On Unicast QoS Routing in Overlay Networks

Dragos Ilie

October 2008
Department of Telecommunication Systems,
School of Engineering,
Blekinge Institute of Technology

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Telecommunication Systems
at the Blekinge Institute of Technology (BTH), Karlskrona, Sweden
2008

Thesis adviser: Prof. Adrian Popescu
Thesis co-adviser: Prof. Arne A. Nilsson

Doctoral committee:
Prof. Zhili Sun
Prof. Demetres Kouvatsos
Prof. Do van Thanh
Prof. Michał Pióro
Assoc. Prof. Markus Fiedler (substitute)

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2008:13
ISSN 1653-2090
ISBN 978-91-7295-150-1

Published 2008
Printed by Printfabriken, Karlskrona Sweden

This publication was typeset using LaTeX

To my parents

"I do not know what I may appear to the world; but to myself I seem to have been only like a boy playing on the seashore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me."

Isaac Newton (1642–1727)

Abstract

In the last few years the Internet has witnessed a tremendous growth in the area of multimedia services. For example YouTube, used for videosharing [1] and Skype, used for Internet telephony [2], enjoy a huge popularity, counting their users in millions. Traditional media services, such as telephony, radio and TV, once upon a time using dedicated networks are now deployed over the Internet at an accelerating pace. The *triple play* and *quadruple play* business models, which consist of combined broadband access, (fixed and mobile) telephony and TV over a common access medium, are evidence for this development.

Multimedia services often have strict requirements on quality of service (QoS) metrics such as available bandwidth, packet delay, delay jitter and packet loss rate. Existing QoS architectures (*e. g.*, IntServ and DiffServ) are typically used within the service provider network, but have not seen a wide Internet deployment. Consequently, Internet applications are still forced to rely on the Internet Protocol (IP)'s best-effort service.

Furthermore, wide availability of computing resources at the edge of the network has lead to the appearance of services implemented in overlay networks. The overlay networks are typically spawned between end-nodes that share resources with each other in a peer-to-peer (P2P) fashion. Since these services are not relying on dedicated resources provided by a third-party, they can be deployed with little effort and low cost. On the other hand, they require mechanisms for handling resource fluctuations when nodes join and leave the overlay.

This dissertation addresses the problem of unicast QoS routing implemented in overlay networks. More precisely, we are investigating methods for providing a QoS-aware service on top of IP's best-effort service, with minimal changes to

existing Internet infrastructure. A framework named Overlay Routing Protocol (ORP) was developed for this purpose. The framework is used for handling QoS path discovery and path restoration. ORP's performance was evaluated through a comprehensive simulation study. The study showed that QoS paths can be established and maintained as long as one is willing to accept a protocol overhead of maximum 1.5 % of the network capacity.

We studied the Gnutella P2P network as an example of overlay network. An 11-days long Gnutella link-layer packet trace collected at Blekinge Institute of Technology (BTH) was systematically decoded and analyzed. Analysis results include various traffic characteristics and statistical models. The emphasis for the characteristics has been on accuracy and detail, while for the traffic models the emphasis has been on analytical tractability and ease of simulation. To the author's best knowledge this is the first work on Gnutella that presents statistics down to message level. The models for Gnutella's session arrival rate and session duration were further used to generate churn within the ORP simulations.

Finally, another important contribution is the evaluation of GNU Linear Programming Toolkit (GLPK)'s performance in solving linear optimization problems for flow allocation with the simplex method and the interior point method, respectively. Based on the results of the evaluation, the simplex method was selected to be integrated with ORP's path restoration capability.

Acknowledgments

The five-year long journey towards completing my Ph.D. education has been a most rewarding experience. Many of my research achievements during this period would not have been possible without the direct or indirect support from a number of people.

First and foremost, I would like to express my gratitude and appreciation to Prof. Adrian Popescu from Blekinge Institute of Technology (BTH). Already while I was a M.Sc. student, he encouraged me to pursue graduate studies. His tenacity, enthusiasm and belief in my capacity to get the job done were key elements in finalizing this thesis.

My colleagues and friends, David Erman and Doru Constantinescu, were always there to challenge new ideas, ask difficult questions and encourage me to move forward. Discussions with them over research topics often resulted in fresh, new insights. Additionally, I am grateful for their help with carrying the heavy furniture every time when I changed apartment.

I am indebted to Karel De Vogeleer for his invaluable help with the implementation of the RDP simulator and for numerous suggestions on how to improve the protocol.

Prof. Arne Nilsson has my gratitude for accepting me as a Ph.D. student at the department and for being my secondary adviser.

I have benefited from several interesting discussions with Dr. Markus Fiedler. For this, I thank him very much.

My fellow graduate students Stefan Chevul, Lennart Isaksson, Patrik Arlos and Henric Johnson deserve acknowledgments for encouragement and many interesting discussions.

I would like to thank our head of department, Civ. Eng. Anders Nelsson, and our department economist, Eva-Lotta Runesson, who dealt admirably with practical issues related to my studies, such as literature, equipment and conference travel.

Dr. Parag Pruthi, CEO of Niksun Inc., has my gratitude for helping me with the transition from being a software engineer in his company to becoming a Ph.D. student at BTH.

Much of my early scientific skills were trained by Dr. T. V. Kurien, now with Microsoft Corp. He often reminded me that if I do not start graduate studies before the age of 30, I probably never will. Looking back, I know he was right.

My dear friends, Bob & Hana Pruthi, Zohra Yermeche, Alina Tatu, Gabriela Şerban and Mihaela Chirilă, deserve huge recognition for help, encouragement and advices during my studies.

I would like to express my deep gratitude to my parents who were always there for me. Without their love, help and encouragement I would not have made it this far.

My late grandfather Constantin is responsible for cultivating my interest towards science and discovery during my childhood, through careful selection of books to read. I will always remember him as the man who taught me to love books.

Dragos Ilie
Karlskrona, September 2008

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Multimedia services such as voice over IP (VoIP), IP Television (IPTV), video-conferencing, and video on demand (VoD) have progressed considerably during the last decade in replacing similar functionality offered by traditional analog networks. These IP-based services have strict requirements on how the media streams must be handled during transit in the network. The requirements are typically expressed in the form of constraints on bandwidth[1], packet delay, delay jitter and packet loss. Consequently, multimedia traffic must be transferred over network paths selected such that the media stream requirements are satisfied. This can be done by QoS routing, which is a mechanism for optimizing network performance by a combination of constrained-path selection and traffic flow allocation.

This thesis is about unicast QoS routing in overlay networks. More precisely, we are investigating methods for providing a QoS-aware service on top of IP's *best-effort* service, with minimal changes to existing Internet infrastructure.

1.1 QoS

The term QoS can be interpreted intuitively as an indication for how well a service performs. In reality, QoS is an overloaded term and, when used outside

[1] In the field of computer networking, the term *bandwidth* is used to denote data rate or capacity, unless specified otherwise.

a specific context, it can refer to a quantitative metric related to the "wellness" of the network (or service) or to a mechanism or architecture aimed at improving the well-being of the network (or service). Out of several definitions, we have selected the following two which we consider best at capturing the notion of QoS:

- "*The capability to provide resource assurance and service differentiation in a network is often referred to as quality of service (QoS)*" [3].
- "*Quality of Service (QoS) refers to the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Asynchronous Transfer Mode (ATM), Ethernet and 802.1 networks, SONET, and IP-routed networks that may use any or all of these underlying technologies. The primary goal of QoS is to provide priority including dedicated bandwidth, controlled jitter and latency (required by some real-time and interactive traffic), and improved loss characteristics. Also important is making sure that providing priority for one or more flows does not make other flows fail*" [4].

Consider the following scenario that attempts to illustrate the necessity to implement QoS support in networks and services. Two nodes engage in a voice conversation over a computer network. At each node the continuous voice signal is sampled into a digital signal. The digital signal is compressed and encoded by a codec into a sequence of packets that are sent over the network. In a packet-switched network, individual packets may reach the destination over different paths, within different time durations, possibly arriving out-of-order or not at all. The receiver attempts to cope with these limitations by using for example a playback buffer and error correcting codes. However, each codec has a number of requirements, *e. g.* , bitrate, delay, delay jitter, that must be met if the signal is to be decoded successfully. Additionally, if the packet delay grows too large it gravely affects the interactivity between the speakers, thus rendering the conversation useless. When the network load increases from medium to high, packet queues start building up. This increases the packet delay and also the number of Transmission Control Protocol (TCP) retransmissions, and nodes where the queues reach critical length start dropping packets. Clearly, in this scenario it becomes difficult to guarantee that codec requirements are

maintained, unless care is taken to prevent multimedia flows from being affected by these conditions.

Pure IP-based networks offer the weakest form of QoS, namely best-effort service. In best-effort service no guarantees are provided. The network tries to transport the data to the destination, but sometimes may fail to do even that. Perhaps "poor-effort service" is a more accurate name, but the terminology is too entrenched to be changed.

With the increased popularity of multimedia services, the ability to provide better than best-effort service gained importance. In this context, work begun on architectures for QoS.

The first proposed QoS architectures used on top of IP is called Integrated Services (IntServ) [5]. In IntServ, resources are allocated along the path by using the Resource Reservation Protocol (RSVP) [6, 7]. IntServ performs *per-flow* resource management. This has led to skepticism towards IntServ's ability to scale, since core routers in the Internet must handle several hundred thousands flows simultaneously [8]. A newer report [9] corroborates this number. However, the authors of the report argue that per-flow management is feasible in these conditions due to advances in network processors, which allow over a million concurrent flows to be handled simultaneously.

A new architecture called Differentiated Services (DiffServ) [10] was developed, due to concerns about IntServ's scalability. DiffServ attempts to solve the scalability problem by dividing the traffic into separate forwarding classes. Each forwarding class is allocated resources as stipulated in the service level agreement (SLA) between provider and customer. Packets are classified and mapped to a specific forwarding class at the edge of the network. Inside the core, routers handle the packets according to their forwarding class. Since routers do not have to store state information for every flow, but only have to inspect certain fields in the packet header, it is expected that DiffServ scales much better than IntServ. A major problem with the DiffServ architecture has to do with end-to-end QoS provisioning over multiple DiffServ domains. Premium services cannot be offered unless bilateral SLAs exist between peering domains over the entire end-to-end path. Currently, technical difficulties coupled with the providers' lack of incentive to engage in bilateral SLAs has prevented widespread deployment of DiffServ [3, 11].

Generally, a QoS architecture must address two issues: *resource allocation* and *performance optimization*.

Resource allocation is responsible for the reservation and maintenance of QoS resources, foremost bandwidth, but in some cases also host memory buffers and CPU utilization. In IntServ this is achieved by RSVP, while DiffServ relies on bandwidth provisioning.

Efficient resource allocation is important in order to minimize the costs to run the network. By minimizing costs we do not mean solely lowering the monetary value, but also reducing the number of flows for which no QoS commitments can be made because of wasted resources. This is the goal of performance optimization. Optimizing the performance of a network implies taking control over how individual flows are allocated to paths in the network. This is the problem of QoS routing [3].

Routing is the process of finding a path between two hosts in a network. In QoS routing, the path must be selected such that QoS metrics of interest will stay within specific bounds. Such a path is called a feasible path. A network that has the ability to keep the QoS metrics within bounds is said to be able to provide QoS guarantees. In the case when the guarantees are of statistical nature (*i. e.*, for brief periods of time the bounds do not hold) it is said that the system provides soft QoS. If the bounds hold at all time, then the system is said to provide hard QoS.

1.2 Motivation

The predominant form of Internet routing is a combination of shortest-path routing for intradomain environments coupled with policy-based routing for interdomain communication. For the past ten years it has been argued that Internet routing must incorporate elements of QoS in order for the Internet to be used as platform for multimedia distribution. This argument is in part motivated by difficulties in providing a pleasant user experience with multimedia services when relying solely on a best-effort datagram service.

The term quality of experience (QoE) is used to capture the notion of subjective user experience. A typical way to quantify the QoE is through the use of mean opinion scores (MOSs) [12–15]. Contrary to QoE, the QoS term denotes

an objective performance level based on various metrics at the network layer, *e. g.* , bit rate, jitter, packet loss. Lately, QoS has been extended to include application-layer metrics related to call signaling and media handling [16]. QoE is related to QoS in the sense that a desired QoE level can be used to determine values for end-to-end QoS parameters.

Large network operators configure their networks to supply a specific QoS level in order to achieve good QoE and consequently high customer satisfaction. The configuration aspect incorporates techniques such as service prioritization, packet marking, rate control, load balancing and path protection and restoration. Network operators can also choose to implement their services on top of a specific QoS architecture such as IntServ or DiffServ [5, 7, 10]. However, these approaches are of benefit to services and users located in the same network (*e. g.* , the corporate network), but fail to address a more heterogeneous scenario, where the service provider and users are scattered across the Internet. The main reason for this situation is because of the lack of interaction between network providers or difficulties to align premium services to a common denominator among the providers [17–19].

QoS is one of the most debated topics in the areas of computer network engineering and research. It is generally understood that a network providing QoS has the ability to allocate resources for the purpose of implementing services better than best-effort. The major source of debate is on how to provide QoS in IP-based networks [3, 20].

The debate is characterized by two opposing camps. One of them argues that no new mechanisms are required to provide QoS in the Internet, and simply increasing the amount of available bandwidth will suffice. The members of the other camp express their doubts over the idea that bandwidth over-provisioning alone can take care of QoS issues such as packet loss and delay. History has shown that whenever bandwidth has been added to the networks, new "killer" applications were developed to use most of it. Furthermore, over-provisioning may not be an economically viable solution for developing countries and in the long run it may prove to be very expensive even for developed countries. It is also well worth considering the case of mobile networks, *e. g.* , ad-hoc networks or the Universal Mobile Telecommunications System (UMTS), where not only bandwidth is a scarce resource, but additional challenges in the form of power consumption, mobility prediction and handover must be considered.

From a hierarchical point of view, Internet consists of several autonomous systems (ASs). Each AS consists of a number of interconnected networks administered by the same authority. Within an AS routing is performed by using intradomain routing protocols such as Routing Information Protocol (RIP) [21] and Open Shortest Path First (OSPF) [22]. Interconnected ASs exchange routing information using Border Gateway Protocol (BGP) [23]. An AS connects to other ASs through peering agreements. A peering agreement is typically a business contract stipulating the cost of routing traffic across an AS along with other policies to be maintained. When there are several routes to a destination the peering agreements force an AS to prefer certain routes over others. For example, given two paths to a destination where the first one is shorter (in terms of hops) and the second one is cheaper, the AS will tend to select the cheaper path. This is called *policy routing* and is one of the reasons for suboptimal routing [24, 25]. With the commercialization of the Internet it is unlikely that problems related to policy routing will disappear in the near future.

There seems to be little hope for wide Internet deployment of QoS at network layer, at least in the near future. To cope with this problem several researchers have investigated the possibility to deploy QoS in overlay networks on top of IP [26–30]. This is also the direction chosen for the research presented in this thesis.

At BTH, we are working towards an architecture for multimedia distribution in overlay networks. The work includes evaluation and enhancement of various parts required by the targeted architecture. An important such part is QoS routing in overlay networks. Under the Routing in Overlay Networks (ROVER) project we are developing a platform to facilitate development, testing, evaluation and performance analysis of different solutions for overlay routing, while requiring minimal changes to the applications making use of the platform [31]. The project aims to do this by implementing a middleware system, and exposing two set of application programming interfaces (APIs) – one for application writers, and one for interfacing various overlay solutions.

Overlay routing frameworks have been been the subject of much research in recent years. Systems such as Chord [32], *i3* [33], and Kademlia [34] have been proposed and studied from various aspects. The similarities in the functionality of these and other structured overlay routing systems have resulted in a suggestion for a common API for structured overlays [35]. The ROVER research

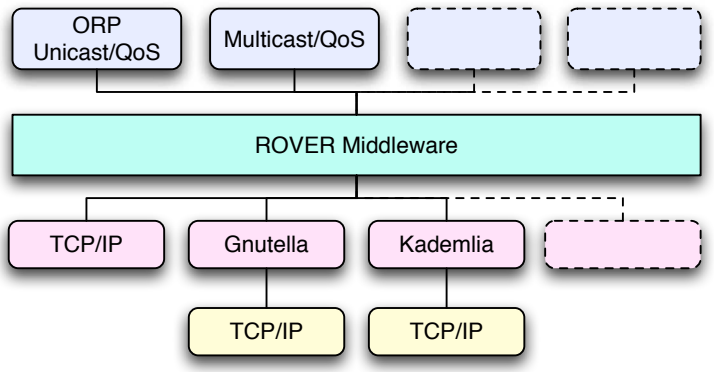


Figure 1.1: ROVER architecture.

group uses this API as a starting point for the development of the ROVER middleware.

The common API is designed to abstract structured overlays, which are overlays with topologies that follow a specific geometry imposed by the distributed hash table (DHT) they use. These overlays are in contrast with unstructured overlays, in which there is no internal structure, and the system can be viewed as emergent. An important goal of the ROVER middleware is to abstract both structured and unstructured overlays.

The ROVER architecture is shown in Fig. 1.1. The top layer represents various protocols and applications using the ROVER API. The middle layer is the ROVER middleware with associated API. Finally, the bottom layer represents various transport protocols that can be used by the ROVER middleware. Only the left box, denoted ORP, in the top layer in the figure is within the scope of this thesis. ORP is a framework that allows us to study specific problems and solutions related to unicast QoS routing [31]. The details of ORP are presented in Chapter 5.

The long term goal is to combine ORP together with additional QoS mechanisms, such as resource reservation and admission control, into a QoS layer. User applications that use the QoS layer can thus obtain soft QoS guarantees. These applications run on end-hosts without any specific privileges such as the

ability to control the internals of TCP/IP stack, the operating system, or other applications that do not use the QoS layer. In terms of the OSI protocol stack, the QoS layer is a sub-layer of the application layer. Applications may choose to use it or to bypass it.

We consider that a QoS routing application based on the architecture described here can be deployed efficiently since it requires no changes to existing IP routers and it relies solely on resource management on end-nodes.

1.3 Related Work

In this section we discuss related work in the area of measurement and modeling of Gnutella traffic and in the area of overlay-based QoS routing. The selection criteria for related work is that it should either have produced strong contributions or influenced our own work, preferably both.

1.3.1 Gnutella Traffic Measurements and Models

Perhaps the oldest and most cited paper on Gnutella measurements is [36], which looks into the social aspects of the Gnutella network. The authors instrumented a Gnutella client to log protocol events. The main contribution of the paper was to show that only a few peers contribute with hosting or adding new content to the Gnutella network, whereas the majority of nodes would retrieve content without sharing any. The authors used the term *free-riding* to describe this behavior and showed that it was just another form of the tragedy of the commons phenomenon described more than three decades earlier [37]. The conclusion of the paper was that the common belief in Gnutella network being more resilient to shutdowns due to distributed control does not hold very well when only few nodes host the majority of content.

A dooms-day prediction was made by [38]. Through mathematical analysis, the author argued that due to its architectural design, in particular the volume of signaling traffic, the Gnutella network will not be able to scale to more than a few hundred users. Enhancements in message caching, flow control and dynamic hierarchical routing implemented by major Gnutella vendors have however rendered most of the conclusions in [38] obsolete.

In [39] the authors created *crawlers* for Napster and Gnutella networks. A crawler is a special purpose software agent, which discovers and records the network topology through an automated, iterative process. The authors used information from crawlers to measure properties of individual peers (*e. g.*, bandwidth and latency). The data from their measurements indicated that both Gnutella and Napster exhibit highly heterogeneous properties, *e. g.*, in connectivity, speed, shared data. This is contrary to the design assumptions used when those systems were built. Another important finding is that users are typically unwilling to cooperate with each other, few of them acting as servers and the remaining majority acting as clients.

A different approach was taken in [40]. The authors performed non-intrusive flow measurements at a large Internet service provider (ISP) instead of using a crawler. The goal was to analyze FastTrack[2], Gnutella and DirectConnect networks. Flows belonging to any of these networks were identified by well-known port numbers. The major findings in the paper are that all three networks showed increases in the traffic volume across consecutive months, skewed distributions for traffic volume, connectivity and average bandwidth, few hosts with a long uptime, and uniformity in terms of number of P2P nodes from individual network address prefixes.

Measurements from a 1 Gbps link in the France Telecom IP backbone [41] network revealed that almost 80 % of traffic on the link in question was produced by P2P applications. Further, the authors showed that flows were partitioned into "mice" — short flows, mostly due to signaling, and "elephants" — long flows due to data transfers.

The P2P traffic identification in [40, 41] assumes that applications use well-known ports. This assumption rarely holds nowadays, when P2P applications use dynamic ports in order to camouflage themselves. Karagiannis *et al.* [42–44] used better heuristics to detect P2P traffic. Their measurement results showed that, if anything, P2P traffic was not declining in volume. Further, they showed that P2P traffic is predominantly using dynamic ports. Applications that currently use or will use encrypted connections would make the P2P flow identification task even harder, if not impossible.

[2]FastTrack is a protocol used by Kazaa and Grokster.

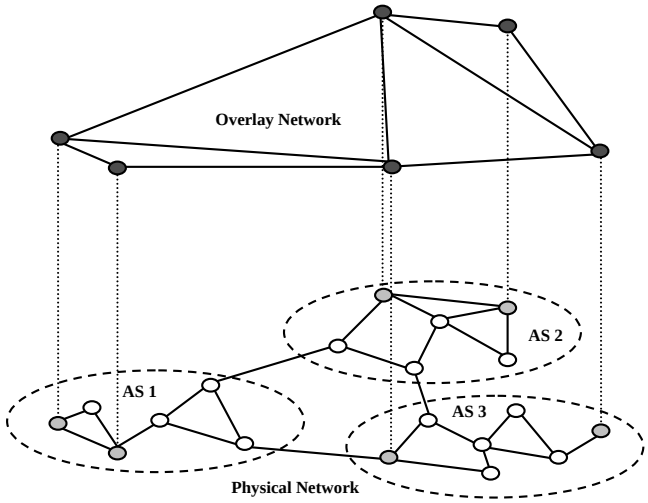


Figure 1.2: Overlay network.

1.3.2 Overlay Networks for QoS

An overlay network utilizes the services of an existing network in an attempt to implement new or better services. An example of an overlay network is shown in Figure 1.2. The physical interconnections of three ASs are depicted at the bottom of the figure. The grey circles denote nodes that use the physical interconnections to construct virtual paths used by the overlay network at the top of the figure.

The nodes participating in the overlay network perform active measurements to discover the QoS metrics associated with the virtual paths. Assume that an overlay node in AS1 wishes to communicate with another overlay node in AS2. Assume further that AS1 always routes packets to AS2 by using the direct link between them, due to some policy or performance metric. The overlay node in AS1 may discover through active measurements that the path crossing AS3 can actually provide better QoS (*e. g.*, smaller delay), than the direct link to AS2. In this specific case, the AS1 node forwards its traffic to the AS3 node, which in turn forwards the traffic to the destination node (or to the next node on the path if multiple hops are necessary). This is the basic idea behind QoS routing in

overlays. Examples of such overlays are the Resilient Overlay Network (RON), OverQoS, the QoS-aware routing protocol for overlay networks (QRON) and the QoS overlay network (QSON).

In RONs [26], strategically placed nodes in the Internet are organized in an application-layer overlay. Nodes belonging to the overlay aid each other in routing packets in such a way as to avoid path failures in the Internet. Each RON node carefully monitors the quality of Internet paths to his neighbours through active measurements. In order to discover the RON topology, RON-nodes exchange routing tables and various quality metrics, *e. g.* , latency, packet loss rate, throughput, using a link-state routing protocol. The path selection is done at the source, which signals to nodes downstream the chosen path. Nodes along the path signal to the source nodes information about link failures pertaining to the selected path. Results involving thirteen sites scattered widely over Internet showed the feasibility of this solution. RON's routing mechanism was able to detect and route around all 32 outages that occurred during the time frame for the experiment, 1 % of the transfers doubled their TCP throughput and 5 % had their loss rate reduced with 5 %.

Following the success of RONs, the authors of [30] propose OverQoS, an overlay-based QoS architecture for enhancing Internet QoS. The key part of the architecture is the controlled-loss virtual link (CLVL) abstraction, which provides statistical loss guarantees to a traffic aggregate between two overlay nodes in the presence of changing traffic dynamics. They demonstrate that their architecture can supply the following QoS enhancements with as little as 5 % bandwidth overhead: smoothing losses, packet prioritization, as well as statistical bandwidth and loss guarantees.

