routing tables and has to be extracted from the table when a new request arrives.

An entry in the QoS routing table consists of two fields. The first is either bandwidth (in case of the BF algorithm) or hop count (in case of Dijkstra's algorithm) of the path, and the second is the neighbor node. The neighbor field has the information about the next router in the path to the destination node.

In the case of the Bellman-Ford algorithm, the route for a request to destination $d$ with a bandwidth requirement $B$ is extracted from the table as follows. The destination node $d$ identifies the row of the QoS routing table. That row is then searched starting from one hop path until an entry with sufficient bandwidth is found. With

Dijkstra's algorithm the destination node similarly identifies the row on the table, and the column is the first column whose bandwidth value is sufficiently large.

### 5.4.2 Explicit routing

While the reference implementation in RFC2676 [7] uses hop-by-hop routing, it is possible to modify the algorithms to use explicit routing, that is the route computation in the source node would return the entire path, not just the next hop. The routing algorithm is changed so that it uses the neighbor field to identify the previous router on the path instead of the adjacent router. This being done, the explicit routes can be extracted recursively. The first entry is identified as in the hop-by-hop case, and the recursive algorithm starts from there.

Bellman-Ford: Starting from the table entry $(n, h)$, where $n$ is the destination node and $h$ is the hop count associated with the path with sufficient bandwidth, the neighbor field of that entry identifies the previous node, say $m$, on the path. Consequently, the next table entry is $(m, h-1)$. The path is extracted by recursively proceeding through the columns from $h$ to 1, each time using the neighbor field information to identify the next node.

Dijkstra: Starting from the table entry $(n, q)$, where $n$ again is the destination node and $q$ is the quantized bandwidth value that can satisfy the QoS requirements, the process similarly uses the neighbor field information about the previous node to identify the row for the next entry. However, since columns now correspond to bandwidth instead of hop count, the process does not proceed from column to column, but stays in column $q$, recursively moving from row to row, and ends when the neighbor field points to the source node.

# Chapter 6

# Problems arising from QoS routing

## 6.1 Introduction

This chapter concentrates on some of the problems arising from using QoS routing and possible solutions to those situations. The following section looks into routing under insufficient or inaccurate link state information. Then the instability caused by routing is discussed. The last section of the chapter studies interaction between the traffic class routed with QoS routing algorithms and lower class traffic.

## 6.2 Routing with inaccurate information

> ”Most existing routing algorithms assume the availability of precise state information. However, the state information is inherently imprecise in a distributed network environment. The imprecision directly affects the routing performance. Therefore, the design of routing algorithms for large networks should take the information imprecision into consideration.” *Chen, Nahrstedt* [16]

### 6.2.1 Origins of inaccuracy

The origins for inaccuracy of routing information are listed in [32] as follows:

**Network dynamics** Parameters advertised by a link can be based on an average behavior or out-dated information, as the information may change rapidly. But as discussed in section 4.3, it is not possible, or at least not practical concerning cost, to constantly advertise the latest information.

**Aggregation** As mentioned in section 4.2.3, in hierarchical routing schemes the information is aggregated and a node has precise information only about nodes in the same group. This causes imprecision about routing information in remote parts of the network.

**Hidden information** This can be caused by private networks hiding their information. Another reason could be subnetworks hiding information to maintain some degree of freedom in internal routing.

**Approximate calculation** Even if we have information about the current parameters, it is not exact, but only an approximation of the real parameters.

### 6.2.2 Proposed solutions

Paper [20] investigates both bandwidth requirements and end-to-end delay guarantees for inaccurate routing information expressed in a probabilistic manner. The goal is to find the path most likely to satisfy the QoS requirements. It is concluded that for bandwidth guarantees the impact of inaccuracies is not severe and good paths can be found using shortest path algorithms. For the delay guarantees, however, the problem is NP-hard.

The paper also investigates the delay problem and shows that a rate-based model, although NP-complete, can be used to identify several practical cases of interest for which tractable solutions exist. Also by quantization of metrics near-optimal solutions can be constructed. In [32] Lorenz and Orda study the end-to-end delay case further, and present optimal and $\epsilon$-optimal approximation solutions for the general case.

Chen and Nahrstedt note in [14] that the imprecision model of [20, 32] is based on probability distributions, but how to maintain these distributions was not considered in the work. If the distributions are collected over time they neglect the short term traffic context, which might be very helpful in finding the best paths. Instead, they propose a simpler imprecision model in which every node maintains two values for every destination: an estimation of the metric in question and an estimation of the maximum change during the update period.

The model in [20, 32] tries to maximize the probability of finding a feasible path, but the optimality of the path is not considered. The model in [14] tries to find a low-cost feasible path. Third, the Chen-Nahrstedt approach is a distributed algorithm as opposed to source routing used in the approach above. Paper [14] proposes a ticket based probing scheme, where probes are sent toward the destination to find a path that satisfies the QoS requirements. Simulation results show that high success ratio is achieved and low-cost feasible paths are found. The probing scheme can tolerate a high degree of imprecision.

## 6.3 Stability of QoS routing

One problem of dynamic routing is oscillation of traffic between paths. If path selection relies on link state information about available bandwidth it can easily lead to a situation where all the traffic is routed to path with a lot of available bandwidth. This path will then get congested and the traffic is routed to another path. Compared to traditional dynamic routing, QoS routing has some advantages in handling this problem.

As discussed in section 2.4.3, a reservation protocol is usually used along with QoS routing and it is possible to pin the paths [19, 51]. By this way, even if some other path becomes a better choice, the connections already routed to some path will not switch to the better path. Still all the new request will be routed to the path with more available bandwidth. But because of the probabilistic nature of inter-arrival times and holding times, the fluctuation is not as rapid as it would be if all the existing connections would be re-routed.

If accurate link state information is not available, oscillation can still be a problem.

If one path is marginally better than the other, all the new requests are routed onto that path, until the router receives new link state information. If that takes too long, the path could get congested [42]. This depends on the sensitivity of the update triggers, see section 7.2.3. If accurate information is not available, one solution is to use quantized bandwidth values, as section 5.3.3 describes. By this way there is likely to be a few equally good paths and the traffic would be divided between them. The random routing technique to be described in section 7.2.3, has similar effects. It considers other than just the best path and uses weighted randomization to select between the paths. So not all the arriving requests are routed to the same path.

## 6.4 Inter-class effects in QoS routing

### 6.4.1 Impact of QoS guaranteed traffic on best-effort traffic

It is not reasonable to assume that in a network utilizing Quality of Service routing there would be only QoS guaranteed traffic. Most likely some, if not the majority of traffic flows would still be routed on the best effort basis. When a network has multiple classes of traffic, the paths used for routing high priority traffic, like QoS guaranteed flows, have a significant effect on the performance of lower priority traffic [33].

In an environment with both QoS guaranteed traffic and best-effort traffic, the task of routing is to maximize the resource efficiency. This can be described by two objectives:

1. Minimizing the call-blocking ratio of QoS flows
2. Optimizing the throughput and fairness for best-effort flows

Since the first objective considers only QoS traffic and the second only best-effort traffic, this could lead to a contradiction. Chen [12] proposes a two-level hierarchical scheduling that provides QoS flows with required guarantees and uses maxmin fair allocation for best-effort flows.

In [33] the effect on best effort flows is simulated with different path selection algorithms used for the QoS flows. It is concluded that for a light traffic load the impact is small no matter what algorithm is used, since resources are available. For higher traffic loads the shortest-distance path algorithm is found to perform best.

In [36] Ma and Steenkiste point out that even though poor path selection for QoS flows could lead to congestion and even starvation of best-effort flows, little attention has been given to the problem of inter-class resource sharing. They propose a multi-class QoS routing algorithm that discourages high priority traffic from using links loaded with low priority traffic. Simulations show that the algorithm can improve performance for best-effort traffic without affecting the performance of QoS guaranteed traffic.

### 6.4.2 Inter-class effects in DiffServ

In [47] a DiffServ network with three traffic classes is considered. If all traffic of different classes between specific source-destination pairs follow the same path, the Premium traffic class imposes negative effect on the other traffic classes when traffic load is heavy. These are called inter-class effects. To optimize premium traffic routing and minimize negative inter-class effect, the Optimal Premium-class Routing (OPR) Problem is defined. It attempts to find the optimal routing scheme among all the possible loop-free hop-by-hop routing schemes.

The required bandwidth is attempted to be reserved for every Premium-class traffic flow. Let us assume for simplicity that every node tries to reserve the same amount of bandwidth, say $B$, to every destination node in the network. The maximal value of $B$ is called the saturate bandwidth ($B_s$) and, given a topology, it is fixed for each routing scheme, and depends on the capacities of the links.

In some cases shortest path algorithms produce significantly lower saturate bandwidth than more complex routing schemes. For instance, if one link in the network between nodes $A$ and $B$ has much lower capacity than others, this link would be the bottleneck. Node $A$ would route traffic to $B$ through this link, since it is the shortest path to $B$. A QoS routing algorithm could find a longer path around the link, thus enabling higher $B_s$ values.

A routing scheme with a larger saturate bandwidth is able to accommodate more premium traffic, and if the premium traffic does not use all of the available bandwidth it will yield higher residual bandwidth, thus lowering inter-class effects on lower priority traffic classes and helping to balance the load in the network.

The optimal routing scheme is the one that finds the maximal saturated bandwidth $B_{max}$. If we call the different routing schemes by $R_1, R_2, \ldots, R_n$ and the saturate bandwidth achieved by a particular routing scheme $R$ by $B_s^R$, then the maximal saturate bandwidth is

$$B_{\max} = \max\left(B_s^{R_1}, B_s^{R_2}, \ldots, B_s^{R_n}\right) \tag{6.1}$$

This defines the OPR problem. A simple example is shown in figure 6.1. In this example the network consists of three nodes. The link capacities are similar in both directions and are marked in the figure. In the situation on the left, the shortest path routing achieves the saturate bandwidth $B_s^{sp} = 10$, with link AC as the bottleneck link. In the example presented on the right side of the figure, Optimal Premium Routing achieves a higher saturate bandwidth by routing the traffic from node $A$ to node $C$ (and vice versa) through node $B$. Now $AB$ is the bottleneck link, and its capacity is divided between traffic flows $AB$ and $AC$. The saturate bandwidth is now $B_s^{OPR} = 50$. So in this example equation 6.1 becomes

$$B_{\max} = \max(B_s^{sp}, B_s^{OPR}) = \max(10, 50) = 50 \tag{6.2}$$

and the maximal saturate bandwidth is achieved as described on the right side of figure 6.1.

The OPR problem is NP-complete [47]. Subsequently, the authors study the performance of three heuristic approximation algorithms.

**Widest-shortest path** performs slightly better than the traditional shortest path algorithm, because in tie situations it selects the widest of the paths.

**Bandwidth-inversion shortest path** algorithm *bsp*, or shortest-distance path as it was called in section 3.6, achieves generally a better saturate bandwidth than the shortest path or widest-shortest path algorithms but it is unstable and can also get values even lower than the shortest path algorithm. This is caused by the fact that it prefers wider paths too much without sufficient concern for the

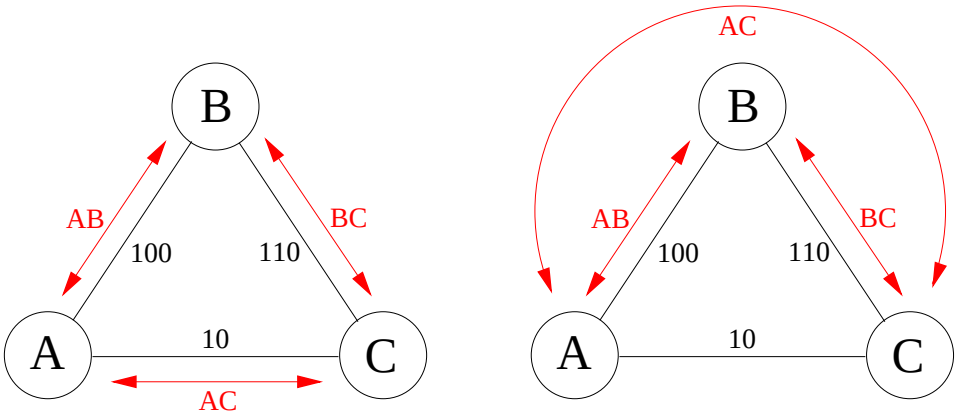



Figure 6.1: An example of Optimal Premium Routing. The arrows represent routing decisions. On the left shortest path, on the right OPR.

length of the path. Obviously, the longer the paths become, the more flows share the bandwidth of the links, and the saturate bandwidth is lower.

**Enhanced bandwidth-inversion shortest path** algorithm *ebsp* was proposed specifically for the problem mentioned above. The penalty term prevents the paths from becoming excessively long. It is found through simulations that for a simple homogeneous network the widest-shortest path algorithm is preferable because of its stability, but for a more complex heterogeneous network, like DiffServ networks tend to be, the *ebsp* is better since it yields much higher values of the saturate bandwidth.

But as will be seen in the example in section 8.3 the *ebsp* is just a heuristic algorithm and is by no means perfect.

### 6.4.3 Bandwidth reservation

One way to prevent the starvation of lower priority traffic even under heavy congestion is to limit the amount of bandwidth that high-priority traffic can reserve on a link. In [33] different routing algorithms were compared and the throughput of the best effort traffic was one of the considered aspects. In that case the maximum reservable bandwidth was limited to $90\%$ of the links capacity. The best effort traffic thus gets at least $10\%$ of the bandwidth, but is free to use all the unreserved bandwidth.

In section 8.2 a simplified example case is formed and solved to study the effects that this kind of bandwidth reservation has, and particularly to see how different values of the reservation level affect the performance.

# Chapter 7

# Review of evaluations of QoS routing

## 7.1 Introduction

This chapter discusses simulation results and measurements from a test implementation found in recent studies. Section 7.2 studies the costs involved in QoS routing and section 7.3 concentrates on simulation results evaluating the different routing algorithms presented in chapter 3.

The topologies that this chapter refers to, are presented in appendix C.

## 7.2 Cost of QoS routing

When evaluating the usefulness of Quality of Service routing or constraint-based routing, it is practical to separate between cases where resource reservation is used, and those where it is not used.

The former implies an IntServ-type environment, where we can be sure of one thing: the traffic that is routed by QoS routing scheme gets the resources necessary to receive the specified quality of service. What remains to be decided are the following issues.

1. Does QoS routing for high priority traffic cause too much difficulties for lower

priority traffic? This was studied in section 6.4.1.

2. What are the costs involved in enabling QoS routing? For instance, how large are the computational cost and protocol overhead. To put it simply: is it worth it? Section 7.2.1 concentrates on this.

The latter case, with no resource reservations, is more oriented to load balancing and congestion prevention, which improves the performance of the network in general, but does not guarantee quality of service to any particular traffic flow. For instance DiffServ architecture does not include resource reservation possibilities, but QoS routing can be used to balance the load within the DiffServ domain to make sure that congestion does not occur. Unfortunately, it seems that there are not yet implementations of this kind of a framework, and simulations are more concentrated on performance than costs. Thus, there are no results on the costs involved.

### 7.2.1 Factors contributing to cost and overhead

The cost of QoS routing comes mainly from two sources. First, it includes *computational cost*. The computations needed in QoS routing are much more complex than those used in regular shortest path routing. Using more sophisticated path selection algorithms and pre-computing the paths more frequently leads to even higher costs. Second, the link state updates contribute to *protocol overhead*. More frequent updates lead to larger overhead.

1. Computational cost [4]

    - Choosing the path selection algorithm. Sophisticated algorithms are able to find better paths but are, of course, more costly.
    - Choosing when the paths are computed, on-demand or beforehand by pre-computation, and what is the pre-computation period. Routing is more effective if the computation uses the most recent information but the cost per request gets higher.
    - Flexibility in routing, supporting alternate path selection, accounting for inaccuracy etc.

2. Protocol overhead [4]

   - Triggers for network state updates. How does a node decide when to inform about changes in link states. State information, and thus also routing, is more accurate if every change is informed, but it is not practical to flood the network constantly with link state updates, since this leads to a larger protocol overhead. The triggers include:

     **Threshold based triggers,** which trigger an update when the relative change between the advertised and the current link state exceeds some predetermined value.

     **Class based triggers,** which divide the link state values to classes. An update is triggered when the current value crosses a class boundary.

     **Timer based triggers,** which can be used to periodically trigger an update. Another kind of timer based trigger is the *clamp-down timer,* which is used with threshold based triggers to prevent too frequent updates by enforcing a minimum hold-down time between updates.

   - Scope of a link state update message. When a trigger activates a node to send a link state update message about changed values in one of its links, should the message include information about other links? On one hand, adding this information increases unnecessary traffic, on the other hand it could eliminate the need for some future updates.

Of course, using better computation schemes and more frequent link state updates obviously improves the performance of the network. So all the items listed above are basically tradeoffs between performance and cost. The better the performance, the higher the cost, and vice versa. A lot of work has been done on studying the effects of these trade-offs, and to find optimal strategies and values.

### 7.2.2 Evaluation of the significance of the different factors

Apostolopoulos et al. [1] study an implementation of the QOSPF protocol for cost and overhead. Their router uses pre-computation of paths and hop-by-hop routing. The implementation is made with a public domain program called Gate Daemon, which provides a platform for implementing routing protocols on UNIX hosts. The

stand-alone test router is then combined with simulation of a network in order to capture the effects of a large network. They divide the costs into three categories:

1. processing costs, consisting of path pre-computation and path selection costs,
2. message generation and reception cost,
3. memory requirements.

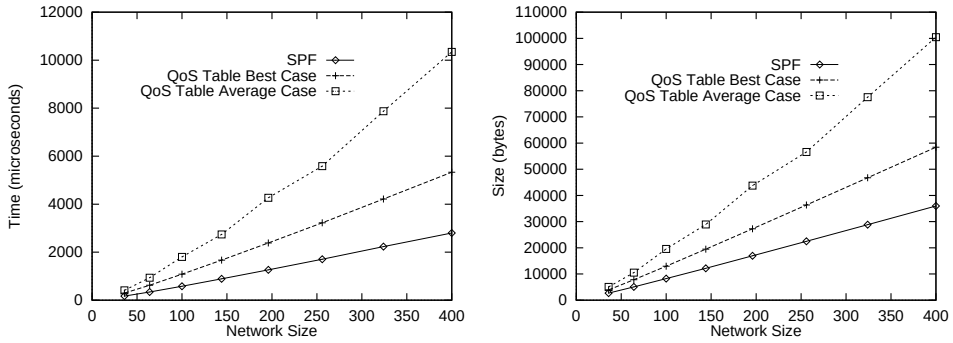



Figure 7.1: Processing time of path computation and comparison of memory requirements [1]

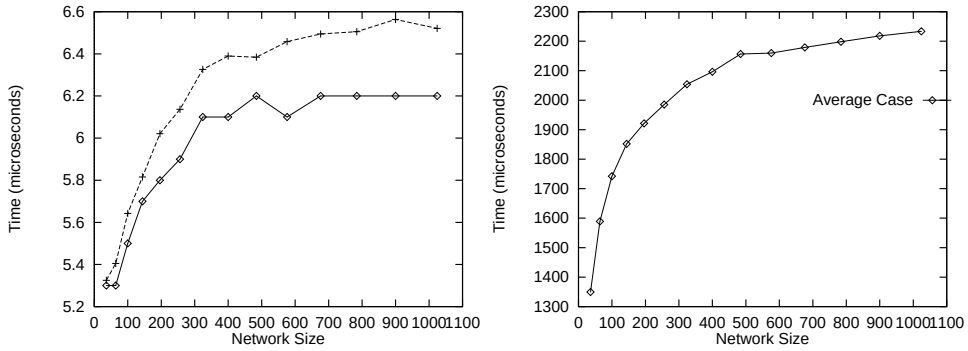



Figure 7.2: Time consumption of path selection and link state database [1]

Figure 7.1 shows the results for the processing time of the path pre-computation. The pre-computation algorithm is executed for different link states. The best case and the average case results in different network sizes are compared against the processing time of the shortest-path computation. The time consumed by the QoS routing pre-computation is larger, but not excessively large compared to the shortest

path computation. This does not take into consideration the fact that QoS paths have to be computed much more frequently than paths for best effort traffic, since the values of the metrics such as available bandwidth, change very rapidly compared to topology changes that trigger a new computation in the traditional routing.

Figure 7.1 shows also the memory requirements of maintaining the QoS routing table in the study. The difference is again clear but not dramatic, as the required memory space is about two or three times larger than in a standard routing table.

Figure 7.2 shows the cost of the path selection and accessing link state database. The results are for the best case, where there is only one entry per path structure, and the average case. Figure 7.2 shows that the time spent on path selection is negligible compared to the time used on accessing the link state database for receiving and generating link state advertisement messages.

As in the case of the path computation, it is also important to consider how often the LSA messages have to be generated. To take this into considerations Apostolopoulos et al. study the utilization of the router with different update triggers and pre-computation periods. Results in table 7.1 display the average and maximal router utilization for the most heavily utilized router in the network, for both the isp topology and the mesh topology. The table shows results for all combinations of three different pre-computation periods and two trigger thresholds. Although momentary maximum values of load were over $30\%$ in the case of the more realistic isp topology, and over $80\%$ for the mesh topology, the average values are low. The results indicate that changes in computation period are much less significant than changes in trigger thresholds. See [1] for more extensive results on router utilization.

Based on the evaluation, the cost increases caused by implementing QoS routing extensions for the OSPF protocol are not excessive for today's processors. Already in 1999 the authors concluded that the cost "remains well within the capabilities of medium-range processors." The single most significant cost source is the time consumed on LSA generation and reception. Table 7.1 shows that the average router load is much more sensitive to changes in the update trigger threshold than changes in the pre-computation period. So the sensitivity of link state update triggers is the most important of the tradeoffs situations considered.

| | Topology: | isp | | mesh | |
|---|---|---|---|---|---|
| | Trigger threshold: | 10% | 80% | 10% | 80% |
| Computation period: | 1s | 36% / 0,27% | 31% / 0,02% | 84% / 1,60% | 82% / 0,22% |
| | 5s | 36% / 0,26% | 31% / 0,02% | 84% / 1,27% | 82% / 0,15% |
| | 50s | 30% / 0,24% | 30% / 0,02% | 84% / 1,26% | 80% / 0,12% |

Table 7.1: Maximal and average router loads for different topologies, LSA update trigger thresholds and pre-computation periods

The next section discusses various simulation results concerning these triggers. Since the cost issues could prevent the network from always using the latest link state information, it is an important area of research to see how the routing is affected by the lack of accurate information. Section 6.2 took a brief look on routing with inaccurate information.

### 7.2.3 Evaluation of trigger policies

In [4] Apostolopoulos et al. evaluate trigger policies and study routing performance and cost under different policies through simulation. The results suggest that the volume of the link state update traffic does not depend on the type of value advertised; quantized classes or non-quantized values updated based on relative change.