Another approach involving strategically placed nodes in the Internet is presented in [29]. The authors propose an architecture where each AS has one or more overlay brokers. The overlay brokers are organized into clusters that interconnect with each other to form an overlay service network that runs a QRON. The purpose of QRON is to find an overlay path satisfying a bandwidth constraint. QRON nodes use source routing and a number of backup paths to cope with bandwidth fluctuations. The authors were able to show that the QRON algorithms perform well under a variety of traffic loads while balancing the load among overlay brokers.

In a similar spirit, the QSON architecture [45] advocates a backbone overlay network for QoS routing. This architecture relies on well-established business relationships of two kinds. The first type of business relationships is defined by end-users who purchase QoS services from the QSON provider. The QSON provider is able to supply these services by engaging in SLAs with several ISPs. This is the second kind of business relationships. The QSON overlay is spanned by QSON proxies located between ISP domains. Each proxy stores a list of paths to the other proxies. The proxies use probes to reserve bandwidth and to inform each other about changes in available bandwidth. Simulation results have shown that QSON is able to provide bandwidth reservation with low control overhead.

1.4 Main Contributions

In this thesis we investigate the possibility of providing QoS-aware routing for end-users on top of IP's best-effort service. We focus on bandwidth management, but our framework is applicable to other QoS parameters as well. Our solution is based on using an overlay network for QoS routing that combines constrained-path discovery with flow allocation. In this context we present the following contributions:

- Highly detailed statistical models and characteristics for Gnutella traffic crossing an ultrapeer.
- The Route Discovery Protocol (RDP), which is used for constrained-path discovery by selective diffusion.
- The Route Management Protocol (RMP), which is used to handle node churn in the overlay.
- A software library based on the GLPK for solving network flow problems.
- A performance testbed for network flow algorithms utilizing the solver library above.
- Performance results for the simplex method and the interior point method on linear problems of network flow allocation.
- A flexible software library for P2P traffic decoding, based on `tcptrace`.

1.5 Thesis Outline

The thesis is organized as follows. In the current chapter we described the motivation for this thesis. Additionally, we presented related research work and an outline of our own main research results.

In the next chapter we lay the theoretical ground for the reminder of the thesis. In particular, we define notation and terminology for elements of graph theory and discuss algorithms and complexity. This is followed by a brief presentation of shortest-path algorithms, which are used by ORP.

Chapter 3 begins with a short overview of notation and terminology for linear optimization problems involving network flows. This is followed by a presentation of our performance testbed for algorithms used in solving network flow problems. The remainder of the chapter describes performance results for the simplex method and the interior point method. Based on these results we selected the simplex method for solving the optimization problems in Chapter 5.

The Gnutella P2P protocol is presented in Chapter 4. In addition, we describe the measurement infrastructure used to capture Gnutella traffic and our software library for P2P traffic decoding. The chapter reports on the models and characteristics obtained from the recorded traffic. The statistical models for session duration and session interarrival time are further used to generate churn for our ORP simulations presented in Chapter 5.

The subject of Chapter 5 is the ORP framework, which is composed of two protocols: Route Discovery Protocol (RDP) and Route Management Protocol (RMP). We describe their design and implementation and present performance results based on simulations.

In Chapter 6 we share our conclusions and ideas for future work.

There are a number of appendixes at the end of the thesis. In Appendix A we provide a list with acronyms encountered throughout the text. The next appendix summarizes the notation used in preceding chapters. Appendix C outlines the probability distributions relevant for this work.

1.6 Publications

The thesis reports on the author's research activities in the areas of QoS routing, applied optimization algorithms and P2P traffic measurements and analysis. The work was done at the Department of Telecommunication Systems (ATS) at Blekinge Institute of Technology (BTH) in the context of the following research projects:

- Internet Next Generation Analysis (INGA), funded by the Swedish Agency for Innovation Systems (VINNOVA), 2003–2005.
- Routing in Overlay Networks (ROVER), funded by the European Next Generation Internet (EuroNGI)-Network of Excellence (NoE), 2006.
- ROVER, funded by the Swedish Foundation for Internet Infrastructure (IIS), 2007–2008.

Parts of this thesis are based on the following previously published material:

1. K. De Vogeleer, D. Ilie, and A. Popescu, "Constrained-path discovery by selective diffusion," in *Proceedings of HET-NETs*, Karlskrona, Sweden, Feb. 2008.
2. D. Ilie, D. Erman, and A. Popescu, "Passive application layer measurements," *Communications of the ACM*, Submitted for publication.
3. D. Ilie and A. Popescu, "Statistical models for Gnutella signaling traffic," *Journal of Computer Networks*, vol. 51, no. 17, pp. 4816–4835, Dec. 2007.
4. D. Ilie, "Optimization algorithms with applications to unicast QoS routing in overlay networks," Blekinge Institute of Technology, Karlskrona, Sweden, Research Report 2007:09, Sep. 2007.
5. D. Erman, D. Ilie, and A. Popescu, "BitTorrent Session Characteristics and Models," to appear in the Journal of Computer Communications, COMCOM HET-NETs Special Journal Issue 2.
6. D. Ilie and A. Popescu, "A framework for overlay QoS routing," in *Proceedings of 4th Euro-FGI Workshop*, Ghent, Belgium, May 2007.

7. A. Popescu, D. Constantinescu, D. Erman, and D. Ilie, "A Survey of Reliable Multicast Communication," in Proceedings of Euro-NGI NGI, Trondheim, Norway, May 2007.

8. D. Ilie and D. Erman, "Peer-to-Peer Traffic Measurements," Research report No. 2007:02, Blekinge Institute of Technology. Feb. 2007.

9. D. Constantinescu, D. Erman, D. Ilie, and A. Popescu, "Congestion and Error Control in Overlay Networks," Research report No. 2007:01, Blekinge Institute of Technology, Jan. 2007.

10. A. Popescu, D. Erman, D. Ilie, D. Constantinescu, and A. Popescu, "Internet Content Distribution: Developments and Challenges," in Proceedings of SNCNW, Luleå, Sweden, Oct. 2006.

11. D. Erman, D. Ilie, and A. Popescu, "BitTorrent Traffic Characteristics," in Proceedings of IEEE ICCGI, Bucharest, Romania, Aug. 2006.

12. D. Ilie, "Gnutella network traffic: Measurements and characteristics," Licentiate Dissertation, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, Apr. 2006, ISBN: 91-7295-084-6.

13. D. Ilie, D. Erman, and A. Popescu, "Transfer rate models for Gnutella signaling traffic," in *Proceedings of ICIW*, Guadeloupe, French Caribbean, Feb. 2006.

14. D. Erman, D. Ilie, and A. Popescu, "BitTorrent Session Characteristics and Models," in Proceedings of HET-NETs, Ilkley, United Kingdom, Jul. 2005.

15. D. Ilie, D. Erman, A. Popescu, and A. A. Nilsson, "Traffic measurements of P2P systems," in *Proceedings of SNCNW*, Karlstad, Sweden, Nov. 2004, pp. 25–29.

16. D. Ilie, D. Erman, A. Popescu, and A. A. Nilsson, "Measurement and analysis of Gnutella signaling traffic," in *Proceedings of IPSI*, Stockholm, Sweden, Sep. 2004.

17. D. Erman, D. Ilie, A. Popescu, and A. A Nilsson, "Measurement and Analysis of BitTorrent Signaling Traffic," in Proceedings of NTS, Oslo, Norway, Aug. 2004.

18. P. Pruthi, D. Ilie, and A. Popescu, "Application Level Performance of Multimedia Services," in Proceedings of SPIE International Conference on Quality of Service Issues Related to Internet, Boston, USA, Sep. 1999.

The origin of ORP can be traced back to an unpublished design document [46]. High-level details were presented for the first time at the 4th Euro-FGI workshop in Ghent and more details were included in Karel De Vogeleer's M.Sc. thesis "*QoS Routing in Overlay Networks* [47], for which I acted as main adviser.

Chapter 2

Graph Algorithms

The goal of this chapter is to introduce theoretical elements, terminology and notation that will be used in the reminder of the thesis. In the first part of the chapter we focus on graph theory and graph algorithms. QoS with emphasis on QoS routing is the subject of the second part of the chapter.

2.1 Definitions and Notation

Routing and network flow problems can be defined rigorously using graph theory notation. This allows in turn concise, non-ambiguous specification of algorithms that can solve such problems. We use therefore this opportunity to introduce basic graph theory definitions.

Definition 2.1 (Undirected graph)**.** An *undirected graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$ consists of a nonempty set $\mathcal{V}$ of *vertices* (also called *nodes*) and a collection $\mathcal{E}$ of pairs of distinct vertices from the set $\mathcal{V}$. The elements of $\mathcal{E}$ are called *edges*, *links* or *arcs*. In an undirected graph the edge (u, v) between node u and node v is indistinguishable from the edge (v, u). □

Sometimes, in the interest of brevity we write $\mathcal{G}$ instead of $\mathcal{G}(\mathcal{V}, \mathcal{E})$. We denote by the V number of vertices in $\mathcal{G}$ and similarly we denote by E the number of graph edges.

Definition 2.2 (Directed graph)**.** In the case of a *directed graph* (also called

digraph) the edges (u, v) and (v, u) are distinct [48–50]. For the directed edge (u, v) we say that the edge is *outgoing* from the perspective of node u and *incoming* for node v. □

In a computer network vertices represent *hosts*, also called *nodes*, while edges represent communication *links* connecting two hosts. Since various network traffic characteristics are dependent upon the direction in which the traffic flows, we focus exclusively on digraphs. An undirected graph can be converted to a digraph by replacing each undirected link with a pair of directed links, each of them pointing in the opposite direction of the other.

If (u, v) is an edge in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ then we say that edge (u, v) is *incident* to node u and v. Additionally, we say that u and v are *adjacent nodes* (or *neighbors*).

The number of outgoing edges (u, v) is called the *outdegree* of node u. Similarly, the *indegree* of node v is defined as the number of incoming edges at v from various nodes u. When edge direction is not relevant, the term *degree* denotes the number d_i of links associated with a node u.

A graph has two basic forms of representation. *Adjacency-list* representation consists of an array of elements $Adj[v]$, one for each vertex v in the graph. Each element $Adj[v]$ is a list consisting of nodes adjacent to v. In *adjacency-matrix* representation the graph is represented by a $V \times V$ matrix A, where each element $a_{u,v}$ is equal to one if the nodes u and v are adjacent (*i. e.*, the edge $(u, v) \in \mathcal{E}$) and zero otherwise. The adjacency list representation is preferred when the graph is *sparse* (*i. e.*, when having a graph where $V^2 \gg E$) since it requires less memory than the adjacency-matrix representation. For a dense graph (*i. e.*, a graph where $V^2 \approx E$) the adjacency-matrix representation tends to be more computationally efficient when searching for the existence of an edge (u, v) in the graph, but has higher memory requirements [51].

Definition 2.3 (Weighted graph)**.** In a *weighted graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$ all edges have an associated number $w \in \mathbb{R}$ called the *weight*, which represents a metric of interest, *e. g.*, cost, bandwidth, delay. Clearly, if we consider $n > 1$ metrics simultaneously, the weight is a vector $\boldsymbol{w} = [w_1, \dots w_n]$. The link weights in a weighted graph can be represented by a symmetric matrix $\boldsymbol{W} = \left[\boldsymbol{w}_{u,v}\right]$, where $\boldsymbol{w}_{u,v}$ is set to a suitable value (*e. g.*, 0 or ∞) if there is no edge (u, v) in E. □

We use the terms *graph* and *topology* to denote the same thing, which is

a complete network description that includes nodes and links along with additional properties.

Definition 2.4 (Path)**.** A *path* $P(v_1, v_k)$ in a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a sequence of vertices $(v_1, v_2, \ldots, v_k)$ with $k \geq 2$. This definition is equivalent to saying that the path P is a sequence of $(k-1)$ links $(e_1, \ldots, e_{k-1})$. The number of edges in a path P defines the *length* of the path, which is $(k-1)$ in this case. □

A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is said to be *connected* if, for each pair of vertices $u, v \in \mathcal{V}$ and $u \neq v$, there is a path $P(u, v)$. If each vertex pair is connected also by a path $P(v, u)$, the graph is said to be *strongly connected*.

In a path $P(v_1, v_k)$, the node v_1 is called the *source* or *origin* node and v_k is called the *destination* node. For a node v_i in P, all nodes $\{v_j : 1 \leq j < i\}$ (if any) are called *upstream* nodes and all nodes $\{v_m : i < m \leq k\}$ (if any) are called *downstream* nodes.

A *simple path* $P(v_1, v_k)$ is a path without loops (cycles), meaning that each element in the sequence $(v_1, v_2, \ldots, v_k)$ is distinct.

We denote a path by a single italic letter P when the node sequence is implicitly defined or when it is irrelevant to the context.

Definition 2.5 (Characteristic path length)**.** The characteristic path length, L, of the graph $\mathcal{G}$ is the number of edges in the shortest path between two nodes, averaged[1] over all pairs of nodes in the graph [53, 54]. □

Definition 2.6 (Clustering coefficient)**.** Let a node i have k neighbours. These neighbours share at most $k(k-1)/2$ links. Denote by C_i the fraction of links that actually exist. The clustering coefficient, C, of the graph $\mathcal{G}$ is the average value of C_i taken over all nodes with degree larger than one [53, 54]. □

Definition 2.7 (Node rank)**.** The rank r_i of a node v is the index of the node v in a node list sorted by node degree in decreasing order. □

[1] In [52], L is defined as the median of the means of the shortest paths from each node to the other nodes in the graph.

2.2 Network Models

A network model is a set of rules describing network properties such as vertex degree, edge length, clustering factor and growth. Typically, the purpose of a network model is to mimic characteristics from existing networks. However, a good network model may also lead to a better understanding of processes responsible for the formation and development of the network.

Some of the early efforts to create realistic network are based on random graphs following the Erdös-Rény model [55]. In its simplest form, the model specifies that two nodes in a undirected graph $\mathcal{G}$ are connected with a probability p. Thus, such a graph has on average $p\frac{V(V-1)}{2}$ edges, where V is the number of vertices [55]. In *Waxman* graphs [56] the probability p is inversely proportional to the Euclidean distance λ between the two nodes, such that

$$p(\lambda) \propto \beta \exp^{-\frac{\lambda}{\Lambda\alpha}} \tag{2.1}$$

where Λ denotes the maximum distance between two nodes in the graph and $0 < \alpha, \beta \leq 1$.

Although appealing for their simplicity, models based on random graphs are unable to capture non-random structural characteristics observed in the Internet, such as routing locality and hierarchy. The phenomenon of routing locality appears because the path between two nodes in a routing domain is confined entirely to that domain. Routing domains are either stub domains or transit domains, which imposes a two-level hierarchy on the nodes in the graph [57]. The desire to incorporate these characteristics into generated topologies led to the development of structural topology generators, as for example the Georgia Tech - Internetwork Topology Models (GT-ITM) generator [58].

More recent research [59–61] on Internet topologies concludes that some specific topological elements are better described by power-law distributions, also known as heavy-tail distributions. For example, results from this research indicate that the degree d_v of a node v is proportional to its rank r_v to the power of a constant R:

$$d_v \propto r_v^R \tag{2.2}$$

Similarly, the total number of pairs of nodes $\mathsf{P}(h)$ within h hops is proportional

to the power of a constant H:

$$\mathsf{P}(h) \propto h^H, \quad h \ll \delta \tag{2.3}$$

where δ is the diameter of the graph.

Barabási and Albert [62] showed that two generic mechanisms can be hold responsible for the appearance of power-law distributions: *incremental growth* and *preferential connectivity*. Incremental growth refers to continuous expansion of the network by adding new nodes to existing ones. This is in sharp contrast to Erdös-Rény and Waxman networks, where the number of nodes is kept unchanged and new links are added or old links are rewired. On the other hand, preferential connectivity denotes the tendency of new nodes to be connected to existing high-degree nodes [60].

Incremental growth and preferential connectivity together lead to the appearance of what is called *small-world* networks [53]. These networks are characterized by short paths lengths between arbitrary pairs of nodes and by strong clustering behavior. Compared to random graphs, small-world graphs tend to have shorter characteristic path length and a much larger clustering coefficient (see Def. 2.5 and Def. 2.6 in Section 2.1) [54].

We have used the BRITE [63] software to generate network models according to the Barabási-Albert model. BRITE is a topology generator developed at Boston University, designed to be flexible, extensible, interoperable, portable and user friendly. We have chosen BRITE because:

i) it has supports for realistic topology models based on power-law distributions,

ii) it can generate router level topologies,

iii) it is supported under OMNeT++, and

iv) the source code is freely available.

Each BRITE topology is embedded on a two-dimensional Euclidean plane divided into $HS \times HS$ high-level squares, where HS is a configurable BRITE parameter. Each high-level square is divided into $LS \times LS$ low-level squares, where LS is also configurable. A low-level square can be occupied by at most

one node. BRITE has two modes of laying out nodes: *random* and *heavy-tailed*. In random mode, BRITE assigns each node to a random low-level square while avoiding collisions. In heavy-tailed mode, BRITE selects the number of nodes in a high-level square according to the bounded Pareto probability density function [60]

$$f(x) = \frac{a\kappa x^{-a-1}}{1-(\kappa/P)^a}. \tag{2.4}$$

These nodes are then distributed randomly within the high-level square, such that each node occupies exactly one low-level square.

We have configured BRITE to use both incremental growth and preferential connectivity. With this configuration, BRITE starts with an initial set of m_0 randomly connected nodes[2]. The remaining nodes are added to the graph, one by one. Each of these nodes selects an existing node u with probability

$$\frac{d_u}{\sum_{v\in\mathcal{C}} d_v} \tag{2.5}$$

where $\mathcal{C}$ is the set of nodes already added to the network and d_i and d_v are the outdegree of node u and v, respectively. This process is repeated m times to connect node u to m other nodes. The parameter m is configurable.

2.3 Algorithms

Many problems can be converted to analytical expressions suitable for direct calculation. However, a significant amount of problems are either too large or too complex to be solved analytically. In this case it is more reasonable to attempt a computer-based *algorithmic* approach to find the solution.

Definition 2.8 (Algorithm)**.** An algorithm is a well-defined step-by-step procedure to solve a problem [51, 64, 65].

We differentiate between a *problem* and a *problem instance*. The first case denotes a general inquiry with some parameters left unspecified (*e. g.*, "What is the shortest path between two nodes in a graph?"). An instance of this problem requires complete specification of the nodes, edges and edge weights contained

[2] In BRITE's source code $m_0 = m$.

in the graph as well as specification of the two nodes we are interested in. An algorithm can solve either all instances of a problem or only some subset of them. For example, Dijkstra's shortest path algorithm can solve only problem instances where all edge weights are non-negative.

For computer algorithms, the problem instance must be encoded into a string ω that serves as input to the algorithm. In the case of problem instances with parameters defined on a continuous space, the encoding process involves input conversion to a discrete space. In some cases the conversion can introduce an error called *discretization error*. It may be possible to reduce the discretization error by making the conversion more "fine-grained", albeit at an increased computational cost.

The general form of an algorithm is

$$x = f(\omega) \tag{2.6}$$

where f denotes the algorithm and x is the result.

Algorithms can often be classified either as *direct methods* or as *iterative methods*. A direct method finds the solution in a finite number of steps whereas an iterative method converges asymptotically to the solution. Typically, an iterative method searches some space $\mathcal{X}$ defined by ω in order to find the solution. The simplex algorithm and Newton's method for unconstrained optimization [66] are examples of a direct and a iterative method, respectively.

An iterative method can be specified as

$$x_{k+1} = f(x_k) \quad \text{for } x_k \in \mathcal{X} \tag{2.7}$$

where $\mathcal{X}$ is the space to be explored by the algorithm, x_k denotes the position in the search space at step k and the algorithm f is a mapping from $\mathcal{X}$ to $\mathcal{X}$. When applied to a vector $\boldsymbol{x} \in \mathcal{X}$, f produces another vector $\boldsymbol{y} \in \mathcal{X}$.

Another way to classify algorithms is to place them in one of the following four classes:

i) numerical methods,

ii) exact algorithms,

iii) heuristics,

iv) meta-heuristics.

Numerical methods solve problems by using function approximation, finite differential calculus or a combination thereof [67, 68]. For example, Newton's method for unconstrained optimization approximates a function g in the neighbourhood of a point x with the polynomial generated by the second-degree Taylor series [66]. The first and second order derivatives from the series allow Newton's method to estimate the direction towards the optimum. Numerical methods require that certain conditions apply to functions involved in solving the problem or else the methods may not converge to the solution [69]. They are susceptible to round-off and truncation errors and to stability problems related to the feedback loop in Equation 2.7 [67]. It is worth noting that algorithms in the remaining three classes may be susceptible to these problems as well. The steepest descent and Newton's method are examples of numerical methods for unconstrained optimization [66].

Exact algorithms are iterative procedures that always find the correct solution, provided there is one. They are different from numerical methods in the sense that they do not require function approximation or finite differential calculus, but rely instead on properties specific to the problem they solve. For example, the Bellman-Ford shortest path algorithm and Dijkstra's shortest path algorithm, both described in Section 2.5, are exact algorithms that rely on the property that sub-paths of shortest paths in one dimension are also shortest paths [55].

If numerical methods or exact algorithms do not work well on the problem at hand, it may be possible to apply a heuristic [69]. Heuristics are algorithms that explore the search space in an intelligent way, albeit without guarantees for convergence to the correct solution. Often, they involve a trade-off between computation time and accuracy: fast heuristics sometimes cannot find the optimal solution and accurate heuristics always find the optimal solution, but for some problem instances they can take an unreasonable amount of time to finish. Local search [50, 70] is an example of optimization heuristic.

Metaheurstics are algorithms that combine various heuristics in an effort to obtain an approximate solution even in the case of difficult problem instances [69, 71]. They often employ a probabilistic element in order to avoid being trapped in a local minimum. The particle swarm optimization (PSO) method

described in [72] is an example of metaheuristic.

Direct methods find the solution to a problem instance in a finite number of steps, but iterative algorithms require some form of *convergence criteria*. Intuitively, we say that an algorithm has converged when $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ or when $f(\boldsymbol{x}_{k+1}) = f(\boldsymbol{x}_k)$. However, this may not happen at all when the algorithm is executed on a computer. Since computers are finite-state machines, they have finite precision in representing real numbers *i. e.*, they allocate a finite number of bits to represent numbers. This implies that numbers are rounded off, which leads to round-off errors. In practice, it means that the solution found by the algorithm approaches the true solution $\boldsymbol{x}^*$ in an ϵ-neighborhood dictated by the unit round-off (machine precision) for floating-point numbers. Therefore, the convergence criteria should be of one of the formulas shown below [72]:

1. $f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) < \epsilon(1 + |f(\boldsymbol{x}_{k+1})|)$,

2. $\|\boldsymbol{x}_k - \boldsymbol{x}_{k+1}\| < \sqrt{\epsilon}(1 + \|\boldsymbol{x}_{k+1}\|)$,

3. $\|\nabla f(\boldsymbol{x}_{k+1})\| < \epsilon^{1/3}(1 + |f(\boldsymbol{x}_{k+1})|$.

The unit round-off is defined as the difference between 1 and the least value greater than 1 on a specific computer architecture. On a 32-bit Pentium/Athlon architecture a `float` data type uses 32 bits with 1.19209×10^{-7} unit round-off, while a `double` uses 64 bits with 2.22045×10^{-16} unit round-off.

The use of the name *iterative methods (algorithms)* is perhaps unfortunate since direct methods can rely on iterations as well. The difference is that direct methods find the *exact* solution, whereas iterative methods find the solution in the limit. Nonetheless, the name is well established and we will continue using it.

Algorithms require also a *stopping condition*. Obviously if the algorithm converges according to one of the criteria above, then it can stop. Otherwise, one needs an upper bound ζ on the number of steps k performed by the algorithm. If the $k = \zeta$ the algorithm stops. Clearly, the choice of ζ is very important since setting the value too low causes the algorithm to stop prematurely when problems do actually have a solution, whereas if the value is too high the algorithms will run for a long time, even when no solution exists.