The sensitivity of the trigger obviously affects the update traffic volumes. With very large clamp-down timer values the traffic volume is independent of triggering policies altogether, since the timer prevents the generation of LSA messages for so long that no matter what the type of system and values of thresholds, a new message is likely to be sent every time the timer allows it. The situation is thus almost equivalent to that with only periodic updates. This can be seen in figures 7.3 and 7.4, which show results for the number of update messages per second for different settings of thresholds and trigger types.

On the $x$-axis there is the value of the clamp-down timer. The different curves are for different trigger policies and sensitivity levels. The letter T stands for threshold

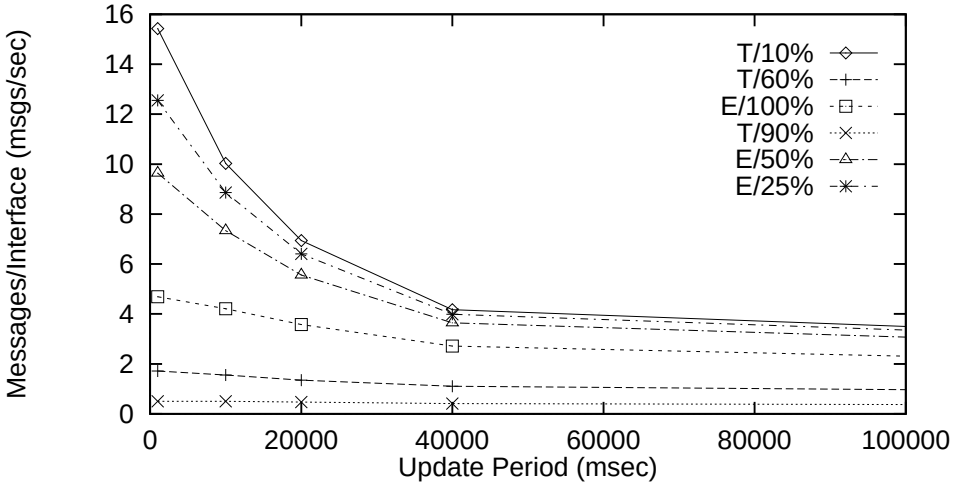



Figure 7.3: Update traffic volume comparison for non-uniform traffic, $5\mathrm{Mb/s}$ requests [4].

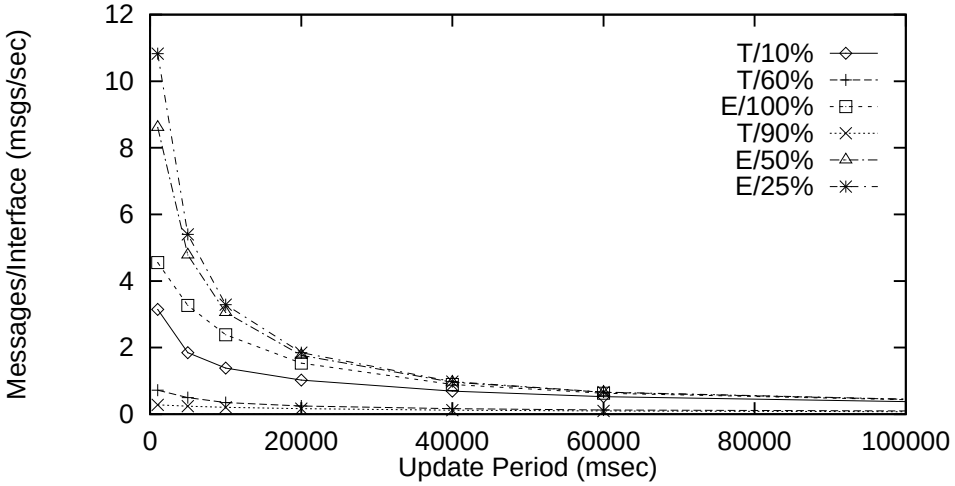



Figure 7.4: Update traffic volume comparison for non-uniform traffic, $1\mathrm{Mb/s}$ requests [4].

based updates and the percentage value is the relative difference between advertised value and current value that is needed to trigger an update. The letter E stands for equal class based updates, a class based trigger where the operating region of available bandwidth is divided into equal size classes. The percentage value here is the size of a class compared to the maximum request size.

Apostopoulos et al. go on to evaluate routing performance under large clamp-down

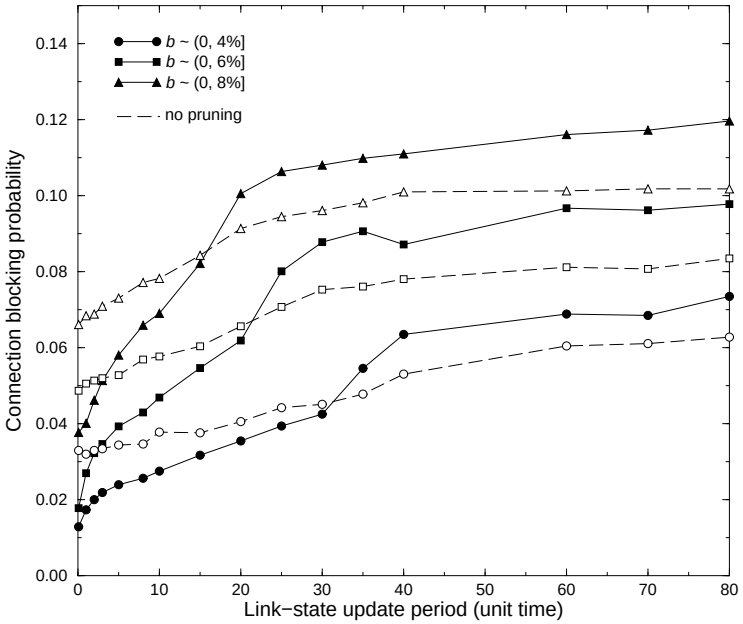



Figure 7.5: Blocking probability vs. update period [42]

values. It is seen that non-pruning routing policies actually outperform the ones that prune the network of links with insufficient available bandwidth. Using *wsp*, the non-pruning policies always select paths among the shortest paths. The pruning policies can select alternative paths when the shortest paths do not have the required resources. But because of the inaccuracy introduced by the low frequency of link state updates, it may select longer alternative paths that consume network resources but do not have the required available bandwidth any longer.

Under substantial inaccuracy the routing performance can be improved by considering several possible paths and using a weighted random selection to choose paths. Choosing sometimes longer paths, even if there appeared to be shorter feasible paths, prevents systematically using a path that is incorrectly assumed to be feasible because of inaccurate link state information. The randomized routing is shown to be particularly effective under a non-uniform traffic distribution.

Shaikh et al. [42] investigate update policies by simulation, as part of evaluation of stale link state impacts on routing. They distinct between *routing failures*, which occur if no feasible path is found, and *set-up failures*, which occur when the selected path cannot support the connection. In an ideal situation where the latest link state information is available, there should be no set-up failures, since the path is selected

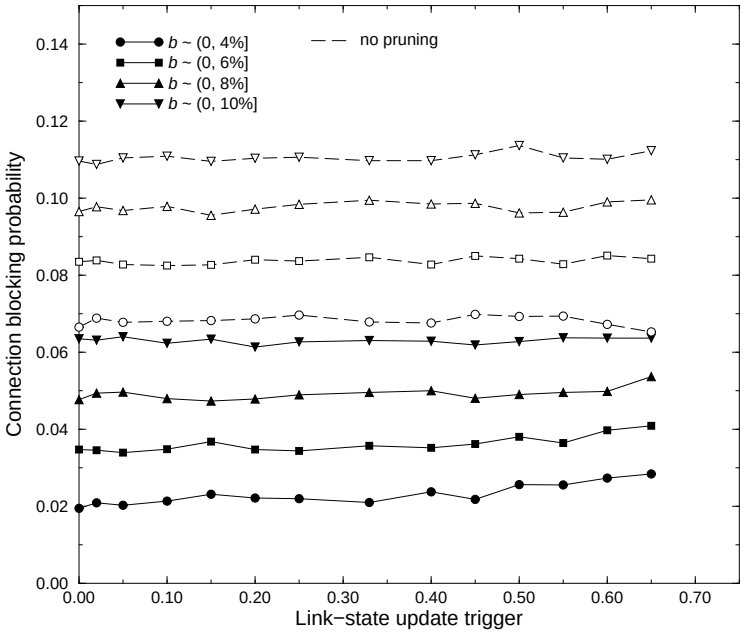



Figure 7.6: Blocking probability vs. trigger sensitivity [42].

based on accurate information. If the path cannot support the connection, this can be seen from the link state information, and either a different path is selected or a routing failure occurs. Also, when using accurate information, the routing failures that do occur are all legitimate in the sense that the requested resources are really not available.

Routing with inaccurate information will increase the amount of set-up failures. Also the cases of false routing failures increase. If the information used for computing the path indicates that some link is infeasible for the connection, it cannot be used for the path, while it may, in reality, be feasible. This can potentially lead to the rejection of the connection, although a path could have been found if the latest information were available.

As the update period increases, more and more of blocking is caused by set-up failures, as can be seen in figure 7.5.[1] In fact it is shown that when update periods are a couple of times larger than the inter arrival time of the connections, the set-up failures dominate the blocking probability. So either more frequent periodic updates

[1]In figures 7.5, 7.6 and 7.7 the legends in the form $b \sim (0, 4\%]$ mean that bandwidth requests used for the simulation are uniformly distributed between zero and four percent of link bandwidth.

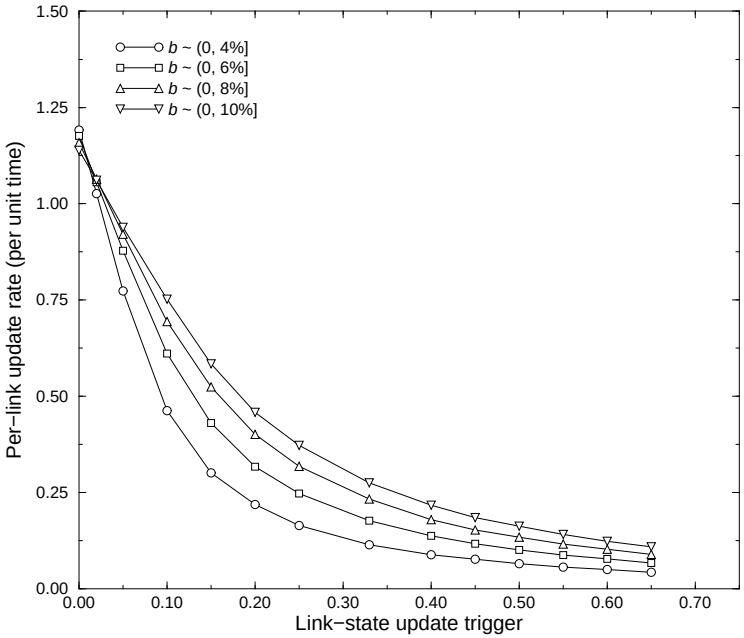



Figure 7.7: Update rate vs. trigger sensitivity [42].

or triggered updates are needed to assure sufficiently accurate information.

Blocking probabilities for using triggered updates are shown in figure 7.6. The sensitivity of the trigger does not seem to make a decisive difference in performance. This result is also seen in [3]. The figures show that the blocking probabilities for the $b \sim (0, 8\%]$ case become larger for periodic updates when the update period is five unit times or larger.