2.4 Algorithm Efficiency

A critical aspect in using algorithms is understanding how efficient they are at finding the solution for the problem at hand. Information about their efficiency helps not only in estimating the computational resources required, but can also be used as a decision factor in selecting one out of several algorithms that can solve the same problem.

For direct methods the efficiency is typically expressed as a function of the input size and it is called *computational complexity*. The complexity refers either to the space (*e. g.*, memory) required to store the input data, to the time required to run the algorithm until a solution is found[3], or, in the case of a distributed system, to the communication volume required by the system to perform its function. In the reminder of this thesis we will focus on time complexity. The word "complexity" will therefore refer to "time complexity" unless stated otherwise.

The goal is to obtain a complexity estimate, which is unaffected by variations in underlying hardware and software (*i. e.*, the computer and operating system running the algorithm) or by variations in the contents of the input. This is achieved by assuming that the algorithm runs inside a mathematical model of a computer instead of a real computer. The mathematical model is typically a Turing machine or a random-access machine [51, 65, 73]. The details of these mathematical models are not directly significant for this thesis. It is sufficient to say that there are several variants of Turing machines, the most important ones being the *deterministic* Turing machine and the *non-deterministic* Turing machine. All computers in use today work according to the principles of a deterministic Turing machine or of a random-access machine. Non-deterministic Turing machines have the ability to clone themselves into multiple copies running simultaneously.

We are foremost interested in *worst-case* complexity, which is an asymptotic upper bound on the number of elementary operations required by the algorithm to complete. In general, the elementary operations include arithmetic operations, comparisons, jumps and subroutine calls. The asymptotic upper bound is a function of the size of the input ω. It is customary to make the simplifying

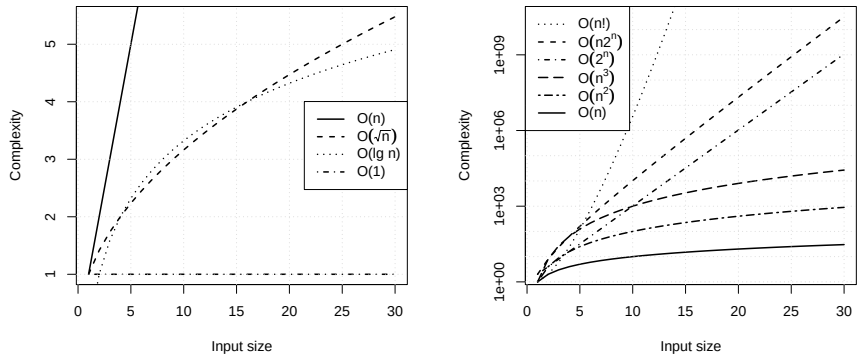
[3] We assume $\zeta = \infty$ in this case.

assumption that each elementary instruction, with the exception of subroutine calls, requires unit time for execution [50]. The time required by a subroutine call depends on the elementary instructions inside the subroutine.

Definition 2.9 (Asymptotic upper bound $\mathrm{O}(\cdot)$ for worst-case run time)**.** Given a function $f(n)$ that estimates the run time of an algorithm, where n is the size of the input ω, $f(n)$ has an asymptotic upper bound $\mathrm{O}(g(n))$ if $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ and provided that the positive constants c and n_0 exist [51]. In practice, $g(n)$ is obtained by removing from $f(n)$ the low order terms and the preceding constant of the high order term. □

If the algorithm scales as a constant, then it means that the algorithm is independent of the size of the input, and we write $\mathrm{O}(1)$. *Polynomial-time* algorithms are denoted by $\mathrm{O}(n^x)$, where x is an integer constant. These algorithms are considered *tractable* methods to solve the problem. On the other hand, algorithms belonging to the family of exponential time complexity (*e. g.* ,, $\mathrm{O}(n!)$ and $\mathrm{O}(k^n)$ for $k \in \mathbb{Z}$) are generally regarded as intractable since they tend to require prohibitive amounts of computation resources [50]. In Figure 2.1 we have plotted several types of asymptotic growth functions. The common element between Figure 2.1(a) and Figure 2.1(b) is the linear function $O(n)$, which acts as a border between the regions of sublinear complexity and superlinear complexity. Note that we intentionally use a logarithmic y-axis in Figure 2.1(b) in order to emphasize the growth explosion occurring with superlinear complexity.

Many problems can be rephrased in the form of a *decision* problem, that is a problem which has a "yes" or "no" answer. Most graph optimization problems are of this type. The set of decision problems can be divided into two classes, $\mathcal{P}$ and $\mathcal{NP}$. The class $\mathcal{P}$ contains decision problems that can be solved in polynomial time on a deterministic Turing machine. Similarly, the class $\mathcal{NP}$ contains decision problems that can be solved in polynomial time on a non-deterministic Turing machine using a non-deterministic algorithm [70]. An alternative definition states that the class $\mathcal{NP}$ contains decision problems, whose solution can be verified in polynomial time [50, 73]. Problems belonging to the class $\mathcal{NP}$ are generally regarded as intractable (*i. e.* , it is expected that algorithms solving them will have exponential worst-case complexity). One of the most interesting open questions in computer science is whether $\mathcal{P} = \mathcal{NP}$ or not. In the absence



(a) Sublinear complexity.

(b) Superlinear complexity.

Figure 2.1: Asymptotic worst-case complexity.

of a proof, the empirical evidence seems to indicate that this is not the case and that instead $\mathcal{P} \subseteq \mathcal{NP}$, as shown in Figure 2.2.

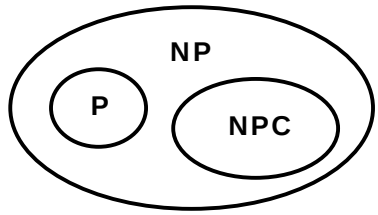


Figure 2.2: Complexity classes.

Some problems in $\mathcal{NP}$ are called *NP-complete* (NPC) because they hold a special status. If a polynomial-time algorithm is found to solve one of these problems, the theory states that all problems in the class $\mathcal{NP}$ will be solvable in polynomial time. Should this occur, it would constitute proof that our current view of the complexity classes is wrong, and in fact $\mathcal{P} = \mathcal{NP}$. The interested reader can find more information in [65, 74].

In terms of the problems described in Chapter 3, the multi-constrained path (MCP) and multi-constrained optimal path (MCOP) problems are NP-complete [75]. However, Kuipers and Van Mieghem have shown [76] that NP-complete behavior is unlikely to occur in realistic communication networks.

The concept of computational complexity is not easily applied to iterative methods [48]. In particular, the computational complexity can be strongly influenced by the convergence criteria used. Some results are provided in [77] for the case of optimization algorithms, although they require strict technical conditions (*e. g.* , Lipschitz-continuity) to apply to the objective function. For iterative algorithms it can be more interesting to examine the *rate of convergence*.

Definition 2.10 (Rate of convergence)**.** For an algorithm that converges through a series of intermediate steps $x_k \in \mathbb{R}$ to a point $x^* \in \mathbb{R}$ in the search space, the algorithm's rate of convergence is given by

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x^k - x^*\|^p} = \alpha \tag{2.8}$$

provided the number p exists and $\alpha \neq 0$. The number p is called *order of convergence*. When $p = 1$ the algorithm is said to have linear convergence [66, 78]. Sometimes linear convergence is also called geometric convergence. In general, the case $p > 1$ denotes superlinear convergence, whereas the specific case $p = 2$ is called quadratic convergence. □

2.5 Shortest-Path Algorithms

In this section we review various shortest-path algorithms. The selection is based on algorithms developed and used in the context of the ORP framework. More specifically, we describe Bellman-Ford and Dijkstra's shortest path algorithms, breadth-first-search (BFS) and Yen's K shortest paths (KSP). Most of the material and notation presented here is based on [51]. In some cases [64] was consulted as well. The description of Yen's KSP is based entirely on his article [79].

Definition 2.11 (Shortest-path problem)**.** Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be a weighted graph, where the scalar $w(u, v)$ denotes the weight of the link connecting node u and

node v. If there is no link connecting node u and v, then $w(u, v) = \infty$. Given a pair of nodes $s, d \in V$, solving the shortest-path problem requires finding a path P^* with the path weight

$$w(P^*) = \sum_{(u,v) \in P^*} w(u, v) \tag{2.9}$$

such that $w(P^*) \leq w(P)$, for all possible paths P. The path weight $w(P)$ is sometimes referred to as the distance between node s and node d over the path P. □

The algorithmic approaches for solving shortest-path problems can be divided into two categories: *label-setting* algorithms and *label-correcting* algorithms. The label denotes an estimated distance assigned to a node at some step in the algorithm. A label-setting algorithm assigns one label at each iteration and does not change it afterwards. This is the same as saying that the algorithm finds the shortest-path to a node at each iteration. A label-correcting algorithm can change assigned labels several times before the algorithm finishes. Dijkstra's shortest-path algorithm is a label-setting algorithm. The Bellman-Ford algorithm and BFS are label-correcting algorithms.

Yen's KSP algorithm does not belong to any labeling class. This algorithm belongs instead to a class of *deviation* algorithms used to solve ranking problems [80]. Solutions produced by deviation algorithms take the form of a tree where the root is the source node. Each branch of the tree constitutes a path from the source node to the destination node, obtained by a deviation in one of the previous paths, *e. g.*, the initial path used to bootstrap the algorithm. Yen's KSP relies internally on a shortest-path algorithm. Our implementation uses Dijkstra's shortest path algorithm.

Both Dijkstra's algorithm and the Bellman-Ford algorithm manage a list π with predecessor nodes and a second list d with distances to various vertices in the graph. Each list, when indexed by a node name (or other form of node identification), allows retrieval of or changes to the entry corresponding to the node. The purpose of the subroutine INITIALIZE shown in Algorithm 1 is to initialize these lists. The subroutine requires as input the graph corresponding to the problem instance, a source node s where from the shortest-path begins and the two (uninitialized) lists, π and d. INITIALIZE sets each entry in the

distance list to infinity, with the exception of the entry for the source node, which is set to zero. The predecessor list is cleared such that each element in it is marked unused (NIL).

Algorithm 1 Initialize.

Require: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, source node s, distance list d, predecessor list π

1: **for** each vertex $v \in \mathcal{V}$ **do**
2: $\quad d[v] \leftarrow \infty$
3: $\quad \pi[v] \leftarrow$ NIL
4: **end for**
5: $d[s] \leftarrow 0$

Algorithm 2 Relax.

Require: Link (u, v), link weight $w(u, v)$, distance list d, predecessor list π

1: **if** $d[v] > d[u] + w(u, v)$ **then**
2: $\quad d[v] \leftarrow d[u] + w(u, v)$
3: $\quad \pi[v] \leftarrow u$
4: **end if**

In addition to the INITIALIZE subroutine, Bellman-Ford and Dijkstra's algorithm share a RELAX subroutine shown in Algorithm 2. This subroutine implements a technique called *edge relaxation*. The purpose of edge relaxation is to test if the distance to a node decreases by inclusion of the link (u, v) into the path. If that is the case, the algorithm updates the distance and predecessor list, respectively. The performance of Dijkstra's and Bellman-Ford algorithms is directly related to how often they perform edge relaxation.

2.5.1 The Bellman-Ford Algorithm

The Bellman-Ford algorithm shown in Algorithm 3 was devised independently by Richard Bellman and Lestor R. Ford Jr, using different methods [81, 82]. Bellman solved the problem using dynamic programming, while Ford used a system of inequality equations.

The input to the algorithm consists of a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and a source node s. The algorithm computes the shortest-path from the source node to every other

node $i \in \mathcal{V}$. It begins by initializing the predecessor and distance lists according to Algorithm 1. Then, for each node in the graph, it loops through all edges calling Algorithm 2 to perform edge relaxation.

When the algorithm reaches line 8 the distance list contains a shortest-path weight entry for each node. The actual path can be obtained by recursive indexing of the predecessor list. For example, for a shortest path $v_0, v_1, \ldots, v_{k-1}v_k$, $\pi[v_k]$ returns the node v_{k-1}, and $\pi[\pi[v_k]] = \pi[v_{k-1}]$ supplies the vertex v_{k-2}. The recursion continues until the predecessor list returns node v_0, thus providing the complete path, or NIL in the case that no path between v_0 and v_k exists.

Algorithm 3 The Bellman-Ford algorithm.

Require: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, source node s

```
1: INITIALIZE(G, s, d, π)
2: V ← |V|
3: for i ∈ V do
4:     for each link (u, v) ∈ E do
5:         RELAX((u, v), w(u, v), d, π)
6:     end for
7: end for
8: for each link (u, v) ∈ E do
9:     if d[v] > d[u] + w(u, v) then
10:        ERROR: Negative-weight cycle discovered
11:    end if
12: end for
```

The Bellman-Ford algorithm is capable of handling graphs with negative weights and positive-weight cycles. The purpose of the algorithm's last `for`-loop is to check that, indeed, the graph contains no negative-weight cycles. The entries in the distance list are shortest-path weights or ∞ in the case no path exists. Therefore, the condition on line 9 must be false when no negative-weight cycle exists (*c. f.*, line 1 in Algorithm 2). The only situation in which the condition on line 9 holds is when negative-weight cycles do exist (see [51] for a formal proof).

The computational complexity of the Bellman-Ford is determined by the

number of vertices and edges in the graph and is in fact $\mathrm{O}(VE)$.

2.5.2 Dijkstra's Algorithm

Dijkstra's algorithm [83] is a different method for finding the shortest-path from a node s to the remaining nodes in the graph. It is less general than the Bellman-Ford algorithm since it requires non-negative weights for all edges of the graph. On the other hand it has better run-time performance when the number of edges in the graph is larger than the number of nodes.

The key element in the algorithm is the min-priority queue Q. The queue is used to store nodes indexed by their corresponding value in the distance list d, such that the node with the shortest distance value can be quickly retrieved.

Algorithm 4 Dijkstra's algorithm.

Require: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, source node s

```
1: INITIALIZE(G, s, d, π)
2: ENQUEUE(Q, V)
3: while Q ≠ ∅ do
4:     u ← EXTRACT-MIN-DISTANCE(Q)
5:     for each vertex v ∈ Adj[u] do
6:         RELAX((u, v), w(u, v), d, π)
7:     end for
8: end while
```

After the initialisation step on line 1, the algorithm adds all vertices from $\mathcal{V}$ to the queue Q. Then, the algorithm enters the `while` loop where it repeatedly removes from Q the vertex with minimum distance, until the queue becomes empty. For each extracted vertex it obtains the set of adjacent nodes $Adj[u]$, and performs edge relaxation for the edges connecting u to the nodes in $Adj[u]$[4].

Upon completion, the distances from vertex s to the other nodes in the graph are stored in the distance list d. The actual path can be obtained by recursive indexing of the predecessor list, as it was shown in the case of the Bellman-Ford

[4]The algorithm, as presented here, assumes the graph is represented using an adjacency list. In the case of an adjacency matrix, $Adj[u]$ can be easily computed by selecting the non-zero elements from row u of the matrix.

algorithm. Dijkstra's algorithm has $\mathrm{O}\left(V^2\right)$ computational complexity. It is possible to achieve a $\mathrm{O}\left(V \log_2 V + E\right)$ complexity if the algorithm is modified to use a more sophisticated min-priority queue based on Fibonacci heaps [84]. However, our implementation follows the original algorithm.

2.5.3 Breadth-First Search (BFS)

BFS is typically presented as a method to explore graphs rather than a traditional shortest-path algorithm. This is because BFS implicitly assumes that all links have equal weights, which can be seen on line 14 in Algorithm 5. Therefore, this method can be used only in the case where our interpretation of a shortest-path is that of a path with minimum number of hops (*i. e.*, a path with minimum length according to Definition 2.4 on page 19).

Algorithm 5 Breadth-first search (BFS).

Require: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, source node s

```
 1: for each vertex v ∈ V do
 2:     c[v] ← WHITE
 3:     d[v] ← ∞
 4:     π[v] ← NIL
 5: end for
 6: c[s] ← GREY
 7: d[s] ← 0
 8: ENQUEUE(Q, s)
 9: while Q ≠ ∅ do
10:     u ← DEQUEUE(Q)
11:     for each v ∈ Adj[u] do
12:         if c[v] = WHITE then
13:             c[v] ← GREY
14:             d[v] ← d[u] + 1
15:             π[v] ← u
16:             ENQUEUE(Q, n)
17:         end if
18:     end for
19: end while
```

In addition to predecessor and distance lists, BFS manages a color list as well. When the algorithm starts, all nodes are labeled WHITE, as it can be observed on line 2. WHITE nodes are nodes that have not been visited yet by the algorithm. When a node is visited, its label changes to GREY. BFS uses a queue Q also, but compared to the min-priority queue used by Dijkstra's algorithm, this is a regular first in first out (FIFO) queue. The EXTRACT-MIN-DISTANCE operation is therefore replaced by the DEQUEUE operation, which retrieves the element at the head of queue.

At the end of the initialization phase, lines 1–8, the node s has been labeled GREY and added to the queue and the algorithm proceeds to the discovery phase shown on lines 9–19. During the discovery phase, BFS dequeues one node at a time until the queue is emptied. The algorithm verifies if each adjacent node to the dequeued vertex has been visited before. If that is the case, the adjacent node is colored GREY, its entry in the distance and predecessor list are updated and then the node is added to the queue Q. Essentially, what BFS does is to discover at first all nodes located one hop away from s, then all nodes located two hops away from s and so on until the entire network has been explored.

Compared to the previous two algorithms, BFS is much more efficient. Its $\mathrm{O}(V + E)$ computational complexity makes it an ideal candidate for problems with minimum-length paths.

2.5.4 Yen's K Shortest Paths Algorithm

The optimization problems described in the next chapter are concerned with the distribution of a set of bandwidth demands over multiple paths. For large networks it becomes impractical to manually describe for each demand the paths connecting the source with the destination node. An algorithm that performs this task automatically is more efficient. This is where Yen's KSP algorithm comes into the picture.

Yen's algorithm can find up to K shortest paths between a source node and a destination node, such that path $(i-1)$ is shorter than path i for $1 < i \leq K$. Depending on the network topology in question, a node pair can be disconnected or connected by less than K paths. The algorithm handles correctly both special cases [79].

Algorithm 6 Yen's K shortest paths algorithm.

Require: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, source v_s and destination v_d, K

$P^1 \leftarrow$ DIJKSTRA($\mathcal{G}$, v_s, v_d)
for $k \leftarrow 2, \ldots, K$ **do**
    for $v_i \leftarrow v_s, v_2^{k-1}, \ldots, v_{d-1}^{k-1}$ **do**
        for $j \leftarrow 1, 2, \ldots, (k-1)$ **do**
            if $R_{v_i}^{k-1} \equiv R_{v_i}^{j}$ **then**
                $\mathrm{w}(v_i^j, v_{i+1}^j) \leftarrow \infty$ in the graph $\mathcal{G}$
            end if
        end for
        $S_{v_i}^k \leftarrow$ DIJKSTRA($\mathcal{G}$, v_i, v_d)
        if $S_{v_i}^k \neq \emptyset$ **then**
            $P_{v_i}^k \leftarrow R_{v_i}^{k-1} \bigcup S_{v_i}^k$
            $\mathcal{C} \bigcup P_{v_i}^k$
            if $\mathcal{C}$ contains $K - k + 1$ paths **then**
                break
            end if
        end if
    end for
    $P^k \leftarrow \min \mathcal{C}$
    restore all changed weights w to original values
end for

The pseudo-code for Yen's KSP is shown in Algorithm 6. The algorithm requires a graph description, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a source node v_s, a destination node v_d and the desired number of shortest paths, K. We denote the kth shortest path between v_s and v_d by P^k. Furthermore, the ith node (vertex) on the kth path is denoted by v_i^k. For $k > 1$, each path P^k consists of a root $R_{v_i}^k$ that extends from the source node v_s to an intermediate node v_i^k and of a spur $S_{v_i}^k$ from node v_i^k to the destination node v_d. A candidate path $P_{v_i}^k$ coincides with the $(k-1)$st shortest path, P^{k-1}, in nodes $v_s, \ldots, v_i^{k-1}$ and differs from the remaining portion of P^{k-1} that consists of nodes $v_i^{k-1}, \ldots, v_{d-1}^{k-1}, v_d$. Node v_{d-1}^{k-1} is the predecessor of the destination node on path P^{k-1}. The candidate path $P_{v_i}^k$ is also called a *deviation* from the path P^{k-1} at the node v_i.

The first shortest path P^1 is computed using Dijkstra's shortest path algorithm [83], as shown on line 1. Subsequent shortest paths are computed iteratively. The idea is to find new deviations, searching from the source node towards the destination node. The **for**-loop on line 3 computes the root $R_{v_i}^{k-1}$ of the shortest path P^{k-1}. On each loop iteration, the root is allowed to extend one node further towards the destination, up to the node v_{d-1} preceding the destination. During each iteration this root is compared to the roots of equal length for all shortest paths computed so far, as shown on lines 4–5.

If the compared roots consist of the exact same sequence of nodes, then the algorithm removes the link connecting the nodes v_i^j and v_{i+1}^j from the graph. This is shown on line 6, where the algorithm sets the link weight w to infinity. This has the effect of forcing Dijkstra's algorithm on the next line to find a new shortest path from v_i to v_d, different from $R_{v_i}^{k-1}$ (note that nodes v_i^j and v_i are the same at this point). If Dijkstra's algorithm is successful in finding a shortest path, then we concatenate the path to the root $R_{v_i}^{k-1}$ and save that in the list $\mathcal{C}$ with candidate paths, as shown on lines 11–12. Since it is not necessary to store more than $(K - k + 1)$ paths in the candidate list [79], we terminate the `for`-loop started on line 3 when the list grows to this size.

On line 18 Yen's KSP extracts the shortest path from the candidate list and this becomes our kth shortest path. Before starting a new iteration, the algorithm restores the weights in the graph as shown on the next line.

Yen's KSP algorithm has $\mathrm{O}\left(KV^3\right)$ worst-case time complexity. In the case when Dijkstra's algorithm uses Fibonacci heaps the complexity is reduced to

$\mathrm{O}\left(KV(V \log_2 V + E)\right)$ [80]. There have been reports [80, 85] of newer implementations that share the same worst-case complexity, but with the claim that in practice they perform better than Yen's KSP.

2.6 Summary

In this chapter we established graph theoretical notation and terminology. We provided also a brief overview of various graph models used to represent real networks and a short description of the BRITE topology generator. Furthermore, we discussed different classes of algorithms and methods used to quantify algorithmic efficiency.

In the final part of the chapter our attention was focused on shortest path algorithms, which are used in our ORP framework. We provided algorithmic formulation and worst-case run-time estimates for the Bellman-Ford algorithm, Dijkstra's algorithm, breadth-first search and Yen's K shortest paths algorithm.

Chapter 3

Optimization Algorithms

The ORP framework described in Chapter 5 relies on three different types of optimization algorithms:

i) path selection,

ii) K shortest paths,

iii) flow allocation.

The purpose of the path selection algorithm is to find a path from a source node s to a destination node d, subject to a number of QoS constraints. In the event that one or more paths are no longer able to satisfy the QoS constraints, the K shortest paths algorithm is used to discover multiple backup paths. The flow allocation algorithm is then used to distribute the affected flows over the backup paths, such that the traffic volumes are accommodated [86].

To quantify the time and memory requirements for this type of computations we have implemented a performance tested for network flow algorithms. Using this testbed we have analyzed the performance of Yen's K shortest paths (KSP) algorithm [79] as well as the performance of linear optimization methods (*i. e.*, the simplex method and the interior point method) in conjunction with the pure allocation problem (PAP) and the PAP with modified link-path formulation (PAP-MLPF) [70]. These problems are described in Section 3.2. The end goal in this chapter is to select, based on empirical data provided by the testbed, an optimization algorithm suitable for use with ORP.

In the next section we describe linear programming terminology and then proceed to outline optimization problems related to path selection and flow allocation. The remaining part of the chapter describes the performance testbed and the performance results obtained with it.

3.1 Linear Programming

The expression *linear programming* refers to the design and analysis of algorithms for solving linear optimization problems [87]. The *general form* of a linear optimization problem is shown in Table 3.1.

minimize	$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$	
subject to	$\boldsymbol{E}\boldsymbol{x} = \boldsymbol{p}$	(1)
	$\boldsymbol{G}\boldsymbol{x} \geq \boldsymbol{s}$	(2)
	$\boldsymbol{H}\boldsymbol{x} \leq \boldsymbol{t}$	(3)
	$x_i \in \mathbb{R}$ for $i = 1, \ldots, n$	

Table 3.1: Linear optimization problem in general form.