The question is which is the better alternative for minimizing cost: frequent periodic updates, or triggered updates? In the case of the $b \sim (0, 8\%]$, the blocking probability in figure 7.6 is approximately $0.05$ for the trigger case. Comparing to figure 7.5, it is seen that to achieve similar blocking probability the required update period is $4$ unit times, or $0.25$ updates per unit time. For triggered updates, the update per unit time graph for the same topology and traffic load in 7.7 shows that using trigger values of $0.40$ or larger lead to substantially lower amount of updates. So the cost efficient solution is to use triggered updates, but not too sensitive triggers.

## 7.3 Evaluation of routing schemes

### 7.3.1 Bandwidth guaranteed algorithms

Ma et al. [34] use simulation to compare four different algorithms for routing bandwidth guaranteed traffic: widest-shortest path, shortest-widest path, shortest distance path and dynamic alternative path, all of which were described in section 3.6. The results indicate that the *swp* algorithm is not competitive with the others. No matter what topology or traffic load was used the shortest-widest path always leads to the highest call blocking rate among the four. See figures 7.8 and 7.9.

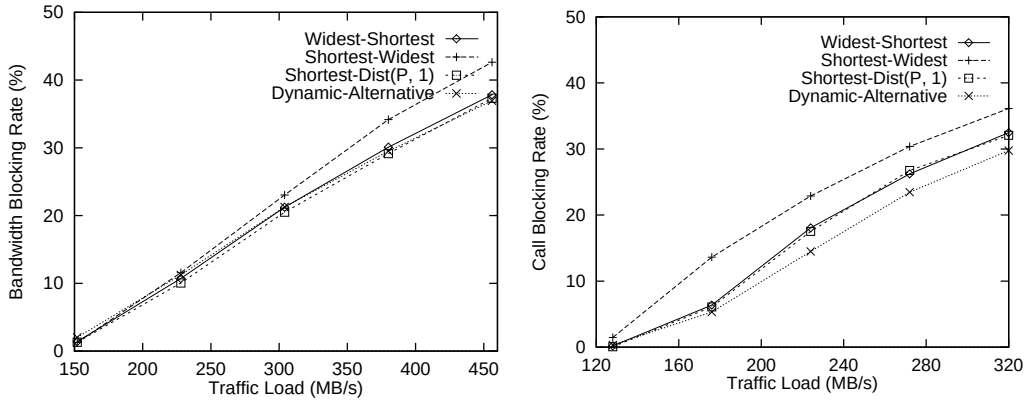



Figure 7.8: Blocking comparison with different topologies, uniform traffic distribution [34].

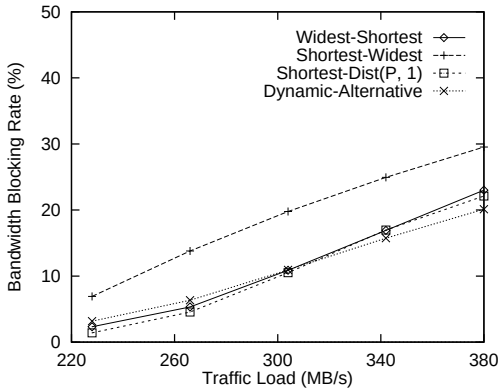



Figure 7.9: Blocking comparison with different algorithms, uneven traffic distribution [34].

The problem with emphasizing the widest path is that the algorithm always routes along the widest path, however long it might be. Certainly this balances the load in the network by avoiding congested links, but it may use much more resources than a shortest path, which leads to higher blocking for other requests, especially if the traffic load is heavy. The three other algorithms have almost similar performance. The *dap* algorithm has slightly lower blocking probability in some cases. See the graph on the right in figure 7.8, which shows blocking of video sessions in the clustered topology. Compared to the MCI topology, in the clustered topology there are always alternate paths available. So it is even more important to limit the hop count.

When the traffic load is light, limiting hop count increases blocking probability, because resource reservation is of no importance and the hop count limit can cause some requests to be rejected. However, the blocking probability in light traffic is only a fraction of the blocking probability under heavy traffic no matter what the routing algorithm is. In the situation shown in the left side of figure 7.8, if the traffic load would be only $114\mathrm{MB/s}$, the blocking probability of the *dap* algorithm would be $0,68\%$ while with *wsp* it would be $0,23\%$ and for *sdp* and *swp* even lower [33].

So it seems that, in general, limiting hop count lowers the blocking probability. It is important to notice that the shortest paths used here are in fact shortest-feasible paths. So unlike best effort traffic, the feasibility condition already forces to route around heavily congested links, and thus algorithms emphasizing available bandwidth are not as effective. Indeed in [35] the *sdp* algorithm was found the most effective for best-effort routing, because of the ability to route around congestion. But that ability does not hold such importance in QoS routing for the above reason.

In [33] Ma et al. add also best effort traffic to their simulation model. The situation of figure 7.8(a) is studied separately for audio and video traffic for which bandwidth requests are from $16\mathrm{kb/s}$ to $64\mathrm{kb/s}$ and from $1\mathrm{Mb/s}$ to $5\mathrm{Mb/s}$, respectively. The call blocking rate of video sessions is over hundred times larger than the rate of the audio sessions. It is shown that the call blocking rate is linearly dependent on the requested bandwidth, such that the algorithms favor connections with lower bandwidth requirements.

### 7.3.2 Evaluation of heuristic approaches

Kuipers et al. [29] present a performance evaluation of the heuristic approaches presented in section 3.5. The evaluation is done using the Waxman graph topology. The simulations are done for different size topologies and each simulation consists of $10000$ different topologies of the same size.

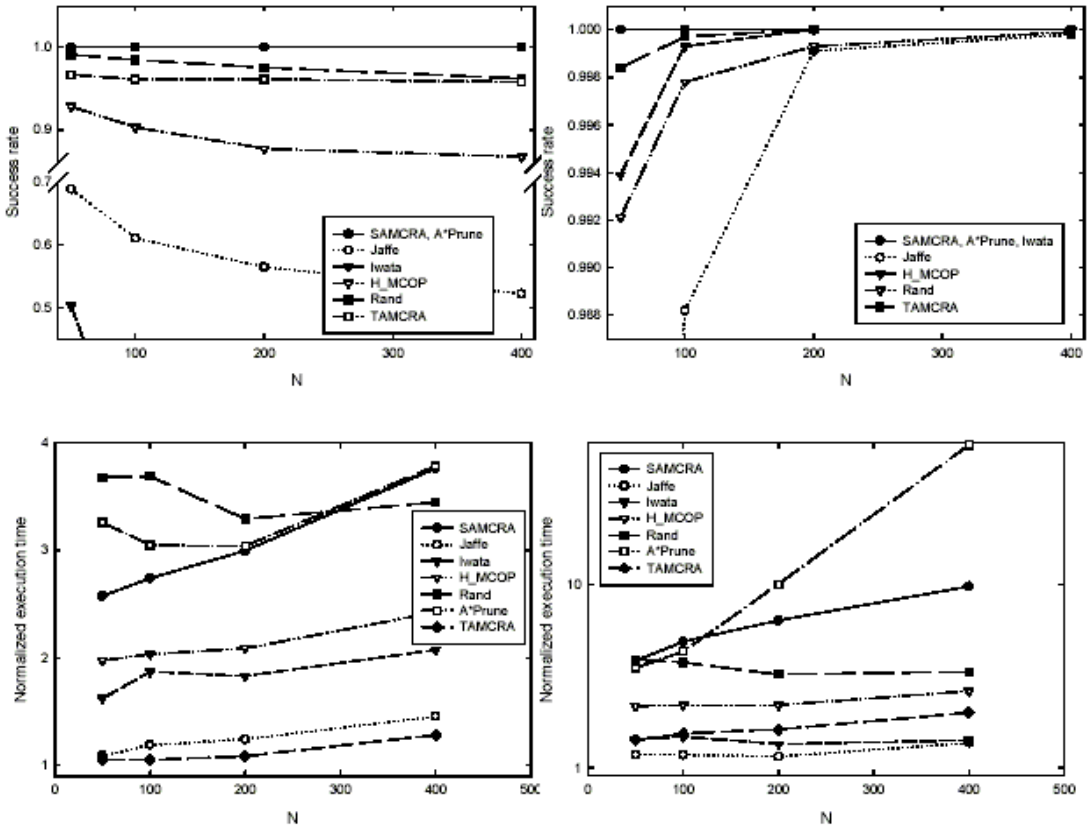
Each link is randomly assigned values between zero and one, for all the metrics. The constraints are chosen in two different ways. In the first case, the constraints are selected so that there is exactly one feasible path. The set of constraints obtained in this way are referred to as constraints L1. In the second case the constraints are selected as follows: First the shortest paths with regard to each metric $i$ are computed. This yields as many different paths as there are constraints used. Let $P^i$ denote the shortest path with regard to metric $i$. The constraint $C_i$ selected for metric $i$ is the maximal value $w_i$ found among those paths.

$$C_i = \max_j w_i(P^j).$$

These are called constraints L2.

The simulation results include the success rate and execution time. The success rate is the number of times a feasible path is found divided by the number of graphs examined. Since the constraints are selected such that a feasible path always exists in the graph, the best possible success rate is $1.00$. The execution time shown in figure 7.10 is a normalized execution time referring to the algorithm's execution time over all graphs examined divided by the execution time that Dijkstra'a algorithm would take computing shortest paths for the same graphs. The authors find that Bellman-Ford based algorithms perform significantly worse regarding execution time than Dijkstra-like algorithms. Therefore the Bellman-Ford based algorithms are omitted from their results. Of the heuristic algorithms presented in section 3.5, only Chen and Nahrstedt's algorithm is such an algorithm.

Figure 7.10 shows the results for the case with two metrics for different topology sizes. $N$ on the x-axis denotes the amount of node in the topology. The results for the stricter set of constraints L1 are on the left side of the figure, and results for constraints L2 on the right side. SAMCRA and A*Prune always give the highest possible success rate, since they are exact algorithms. For the constraint set L2,




Constraints L1 Constraints L2

Figure 7.10: Performance of heuristic algorithms with two metrics [29]

Iwata's algorithm also always gives perfect success rate. This is, however, due to the process of selecting the constraints, which is very convenient for Iwata's algorithm. The selection process is in essence the reverse of the search process in Iwata's algorithm. For the stricter constraint set L1, Iwata performs much worse than the other algorithms. Jaffe's algorithm achieves also significantly lower success rate than the others for constraints L1. For the looser set of constraints, the differences in performance are much smaller. Again, Jaffe's algorithm is the worst performing algorithm. The difference in success rates decreases as a function of the topology size.

Figure 7.10 shows that execution times are longer for the looser constraints L2. One reason for this is that, since more paths are feasible, it is harder for MCOP algorithms to find the optimal among those.

Even though the exact algorithms, SAMCRA and A*Prune, have exponential complexity, their execution times are not overwhelmingly larger than the other algorithms. Only in the case of the looser constraints, for very large topologies they experience significantly larger execution times, but the growth still seems linear. TAMCRA, along with the low success rate Iwata's algorithm and Jaffe's algorithm, performs best in the execution times comparison.

# Chapter 8

# Remarks on QoS routing techniques

## 8.1 Introduction

This chapter includes the parts of the thesis that are new contributions by the author. They are referred to in the other chapters considering these subjects. But as this thesis is mostly a literature review, the own contributions are separated as their own chapter.

## 8.2 Use of bandwidth reservation to prevent starvation of low priority traffic

In section 6.4 the impact of QoS guaranteed traffic on lower priority traffic was discussed. One way to prevent the starvation of low priority traffic is to use a bandwidth reservation scheme. Some percentage of the bandwidth is reserved exclusively for low priority traffic. In this section the effect of using this kind of reservation technique is studied by formulating a Markov-model approximation of the situation.