The function $f(\boldsymbol{x})$ is called the objective function. The unknown variables are denoted by the column vector $\boldsymbol{x}$ with n elements. The vector $\boldsymbol{c}^T$ holds the coefficients for the unknown variables. The matrix $\boldsymbol{E}$ contains the coefficients for the equality constraints. Similarly, the matrices $\boldsymbol{G}$ and $\boldsymbol{H}$ represent the coefficients for the inequality constraints. The vectors $\boldsymbol{p}$,$\boldsymbol{s}$ and $\boldsymbol{t}$ contain the constants on the right hand side of the equalities and inequalities, respectively. Solving this problem, implies finding an optimal point $\boldsymbol{x}^*$ (also called optimal solution) such that the optimal value $f(\boldsymbol{x^*})$ is a minimum and all constraints are satisfied. In the description of the linear problem any combination of the constraints (1)–(3) may appear. If there are no constraints of the type (1)–(3), then the linear problem is unconstrained.

Optimization algorithms (*e. g.*, the simplex method) require the problem description to be converted from the general form to the *standard form* shown in Table 3.2. To do that, the inequality constraints (2) and (3) are changed to equality constraints and are stored together with (1) in matrix $\boldsymbol{A}$ [66].

minimize	$f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}$
subject to	$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$
	$x_i \geq 0$ and $b_i > 0$ for $i = 1, \ldots, N$

Table 3.2: Linear optimization problem in standard form.

Inequality constraints can be changed to equality constraints by adding *slack* or *surplus* variables to the problem. For example, given the constraint row i in matrix $\boldsymbol{G}$

$$g_{i1}x_1 + g_{i2}x_2 + \cdots + g_{in}x_n \geq s_i \tag{3.1}$$

this row can be converted into standard form by subtracting from it a surplus variable y_1 such that

$$g_{i1}x_1 + g_{i2}x_2 + \cdots + g_{in}x_n - y_i = s_1. \tag{3.2}$$

There are just as many surplus variables as rows in the matrix $\boldsymbol{G}$. Similarly, for the constraints row i in matrix $\boldsymbol{H}$

$$h_{i1}x_1 + h_{i2}x_2 + \cdots + h_{in}x_n \leq t_i \tag{3.3}$$

we add a slack variable w_i on the right side of the equality sign such that

$$h_{i1}x_1 + h_{i2}x_2 + \cdots + h_{in}x_n + w_i = t_i. \tag{3.4}$$

Geometrically, the feasible region of the problem described in Table 3.2 is always a *convex polytope*, also called *simplex*. The vertices of the simplex constitute feasible solutions to the optimization problem. For a problem with n variables and m constraints there are

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \tag{3.5}$$

feasible solutions.

The simplex method, which is the classical algorithm for solving linear optimization problems, attempts to find the optimal solution by searching the vertices of the polytope in an efficient manner. The algorithm starts in one of the simplex vertices, and it evaluates how the objective function changes if it

were to "move" into one of the $(n-m)$ neighboring vertices. The move is always along the edges of the simplex. Although in practice the simplex method appears to have $\mathrm{O}(m+n)$ computational complexity, it can approach exponential complexity on some specific problems [50, 70, 87].

A newer class of optimization algorithms, called interior point methods (IPMs), has $\mathrm{O}(n \log 1/\epsilon)$ theoretical worst-case complexity, where $\epsilon > 0$ is a small tolerance factor. Contrary to the simplex method, IPMs approach the solution asymptotically from the interior or exterior of the polytope. Many practical IPM implementations, including the one used here, are based on techniques suggested by Mehrotra [88]. These techniques are aimed at improving the performance of the algorithm. Their side effect is that certain theoretical aspects of the algorithm are modified. Hence, it is not yet known if the complexity estimate apply to Mehrotra-based implementations [87].

3.2 Optimization Models

In this section we focus on optimization models that are used within the ORP framework to solve multi-constrained path (MCP) and multi-constrained optimal path (MCOP) selection problems [89] as well as flow allocation problems. Comprehensive surveys on additional types of optimization models and algorithms are available in [64, 69, 70].

The starting assumption is that information about network topology is available in the form of a weighted digraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The weight of each link represents a set of metrics of interest, such as bandwidth, delay, jitter, packet loss and cost. In addition to the graph and link weights, information about the *flow demands* is available as well. A flow demand is a set of *path constraints* for the path $P(s, d)$, where $s \in \mathcal{V}$ is the source node and $d \in \mathcal{V}$ is the destination (sink) node. In its simplest form, the flow demand contains only the bandwidth required to transfer data from s to d. In our implementation, the flow demands are tied to the direction of the path.

3.2.1 Multi-Constrained Path Selection

In the case of a MCP problem we attempt to find one constrained path at a time. This is a feasibility problem. Each link weight in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a vector of QoS metrics, where each metric belongs to one of the following types:

additive: delay, jitter, cost

multiplicative: packet loss

min-max: bandwidth, policy flags

Multiplicative weights can be turned into additive weights by taking the logarithm of their product. The constraints on min-max metrics can be dealt with by pruning the links of the graph that do not satisfy the constraints [89, 90]. Therefore, in the remainder of this section we focus on additive link weights only.

For $i = 1, \ldots, m$ we denote by $w_i(u, v)$ the ith additive metric for the link (u, v) between nodes u and v such that $(u, v) \in \mathcal{E}$. The MCP optimization problem for m additive constraint values L_i on the requested path is shown in Table 3.3.

find	path P
subject to	$w_i(P) = \sum_{(u,v)\in P} w_i(u, v) \leq L_i$ for $i = 1, \ldots, m$ and $(u, v) \in \mathcal{E}$

Table 3.3: Multi-constrained path selection problem (MCP).

The MCP selection problem problem can be converted to a multi-constrained optimal path (MCOP) selection problem by minimizing or maximizing over one of the metrics w_i. It is also possible to define a path-weight function f over all metrics [89, 90] and to optimize over the path-weight function itself, as shown in Table 3.4.

Wang and Crowcroft proved in [75] that MCP problems with two or more constraints are $\mathcal{NP}$-complete. By extension, MCOP problems with two or more constraints are $\mathcal{NP}$-complete as well. The apparent intractability of these problems suggests abandoning the search for exact solutions in the favour of heuris-

$$\begin{array}{ll} \text{minimize} & f\left(\boldsymbol{w}(P)\right) \\ \text{subject to} & w_i(P) = \sum\limits_{(u,v)\in P} w_i(u,v) \leq L_i \text{ for } i = 1,\ldots,m \text{ and } (u,v) \in \mathcal{E} \end{array}$$

Table 3.4: Multi-constrained optimal path selection problem (MCOP).

tics that have a better chance of running in polynomial time. Chen and Nahrstedt suggest a $\mathrm{O}\left(2L\right)$ heuristic [91] for the MCP problem, where L is the length of the feasible path. The path selection algorithm in ORP is based on this heuristic.

The results of a study [76] on the $\mathcal{NP}$-complexity of QoS routing found four conditions leading to its appearance:

i) graphs with long paths (large hop-count),

ii) link weights with infinite granularity, or excessively large or small link weights,

iii) strong negative correlation among link weights,

iv) "critically constrained" problems, which are problems with constraint values close to the center of the feasible region.

The authors of the study consider that these conditions are unlikely to occur in typical networks. If they are right, the consequence is that the exponential run time behavior of exact algorithms is bound to occur only in some few pathological cases.

3.2.2 Flow Allocation

In the flow allocation problem, it is assumed that we know about one or more directed paths connecting a source node s and a destination node d. These paths can be discovered automatically, for example with a K shortest paths algorithm. We consider the following type of optimization problems: given a digraph $\mathcal{G}(\mathcal{V},\mathcal{E})$, a set $\mathcal{P}$ of directed paths and a set $\mathcal{D}$ of flow demands for bandwidth, we would like to allocate bandwidth on the paths in $\mathcal{P}$ such as to simultaneously satisfy all demands.

If the traffic volume pertaining to a specific flow is allowed to be distributed over several paths to the destination, this is said to be a feasibility problem for *bifurcated flows*. This is the flow allocation problem that we are interested in.

On the other hand, if the problem includes the requirement that the entire traffic flow between two nodes must be transmitted on a single path, instead of being spread across several, we have a feasibility problem for *non-bifurcated flows*. This problem is known to be computationally intractable [70] for large networks (it is in fact $\mathcal{NP}$-complete). Hence, it will not be considered here.

We adopt a notation called *link-path formulation* [70] to formalize our problem statement. Using this notation, we let the variable x_{dp} denote bandwidth allocated to demand d on path p. Recall that a demand is a request for a specific amount of bandwidth, h_d, from a source node to a destination node. The source node and the destination node can be connected by more than one path, which explains the use of the index variable p. We use the variables D and E to denote the number of demands in the demand set $\mathcal{D}$ and the number of edges (links) in the set $\mathcal{E}$, respectively. Further, the capacity of a link e is denoted by c_e. The indicator variable δ_{edp} is defined as

$$\delta_{edp} = \begin{cases} 1 & \text{if link } e \text{ is used by demand } d \text{ on path } p, \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

Our problem statement can now be written as shown in Table 3.5.

find	x_{dp}	for all $d \in \mathcal{D}, p \in \mathcal{P}$
subject to	$\sum_p x_{dp} = h_d,$	$d = 1, 2, \ldots, D$
	$\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e,$	$e = 1, 2, \ldots, E$

Table 3.5: Pure allocation problem (PAP).

PAP is a linear feasibility problem. Finding a solution to it entails solving a system of linear equations of the type $\boldsymbol{Ax} = \boldsymbol{b}$. If the number of variables equals the number of equations we can use LU-decomposition or the conjugate gradient method [92] to solve the system of equations. If there are fewer equations than variables, then the system is *underdetermined* and has an infinity of solutions or no solution at all. In the first case, this type of systems can be solved

with singular value decomposition (SVD). *Overdetermined* systems of linear equations (*i. e.*, systems with more equations than variables) in general lack an exact solution. In this case, one can transform the system into a linear least-square optimization problem to obtain an approximation of the true solution. The goal of the least-square problem is to minimizes the residual error $\|\boldsymbol{Ax} - \boldsymbol{b}\|$ due to the approximation. The least square problem can be solved for example with SVD or with QR-decomposition [68, 93].

The first phase (phase I) of the simplex method is another approach for solving the system of linear equations. The purpose of phase I is to obtain an initial feasible solution from which the simplex method can start. This involves augmenting the original system $\boldsymbol{Ax} = \boldsymbol{b}$ with a set of artificial variables $\boldsymbol{y}$ and using the simplex method to solve the modified system $\boldsymbol{Ax} + \boldsymbol{y} = \boldsymbol{b}$ [66, 86, 87].

In [70], it is suggested that the PAP described in Table 3.5 can be reformulated in the form of the linear optimization problem shown in Table 3.6. The new problem, PAP with modified link-path formulation (PAP-MLPF), has an additional variable z to be modified. Unlike the PAP, this problem *always* has a feasible solution in the sense that a minimum value for z can be found. If $z < 0$ in the solution, we have a successful bandwidth allocation. Otherwise the value of z indicates how much additional bandwidth is required to obtain feasibility.

minimize	z	for all $d \in \mathcal{D}, p \in \mathcal{P}$
subject to	$\sum_p x_{dp} = h_d,$	$d = 1, 2, \ldots, D$
	$\sum_d \sum_p \delta_{edp} x_{dp} \leq z + c_e,$	$e = 1, 2, \ldots, E$

Table 3.6: PAP with modified link-path formulation (PAP-MLPF).

The PAP-MLPF can be solved with the linear programming algorithms presented in Section 3.1.

3.3 Performance Testbed

The performance testbed for network flow algorithms consists of five main parts:

i) network topology parser,

ii) flow demands module,

iii) graph data types and algorithms,

iv) optimization methods,

v) resource utilization counters.

Currently, the topology parser is able to read network graphs created with the BRITE [63] topology generator. BRITE uses a simple text-based file format, which allows users to define their own topologies using a text editor.

The flow demand module is used to either randomly generate flow demands or to read demands from a file. A flow demand consists of a source node identifier, a destination node identifier and a set of QoS values: bandwidth (*i. e.*, bit rate or throughput), delay and loss. The set of QoS values can be easily extended to incorporate other metrics. A collection of flow demands is tied to a specific topology since the source and destination nodes must have the same identifier as in the corresponding graph. For random flow demands the value of each component (*i. e.*, node identifiers and QoS parameters) is chosen according to an uniform distribution. We are using for this purpose the Mersenne-Twister random number generator [94] from the GNU Scientific Library (GSL) version 1.9 [95]. The user can choose to initialize the random number generator using the default seed or with a seed from device `/dev/urandom` [96]. The Linux kernel maintains an entropy pool containing environmental noise from device drivers and other sources. The entropy pool is used to create random numbers that can be accessed via `/dev/urandom`.

We have implemented a number of graph algorithms for performing various operations on `Graph` objects. The `Graph` object is used to store data imported from a BRITE topology file. Additionally, it provides various methods to obtain information about or modify the contents of `Graph` objects (*e. g.*, finding the link connecting a pair of nodes if one exists, adding and removing nodes and links). A summary of the implemented graph algorithms consists of: BFS, depth-first-search (DFS), topology closure, connectivity map, Bellman-Ford shortest-path, Dijkstra's shortest-path and Yen's KSP [51, 79, 81–83].

There can be several optimization algorithms that are able solve the same type of optimization problem. Furthermore, an optimization algorithm may

have different vendor implementations. The combination of these three factors (*i. e.*, problems, algorithms and implementations) is difficult to handle in a consistent way, since vendor implementations use different APIs and different input formats for the problem specification. We have designed a common *solver* interface in order to improve this situation. A solver is the encapsulation of an optimization problem description with the implementation of the optimization algorithm used to solve the problem. A solver has three functions that can be called by the program:

`init()`: loads the `Graph` object, the flow demands and a set of paths (*e. g.*, the three shortest paths for all node pairs) into the solver. The solver converts all this information into an optimization problem (*i. e.*, into a system of equations and a corresponding objective function)

`solve()`: runs the optimization algorithm on the optimization problem, extracts the solution, if one can be found, and returns the status to the caller

`clear()`: resets the solver and frees up the memory

We have developed solvers for the PAP and PAP-MLPF optimization problems [70] using the simplex and interior point methods from the GNU Linear Programming Toolkit (GLPK) version 4.28 [97]. Additional solvers can be integrated with the rest of the testbed provided that they implement the common interface.

The solvers have the ability to output a problem statement in detail and in Table 3.7 we show an example of it. The group of equality equations corresponds to the first constraint equation in Table 3.5 and ensures that for each demand the sum of bandwidth allocations on each path satisfies the demand. The set of inequality equations correspond to the second equation in Table 3.5 and their purpose is to ensure that bandwidth allocations do not exceed link capacity.

Since the solvers are intimately tied to the problem being solved, they automatically convert the problem from the general form shown in Table 3.7 to the standard form, adding slack variables when necessary. This leaves us in fact with a system of linear equations. We solve the system of linear equations using the phase I of the simplex algorithm. In GLPK this is achieved by setting the objective function equal to zero [97]. The same procedure works for GLPK's

find	x_{dp} for all $d \in \mathcal{D}, p \in \mathcal{P}$
subject to	$x_{0,0} + x_{0,1} + x_{0,2} = 289.401$
	$x_{1,0} + x_{1,1} + x_{1,2} = 496.613$
	$x_{2,0} + x_{2,1} + x_{2,2} = 553.005$
	$x_{1,2} \leq 9269.23$
	$x_{0,1} \leq 899.3$
	$x_{1,1} \leq 402.51$
	$x_{2,2} \leq 6661.46$
	$x_{0,2} \leq 2505.57$
	$x_{0,2} + x_{2,2} \leq 8457.03$
	$x_{1,1} + x_{2,0} \leq 902.17$
	$x_{0,0} + x_{1,0} + x_{1,2} + x_{2,1} \leq 213.26$
	$x_{0,1} \leq 5685.38$
	$x_{1,2} \leq 9323.55$
	$x_{0,0} \leq 3240.66$
	$x_{1,0} \leq 3220.41$
	$x_{2,2} \leq 5912.13$
	$x_{2,1} \leq 5482.41$

Table 3.7: PAP in detail for a network with 10 nodes.

implementation of the interior point method. This is the approach used by our solvers in handling feasibility problems.

The final part of the testbed, the resource utilization counters, are used to measure the run-time performance for graph algorithms and solvers. The performance is expressed in memory and central processing unit (CPU) time usage as reported by the operating system.

Memory usage is harvested using the `statm` node in the Linux `/proc` pseudo-filesystem [98]. This information is an aggregation of memory reservations within our own code and also of memory reserved by software libraries we link with, *e. g.*, GSL and GLPK libraries. The counters record memory usage among consecutive counter readings as well as the largest value since the counter was started.

CPU time usage is recorded using `getrusage` [99]. This system call returns the CPU time spent by the current process in kernel space and user space, respectively. The counters keep track of time usage between consecutive counter readings as well as the total time used since the counter was started.

In addition of CPU time the counters record absolute time durations (*i. e.*, wall clock time) from the `gettimeofday` system call [100]. The difference between the wall clock time and the total amount of CPU time is due to other processes running concurrently on the same host.

All main testbed components are developed in C++ and make heavy use of the C++ Standard Template Library (STL) [101, 102]. The testbed software is organized into a linkable library called `liboptim`. This facilitates integration at source code level with other applications (*e. g.*, simulators or network-based utilities) as we have done for ORP.

3.4 Experiment Setup and Results

The experiments described here were performed on a host equipped with an Intel Core2 2.0 GHz CPU with 4 MB cache memory and 2 GB RAM. The host was running the Gentoo Linux 32-bit operating system with kernel 2.6.24. The testbed and the required libraries were compiled with the GNU Compiler Collection (GCC) version 4.1.2 with optimization flag -O2. This is the form in which the libraries would be used by a "real-world" application. Consequently,

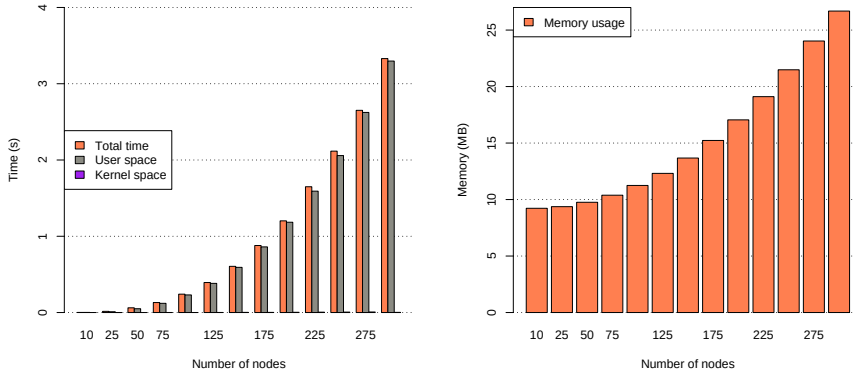
the results presented here are what should be expected in a "real-world" environment, when the same experiments are performed with similar network topologies, and on a host with similar hardware and software specifications.

We used BRITE to generate a set of network topologies with $10, 25, 50, 100, 125, \ldots, 300$ nodes. Apart from the number of nodes, BRITE was configured to generate one level "ROUTER (IP) ONLY" graphs using the Barabási-Albert (BA) model [62] with incremental growth type and preferential connectivity. The nodes were placed on a plane of size 10000×10000 divided into squares of size 100×100. We selected heavy-tailed node placement, which uses the bounded Pareto distribution to choose the number of nodes in a square. The number of links per node (*i. e.,* BRITE's parameter m) was set to 4. Furthermore, we selected uniform bandwidth distribution between 10 and 10000.

In BRITE all links in a graph are undirected. When we import a BRITE topology in the testbed each link is replaced by a pair of links, one in each direction.

After the topology is imported we run Yen's KSP algorithm to compute 3, 5, and 7 shortest paths, respectively, from each node to the other nodes in the graph. For example, in a graph with 100 nodes for each node we call Yen's KSP algorithm 99 times computing up to a maximum of $K \times 99$ paths. In our case $K = 3, 5, 7$ and when referring to a specific K instance of Yen's KSP we use the shorthand notation 3SP, 5SP and 7SP, respectively. Each time the testbed has computed the KSPs from a node to all other nodes we record the time spent in doing that as well as the memory usage.

When the algorithm has finished iterating over all nodes, we compute an average value of the computation time. The computed values are shown in Figure 3.1(a), Figure 3.2(a) and Figure 3.3(a). Each figure consist of a group of three bars. Each group corresponds to a network with a different number of nodes, increasing from the left to the right on the bar diagram. The height of each bar represents the average time duration in seconds. In each group, the first bar to the left corresponds to the measured wall-clock time. The bar in the middle denotes time spent in user space and is the actual time spent by the processor in performing the computation. The last bar shows the amount of time spent in kernel space. This occurs, for example, when I/O operations take place, such as when the testbed process logs data to a file. In the tests



(a) Computation time.

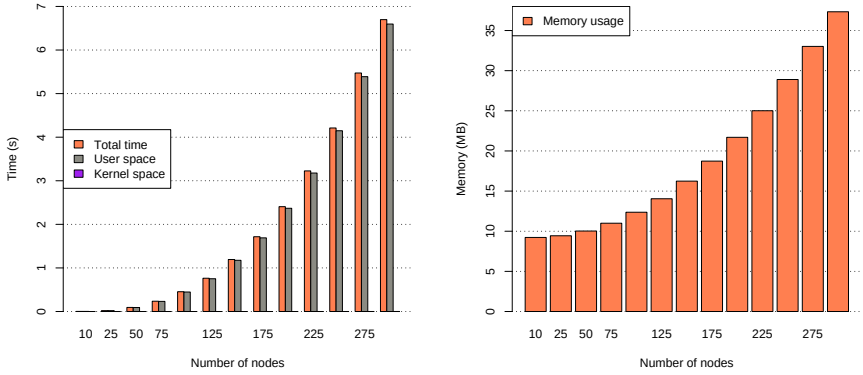
(b) Memory usage.

Figure 3.1: Yen's 3SPs.

considered here, kernel-space time is generally much lower than wall-clock time and user-space time. Consequently, in some of the graphs the bars corresponding to kernel-space time are hardly visible.

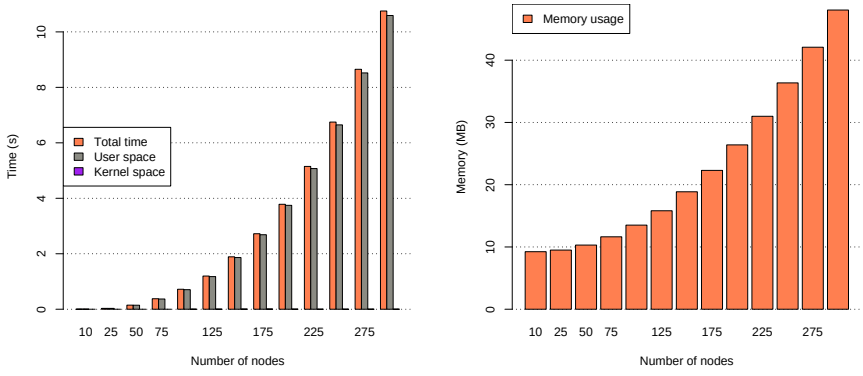
A rapid increase in computation time can be observed in all three figures. This is expected given the cubic worst-case time complexity of Yen's KSP. We have plotted in Figure 3.4 the average user-space time, on the left side using linear scale on the axes, and with logarithmic scale on the right side. The almost straight line in Figure 3.4(b) indicates a power-law relationship between the number of nodes and the user-space time required for computation.