To simplify matters it is assumed that the guaranteed traffic consists of video connection requests of the same size. Poisson arrivals and exponential holding times with parameters $\lambda$ and $\mu$, respectively, are assumed. The bandwidth requirement of a video stream is here $d = 5\mathrm{Mb/s}$ and the capacity of the link is chosen as

$C = 100\mathrm{Mb/s}$. So if all of the link capacity were available to guaranteed traffic, a total of twenty connections could be accepted at the same time.

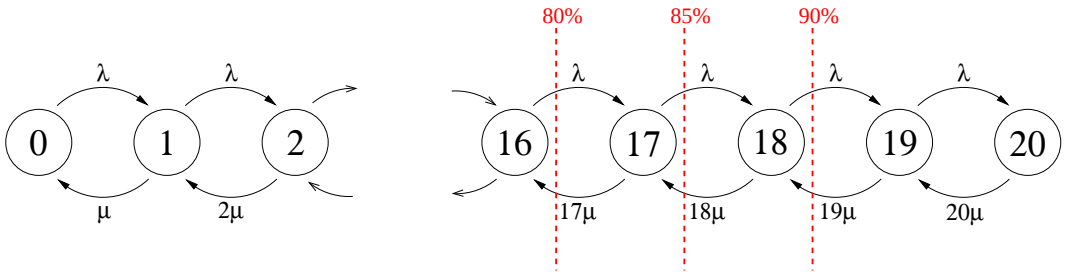



Figure 8.1: The bandwidth reservation Markov-model

If we limit the percentage of link capacity allowed to be reserved by this bandwidth guaranteed priority class, every five percent change is equal to a drop of one connection. The situation is illustrated in figure 8.1 where the dashed lines show where the capacity limit of the priority class is positioned with different values of maximal reservation.

The state probabilities can be easily solved from the balance equations. From the state probabilities we can calculate the average bandwidth available for low priority traffic:

$$b_{\text{available}} = \sum_{i=0}^{20} \pi_i (C - i \cdot d), \tag{8.1}$$

from which the link capacity $C$ and request size $d$ can be taken out from the sum,

$$b_{\text{available}} = C - d \cdot \sum_{i=0}^{20} \pi_i \cdot i. \tag{8.2}$$

Now the sum-term is the average number of connections on the link. Using Little's formula

$$\overline{N} = \lambda_c \overline{T}, \tag{8.3}$$

which says that the average number of connection is the same as carried arrival intensity $\lambda_c$ times the average holding time, which is $\overline{T} = \frac{1}{\mu}$. Carried arrival intensity is the arrival intensity of traffic that is not blocked, that is $\lambda_c = (1 - B)\lambda$. So

$$\sum_{i=0}^{20} \pi_i \cdot i = \lambda_c \cdot \overline{T} = (1 - B)\lambda \cdot \frac{1}{\mu} = (1 - B) \cdot \rho. \tag{8.4}$$

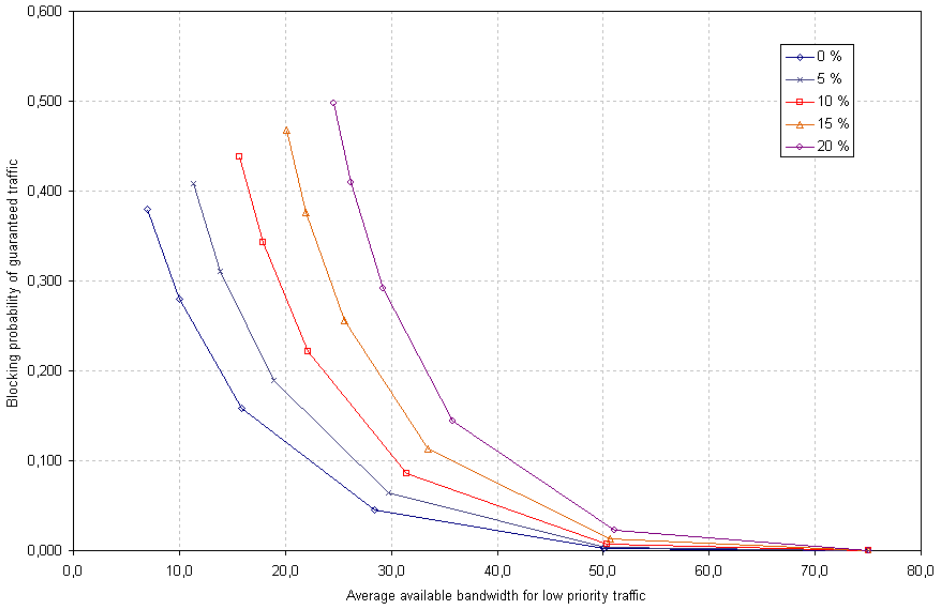



Figure 8.2: Available lower priority bandwidth vs blocking in the bandwidth reservation Markov-model

And substituting that to (8.2), the average available bandwidth for lower priority traffic is then

$$b_{\text{available}} = C - d \cdot (1 - B) \cdot \rho, \tag{8.5}$$

which is on the x-axis of figures 8.2 and 8.3.

The blocking probability of guaranteed traffic can be calculated with the Erlang-formula:

$$B = E(n, \rho) = \frac{\frac{\rho^n}{n!}}{1 + \rho + \frac{\rho^2}{2!} + \ldots + \frac{\rho^n}{n!}}, \tag{8.6}$$

where $n$ depends on the bandwidth reservation level and $\rho$ is the offered traffic intensity on the link for the guaranteed class. Blocking-bandwidth pairs are calculated for traffic intensities from $0$ to $1.5\,C$ and bandwidth reservation levels from $0\%$ to $20\%$, where in the first case there is no reservation at all. The equivalent maximal reservable bandwidths are thus from $80\%$ to $100\%$ of the capacity. The results can be seen in figures 8.2 and 8.3.

Both figures have exactly the same data points, but the connecting graphs are drawn differently to clarify the situation. On the $x$-axis there is the average available band-

width for lower priority traffic, on the $y$-axis the blocking probability of the guaranteed traffic requests.

In figure 8.2 the curves indicate the bandwidth reservation level used. The traffic intensity in the left-most points is $1.5$ times the link capacity $C$, and on the right end of the curves it is $0.25\,C$. This is displayed in figure 8.3 where the curves now represent the trade-off situation between bandwidth reservation levels. Given a traffic load, a change in bandwidth reservation corresponds to moving along the curves in figure 8.3.

The curves are linear and the slope $\alpha$ depends inversely on traffic intensity. In fact it is exactly a function of $\frac{1}{\rho}$ where $\rho$ is the traffic intensity.

This can be seen by solving (8.5) for blocking probability:

$$B = 1 - \frac{C - b_{\text{available}}}{d\rho} = \frac{1}{d\rho} \cdot b_{\text{available}} + \left(1 - \frac{C}{d\rho}\right), \tag{8.7}$$

in which the latter term is a constant, and the dependence between $B$ and $b_{\text{available}}$ is linear and inversely proportional to $\rho$. □

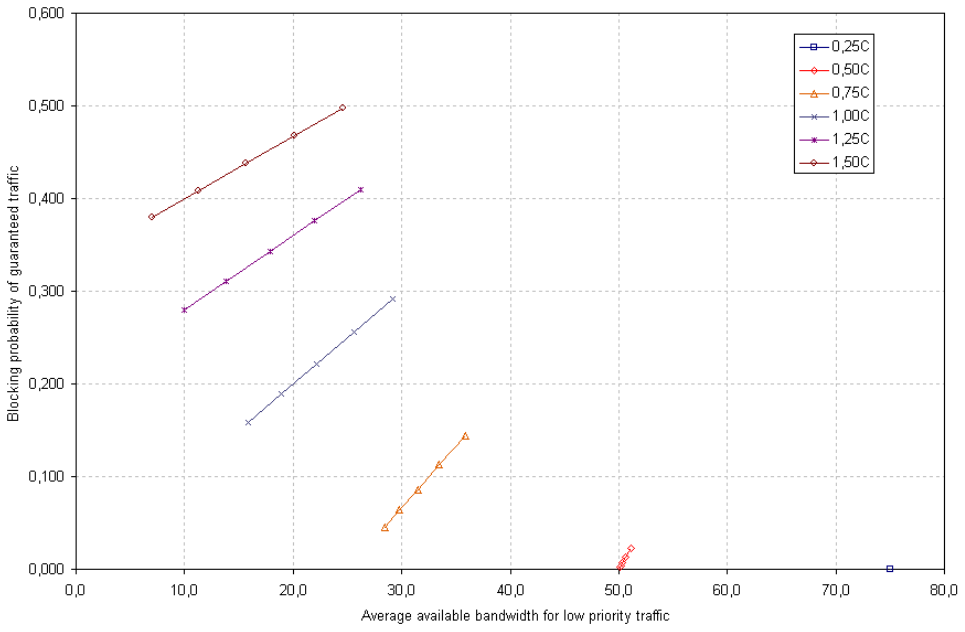



Figure 8.3: Trade-offs between available lower priority bandwidth and blocking in the bandwidth reservation Markov-model

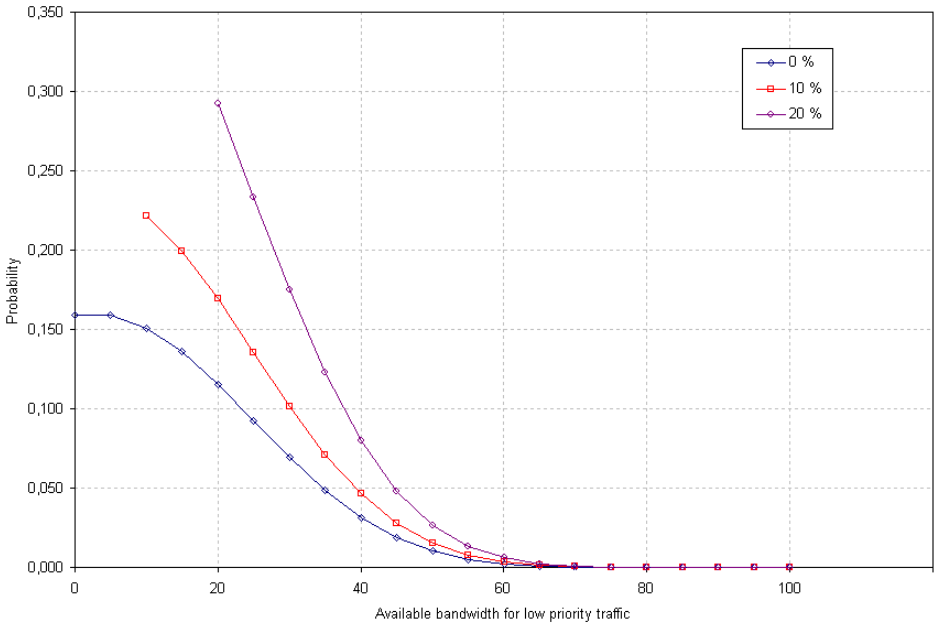



Figure 8.4: State probabilities for values of available bandwidth to low priority traffic for $\rho = 1.00C$

So the curves in figure 8.3 become increasingly flat as traffic load increases, so that increasing the reservation value leads to larger improvement while distributing less to blocking. So with heavier traffic load the trade-off is more favorable. However, for reasonable traffic load, like $0.50$C the effect remains rather small. Figures 8.2 and 8.3 show the average available bandwidth. The minimum available bandwidth is, of course, always the same as the bandwidth reservation level. Figures 8.4 and 8.5 plot the amount of available bandwidth against the probabilities of that much bandwidth being available for traffic load $1.00\,C$ and $0.50\,C$ respectively. The effect of the reservation in the case of traffic load $0.50\,C$ is again smaller. The curves seem almost identical, but there is approximately $2.5\%$ probability for $0\%$ reservation that the available bandwidth is lower than the minimum bandwidth for the $20\%$ case.

There is no clear cut choice for the value of the reservation level, since it depends on the emphasis put on guaranteed class blocking compared to lower priority class throughput.

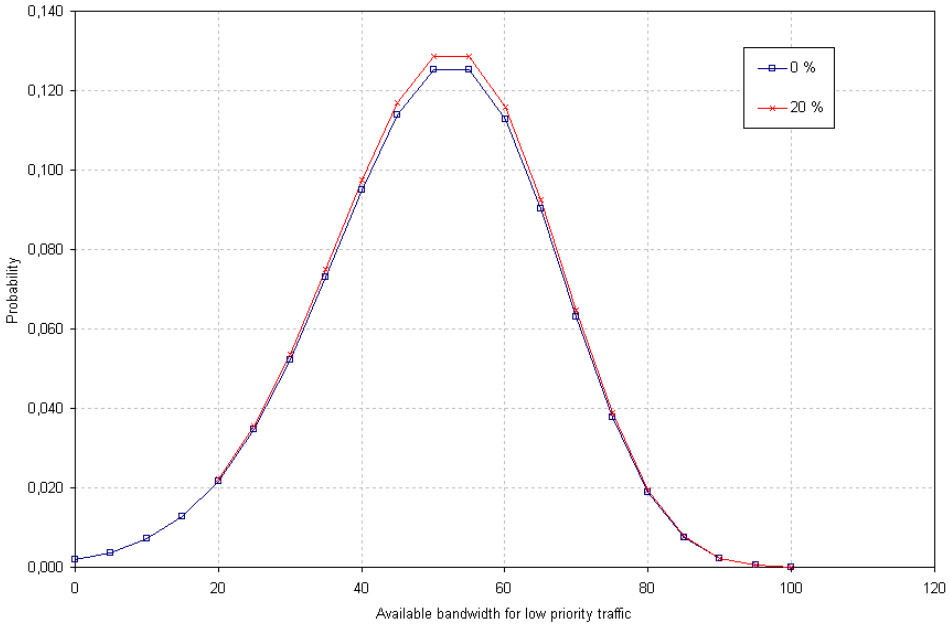



Figure 8.5: State probabilities for values of available bandwidth to low priority traffic for $\rho = 0.5\,C$

## 8.3 Critique on path selection algorithms

The *dap* algorithm, which is just the *wsp* algorithm with a hop count limit, in general performs best out of the four algorithms considered in section 7.3.1, mostly because by limiting excessively long paths, it saves resources for future requests.

In section 6.4.2 DiffServ's inter-class effects were studied. The best performing path selection algorithm for premium class traffic in Wang and Nahrstedt's simulation study [47] is the Enhanced bandwidth-inversion shortest path algorithm, *ebsp*, which also makes modifications to limit hop count. However, the limit of one hop over the shortest path, while leading to a better performance under heavy traffic, decreases performance by not allowing longer paths even if the network is not congested. The *ebsp* also has some unwanted qualities. Consider the example shown in figure 8.6. A connection with bandwidth requirement $1$ is to be routed from node $A$ to destination node $D$. The *ebsp* path weight is calculated as follows

$$w(P) = \sum_{i=1}^{k} \frac{2^{i-1}}{r_i}.$$

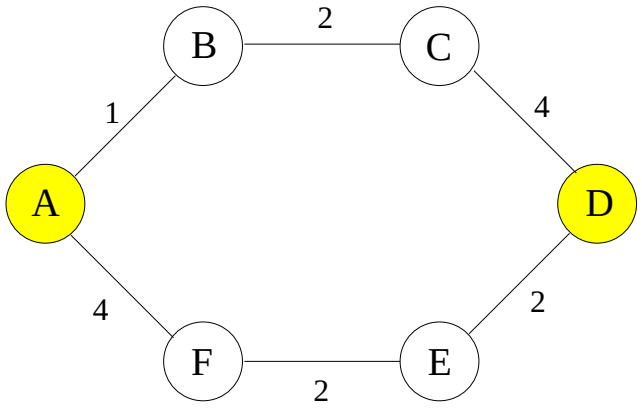



Figure 8.6: Example of Enhanced Bandwidth conversion Shortest Path algorithm.

Now there are two paths from node $A$ to node $D$: through $B$ and $C$ or Through $F$ and $E$. The available bandwidths $r_i$ are shown in the figure. In the first case the path weight is

$$w(P_{ABCD}) = \frac{1}{1} + \frac{2}{2} + \frac{4}{4} = 3,$$

and in the second case

$$w(P_{AFED}) = \frac{1}{4} + \frac{2}{2} + \frac{4}{2} = 3.25.$$

So the shorter of the paths according to the algorithm is the upper path, because the order of the weights affects the outcome. Of course this does not make much sense.

Although the *ebsp* algorithm, modified from the *dsp* algorithm, does perform well in more realistic topologies, this example shows that the enhancement is somewhat arbitrary in some situations. The other successful algorithm, *dap*, is also a modification of another algorithm, the *wsp*.

In both of these algorithms the modification works as a sort of admission control, which prevents selecting paths that increase the blocking probability of future requests so much that it is more costly to accept the request than to reject it. If a higher level admission control scheme could somehow be included to the path selection and resource reservation process, perhaps using different bandwidth reservation levels for different paths based on the hop count of the path (see section 2.4.4), these kinds of attempts to achieve admission control through routing algorithm might not be needed. It is difficult to say whether that would be a better solution.

# Chapter 9

# Conclusion

## 9.1 Summary

Quality of Service routing is needed in the quality of service framework to find paths with sufficient resources available to guarantee the quality that connections request, and to help balance the load in the network thus preventing congestion.

The path selection problems are divided into several classes, based on the nature of the metrics they use, whether there is one or several constraints, and whether optimization is required or any solution satisfying the constraints is feasible.

The routing problems for one and two constraint combinations were formulated in chapter 3. For the problems whose computational complexity is polynomial, the exact solution algorithms were presented. Routing problems with several path constraints are computationally very complex. For those problems heuristic algorithms need to be developed. Several heuristic algorithms were presented in the chapter.

The most important, and most used, metrics are bandwidth along with hop count. Several different algorithms for bandwidth guaranteed traffic were presented in chapter 3. The goal of the algorithms is to find paths that while providing the requested bandwidth, use network resources as little as possible, so that the blocking probability would be low and congestion avoided. The performance of the algorithms was studied in chapter 7. It was found that algorithms that emphasize limit-

ing hop count perform the best.

In chapter 4 various strategies and approaches to QoS routing were introduced. The routing algorithms are divided into three classes:

**Source routing,** where the whole path is computed in the source node, and intermediate nodes just forward the flow,

**distributed routing,** where each node computes just the next hop, and

**hierarchical routing,** where information is aggregated hierarchically to solve the scalability issues in source routing, though aggregating information about several metrics is problematic.

The chapter also discussed the choice between pre-computation, where paths are periodically pre-computed, and on-demand computation, where the path is computed when a request arrives.

Chapter 5 introduced the Quality of Service extensions to OSPF-protocol. It is the QoS routing protocol that has received the most attention in research in recent years. Test implementations have been made, and it is the most likely candidate for a first QoS routing scheme.

Chapter 6 took a look into some problems arising from the use of QoS routing: routing with inaccurate information, instability in path selection and impact on low priority routing.

It was concluded in chapter 7 that inaccuracy in link state information is an important area of research, because the most significant cost issue in QoS routing is the time consumed on link state advertisements, and thus they need to be held to a moderate frequency. Thus, accurate information cannot always be at the disposal of the path selection algorithms. It was learned that for traffic with bandwidth requirements only, the impact of inaccuracy is not significant. For more complicated QoS requirements, however, the inaccuracies have a bigger impact, and the problem is in general intractable. Fortunately, solutions for some important special cases can be found.

Instability is a problem in the best effort routing as well as in traditional dynamic

routing. In QoS routing this can be avoided by using path pinning, a technique that prevents connections to be re-routed when better paths appear, as long as the current path is feasible. Random routing and quantized metric values are also proposed to prevent instability and congestion caused by routing all the arriving requests to the path that is best according to the latest link state information, but may not be for long if all traffic is routed there.

To prevent congestion and even starvation of low priority traffic, several approaches have been proposed. Out of the basic path selection algorithms the shortest-distance path algorithm was found to be the most efficient. For DiffServ inter-class effects the Optimal Premium-class routing problem was defined. The problem is NP-complete. So heuristic approaches were needed. One way to guarantee that the lower priority traffic will not get congested is to reserve some percentage of bandwidth for it. In chapter 8 a model was formulated and solved to study the affect of this approach. No clear-cut choice for the reservation level was found, but it depends on the emphasis put on the blocking probability of QoS traffic compared to the available bandwidth of low priority traffic.

Chapter 7 concluded that pre-computation does not consume resources excessively, and is not a major cost contributor. On-demand computation on the other hand has the benefit of always using the most recent information. It is not, however, efficient to use, if requests arrive very frequently. Also, if the link state updates are less frequent than the pre-computations, the bottleneck on accuracy is not the computation, but the link state advertisements, so on-demand computation is probably not very useful.

Chapter 7 also discussed the cost of QoS routing. Several aspects contribute to the cost and overhead involved in QoS routing, but it has been shown that the most significant source of cost is the time consumed on the link state advertisement generation and reception. The time consumption of the LSA's depends on the the types of update triggers used. The most efficient way seems to be using triggered updates, but not very sensitive triggers. Based on the results, QoS routing is not too costly or time consuming to use.

## 9.2 Further work

The path selection algorithms that perform best are the ones that emphasize limiting hop count. So far there have been simulation studies showing that some particular algorithm performs slightly better than, for instance, the widest-shortest path algorithm, which is often used as a benchmark in comparison. For instance the *dap* algorithm performs very well, but how would an algorithm do that allows longer paths, for example $n + 2$ hops as opposed to $n + 1$ hops allowed by the *dap*. Also, there is little motivation for the penalty terms of *ebsp*, or the optimal value of $n$ in *sdp* algorithm extension $\mathrm{dist}(P, n)$. Perhaps some other kind of weights would perform just as well, or better, and solve the irregularity issues of *ebsp*. As these are the most efficient algorithms, a comprehensive study of similar type of algorithms with different parameters could be beneficial.

Co-existence of QoS guaranteed traffic and low priority traffic has gotten surprisingly little attention so far, even though it is recognized as an important problem. This problem is not different from the path selection problem, where resource reservation is important to accommodate future request, even if lower priority traffic is not considered. So it would be reasonable to combine the two and consider routing schemes with regard to both aspects.

For algorithms with additional weights to penalize for longer hop counts, the behavior of the path weights that the algorithms yield for typical paths could be studied. The performance of this kind of algorithm would, however, need to be evaluated by extensive simulations.

The bandwidth reservation approach could be extended to multidimensional Markov process for cases with several routing classes. Throughput of classes could be roughly approximated by simulating only the class of interest, and letting the available bandwidth change based on the state probabilities in the Markov process.

The bandwidth reservation can be used as an admission control, such that it allows longer paths, but sets higher reservation levels. For instance if the bandwidth reservation for the purpose of low priority traffic is $10\%$, an additional $x_1\%$ would be added for paths one hop longer than the shortest path, an addition of $x_2\%$ for paths two hops longer, and so on. The optimal values of $x_i$ could be evaluated with traf-

fic theoretic methods, such as dynamic state dependent routing. The performance study would be done by simulation.