It can be observed that, for a network of 300 nodes, the average user-space time required to compute 5SPs to 299 nodes is almost double the duration of that required by 3SPs (*i. e.*, ≈ 7 seconds versus ≈ 3.5 seconds). When comparing computation time between 5SPs and 7SPs, again for 300 nodes, it is observed that the 7SPs require only about 50 % more time than the 5SPs. This happens when, due to previous iterations, $(K - k + 1)$ shortest paths accumulate in the candidate list $\mathcal{C}$ maintained by the algorithm (see Algorithm 6 on page 36). When this occurs, the number of calls to Dijkstra's algorithm is reduced. In short, the algorithm performs more work in the early stages while it discovers



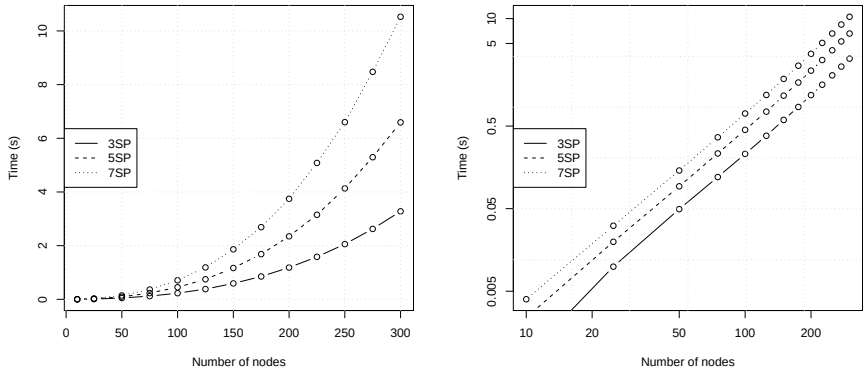
(a) Computation time. (b) Memory usage.

Figure 3.2: Yen's 5SPs.



(a) Computation time. (b) Memory usage.

Figure 3.3: Yen's 7SPs.



(a) Linear scale on the axes.

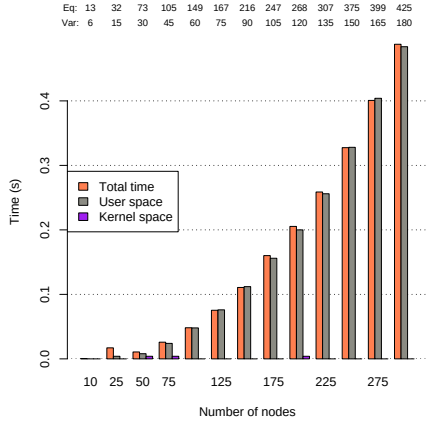
(b) Log-log scale on the axes.

Figure 3.4: KSP user-space time comparison.

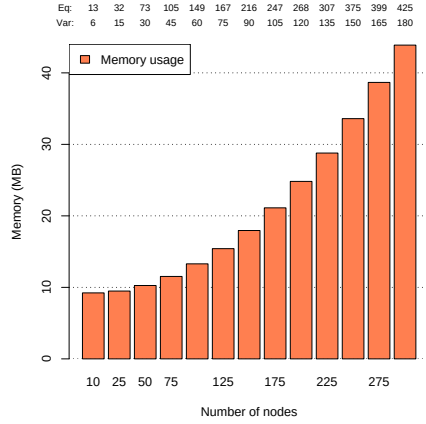
initial paths k, such that $1 \leq k \ll K$, than in the later stages, where for paths k such that $1 \ll k \leq K$, it can exploit knowledge from the earlier stages.

We now turn our attention to the corresponding memory usage shown in Figure 3.1(b), Figure 3.2(b), and Figure 3.3(b). Each bar in a figure shows the memory used for storing all KSPs for all nodes in the corresponding network, including memory used by libraries loaded by the testbed. The bar corresponding to a network of 100 nodes in Figure 3.1(b) indicates the memory used by the testbed process after we have computed a maximum of $3 \times 99 \times 100$ paths, while the same bar in Figure 3.2(b) and Figure 3.3(b) shows the memory usage after computing a maximum of $5 \times 99 \times 100$ paths and $7 \times 99 \times 100$ paths, respectively. The bar diagrams show a growth pattern similar to that of computation time. The memory usage for a network with 300 nodes increases with $\approx 40\,\%$ when going from 3SPs to 5SPs and with $\approx 30\,\%$ for the step from 5SPs to 7SPs.

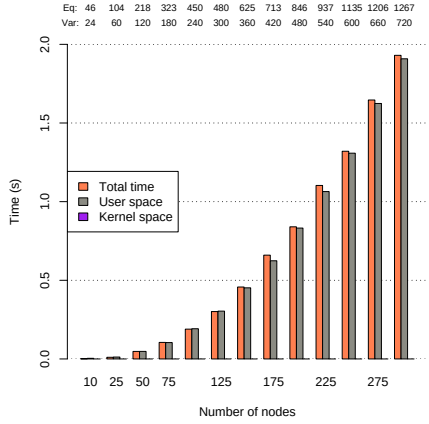
The next target in our performance study is the `init()` function in the solver interface. The GLPK API uses a data type called *problem object* to represent a problem statement. The problem object must be initialized with the number of rows and columns belonging to the problem statement, before the actual problem can be loaded. The number of rows includes any slack or



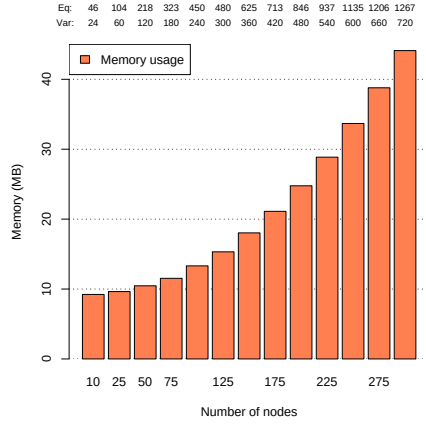
(a) Computation time, 20 % demands.



(b) Memory usage, 20 % demands.

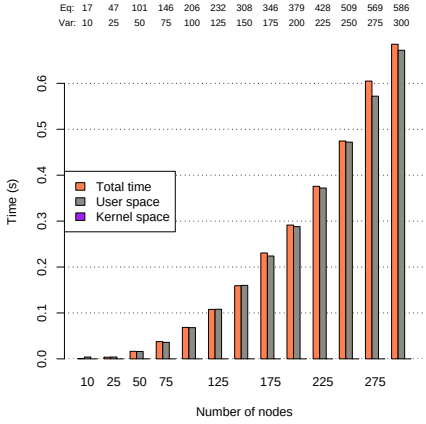


(c) Computation time, 80 % demands.

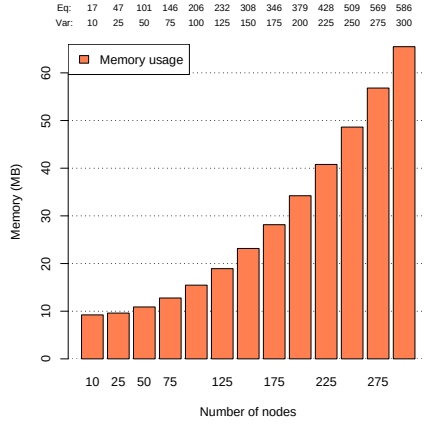


(d) Memory usage, 80 % demands.

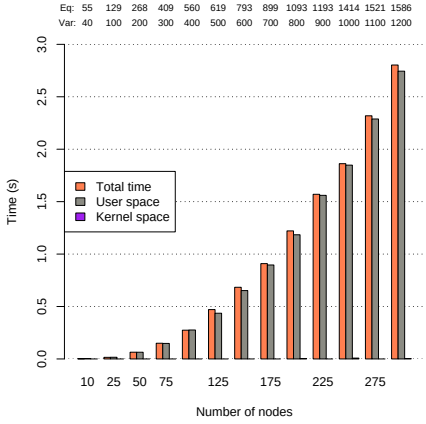
Figure 3.5: Solver `init()` subroutine with 3SPs.



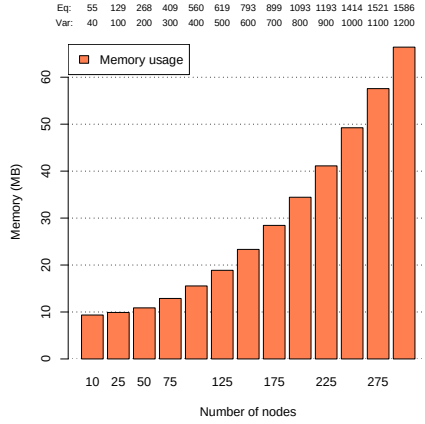
(a) Computation time, 20 % demands.



(b) Memory usage, 20 % demands.

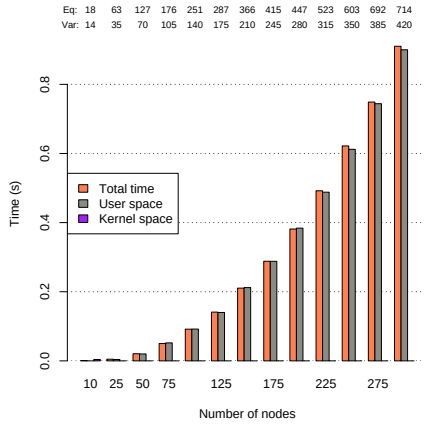


(c) Computation time, 80 % demands.

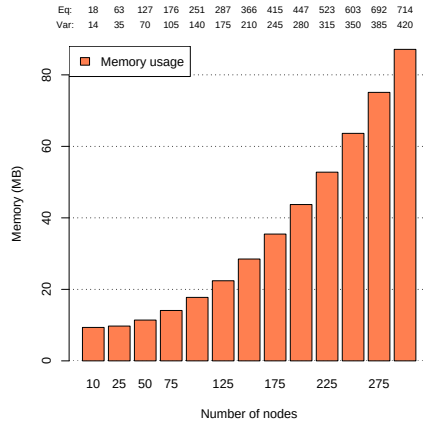


(d) Memory usage, 80 % demands.

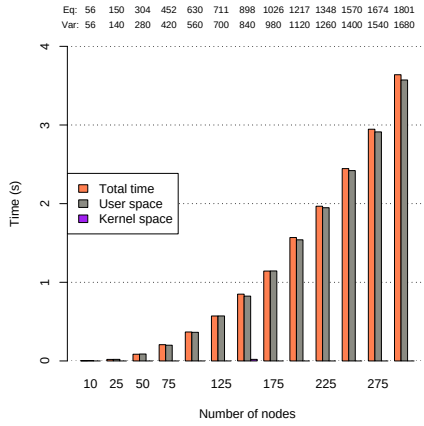
Figure 3.6: Solver `init()` subroutine with 5SPs.



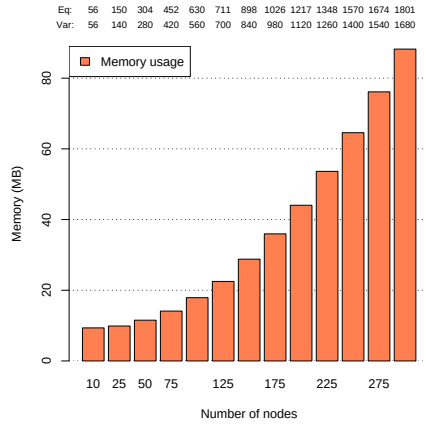
(a) Computation time, 20 % demands.



(b) Memory usage, 20 % demands.



(c) Computation time, 80 % demands.



(d) Memory usage, 80 % demands.

Figure 3.7: Solver `init()` subroutine with 7SPs.

surplus variables required to bring the problem into standard form. This allows GLPK to reserve enough memory for the problem object. Consequently, the `init()` function performs two passes. During the first pass we iterate through the flow demands and through the KSPs and compute the amount of memory needed. During the second pass we iterate again through the same items and load the variable coefficients, the constraint values and the objective function into the reserved memory. The time spent in `init()` is computed over both passes.

There are several factors that decide the number of equations and the number of variables in a problem statement:

i) amount of flow demands,

ii) number of paths associated with each flow demand,

iii) number of links traversed by the flow demand paths.

Item ii) can be controlled loosely by a suitable choice of K for Yen's KSP. Item iii) is indirectly decided by the choices in i) and iii). To test the influence of item i) on the problem statement we have created two scenarios corresponding roughly to a quiescent network and to a busy network, respectively. In the quiescent network scenario we create $\lceil 0.2V \rceil$ demands, where V is the number of nodes in the graph[1]. We call this the "20 % demands" scenario. In the busy network scenario we let the number of demands go up to $\lceil 0.8V \rceil$ and we call it the "80 % demands" scenario. For each demand, we randomly choose the bandwidth required, the source and the destination nodes. The only requirement is that the source and the destination nodes are distinct from each other. The bandwidth values are selected from a uniform distribution on the half-open segment $(0, 256]$.

Figures 3.5–3.7 show each a group of bar diagrams for the run-time behavior and memory usage of the `init()` function in the case of 3SPs, 5SPs and 7SPs, respectively. In each group we show the 20 % demands and 80 % demands scenarios on top of each other. At the top of each bar diagram there are two rows with numbers aligned to the bars. The first row shows the number of equations corresponding to the bar in question and the second line shows the number of variables. We want to emphasize that the number of variables on

[1]The notation $\lceil x \rceil$ denotes that x is rounded upwards to the nearest larger integer.

top of the diagrams does not include slack variables. To compute the number of slack variables we can use the property that for a PAP problem with n demands, n of the equations are equality equations for satisfying the demands, and the reminder are inequality equations for satisfying link capacity constraints. The inequality equations require a slack variable each. The largest system of linear equations encountered in our test is shown in Figure 3.7(c) and Figure 3.7(d) and contains 1801 equations and 1680 variables. In addition, this system includes also $1801 - (0.8 \times 300) = 1561$ slack variables.

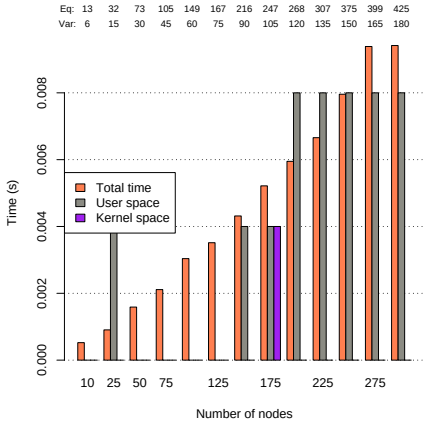
We can observe that the computation time for `init()` is directly proportional to the number of demands: when the number of demands grows four times (*i. e.*, from 20 % to 80 % demands) so does the computation time. The same relation holds between computation time and number of variables. This is expected because the number of variables is equal to the number of demands multiplied by the number of paths used by all demands. The BRITE topologies are strongly connected, which allows Yen's KSP to always find K paths between a pair of nodes. This keeps the number of paths per demand constant (*i. e.*, 3, 5, and 7).

The memory usage displayed in the diagrams is constituted of the following components:

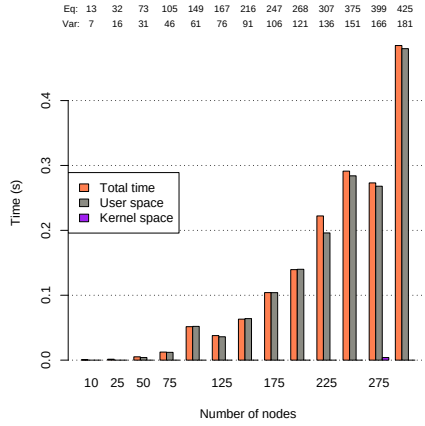
- memory used for storing all KSPs as shown in Figure 3.1(b), Figure 3.2(b), and Figure 3.3(b),
- memory reserved for all flow demands,
- memory reserved for GLPK's problem object.

In all diagrams the memory usage shows roughly 20 MB increase for networks with 250–300 nodes, when compared to the memory used for storing KSPs. For smaller networks the increase is smaller because the STL data types allocate a reasonable amount of memory from the start. Additionally, `/proc/statm` reports memory usage in terms 4 KB pages reserved by the Linux kernel. When the testbed is started a number of pages are automatically reserved, which can be enough to hold our data for small networks.

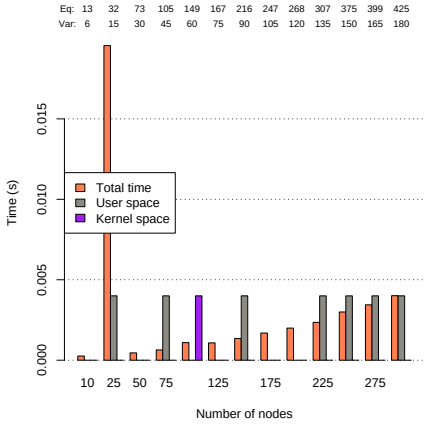
The next six groups of graphs, Figure 3.8–3.13, show the computation time required to solve the PAP and PAP-MLPF problems using the simplex method and the interior point method (IPM), respectively. Some of the bars displaying



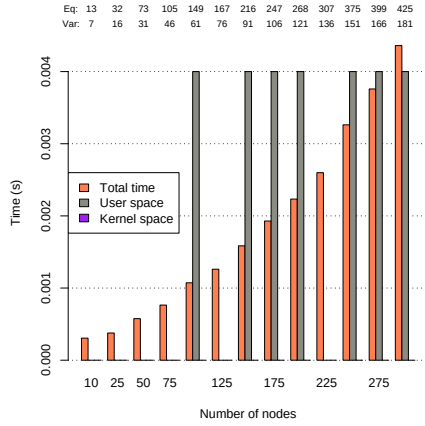
(a) PAP-IPM computation time.



(b) PAP-MLPF-IPM computation time.

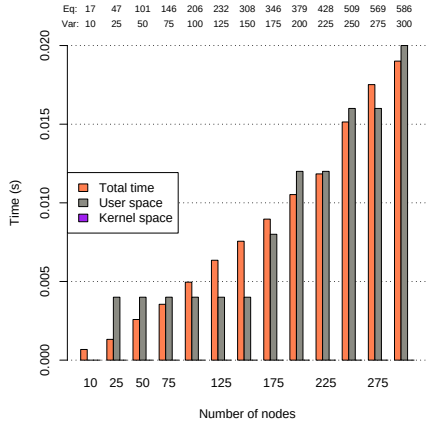


(c) PAP-SIMPLEX computation time.

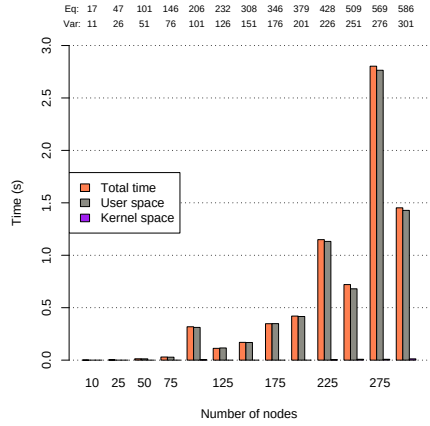


(d) PAP-MLPF-SIMPLEX computation time.

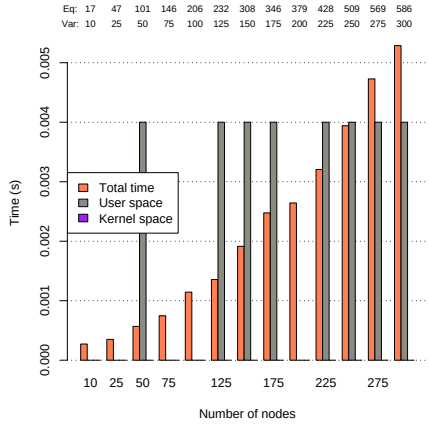
Figure 3.8: Solver `solve()` subroutine with 3SPs, 20 % demands.



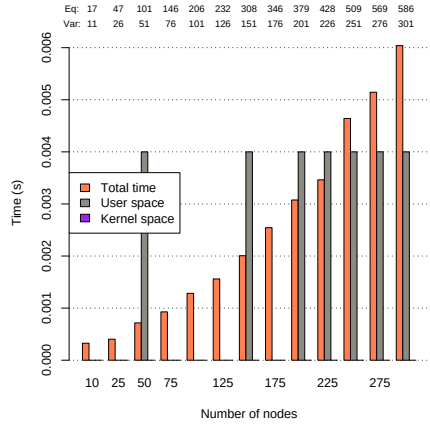
(a) PAP-IPM computation time.



(b) PAP-MLPF-IPM computation time.

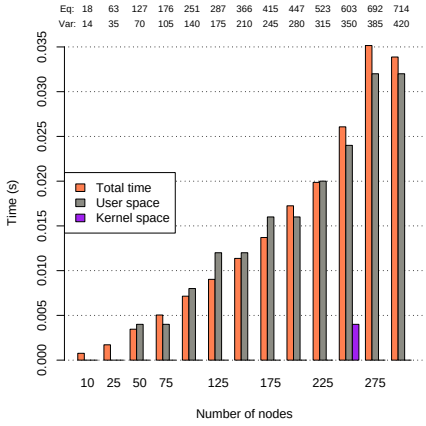


(c) PAP-SIMPLEX computation time.

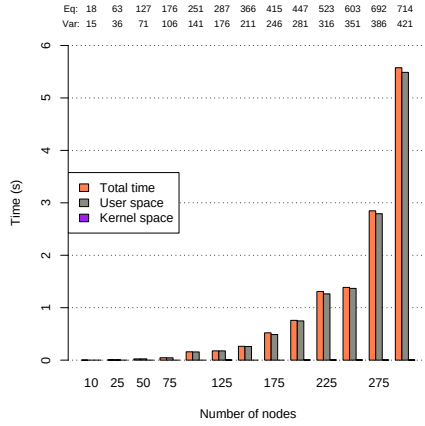


(d) PAP-MLPF-SIMPLEX computation time.

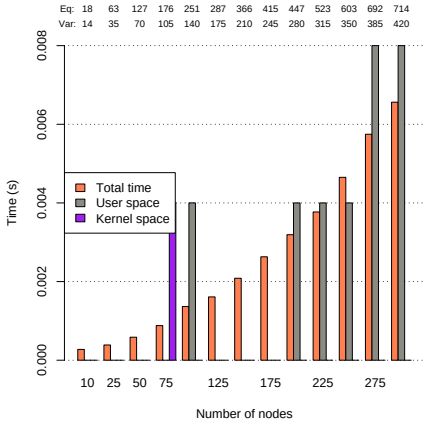
Figure 3.9: Solver `solve()` subroutine with 5SPs, 20 % demands.



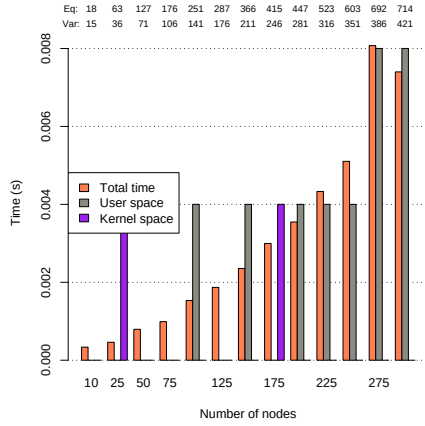
(a) PAP-IPM computation time.



(b) PAP-MLPF-IPM computation time.

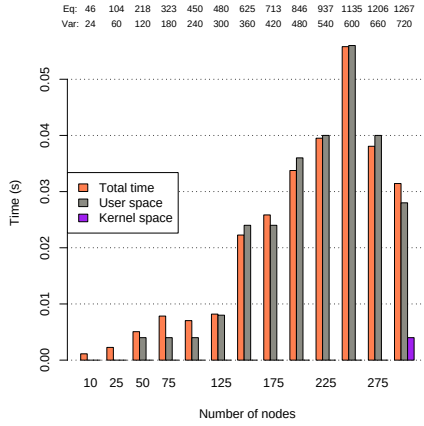


(c) PAP-SIMPLEX computation time.

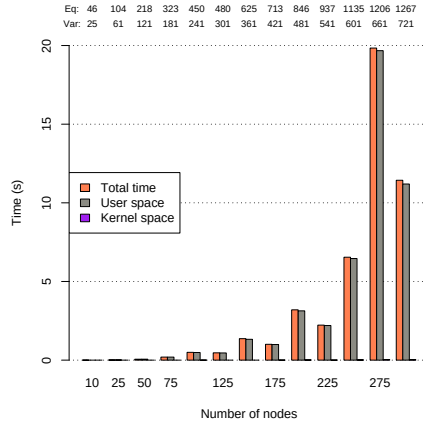


(d) PAP-MLPF-SIMPLEX computation time.

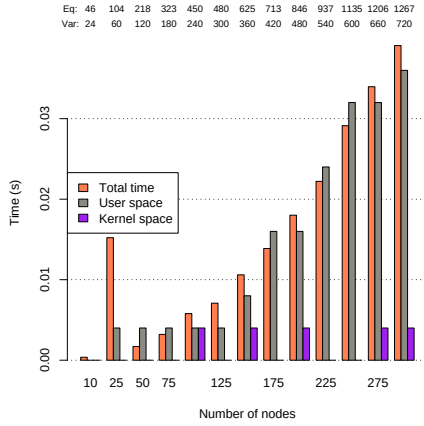
Figure 3.10: Solver `solve()` subroutine with 7SPs, 20 % demands.



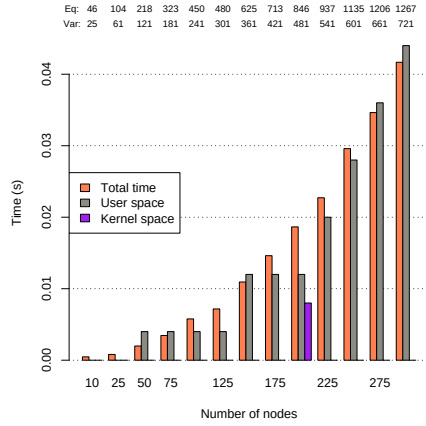
(a) PAP-IPM computation time.



(b) PAP-MLPF-IPM computation time.

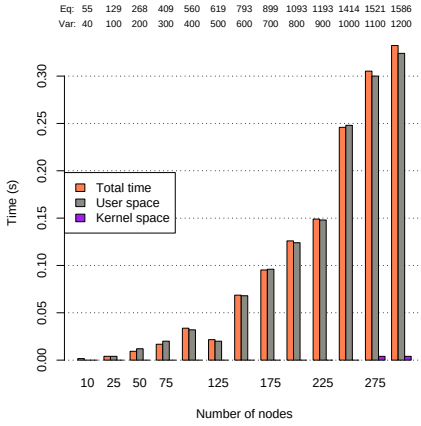


(c) PAP-SIMPLEX computation time.

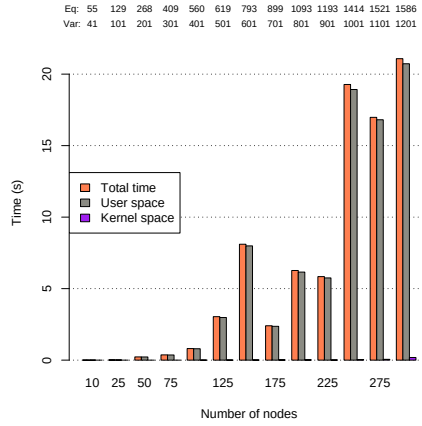


(d) PAP-MLPF-SIMPLEX computation time.

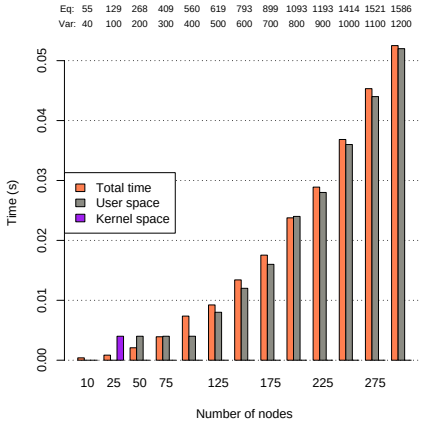
Figure 3.11: Solver `solve()` subroutine with 3SPs, 80 % demands.



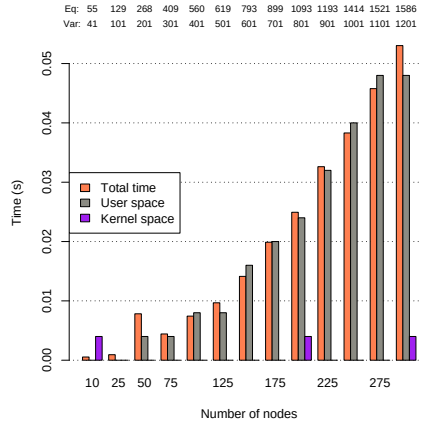
(a) PAP-IPM computation time.



(b) PAP-MLPF-IPM computation time.

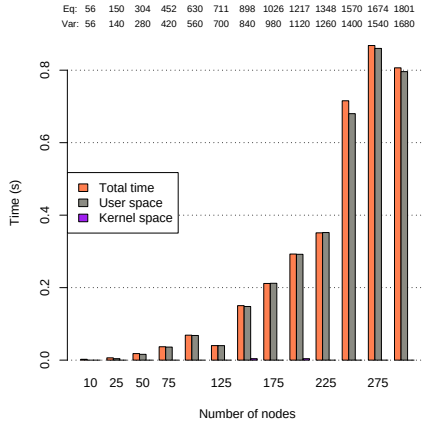


(c) PAP-SIMPLEX computation time.

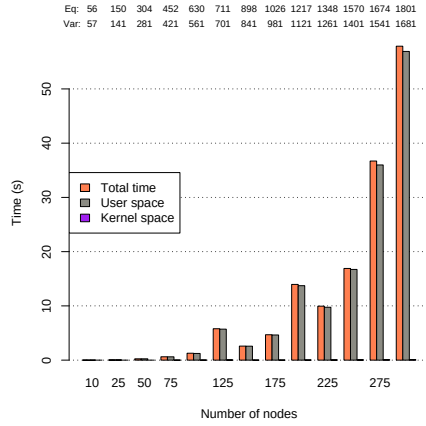


(d) PAP-MLPF-SIMPLEX computation time.

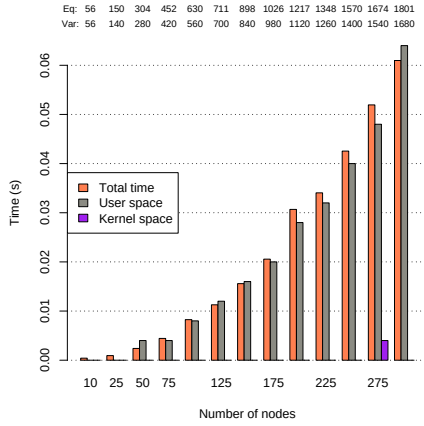
Figure 3.12: Solver `solve()` subroutine with 5SPs, 80 % demands.



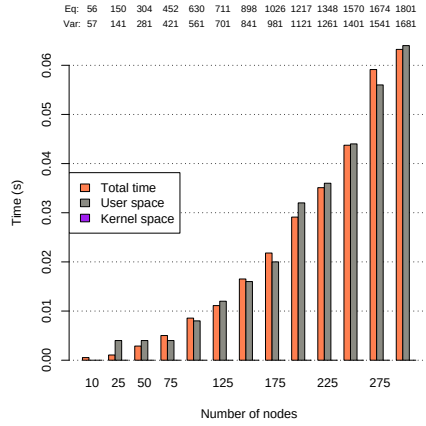
(a) PAP-IPM computation time.



(b) PAP-MLPF-IPM computation time.



(c) PAP-SIMPLEX computation time.



(d) PAP-MLPF-SIMPLEX computation time.

Figure 3.13: Solver `solve()` subroutine with 7SPs, 80 % demands.

entries in the millisecond range show user-space time or kernel-space time larger than the total time value. This pathological behavior is caused by the `getrusage` system call, which uses the system timer that runs at a frequency of 250 Hz. The time reported by `getrusage` is consequently incremented by 4 ms units. The total time provided by the `gettimeofday` system call has microsecond accuracy [103].

The name in the caption of each graph denote the type of problem and solved and the algorithm used:

PAP-IPM: pure allocation problem solved with the IPM,

PAP-MLPF-IPM: pure allocation problem with modified link-path formulation solved with the IPM,

PAP-SIMPLEX: pure allocation problem solved with the simplex method,

PAP-MLPF-SIMPLEX: pure allocation problem with modified link-path formulation solved with the simplex method.

The obvious observation is that the simplex method outperforms the IPM in every tested scenario. We suspect that this happens because GLPK's implementation of the interior point method is not as mature as the implementation of the simplex method. In particular, there is a huge difference, almost two orders of magnitude in some scenarios, between the IPM's computation time for the feasibility problem, PAP, and the computation time for the optimization problem, PAP-MLPF. When the simplex method is used to solve these problems, the diagrams show minor differences (1–20 ms) between the computation time of the feasibility problem versus the computation time of the optimization problem.

We show no corresponding memory graphs since any fluctuations in the memory usage while solving the problems are too small to distinguish from the memory usage displayed in Figure 3.8–3.13.

3.5 Summary

The chapter started with a brief overview of linear programming notation. Using this notation, we formulated optimization problems concerning path selection

and flow allocation. Furthermore, we described the performance testbed for methods used in solving network flow problems.

We used the testbed to obtain empirical results for the performance of Yen's KSP, the simplex method, and the IPM. The purpose for obtaining these results was two-fold. First, we wanted to know the cost in terms of resource usage associated with running these algorithms. Secondly, but no less important, the test results provided guidance for choosing an optimization algorithm for the ORP framework presented in Chapter 5.

Based on the results in final part of Section 3.4 we selected the simplex algorithm to be used in ORP because it showed consistent good performance when compared to the performance of the IPM.

Chapter 4

Gnutella Traffic Models

In the design of the ORP framework described in Chapter 5 it is assumed that implementations run on top of existing overlay networks spawned by end-nodes. The Gnutella P2P network is a typical example of this type of overlays. Consequently, Gnutella's traffic characteristics were the subject of a detailed study [104, 105].

The goal in studying the characteristics of Gnutella traffic was to obtain a better understanding of P2P dynamics and to construct simple statistical models that can be used for synthetic traffic generation.

In Section 4.1 we describe the Gnutella protocol with emphasis on message format, bootstrap, connection establishment, topology exploration and resource discovery. The infrastructure for measuring Gnutella traffic is described in Section 4.2, which is followed by a presentation of the methodology for constructing statistical models in Section 4.3. Section 4.4 presents the characteristics and models obtained from the recorded traffic.

4.1 The Gnutella Protocol

Gnutella is a heavily decentralized P2P system. Nodes[1] can share any type of resources, although the currently available specification covers only computer

[1] A Gnutella node is also called a servent, which is a combination of the words server and client.

files [106].

The network spawned by Gnutella nodes consists of an unstructured topology with a two-level hierarchy: ultrapeers (UPs) and leaf nodes (LNs). UPs are faster nodes in the sense that they are connected to high-capacity links and have a large amount of processing power available. LNs maintain a single connection to their UP. An UP maintains 10-100 connections, one for each LN and 1-10 connections to other UPs [107]. The UPs perform signaling on behalf of the LNs, thus shielding them from large volumes of signaling traffic. An UP does not necessarily have LNs, in which case it works standalone.

The activities of Gnutella peers can be divided into two main categories: signaling and user-data transfer (further referred to as data transfer). Signaling activities are concerned with peer discovery, overlay topology maintenance, content search and other management functions. Data transfer occurs when a peer has localized during content search one or more files of interest.

According to the Gnutella Development Forum (GDF) mailing list, the Gnutella community has recently adopted what is called support for high outdegree [108]. This implies that UPs maintain at least 32 connections to other UPs and 100–300 connections to different lead nodes. LNs are recommended to maintain approximately 4 connections to UPs. The numbers may slightly differ between different Gnutella vendors. The claim is that high outdegree support allows a peer to connect to the majority of Gnutella peers in 4 hops or less [109].

4.1.1 Bootstrap

A Gnutella node that attempts to join the overlay for the first time must bootstrap itself into the overlay. This implies finding and connecting to one or several peers that are already part of the overlay. A list of active servents can be obtained from a Gnutella Web Cache (GWC) [110] server. A GWC server is essentially an Hypertext Transfer Protocol (HTTP) server maintaining a list of active peers with associated listening sockets. A listening socket is an IP address and port number pair that can be used to connect to the corresponding servent. UPs update the list continuously, ensuring that new peers can always join the overlay.

Once the node joins the overlay, additional peers can be found through the

exchange of PING and PONG messages. The servent saves peer addresses in a local host cache in order to avoid connecting to a GWC server upon restart. The local host cache is used also if the servent supports the UDP Host Cache (UHC) protocol. The protocol works as a distributed bootstrap system, transforming UHC-enabled servents into GWC-like servers [111] and off-loading the actual GWC servers.

4.1.2 Connection Establishment

Peer signaling occurs over TCP connections. Once a TCP connection has been setup, the peers at each end of the TCP connection perform a three-way Gnutella handshake. The Gnutella handshake allows the negotiation of a common set of capabilities to be used during the session. The type of capabilities negotiated are UP - LN relationship, support for high outdegree, traffic compression, etc.

If the handshake fails the TCP connections is teared down. Otherwise, the client and the server start exchanging binary Gnutella messages over the existing TCP connection. The connection lasts until one of the peers decides to terminate the session. At that point the node ending the connection can optionally send a BYE message to notify its peer of its departure. The TCP connection will then be closed.

If the capability set used by the peers includes stream compression [112], then all data on the TCP connection is compressed, with the exception of the initial Gnutella handshake. The type of compression algorithm can be selected during the handshake, but the currently supported algorithm is *deflate*, which is implemented in `zlib` [113].

4.1.3 Messages

Each Gnutella message starts with a generic header that contains the fields shown in Figure 4.1 (the numbers in the figure denote bytes):

- message ID using a globally unique identifier (GUID), to uniquely identify messages on the Gnutella network [114],
- payload type code, denoted by P in Figure 4.1, which identifies the type of Gnutella message. The currently supported messages are: PING, PONG,

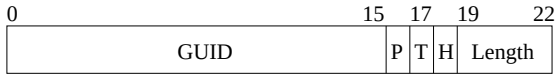


Figure 4.1: The Gnutella header.

BYE, QRP, VEND, STDVEND, PUSH, QUERY, QUERY_HIT and HSEP,

- time-to-live (TTL), to limit the signaling radius and its adverse impact on the network. Messages with TTL > 15 are dropped[2]. This field is denoted by T in Figure 4.1,
- hop count to inform receiving peers how far the message has traveled, denoted by H in Figure 4.1,
- payload length in bytes to describe the length of the message, not including the header. The payload length indicates where in the byte stream the next Gnutella generic message header can be found.

The generic Gnutella header is followed by the actual message, which may have own headers. Also, the message may contain vendor extensions. Vendor extensions are used when a specific type of servent wants to implement experimental functionality not covered by the standard specifications.

4.1.4 Topology Exploration

Each successfully connected servent periodically sends PING messages to its neighbors. The receiver of a PING message decrements the TTL in the Gnutella header. If the TTL is greater than zero the node increments the hop counter in the message header and then forwards the message to all its directly connected peers, with the exception of the one from where the message came. PING messages do not carry any user data, not even the sender's listening socket. This means that the payload length field in the Gnutella header is set to zero.

PONG messages are sent only in response to PING messages. More than one PONG message can be sent in response to one PING. The PONG message travels in the reverse direction on the path used by the corresponding PING

[2]Nodes that support high outdegree drop messages with TTL > 4.

message. Each PONG message contains detailed information about *one* active Gnutella peer. It also contains the same GUID as the PING message that triggered it.

UPs use the same scheme, however they do *not* forward PINGs and PONGs to and from the LNs attached to them.

Gnutella peers are required to implement some form of flow control in an effort to prevent PING-PONG traffic generated by malfunctioning servents from swamping the network. A simple flow control mechanism is specified in [115].

The BYE message is an *optional* message used when a peer wants to inform its neighbors that it will close the signaling connection. The message is sent only to hosts that have indicated during handshake that they support BYE messages.

4.1.5 Resource Discovery

Gnutella peers use QUERY messages to search for files. The message payload consists of a text string, information about the minimum speed (*i. e.*, upload rate) of servents that should respond to this message, and in some cases additional extensions that are not within the scope of this work. The most important part of the query is the text string, which is used to match files on the nodes receiving the message.

Gnutella v0.6 sends QUERY messages through a form of selective forwarding called *dynamic query* [108]. A dynamic query first probes how popular the targeted content is. This is done by using a low TTL value in the QUERY message that is sent to a small set of directly connected peers. A large number of replies indicate popular content, whereas a low number of replies imply rare content. For rare content, the QUERY TTL value and the number of directly connected peers receiving the message are gradually increased. This procedure is repeated until enough results are received or until an upper bound on the number of QUERY receivers is reached. This form of resource discovery requires all LNs to rely on UPs for their queries (*i. e.*, LNs do not perform dynamic queries).

If a peer that has received the QUERY message is able to serve the resource, it responds with a QUERY_HIT message. The GUID for the QUERY_HIT message must be the same as the one in the QUERY message that triggered the

response. The QUERY_HIT message lists all file names that match the text string from the QUERY message, their size in bytes and some other information [106]. In addition, the QUERY_HIT messages contain the listening socket to be used by the message receiver when it wants to download the matched files. The Gnutella specification discourages the use of messages with size greater than 4 KB. Consequently, several QUERY_HIT messages may be issued by the same servent in response to a single QUERY message.

4.1.6 Other Features

The Query Routing Protocol (QRP) was introduced in order to mitigate the adverse effects of flooding used by the Gnutella file queries [116]. QRP is based on a modified version of Bloom filters [117]. The idea is to break a query into individual keywords and have a hash function applied to each keyword. Given a keyword, the hash function returns an index to an element in a finite discrete vector. Each entry in the vector is the minimum distance (*i. e.*, number of hops) to a peer holding a resource that matches the keyword in the query. Queries are forwarded *only* to leaf nodes that have resources that match *all* the keywords. This procedure substantially limits the bandwidth used by queries. Peers run the hash algorithm over the resources they share and exchange the routing tables (*i. e.*, hop vectors) at regular intervals.

LNs send route table updates only to UPs and the UPs propagate these tables only to directly connected UPs [118].

Data exchange takes place over a direct HTTP connection initiated by the receiver of a QUERY_HIT message. Both HTTP 1.0 and HTTP 1.1 are supported but use of HTTP 1.1 is strongly recommended [106].

PUSH messages can be used when the file owner is protected by a firewall that does not allow incoming TCP connections or if the host is behind a Network Address Translation (NAT) device. In that specific case, the file requester opens a listening socket and puts information about the socket in a PUSH message. The PUSH message is sent over the signaling path to the file owner who, upon message reception, is able to open a TCP connection to the file requester. At that point the HTTP transfer can be performed. The PUSH message does not help if both peers are protected by firewalls or NAT devices that block incoming TCP connections.

The Horizon Size Estimation Protocol (HSEP) [119] is used to obtain estimates on the number of reachable resources (*e. g.,* nodes, shared files and shared kilobytes of data). Hosts that support HSEP announce this as part of the capability set exchange during the Gnutella handshake. If the hosts on each side of a connection support HSEP, they start exchanging HSEP message approximately every 30 seconds. The HSEP message consists of `n_max` triples. Each triple describes the number of nodes, files and kilobytes of data estimated at the corresponding number of hops from the node sending the message. The `n_max` values is the maximum number of hops supported by the protocol and its recommended value is 10 hops [119]. The horizon size estimation can be used to quantify the quality of a connection (*e. g.,* the higher the number of reachable resources, the higher the quality of the connection).

4.1.7 Example of a Gnutella Session

Figure 4.2 shows a simple Gnutella scenario, involving three peers. It is assumed that Peer A has obtained the listening socket of Peer B from a GWC server. Using the socket descriptor, Peer A attempts to connect to Peer B. In this particular example, Peer B already has a signaling connection to Peer C.

The first three messages between Peer A and Peer B illustrate the establishment of the signaling connection between the two peers. The two peers may exchange capabilities during this phase as well.

The next phase encompasses the exchange of network topology information with the help of PING and PONG messages. The messages are sent over the TCP connection established previously (i.e., during the peer handshake). It is observed that PING messages are forwarded by Peer B from Peer A to Peer C and in the opposite direction. Also, it can be observed that PONG messages follow the reverse path taken by the corresponding PING message.

At a later time the Peer A sends a QUERY message, which is forwarded by Peer B to Peer C. In this example, only Peer C is able to serve the resource, which is illustrated by the QUERY_HIT message. The QUERY and QUERY_HIT messages use the existing TCP connection, just like the PING and PONG messages. Again, it is observed that the QUERY_HIT message follows the reverse path taken by the corresponding QUERY message.

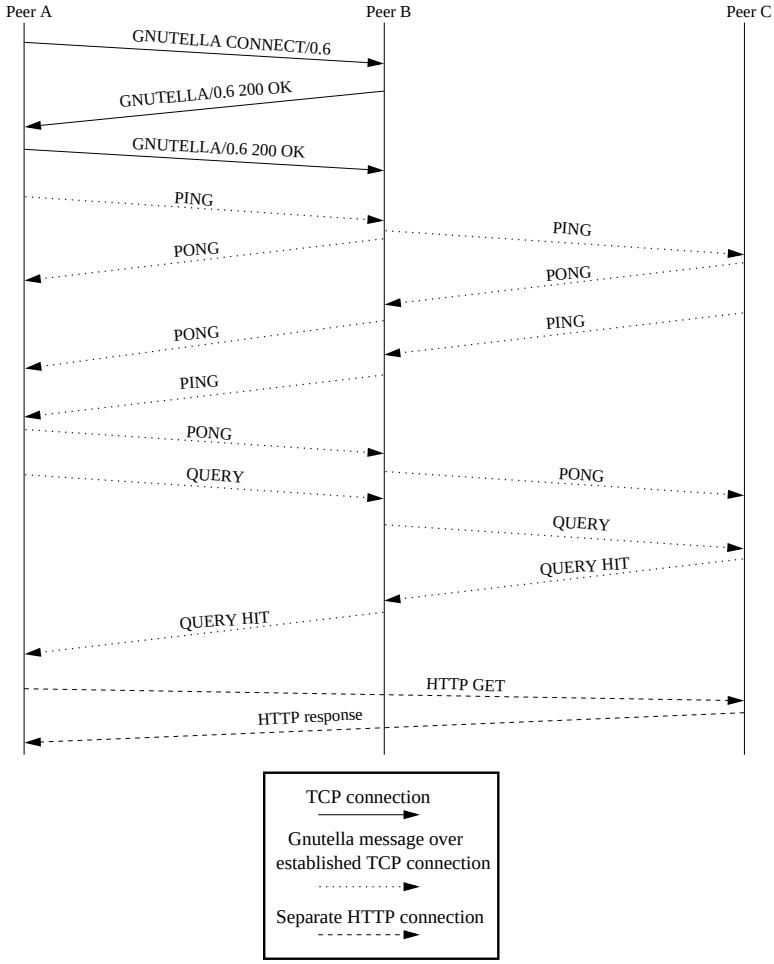


Figure 4.2: Example of a Gnutella session.

Finally, Peer A opens a direct HTTP connection to Peer C and downloads the resource by using the HTTP GET method. The resource contents are returned in the HTTP response message.

The exchange of PING-PONG and QUERY-QUERY_HIT messages continues until one of the peers tears down the TCP connection. A Gnutella BYE message may be sent as notification that the signaling connection will be closed.

4.2 Measurement Infrastructure

Network traffic measurements can be generally divided into active and passive measurements. The main difference between the two is that in active measurements specific patterns of traffic are injected into the network and analyzed when they exit the network. Changes in the injected traffic pattern are used to draw inferences about various properties of the network. In the case of passive measurements, traffic flows seen at specific nodes are observed or recorded, without sending any additional traffic in the network. In general, when the focus is on the characteristics of traffic crossing a single network element, the passive method is more appropriate [120]. This was our choice as well, since we were interested only in the traffic crossing the BTH ultrapeer.

There are two main approaches to perform passive application layer measurements: application logging or link-layer packet capture with application flow reassembly [104]. An important advantage of the link-layer packet capture is that it allows for traffic analysis at any layer in the TCP/IP stack. This enables a more accurate view of how the application affects the network and vice-versa. Another advantage is that packet timestamping is performed in the kernel and not in user space as is the case of application logging [103]. This means that packet timestamps are less affected by, *e. g.* , process preemption due to scheduling in the operating system and queuing and scheduling in the TCP/IP stack. Given these advantages, we use link-layer packet capture with application flow reassembly.