The majority of the studies on QoS routing so far have concentrated on the situation where QoS traffic reserves resources along the path it uses with RSVP. This guarantees that if the request is accepted and the reservation is successful, the requirements will be satisfied. However, this kind of system seems to indicate an IntServ-like framework. In a DiffServ network the end-to-end reservations are harder to achieve. Reservation of resources has to be done through bandwidth brokers, and scalability is an issue. How would QoS routing work in a DiffServ network without resource reservation? The path selection would have to count on link state information to prevent congestion on paths. Inaccurate information would lead to a situation where, from time to time, QoS guarantees would not hold. The question of the trade-off between cost of accurate link state information and probability of failing to give the requested guarantees, needs further attention if QoS routing without resource reservation is to be considered.

# Bibliography

[1] G. Apostolopoulos, R. Guerin, S. Kamat, "Implementation and Performance Measurements of QoS Routing Extensions to OSPF", Proceedings of Infocom ’99, pp. 680–688, 1999.

[2] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, T. Przygienda, D. Williams, "QoS routing mechanisms and OSPF extensions", Internet Draft, December 1998. citeseer.nj.nec.com/apostolopoulos98qos.html

[3] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda and S. K. Tripathi. "Intra-Domain QoS Routing in IP Networks: A Feasibility and Cost/Benefit Analysis", IEEE Network, Vol. 13, No. 5, pp. 42–54, September-October 1999.

[4] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi, "Quality of service based routing: a performance perspective", In Proceedings of ACM SIGCOMM’98 Conference, pp. 17–28, August 1998.

[5] G. Apostolopoulos, R. Guerin, S. Kamat, S.K. Tripathi, "Server Based QoS Routing", Global Telecommunications Conference GLOBECOM ’99, Vol. 1B, pp. 762–768, 1999.

[6] G. Apostolopolus and S. Tripathi, "On Reducing the Processing Cost od On-Demand QoS Path Computation", In Proceedings of ICNP’98, pp. 80–89, Austin, TX, October 1998.

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

[8] ATM Forum, "Private network network interface (PNNI)", v1.0 specification, May 1996.

[9] D. Awduche, B. Jabbari, "Internet traffic engineering using multi-protocol label switching (MPLS)", Computer Networks, Vol. 40, No. 1, pp. 111–129, September 2002.

[10] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, "Requirements for Traffic Engineering Over MPLS", RFC2702, 1999.

[11] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional specification", RFC2205, 1997.

[12] S. Chen, "Routing Support for Providing Guaranteed End-to-End Quality of Service", PhD Thesis, University of Illinois. 1999.

[13] S. Chen and K. Nahrstedt, "Distributed Quality-of-Service routing in High-Speed Networks Based on Selective Probing", Proceedings of LCN'98, pp. 80–89, 1998.

[14] S. Chen and K. Nahrstedt, "Distributed QoS Routing with Imprecise State Information", In 7th Int. Conf. on Computer Communications and Networks, pp. 614–621, October 1998.

[15] S. Chen, K. Nahrstedt, "On Finding Multi-Constrained Paths", ICC'98, pp. 874–879, 1998.

[16] S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for the next generation High-Speed Networks: Problems and Solutions", IEEE Network, Special Issue on Transmission and Distribution of Digital Video, Vol. 12, No. 6, pp. 64–79, November/December 1998.

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

[18] M. Gaery and D. Johnson, Computer and Intractability: A guide to the Theory of NP-Completeness, San Francisco CA : Freeman, 1979.

[19] R. Guerin, S. Kamat, S. Herzog, "QoS Path Management with RSVP", Internet Draft: draft-qos-path-mgmt-rsvp00.txt, 1997.

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

[21] R. Guerin and A. Orda and D. Williams, "QoS Routing Mechanisms and OSPF extensions", IETF Internet Draft, November 1996.
citeseer.nj.nec.com /guerin96qos.html.

[22] C. Huitema, Routing in the Internet, Upper Saddle River (NJ) : Prentice Hall PTR, 2000.

[23] A. Iwata and N. Fujita, "A Hierarchical Multilayer QoS Routing System with Dynamic SLA Management", IEEE Journal on Selected Areas in Communication, Vol. 18, No. 12, pp. 2603–2616, December 2000.

[24] A. Iwata, R. Izmailov, D.-S. Lee, B. Sengupta, G. Ramamurthy and H. Suzuki, "ATM Routing Algorithms with Multiple QoS Requirements for Multimedia Internetworking", IEICE Transactions and Communications E79-B, No. 8, pp. 999–1006, 1996.

[25] J.M. Jaffe, "Algorithms for Finding paths with multiple constraints", Networks 14, pp. 95–116, 1984.

[26] K. Kar, M. Kodialam, T.V. Lakshman, "Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications", IEEE Journal on Selected Areas in Communications Vol. 18, No. 12, pp. 2566–2579, December 2000.

[27] T. Korkmaz, M. Krunz, "Multi-Constrained Optimal Path Selection", in Proceedings of the IEEE INFOCOM 2001 Conference, Vol. 2, pp. 834–843. Anchorage, Alaska, April 2001.

[28] T. Korkmaz, M. Krunz, "A randomized algorithm for finding a path subject to multiple QoS requirements", Computer Networks, Vol. 36, pp. 251–268, 2001.

[29] F.A. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem, "A Review of Constraint-Based Routing Algorithms, Technical Report", Technical Report, June 2002.
http://wwwtvs.et.tudelft.nl/PEOPLE/Fernando/papers/TRreviewqosalg.pdf

[30] Y. Lin, N. Hsu, R. Hwang, "QoS Routing Granularity in MPLS networks", IEEE Communications Magazine, Vol. 40, No. 6, pp. 58–65, June 2002.

[31] G. Liu, K.G. Ramakrishnan, "A*Prune: An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints", IEEE INFOCOM 2001, Vol. 2, pp. 743–749, 2001.

[32] D. Lorenz and A. Orda, "QoS Routing in Networks with Uncertain Parameters", IEEE/ACM Transactions on Networking, Vol. 6, No. 6, pp. 768–778, December 1998.

[33] Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees", in Proceedings of IEEE International Conference on Network Protocols, pp. 191–202, October 1997.

[34] Q. Ma and P. Steenkiste, "Quality-of-Service Routing for Traffic with Performance Guarantees", In IFIP Fifth International Workshopon Quality of Service, pp. 115–126, May 1997.

[35] Q. Ma, P. Steenkiste, H. Zhang, "Routing High-Bandwidth Traffic in Max-Min Fair Share Networks" SIGCOMM'96 Stanford, pp. 206–217, 1996.

[36] Q. Ma and P. Steenkiste, "Supporting Dynamic Inter-Class Resource Sharing: A Multi-Class QoS Routing Algorithm", Proceedings of IEEE INFOCOM '99, pp. 649–660, 1999.

[37] S. Nelakuditi and Z. Zhang, "On Selection of Paths for Multipath Routing", Techincal Report, Department of Computer Science, University of Minnesota, April 2001. citeseer.nj.nec.com/nelakuditi01selection.html.

[38] H. De Neve and P. Van Mieghem, "TAMCRA: A Tunable Accuracy Multiple Constraints Routing Algorithm", Computer Communications, Vol. 23, pp. 667–679, 2000.

[39] A. Orda, "Routing with End-to-End QoS Guarantees in Broadband Networks", IEEE/ACM Transactions on Networking, Vol. 7, No. 3, pp. 365–374, June 1999.

[40] A. Orda and A. Sprintson, "QoS Routing: The Precomputation Perspective", IEEE INFOCOM 2000 - The Conference on Computer Communications, Vol. 1, pp. 128–136, March 2000.

[41] E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture", Internet draft, draft-ietf-mpls-arch-01.txt, March 1998.

[42] A. Shaikh, J. Rexford, K. Shin, "Evaluating the Impact of Stale Link State on Quality-of-Service Routing", IEEE/ACM Transactions on Networking, Vol. 9, No. 2, pp. 162–176, 2001.

[43] W. Sun, "QoS/Policy/Constaint based routing", http://www.cis.ohio-state.edu/ jain/cis788-99/qosrouting/index.html, 1999.

[44] P. Van Mieghem, "Paths in the simple Random Graph and the Waxman Graph", Probability in the Engineering and Informational Sciences (PEIS), Vol. 15, pp. 535–555, 2001.

[45] P. Van Mieghem, H. De Neve and F.A. Kuipers, "Hop-by-Hop Quality of Service Routing", Computer Networks, Vol. 37, No. 3-4, pp. 407–423, November 2001.

[46] Z. Wang and J. Crowcroft, "QoS Routing for Supporting Multimedia Applications", IEEE Journal on Selected Areas in Communications, Vol. 14, No. 7, pp. 1228–1234, September 1996.

[47] J. Wang, K. Nahrstedt, "Hop-by-Hop Routing Algorithms For Premium-class Traffic In DiffServ Networks", In Proceedings of Infocom 2002, Vol. 2, pp. 705–714, 2002.

[48] B.M. Waxman, "Routing of multipoint connections", IEEE Journal on Selected Areas in Communications, Vol. 6, No. 9, pp. 1617–1622, December 1998.

[49] Wikipedia, The Free Encyclopedia. (www.wikipedia.org).

[50] X. Xiao and M. Ni, "Internet QoS, A Big Picture", IEEE Network, Vol. 13, No. 2, pp. 8–18, March 1999.

[51] Z. Zhang, C. Sanchez, B. Salkewicz, E. Crawley, "Quality of Service Extensions to OSPF or Quality Of Service Path First Routing (QOSPF)", Internet-Draft draft-zhang-qos-ospf-01, September 1997.

# Appendix A

# Shortest path algorithms

This chapter introduces the standard shortest path algorithms: Dijkstra's algorithm and the Bellman-Ford algorithm. The algorithms compute shortest paths from a given source node $s$ to all other nodes in the network.

## Dijkstra's algorithm

Let $N(G)$, denoted by $N$ for simplicity, be the set of nodes in the network, by $M$ the set of nodes incorporated by the algorithm, and by A the set of links in the network. Let $T$ denote $N - M$, the set of nodes yet to be incorporated.

When the algorithm starts set $M$ contains only the source node $s$. The cost function $C(n)$ is the weight of the path from node $s$ to node $n$. For each node $n$ in set $T$ let the cost $C(n)$ be the weight of link $(s, n)$ and the predecessor $pred(n)$ be node $s$. If there is no link $(s, n)$, then $C(n) = \infty$ and $pred(n) = 0$.

The cost function values of nodes in set $T$ are called *temporary labels*, and cost functions of nodes in set $M$ are called *permanent labels* since they do not change any more during the execution of the algorithm.

Then, while the set of incorporated nodes $M$ is different from the set of all nodes $N$:
Choose the node $u$ with the smallest cost function of the nodes in $T$. Include node

$u$ to set $M$ and remove it from set $T$. Then update the temporary labels for all the nodes remaining in set $T$ by calculating the new cost functions. The label changes if a node $n \in T$ is reached through node $u$, which was just added to set $M$, with smaller cost than the current label on $n$. That is, if the cost of the path from $s$ to $u$ plus the link weight on link $(u, n)$ is smaller than the current cost on the path from $s$ to $n$, the label is changed. The predecessor of node $n$ is then also changed to $u$.

The nodes are added one by one to set $M$ until all nodes have permanent labels and the set $T$ is empty. The algorithms then stops, having calculated the shortest path from node $s$ to every other node in the network.