A measurement infrastructure dedicated to P2P measurement has been developed at BTH [121]. It consists of peer nodes and protocol decoding software. `Tcpdump` [122] and `tcptrace` [123] are used for traffic recording and protocol decoding. Although the infrastructure is currently geared towards P2P protocols,

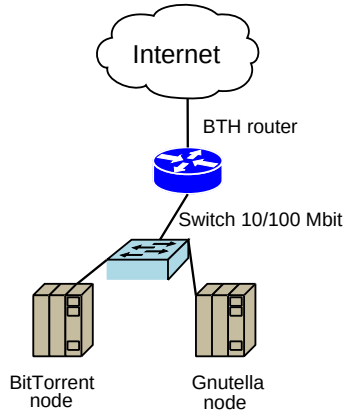


Figure 4.3: Measurement network infrastructure.

it can be easily extended to measure other protocols running over TCP. The measurement infrastructure has been successfully used for Gnutella [104, 124] and BitTorrent measurements [125, 126].

The BTH measurement nodes run the Gentoo Linux 1.4 operating system, with kernel version 2.6.5. Each node is equipped with an Intel Celeron 2.4 GHz processor, 1 GB RAM, 120 GB hard drive, and 10/100 Mbps Ethernet network interface. The network interface is connected to a 100 Mbps switch in the laboratory at the Department of Telecommunication Systems, which is further connected through a router to the GigaSUNET backbone, as shown in Figure 4.3.

Figure 4.4 shows the measurement process flow, which consists of six stages. The data enters each stage sequentially, from top to bottom.

Each measurement node has `tcpdump` 3.8.3 installed on it. When the node is running measurements, `tcpdump` is started before the Gnutella servent to avoid missing any connections. `Tcpdump` collects Ethernet frames from the switch port where the ultrapeer node is connected. The collected data is saved in packet capture (PCAP) format [122]. Since P2P applications tend to use dynamic ports, all traffic reaching the switch port must be collected. In addition, Ethernet frames cannot be truncated since we need the entire payload to decode the signaling traffic.

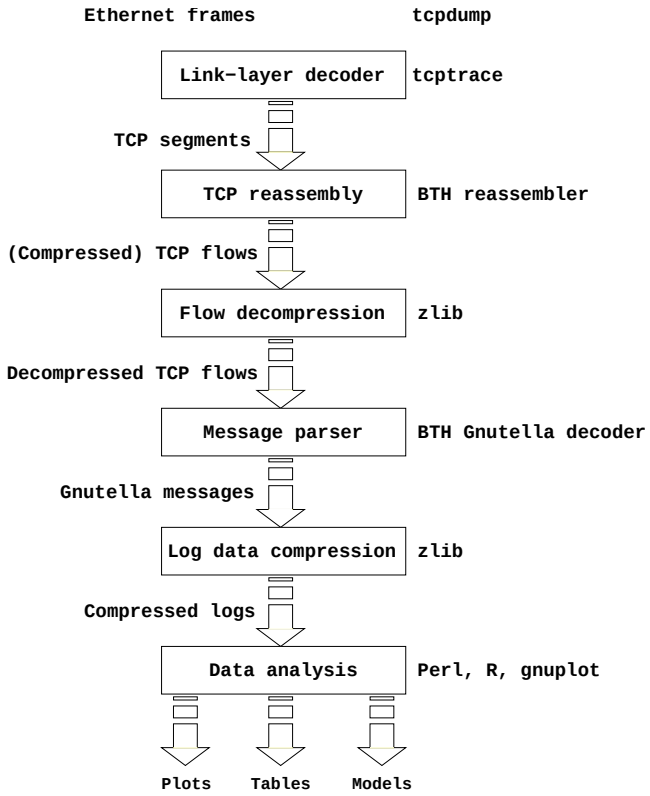


Figure 4.4: Measurement process.

During the first stage of the measurement process, we use `tcptrace` to extract TCP segments from the Ethernet frames.

The TCP segments are then sent to the next stage, whose task is to reassemble them to a flow of ordered bytes. The TCP reassembly module developed at BTH [104] builds on the TCP engine available in `tcptrace` and is similar to the one used by the FreeBSD TCP/IP stack [127]. The reassembly engine is capable of handling out-of-order segments as well as forward and backward overlapping between segments.

When a new Gnutella connection is found, the application reassembly module first waits for the handshake phase to begin. If the handshake fails, the connection is marked invalid and it is eventually discarded by the memory manager.

If the handshake is successful, the application reassembly module scans the capability lists sent by the nodes involved in the TCP connection. If the nodes have agreed to compress the data, the connection is marked as compressed. Subsequent segments received from the TCP reassembly module for this connection are first sent to the decompresser, before being appended to previous data that has not been consumed yet.

The decompresser uses `zlib`'s *inflate()* function to decompress the data available in the new segment [113]. Upon successful decompression the decompressed data is appended to the data buffer.

Immediately after the handshake phase, the application reassembly module attempts to find the Gnutella message header of the first message. Using the payload length field, it is able to discover the beginning of the next message. This is the only way to discover message boundaries in the Gnutella protocol and thus track application state changes [104]. Based on the message type field in the message header, the corresponding decoding function is called, which outputs a message record to the log file. The message records follow a specific format required by the post-processing stage [104].

Since the logs can grow quite large, they can be processed through an optional stage of data compression. The compression is achieved by using the on-the-fly `deflate` compression offered by `zlib`. Additional data reduction can be achieved if the user is willing to sacrifice some detail by aggregating data over time.

The data analysis module interprets the (optionally compressed) log data and it is able to demultiplex it based on different types of constraints: message type, IP address, port number, etc. The data output format of this stage is suitable for input to numerical computation software such as MATLAB and standard UNIX text processing software such as `sed`, `awk` and `perl`.

4.3 Methodology for Statistical Modeling

The measurement infrastructure described in the previous section was used to collect Gnutella traffic crossing the BTH ultrapeer. By decoding the recorded traffic data, flows were recreated at several layers in the TCP/IP stack. The flows consist of discrete protocol data units: IP datagrams at the network layer, TCP segments at the transport layer, and finally, Gnutella messages at the application layer. The Gnutella messages are logically grouped in peer sessions. The time when the protocol data units reached the link layer was recorded together with their size. For peer sessions, the session duration was recorded as well. Due to the complexity of the protocol we used a statistical approach [104, 125] to describe the quantities of interest, which is similar to the methodology introduced by Paxson in [128].

Each quantity of interest is modeled by a random variable X that changes its value whenever a new protocol data unit (or session) is considered. The actual values taken by X are denoted by the small letter x. The random variable X is assumed to have a theoretical cumulative distribution function (cdf) $F_X(x;\theta)$.

Definition 4.1. The theoretical cdf $F_X(x;\theta)$ of a random variable X is defined as

$$F_X(x;\theta) \triangleq P[X \leq x], \tag{4.1}$$

where $\triangleq$ is the equality by definition operator, x is some value on the real line and θ is a set of one or more parameters that control the distribution function *e. g.* , $\theta = \{\mu, \sigma\}$ in the case of the normal distribution. □

Definition 4.2. It is assumed that a cdf $F_X(x;\theta)$ has a corresponding probability density function (pdf) $f_X(x;\theta)$ defined as

$$f_X(x;\theta) \triangleq \frac{dF_X(x;\theta)}{dx}. \tag{4.2}$$

The derivative must exist at all points of interest otherwise impulse functions are required like in the case of discrete distributions [129]. □

It is often useful to observe how fast the cdf decays for large values of x. For that particular purpose it is better to use the complementary cumulative distribution function (ccdf) function.

Corollary 4.1. Assuming a cdf function $F_X(x;\theta)$, the corresponding ccdf function is:

$$\overline{F}_X(x;\theta) \triangleq 1 - F_X(x;\theta) = P[X > x] \tag{4.3}$$

□

For each quantity of interest, the set of values extracted from the recorded traffic is considered to be a *random sample* from the population of the random variable X. The elements of the random sample are denoted by $X_1, X_2, \ldots, X_n$ and the actual recorded values (*data sample*) by $x_1, x_2, \ldots, x_n$. The index n is the number of available values from the measurement.

The modeling methodology employed in this chapter involves three phases:

i) identify a distribution family $F(\cdot)$ through exploratory data analysis (EDA),

ii) using the available data, estimate the parameter(s) θ of the distribution from the previous step. Denote the estimated parameter(s) by $\hat{\theta}$ and the estimated distribution by $\hat{F}_X(x;\hat{\theta})$,

iii) quantify the quality of the fit.

4.3.1 Exploratory Data Analysis

The first step in the modeling methodology is to identify a distribution family $F_X(x,\theta)$. This is done through an EDA approach that combines graphs of the data such as histograms and distribution plots and summary statistics *e. g.*, mean, median and standard deviation [130, 131].

The histograms and distribution plots are the main EDA tools. Using them, the EDA user is aided in recognizing a family of distributions that provides good match for the data. The summary statistics provide some quantitative support in the selection of a distribution family.

When unknown parameters of the distribution family are estimated, the candidate distribution is fully specified. At that point the quality of the fit can be assessed by formal numerical methods, as described in Section 4.3.3.

Summary Statistics

Five different types of statistics can be used to summarize a random sample: maximum, minimum, mean, median and standard deviation. All definitions in this section assume a random sample $X_1, X_2, \ldots, X_n$ of length $n > 1$. The corresponding order statistics are $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$.

Definition 4.3. The largest and smallest value of the random sample are denoted by $X_{(n)} = \max[X_1, \ldots, X_n]$ and $X_{(1)} = \min[X_1, \ldots, X_n]$, respectively. The difference $\left(X_{(n)} - X_{(1)}\right)$ defines the range of the data sample. □

Definition 4.4. The sample mean $\overline{X}$ is defined as

$$\hat{\mu} = \overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i \tag{4.4}$$

□

$\overline{X}$ is an unbiased estimator of the first population moment, that is of the expected value $E[X]$. For a symmetric distribution the actual value of $\overline{X}$ represents the "center" of the data range. For a skewed distribution, the median statistic is a more appropriate representation of centricity.

Definition 4.5. If the sample size is odd, *i. e.*, $n = 2k + 1$, the sample median is the middle order statistic $X_{(k+1)}$. If the sample size is even, *i. e.*, $n = 2k$, the sample median is the average of the two middle order statistics

$$\text{median} = \frac{X_{(k)} + X_{(k+1)}}{2} \tag{4.5}$$

□

Definition 4.6. The sample standard deviation is defined as

$$\hat{\sigma} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X})^2} \tag{4.6}$$

□

Histogram Plots

A histogram plot is a graph of tabulated frequencies for an univariate data sample $x_1, x_2, \ldots, x_n$ of length n. If the frequencies are normalized such that the area below the histogram is equal to one, then the histogram can be viewed as a rough estimate of the probability density function.

In order to build a histogram one must begin by dividing the range r of the data into a number of m contiguous bins. Each bin i covers a portion of length[3] L of the data range. The boundaries of the bin i are denoted by b_i and b_{i+1}. Next, the data values are sorted and placed into bins that correspond to their value. The number of entries in each bin represents the frequency f_i of the bin i. To obtain the probability of each bin, the frequencies f_i are normalized such that the probability p_i of bin i is:

$$p_i = \frac{f_i}{n} \tag{4.7}$$

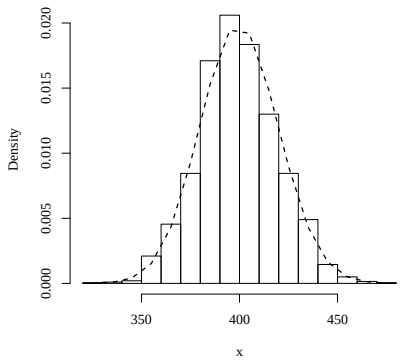
It follows that the probability of the bin i is [132]

$$p_i \approx P[b_i < X \leq b_{i+1}] = \int_{b_i}^{b_{i+1}} f_X(x)\,\mathrm{d}x = f_X(y)\,L \quad \text{for some } y \in (b_i, b_{i+1}) \tag{4.8}$$

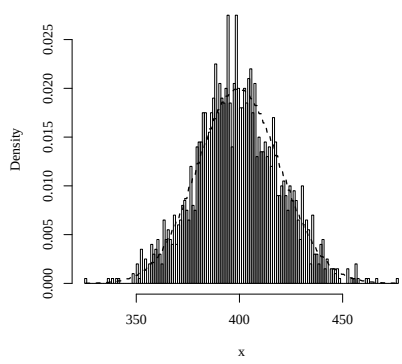
An important question is how to choose the bin length L or equivalently the number of bins m. Histograms using bins that are too wide fail to reveal specific characteristics of the data such as multi-modality (*i. e.* , mixture of distributions) or impulses at the origin. These are called undersmoothed histograms. On the other hand, if the bin width is too small the histogram is likely have a jagged appearance that could complicate the identification of the underlying distribution or even worse, it may present false evidence of multi-modality. In this case the histogram is called an oversmoothed histogram. Research into optimal bin width has lead to the thumb rules [133, 134] presented in Table 4.1.

The terms $\hat{\sigma}$, $q_{.75}$ and $q_{.25}$ denote the estimated standard deviation, the 0.75-quantile, and the 0.25-quantile, respectively. Figure 4.5 shows an example of how the histogram of specific data can look like when the bin width is too large, too small and when it is chosen by using the Friedman-Diaconis method.

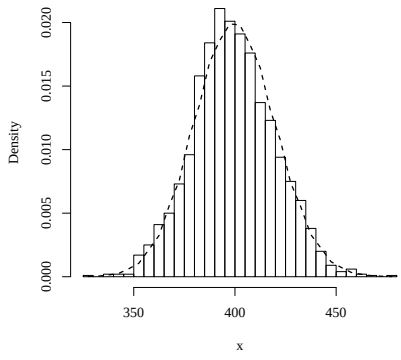
[3] Bins of equal length are assumed.



(a) Undersmoothed



(b) Oversmoothed



(c) Friedman-Diaconis

Figure 4.5: Poisson distribution with $\lambda = 400$: histogram for 2000 samples and superimposed density function.

Name	Bin width
Sturges' formula	$L = r/(1 + \log_2 n)$
Scott's rule	$L = 3.49\hat{\sigma}\, n^{-1/3}$
Friedman-Diaconis	$L = 2\,(q_{.75} - q_{.25})\, n^{-1/3}$

Table 4.1: Various rules for choosing histogram bin width.

The rules in Table 4.1 work well in many situations. Unfortunately, none of them is a panacea. In fact, for some distributions it is necessary to manually adjust the number of bins in order to obtain a smooth histogram [133].

Edf Plots

The empirical distribution function (edf) $\mathcal{F}_n(x)$[4] of a random sample is an approximated representation of the true cdf for the population from which the sample is drawn.

Definition 4.7. Given a random sample $X_1, X_2, ..., X_n$ of length n drawn from a distribution $F_X(x)$, denote the corresponding order statistics by $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$. Then, $\mathcal{F}_n(x)$ is defined as

$$\mathcal{F}_n(x) \triangleq \begin{cases} 0 & x < X_{(1)} \\ \dfrac{i}{n} & X_{(i)} \leq x < X_{(i+1)} \\ 1 & X_{(n)} \leq x. \end{cases} \tag{4.9}$$

□

For large samples, $\mathcal{F}_n(x)$ converges uniformly to the population $F_X(x)$ for all x-values [130].

Corollary 4.2. Assuming an edf $\mathcal{F}_n(x)$, then

$$\overline{\mathcal{F}}_n = 1 - \mathcal{F}_n \tag{4.10}$$

is the corresponding complementary empirical distribution function (cedf). □

[4]Normally, the notation $F_n(x)$ is used to denote an edf. However, this notation could conflict here with the notation used for a cdf and it is therefore written using a calligraphic letter: $\mathcal{F}_n(x)$.

The histogram, the edf and the cedf are complementary views of the sample distribution. The terminology used for ccdf and cedf plots is to denote as "body" the values of $\overline{\mathcal{F}}_n(x)$ for $x \leq \xi$ and as "tail" the values of $\overline{\mathcal{F}}_n(x)$ for $x > \xi$. The point ξ on the x-axis is in general dictated by the type of data analysis performed, but tends to be selected such that for large x-values the corresponding ccdf values are very small, but non-negligible. The decay of the tail allows us to assign the empirical distribution to a particular distribution class.

The *subexponential* or *long-tailed* class contains distributions that decay slower than the exponential distributions. The class of *heavy-tailed* distributions is a more restrictive subclass, since they require infinite variance. A random variable X has a heavy-tailed distribution if:

$$\lim_{x\to\infty} \overline{F}_X(x) = \lim_{x\to\infty} P[X > x] = cx^{-\alpha}, \quad 0 < \alpha < 2,\ c > 0 \tag{4.11}$$

When the *tail index* α is $0 < \alpha < 1$, the heavy-tailed distribution has infinite mean in addition to infinite variance. This is in contrast with the larger class of long-tailed distributions with finite moments [135].

The Pareto distribution is a good example of heavy-tailed distribution. The lognormal and Weibull distributions are subexponential, but not heavy-tailed. In particular, the Weibull distribution has finite variance [135]. Paxson and Floyd provide proof that the lognormal distribution is not heavy-tailed [136]. Gaussian or Gamma and exponential distributions are called *light-tailed* distributions and are not part of the subexponential class [135]. For light-tailed distributions the ccdf values in the tail are negligible.

4.3.2 Parameter Estimation

Parameter estimation is the second phase of the modeling methodology used in this thesis. It is assumed that a distribution family $F_X(x;\theta)$ with a set of unknown parameters θ has been identified as described in Section 4.3.1. The goal is to estimate the parameters by using point *estimators* $\hat{\Theta} = \mathcal{E}(X_1, X_2, \ldots, X_n)$, where $\mathcal{E}$ is a function of the random sample. Point *estimates* $\hat{\theta}$ are obtained by replacing the random variables in $\hat{\Theta}$ with observed values.

The optimality of point estimators is decided by concepts such as bias, efficiency, consistency and sufficiency. An unbiased estimator is one for which

$E[\hat{\Theta}] = \theta$ [137]. Furthermore, an estimator $\hat{\Theta}_1$ is more efficient than an estimator $\hat{\Theta}_2$ if $\text{Var}\left[\hat{\Theta}_1\right] < \text{Var}\left[\hat{\Theta}_2\right]$. By consistency it is meant that a sequence of estimators converges towards the "true" value of the parameter. Sufficiency is concerned with the amount of information intrinsic to the sample, which is lost or kept when a particular estimator is used [137, 138]. These are large topics outside the scope of this thesis. It is sufficient to mention that maximum likelihood estimators are in general at least as good as other estimators for large sample sizes. However, the equations that appear in the course of using the method can be non-linear and difficult to solve. In this case numerical solutions are required [137–139]. Similar problems appear when the method is used with mixture distributions. Therefore, a secondary method, denoted *minimum-absolute-error*, is introduced as well. In addition for providing point estimates, the minimum-absolute-error method is used as goodness-of-fit measure, as described in Section 4.3.3.

Maximum Likelihood Method

The maximum likelihood (ML) method is based on the concept of *likelihood function*, which is defined as the joint pdf of a number of random variables [138].

Definition 4.8. Given n random variables $X_1, X_2, \ldots, X_n$ drawn from a distribution $F_X(x;\theta)$, and the corresponding observed values $x_1, x_2, \ldots, x_n$, the likelihood function $\mathcal{L}(\theta)$ is defined as

$$\mathcal{L}(\theta) \triangleq f_{X_1,X_2,\ldots,X_n}(x_1, x_2, \ldots, x_n; \theta) \tag{4.12}$$

which is the joint distribution of $X_1, X_2, \ldots, X_n$. □

Corollary 4.3. For a random sample $X_1, X_2, \ldots, X_n$ with common distribution $F_X(x;\theta)$

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} f_X(x_i;\theta), \tag{4.13}$$

which follows from the definition of a random sample. □

Intuitively, the likelihood function $\mathcal{L}(\theta)$ for a random sample drawn from a discrete pdf is the probability that the random sample will assume the observed values [139]:

$$\mathcal{L}(\theta) = P[X_1 = x_1; \theta]\, P[X_2 = x_2; \theta] \ldots P[X_n = x_n; \theta] \tag{4.14}$$

It becomes evident that the optimal $\hat{\theta}$ is the value that maximizes $\mathcal{L}(\theta)$. This idea can be applied in a similar manner to random samples from a continuous pdf. Assuming certain regularity conditions [138], the solution $\hat{\Theta} = \mathcal{E}(X_1, X_2, \ldots, X_n)$ to the equation

$$\frac{d\mathcal{L}(\theta)}{d\theta} = 0 \tag{4.15}$$

is the ML estimator. When the random variables are replaced with the actual observed values, one obtains the ML estimate $\hat{\theta} = \mathcal{E}(x_1, x_2, \ldots, x_n)$. Sometimes it is easier to solve the equation

$$\frac{d\, \ln\left[\mathcal{L}(\theta)\right]}{d\theta} = 0 \tag{4.16}$$

instead of Equation 4.15 [138].

Minimum-Absolute-Error Method

The minimum-absolute-error method seeks to find an estimate $\hat{\theta}$ that minimizes the difference between the edf, $\mathcal{F}_n(x)$, and the estimated cdf, $\hat{F}_X(x; \hat{\theta})$, over all x.

Definition 4.9. For a data sample $x_1, x_2, \ldots, x_n$, the difference between $\mathcal{F}_n(x)$ and $\hat{F}_X(x; \hat{\theta})$ is defined as the cumulative absolute error $\epsilon(\theta)$, such that

$$\epsilon(\theta) \triangleq \sum_{i=1}^{n} \left| \hat{F}_X(x_i; \hat{\theta}) - \mathcal{F}_n(x_i) \right| \tag{4.17}$$

The estimate $\hat{\theta}$ is the θ-value that minimizes $\epsilon(\theta)$. □.

Since this method relies on the edf, $\hat{\theta}$ cannot be solved analytically. Numerical algorithms to obtain solutions for it are discussed in Section 4.3.5.

4.3.3 Fitness Assessment

After a probability distribution has been fitted to the data as described in the previous section, the next step is to estimate the quality of the fit. A variety of goodness-of-fit tests can be used for this purpose, *e. g.*, the χ^2, Kolmogorov-Smirnov and Anderson-Darling tests. Their common denominator is the test of the null hypothesis:

$$H_0: \quad \text{The random sample } X_1 \dots X_n \text{ is drawn from the distribution } \hat{F}(x, \hat{\theta})$$

Unfortunately, these tests tend to erroneously reject the null hypothesis when the number of samples is large (Type 1 error) [132, 140, 141], which is the case for our data. Therefore, a different approach is used in which the hypothesis test is avoided.

A goodness-of-fit measure called error-percentage measure ($E_{\%}$) was introduced in [126] and used later in [125, 142]. The method is based on the probability integral transform (PIT).

Definition 4.10 (Probability integral transform)**.** Given a *continuous* random variable R with cdf $F_X(x)$ and $P[Y \leq y] \stackrel{d}{=} U[0,1]$, the transformation

$$F_X(R) = P[X \leq R] = Y \tag{4.18}$$

is called the probability integral transform [132, 138]. The symbol $\stackrel{d}{=}$ denotes equality in distribution and $U[0,1]$ denotes the uniform distribution between zero and one. □

The algorithm to compute $E_{\%}$ is shown in Algorithm 7. If the distribution $\hat{F}$ is a perfect fit, then the PIT transforms the random sample to an uniform distribution, $U[0,1]$. However, since perfect fittings rarely occur in reality, the transformed distribution, $\hat{U}$, is an approximate of the uniform distribution. The discrepancies between U and $\hat{U}$ are computed and their average is normal-

ized [125] to the highest possible error E_{max} for the distribution $U[0,1]$ where,

$$\begin{aligned} E_{max} &= \int_0^1 \sup\{U(x), 1-U(x)\}\ \mathrm{d}x \\ &= \int_0^{1/2} [1-U(x)]\ \mathrm{d}x + \int_{1/2}^1 U(x)\ \mathrm{d}x = \frac{3}{4} \end{aligned} \tag{4.19}$$

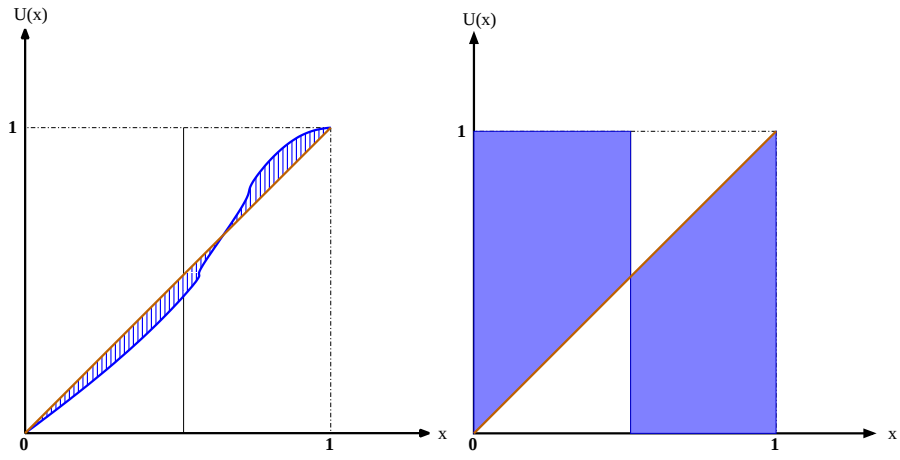
Algorithm 7 Calculate error percentage.