- $M = \{s\}$
- $C(s) = 0$, $\text{pred}(s) = 0$
- for each node $n \in (T)$
  - $C(n) = w(s, n)$, $\text{pred}(n) = s$ if $(s, n) \in A$,
  - $C(n) = \infty$ otherwise
- While $N \neq M$
  - Choose $u \in T$ so that $C(u) = \min\{C(v) : v \in T\}$
  - $M = M \cup \{u\}$, $T = T - \{u\}$
  - For each node $n \in T$ that is a neighbor of $u$
    - if $C(n) > C(u) + w(u, n)$ then
      - $C(n) = C(u) + w(u, n)$, $\text{pred}(n) = u$

# Bellman-Ford algorithm

Let $N$ be the set of nodes in the network, and $A$ the set of links. $C(n)$ is the weight of the path from source node $s$ to node $n$, and $pred(n)$ is the predecessor of node $n$ on that path.

First initialize by setting the path weight from $s$ to itself as 0, and the path weight to every other node to infinity.

Then update the path weights. Check for each link $(u, v)$ in the network if node $v$ is reached with smaller weight through a path to node $u$ and link $(u, v)$. If so update the path weight $C(v)$ and set node $u$ as the predecessor of node $v$. When all links are checked, if at least one of the $C(v)$ values changed repeat the update process. The algorithm finds the shortest path from node $s$ to every node $n$ with most $N$ iterations.

- $C(s) = 0$, $pred(s) = 0$
- $C(n) = \infty$, $\mathrm{pred}(n) = -1$ for every $n \in N$,$n \neq s$
- Loop:
  - For each link $(u, v) \in A$
    - if $C(u) + w(u, v) < C(v)$
      - $C(v) = C(u) + w(u, v)$, $\mathrm{pred}(v) = u$
- if at least one $C(v)$ changed repeat loop

# Appendix B

# Short introduction to NP-completeness

First, the following definitions are needed:

**Turing machine** is used for mathematical modelling of a stored-instructions computational device. It is an abstract model of computer execution and storage introduced in 1936 by Alan Turing to give a mathematically precise definition of algorithm or 'mechanical procedure' [49].

**Non-deterministic Turing machine** is a Turing machine capable of simultaneously pursuing an infinite number of computational paths. This is of course impossible, but it is an useful model for problem classification.

A computational problem is in *class P* (Polynomial) if it can be solved in polynomial time by a deterministic Turing machine [18]. This means that the time of execution is a polynomial function of the problem size. In the case of graphs representing networks, it would be a polynomial of the number of nodes and links in the network.

A computational problem is in *class NP* (Non-deterministic Polynomial) if it can be solved in polynomial time by a non-deterministic Turing machine [18]. Problems in *NP* can be solved in exponential time by real machines. Given an answer, the correctness of the answer can be checked in polynomial time. It has not been math-

ematically proven that problems in *NP* are not solvable in polynomial time, but no algorithm exists. If such an algorithm were found, it would mean that *P=NP*. But from the practical point of view it can be assumed that *P* is different from *NP*, and problems in *NP* can not be solved by computers in polynomial time.

A problem is *NP-complete*, if the following statement is true: If the problem can be solved in polynomial time, then the algorithm could be used to solve all problems in *NP*, that is every problem in *NP* can be reduced to a *NP-complete* problem. Out of all the problems in class *NP*, *NP-complete* problems are the ones most likely not to be in *P*.

# Appendix C

# Topologies used in the simulations discussed in the thesis

## The isp-topology

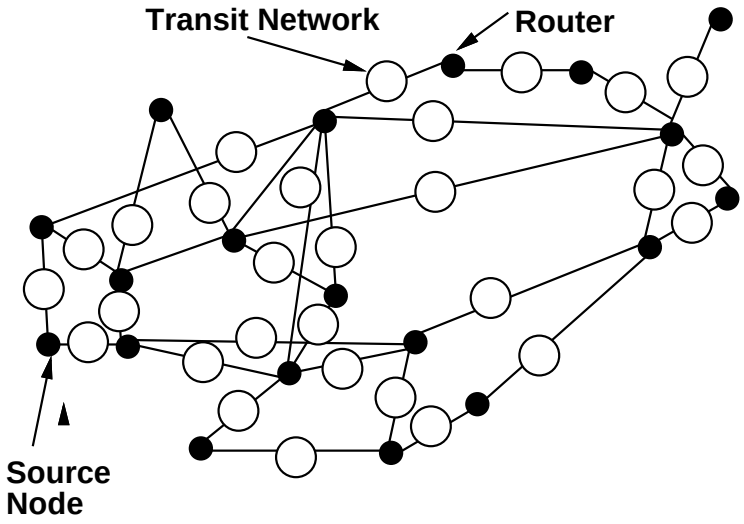



Figure C.1: The isp-topology [1]

The isp-topology, shown in figure C.1 has been used in various simulation studies. The acronym isp stands for Internet service provider, and the topology is a representative of a typical service provider's network in the United States. In paper [1] discussed in section 7.2.1 the link capacities were between $20\mathrm{Mbit/s}$ and $70\mathrm{Mbit/s}$.

## The mesh-topology

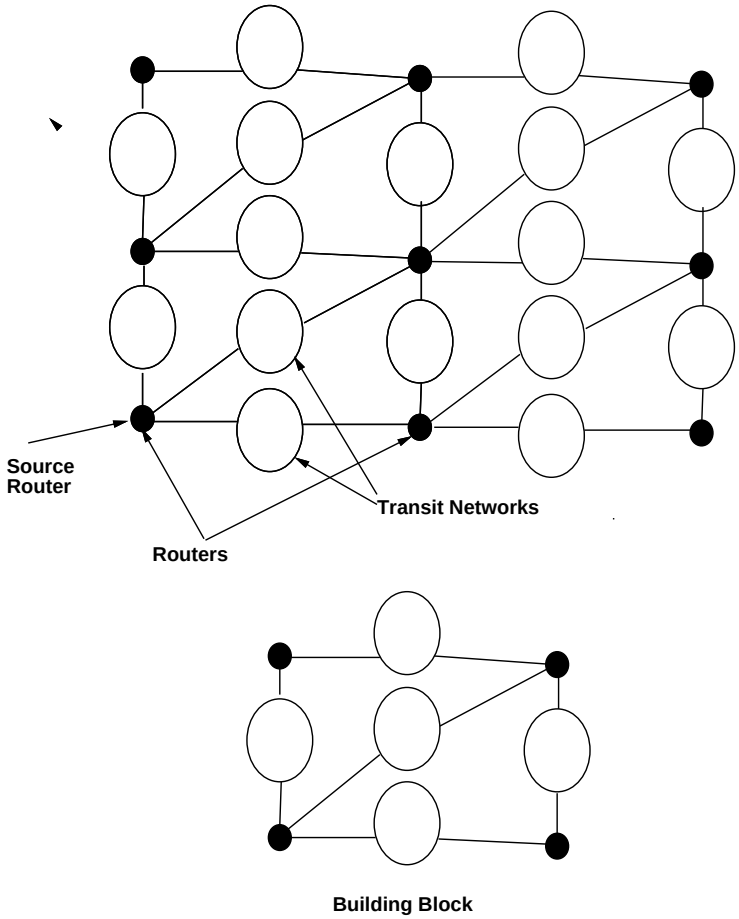



Figure C.2: The $2 \times 2$ mesh-topolgy [1]

The second topology used in [1] is the mesh-topology. The network topology consists of several basic building blocks. One such block has four routers and five transit networks. An example of a two by two blocks topology is shown in figure C.2. The results discussed in section 7.2.1 are for link capacity of $45Mbits/s$.

## The MCI-topology

The MCI-topology used in papers [34], and shown in figure C.3, is a realistic representation of the MCI internet topology in the United States.

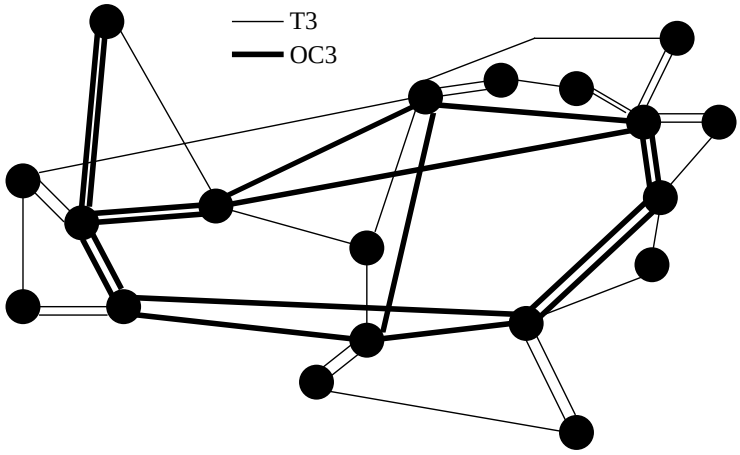



Figure C.3: The MCI-topology [34]

## The clustered-topology

The clustered-topology, figure C.4, used in paper [34] is deliberately constructed so that it has a few clusters of nodes that have many links between them. This allows for alternate routes to be available more often, which is helpful when evaluating the performance of various routing schemes in different situations.

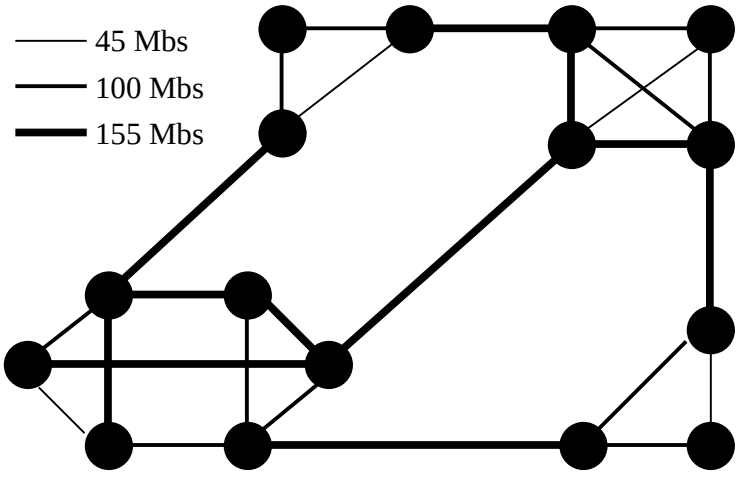



Figure C.4: The clustered topology [34]

# The Waxman graph

The Waxman graph [48, 44] is a random graph with $N$ nodes, which is generated so that for each pair of nodes $i, j$, there is a probability $p_{i,j}$ for a link to connect them. These probabilities are inversely proportional to the distance between the nodes. This leads to a more realistic representative of a network than a random graph without the proportional quality of the probabilities. In a real network, two nodes close to each other are, of course, more likely to be connected by a link. Figure C.5 shows an example of a Waxman graph with $100$ nodes.

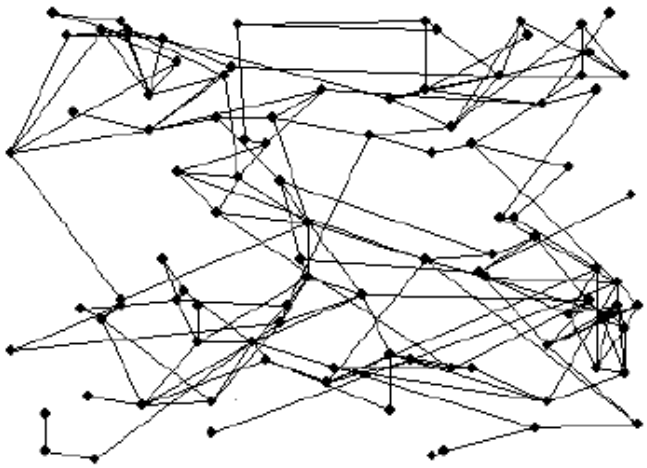


Figure C.5: A typical Waxman graph for $N = 100$ [44]