Fit a distribution $\hat{F}_X(x;\hat{\theta})$ to the random sample $X_1, X_2, \ldots, X_n$
Obtain the order statistics $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$
Transform the random sample with PIT: $\hat{U}_i = \hat{F}_X(X_{(i)};\hat{\theta}), \quad i = 1, \ldots, n$
$E_{\%} = 100\dfrac{\sum_{i=1}^{n}\left|U_i - \hat{U}_i\right|}{n\,E_{max}}$, where $U_i = \dfrac{i}{n} \overset{d}{=} U[0,1]$
return $E_{\%}$

Figures 4.6(a)–4.6(b) provide additional visual clues on how the $E_{\%}$-method works. Figure 4.6(a) shows a hypothetical edf for a random sample transformed with the PIT, *i. e.*, the diagonal straight line. The blue shaded area represents the error (discrepancy) when the edf is compared to the ideal $U[0,1]$ distribution. The size of that area is the $E_{\%}$ score. The size of the shaded area in Figure 4.6(b) is the maximum error E_{max} that can occur when the PIT is applied to a random sample. This is the value that is used to normalize the $E_{\%}$ score.

$E_{\%}$ is expressed in the form of a percentage. The criteria used here to accept a candidate distribution is that $E_{\%} < 6$. We call this value the *accepted error percentage*. The accepted error percentage was decided experimentally by observing that most distributions that provide a visually acceptable fit in both body and tail have $E_{\%} < 6$. Table 4.3.3 presents a mapping between various $E_{\%}$ ranges and qualitative statements about the fit.

The main disadvantage of the $E_{\%}$-method is that it cannot be used with discrete distributions. The reason is that when the PIT method is applied to a discontinuous distribution, the transformed variable is not uniformly distributed [143].



(a) Error in transformed random sample. (b) Maximum error for a PIT transformation.

Figure 4.6: Probability integral transform (PIT).

Range	$0 \leq E_{\%} < 2$	$2 \leq E_{\%} < 4$	$4 \leq E_{\%} < 6$	$E_{\%} \geq 6$
Quality	Excellent	Good	Acceptable	Unacceptable

Table 4.2: Quality-of-fit mapping.

4.3.4 Finite Mixture Distributions

Sometimes a single cdf cannot accurately describe the distribution of the random variables of interest *e. g.* , the $E_{\%}$-method yields an unacceptable score. A more accurate model may be constructed by using a mixture of two distributions or more. In the case of a mixture of two distributions one component of the mixture accounts for the main body of the empirical distribution and a different one describes the behavior in the tail. In the case of more than two components, each cdf accounts for specific modality found in the data. The crux of the problem becomes to find a way to combine the two distributions in a meaningful way. The method used here is based on finite mixture distributions as described in [144]

A mixture distribution $F_X(x)$ with n components has the following distri-

bution function:

$$F_X(x) = \sum_{i=1}^{n} \pi_i \, G_i(x) \qquad \pi_1 + \pi_2 + \cdots + \pi_n = 1 \tag{4.20}$$

where $G_i(x)$ is the ith distribution in the mixture and each π_i is a constant called *mixing weight*. The mixing weight π_i is selected such that $0 < \pi_i < 1$ and it decides how much each component is allowed to influence the distribution $F_X(x)$.

The first step in building a mixture distribution with two components is to identify a distribution family $G_1(\cdot)$ that matches the body of the data, preferably the tail as well. This is done by using the EDA approach, as explained in Section 4.3.1. The parameters of the distribution are estimated, yielding a specific distribution function $\hat{G}_1(x;\hat{\theta}_1)$. $\hat{G}_1(x;\hat{\theta}_1)$ is then visually compared to the true distribution to asses the fit in the tail. If the fit appears good, then the goodness-of-fit measure $E_{\%}$ is computed as explained in Section 4.3.3. Otherwise, it is necessary to find the *cutoff point* x_c and the corresponding *cutoff quantile* q_c, where $\hat{G}_1$ diverges from the true distribution. The probability mass between q_c and 1 is used to identify the distribution family $\hat{G}_2(\cdot)$ that matches the tail. The parameters of the new distribution must be estimated as well, yielding $\hat{\theta}_2$. Then, a finite mixture distribution

$$\hat{F}(x;\hat{\theta}) = \pi\hat{G}_1(x;\hat{\theta}_1) + (1-\pi)\hat{G}_2(x;\hat{\theta}_2) \tag{4.21}$$

is assembled, where $\pi = q_c$. Since the single distributions, $\hat{G}_1$ and $\hat{G}_2$, are now combined in a finite mixture, the parameters $\hat{\theta}_1$ and $\hat{\theta}_2$ must be recomputed[5]. Their original values may be used as a starting point. An optimal value for π must be computed as well. The parameter set $\hat{\theta}$ in $F(x;\hat{\theta})$ is the set containing the parameters for both distributions and π, *i. e.*, $\hat{\theta} = \{\hat{\theta}_1, \hat{\theta}_2, \pi\}$. Numerical methods for computing the set of optimal parameters θ are presented in Section 4.3.6.

It is often the case that a mixture distribution (in particular one with only two components) still cannot describe the data accurately enough. This may be further improved by increasing the number of components in the mixture distribution at the expense of an increase in the number of parameters. However, a different approach was used here.

[5]Recall that G_1 was estimated using the entire probability mass.

Typically, the major discrepancies between the estimated distribution and the true one appear either in the body or in the tail. If, for example, the discrepancies appear in the tail, one can attempt to improve the model accuracy by adjusting the values of the distribution parameters. However, our experience was that this is likely to decrease the accuracy of the model in the body. Similarly, attempts to increase the accuracy in the body may lead to (higher) discrepancies in the tail. Thus, a trade-off is required, accuracy in the body versus accuracy in the tail [104]. Accordingly, a decision must be taken on which part of the distribution (body or tail) is more important to model accurately.

For example, in the case of transfer rates the tail of the distribution models high rates of traffic (bursts) that occur rarely. On the other hand, the body of the distribution models the "average" size of transfer rates. For message or packet size the body accounts for small packets or messages and the tail for large ones. In the case of interarrival and interdeparture times the body accounts for "dense" traffic and the tail for "sparse" traffic. In our models, when a trade-off was required, we favored to accurately model bursty, dense traffic with large packets (messages).

4.3.5 Methodology Review

The goal of this section is to present a formal process for the modeling methodology discussed in the previous sections. The process assumes that the variable of interest has been measured (sampled) n times. The values $x_1, x_2, \ldots, x_n$ resulting from the n measurements are assumed to be the result of a random sample $X_1, X_2, \ldots, X_n$. The complete process for building the statistical models is presented in Algorithm 8 [104].

Step 14–15 in Algorithm 8 may be confusing since no criteria has been provided on how to decide to either select a different quantile or to start over. To solve this, the quantile was changed in increments of 0.05 to either sides of the original value. If this did not result in any improvement, the decision was to start over.

Algorithm 8 Methodology for statistical modeling.

1: Use EDA visual tools, *i. e.,* histogram, edf and cedf plots, to explore the data. The summary statistics provide hints about range, skewness and spread
2: Select a distribution family G_1, which appears to provide a good fit
3: Estimate the unknown parameters θ_1 using ML estimation to obtain a candidate distribution $\hat{G}_{1X}(x, \hat{\theta}_1)$
4: Compare the plots of $\hat{g}_{1X}(x; \hat{\theta}_1)$, $\hat{G}_{1X}(x; \hat{\theta}_1)$, and $\overline{\hat{G}_{1X}}(x; \hat{\theta}_1)$ to the histogram, edf, and cedf plots obtained in Step 1
5: **if** high visual discrepancy **then**
6: Go back to Step 1
7: **end if**
8: Compute $E_{\%}$ for $\hat{G}_{1X}(x; \hat{\theta}_1)$ using $x_1, x_2, \ldots, x_n$
9: **if** $E_{\%} < 6$ **then return** $E_{\%}$ and $\hat{G}_{1X}(x, \hat{\theta}_1)$
10: **end if**
11: Identify the cutoff quantile q_c
12: Fit a distribution $G_2(\cdot)$ to the probability mass $(1-q_c)$ as outlined in Step 1–8
13: **if** $E_{\%} > 6$ **then**
14: Either go back to Step 11 and select a different quantile q_c or,
15: Go back to Step 1. This is equivalent to starting over. Try using a different distribution family $G_1(\cdot)$
16: **end if**
17: Assemble the mixture distribution $F(\cdot) = \pi\, G_1(\cdot) + (1 - \pi)\, G_2(\cdot)$
18: Estimate the unknown parameters $\theta = \{\theta_1, \theta_2, \pi\}$ using $E_{\%}$ method. Use the estimated values from previous steps as initial values
19: **if** $E_{\%} < 6$ **then return** $E_{\%}$ and $F_X(x; \theta)$
20: **else**
21: Go back to step 1
22: **end if**

4.3.6 Numerical Software and Methods

The process presented here was implemented by using the statistical software package `R` [145]. `R` is an interpreted computer language with syntax similar to `S` and `S-PLUS` [133]. The software package contains in addition to the language, a run-time environment with graphics, a debugger and a large library of functions.

As mentioned in Section 4.3.4, the $E_{\%}$-method relies on numerical optimization for finding a minimum. ML estimation requires also numerical optimization in many cases where no closed form ML estimators exist. The `R`-functions `optimize()` and `optim()` have been used for numerical optimization.

The function `optimize()` performs optimization in one dimension. The underlying algorithm is a combination of golden section search and successive parabolic interpolation [93, 145]

General purpose multi-dimensional optimization is performed by the `optim()` function. The function has support for several optimization algorithms. The default algorithm, Nelder-Mead [93, 146, 147], is primarily used. The algorithm does not require any derivatives, and it is quite stable although not very efficient in terms of number of iterations.

When the Nelder-Mead algorithm fails to converge to a solution, the L-BFGS-B [148] algorithm is used instead. This algorithm requires a lower and an upper bound for each variable. The thumb rule used to provide the bounds is to allow variables with initial values $m_0 \geq 1$ a range of $0.2\,m_0$ between the upper and lower bound. For variables with initial values $m_0 < 1$ the range between bounds was $0.1\,m_0$. This thumb rule was designed empirically and it is by no means optimal in any way. In fact, from our experience, the bounds often needed additional adjustment to obtain convergence.

4.4 Characteristics and Statistical Models

In order to keep the mathematical formulas brief we use the following conventions. Cdfs are denoted by capital letters and pdfs by lower case letters, as shown in Table 4.3. The parameters are as follows: μ and σ are related to the distribution mean and standard deviation, while α, β and κ are the shape, scale and location parameters. For the uniform distribution, a and b are the

Uniform	$u_X(x; a, b)$	$U_X(x; a, b)$
Poisson	$\mathrm{po}_X(x; \mu)$	$\mathrm{PO}_X(x; \mu)$
Exponential	$\exp_X(x; \mu)$	$\mathrm{EXP}_X(x; \mu)$
Normal (Gaussian)	$\mathrm{n}_X(x; \mu, \sigma)$	$\mathrm{N}_X(x; \mu, \sigma)$
Log-normal	$\ln_X(x; \mu, \sigma)$	$\mathrm{LN}_X(x; \mu, \sigma)$
Generalized Pareto	$\mathrm{pa}_X(x; \alpha, \kappa, \beta)$	$\mathrm{PA}_X(x; \alpha, \kappa, \beta)$

Table 4.3: Model notation.

lower and upper boundary, respectively, of the range of x-values for which the distribution is valid. In particular, the parameter a is equivalent to a location parameter, while $(b-a)$ is equivalent to a scale parameter [132]. All logarithmic edf plots use $\log_{10}$-transformations for both axes.

The generalized Pareto distribution [149] and the corresponding density function are defined as

$$F_X(x; \alpha, \kappa, \beta) = 1 - \left[1 + \frac{\alpha(x-\kappa)}{\beta}\right]^{-\frac{1}{\alpha}} \tag{4.22}$$

$$f_X(x; \alpha, \kappa, \beta) = \frac{1}{\beta}\left[1 + \frac{\alpha(x-\kappa)}{\beta}\right]^{-\frac{1}{\alpha}-1} \tag{4.23}$$

where $\alpha \neq 0$ is the shape parameter, $\kappa \leq x$ is the location parameter, and $\beta > 0$ is the scale parameter.

4.4.1 Ultrapeer Settings and Packet-Trace Statistics

The results reported here were obtained from an 11-days long link-layer packet trace collected from the BTH network with the methods described in Section 4.2. The `gtk-gnutella` servent at BTH was configured to run as ultrapeer and to maintain 32–40 connections to other ultrapeers and approximately 100 connections to leaf nodes. The number of connections is a vendor preconfigured value, which is close to the suggested values [107, 108]. Although `gtk-gnutella` can communicate using the User Datagram Protocol (UDP), this functionality was turned off. Consequently, the ultrapeer used only TCP for its traffic. No other applications, with the exception of a Secure Shell (SSH) daemon, were running

on the ultrapeer for the duration of the measurements. One SSH connection was used to remotely check on the status of the measurements and the amount of free disk space. The SSH connection was idle for most of the time. The firewall was turned off during the measurements.

The total amount of PCAP data collected with `tcpdump` is approximately 33 GB. The PCAP data generated approximately 45 GB log files. The recorded traffic contains 234 million IP datagrams. The log files show 604 thousand Gnutella sessions that were used to exchange 267 million Gnutella messages. A total of 423 thousand sessions (70 %) were unable to perform a successful Gnutella handshake. The main reasons for the unsuccessful handshakes are filled-up connection queues[6] and refusal to accept uncompressed connections. The remaining sessions consist of 181,805 sessions where both peers used compression, 22 where one of the peers used compression and 10 uncompressed sessions.

4.4.2 Session Characteristics

A Gnutella session is defined as the set of Gnutella messages exchanged over a TCP connection between two directly connected peers that have successfully completed the Gnutella handshake. The session lasts until the TCP connection is closed by either FIN or RST TCP segments.

To describe the Gnutella handshake we have created three pseudo-message types: CLI_HSK, SER_HSK, and FIN_HSK. The CLI_HSK message is the first part of the handshake and it is sent by the peer that opened the TCP connection, *i. e.*, the client. The SER_HSK message is the reply from the peer that received the CLI_HSK, *i. e.*, the server. The FIN_HSK message, which is sent by the client, is the final part of the handshake.

The session duration is computed as the time duration between the instant when the CLI_HSK message is recorded (at link layer) until the time recorded for the last Gnutella message on the same TCP connection.

An incoming session is defined as being a session for which the CLI_HSK message was *received* by the ultrapeer at BTH. Outgoing sessions are sessions for which the CLI_HSK message was *sent* by the ultrapeer at BTH. Tables 4.4

[6] Code 409: "Vendor would exceed 60 % of our slots".

and 4.5 show duration (in seconds), number of exchanged messages and bytes for incoming and outgoing sessions, respectively. Table 4.6 shows the same statistics when no distinction is made between incoming and outgoing sessions.

A Gnutella session is considered valid (in the sense that it is used to compute session statistics) if the Gnutella handshake was successfully completed and at least one Gnutella message was transferred between the two hosts participating in the session. Our data contains 173,711 valid incoming sessions and 7094 valid outgoing sessions.

Type	Max	Min	Mean	Median	Stddev
Duration (s)	767553	0.03	517.30	0.86	6780.99
Messages	7561532	4	585.18	11	22580.99
Bytes	535336627	780	53059	1356	2034418

Table 4.4: Incoming session statistics.

Type	Max	Min	Mean	Median	Stddev
Duration (s)	470422	0.12	3949.86	2459.10	11170.80
Messages	2644660	6	23145.15	15716.50	58627.75
Bytes	182279191	1574	2173564	1457360	4458468

Table 4.5: Outgoing session statistics.

Type	Max	Min	Mean	Median	Stddev
Duration (s)	767553	0.03	651.98	0.87	7036.85
Messages	7561532	4	1470.34	11	25375.64
Bytes	535336627	780	136258	1357	2219411

Table 4.6: Incoming and outgoing session statistics.

The tables show that outgoing sessions transfer about 40 times more data than incoming sessions. Furthermore, by comparing the mean and median values for messages and bytes it can be observed that a few sessions transfer the majority of data. This can be explained by the hierarchy inherent in Gnutella: UPs are bound to transfer more data than their LNs. In addition, most incoming sessions have very short duration (< 1 second), which can be observed by comparing the mean and median duration values for incoming sessions. This translates in little data being exchanged.

These observations confirm earlier results reported in [40, 41, 150]. Following the taxonomy used in [41, 151], we observe that, although we analyze only signaling traffic without considering data transfers, the sessions can be divided into "mice" (*i. e.,* sessions carrying small amounts of data), and "elephants" (*i. e.,* sessions responsible for large volumes of traffic).

The same type of heterogeneity appears when we consider session duration. We observe both "dragonflies", which are very short sessions and "tortoises", which are sessions with very long duration.

4.4.3 Session Interarrival and Interdeparture Times

The statistics and models for session interarrival and interdeparture times are shown in Table 4.7 and Table 4.8. It is observed that interarrival times can be modeled by the lognormal distribution, which is subexponential. In contrast, session interdeparture times require a mixture distribution with a heavy-tailed component (Pareto distribution) to provide an acceptable fit.

A possible explanation for the appearance of the heavy-tailed component is given by the connection cap described in Section 4.4.1. When a Gnutella peer reaches the preset number of connections it does not attempt to establish more connections until existing connections are terminated. This leads to large session interdeparture times that have a non-negligible probability of occurrence.

The *absence* of the heavy-tailed component from the session interarrival times distribution can be explained as follows. We noticed that many of the short duration (< 1 second) incoming sessions presented in Section 4.4.2 transfer one BYE message and are then terminated. This behavior cannot be traced to any of the Gnutella specifications. We assume that the behavior is due to the `gtk-gnutella` implementation, but further study is required to confirm. It appears that `gtk-gnutella` discovers that the connection cap is reached after the handshake is completed. Only then it sends the BYE message to terminate the connection. Normally, this connection should have been aborted during handshake. Nonetheless, since these sessions are considered valid according to our criteria, the session interarrival times are shorter and we can model them without introducing a heavy-tailed component.

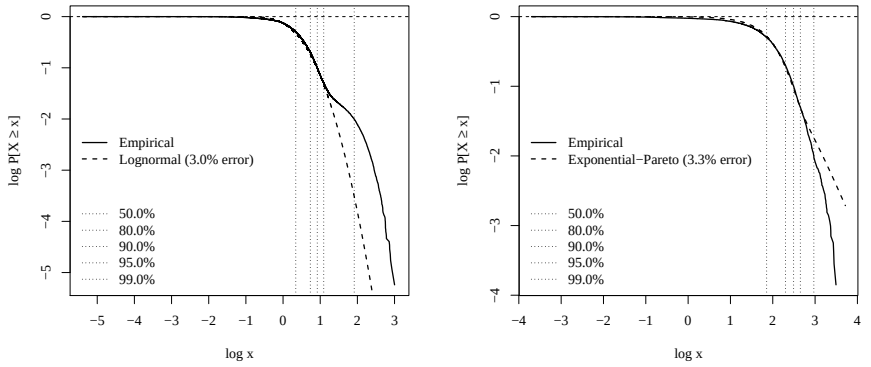
An interesting characteristic was observed when all session interarrival times

DIR	Max	Min	Mean	Median	Stddev
IN	1119.01	4.05e-6	5.47	2.20	20.38
OUT	5192.62	0.20e-3	133.99	71.78	210.34

Table 4.7: Session interarrival and interdeparture times statistics (s).

DIR	Model	$E_{\%}$
IN	$\mathrm{LN}_X(x; 0.71, 1.08)$	3.0 %
OUT	$0.77\,\mathrm{EXP}_X(x; 0.01) + 0.23\,\mathrm{PA}_X(x; 0.7, 0, 132.9)$	3.3 %

Table 4.8: Models for session interarrival and interdeparture times (s).



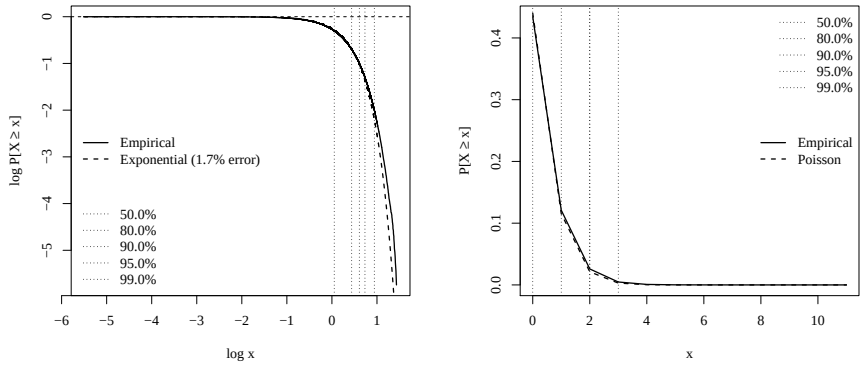
(a) Incoming.

(b) Outgoing.

Figure 4.7: Gnutella session interarrival and interdeparture times (s).

Statistic	Model	$E_{\%}$
Interarrival times (s)	$\mathrm{EXP}_X(x; 0.58)$	1.7 %
Rate (sessions/s)	$\mathrm{PO}_X(x; 0.58)$	N/A

Table 4.9: Gnutella (valid and invalid) session interarrival times.



(a) Interarrival times (s). (b) Incoming session rate (sessions/s).

Figure 4.8: Gnutella (valid and invalid) session interarrival times and incoming session rate.

were considered, that is, even those for invalid sessions. This is equivalent to interarrival times for incoming requests to open a session (incoming CLI_HSK messages). It turns out that the set of all interarrival times is exponentially distributed with parameter $\lambda = 0.58$, as shown in Figure 4.8(a) and Table 4.9. The session arrival rate was analyzed to verify that this is not a measurement error. It is well-known that exponentially distributed interarrival times imply a Poisson arrival rate [129]. As it can be observed in Figure 4.8(b), a Poisson distribution $\mathrm{PO}_X(x; 0.58)$ fits well, at least visually. Unfortunately, no $E_{\%}$ measure can be provided since the method does not work with discrete distributions. However, the edf should leave little doubt that the data is indeed Poisson distributed. The edf is plotted without log-scaled axes, since most of the data, 99.9 % of the probability mass, is clustered around the values $0, 1, \ldots, 4$.

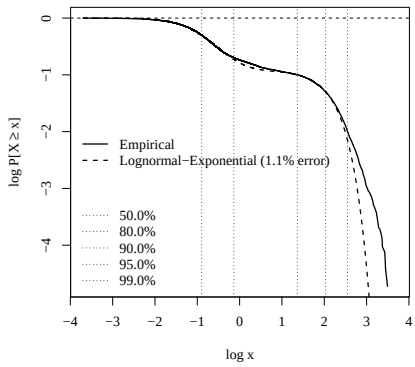


Figure 4.9: Gnutella (valid and invalid) session interdeparture times (s).

The same relation does not hold for outgoing traffic, which is well modeled by a mixture distribution $0.88\,\mathrm{LN}_X(x; -2.32, 1.41) + 0.12\,\mathrm{EXP}_X(x; 0.008)$ with 1.1 % error, as observed in Figure 4.9.

The appearance of the Poisson distribution can be explained by the mixture of arriving CLI_HSK message from different sources. If one assumes that these arrivals are generated by a number of point processes, then the superposition of point processes converges, under some general assumptions, to a Poisson distribution, when the number of sources increases [152–154].

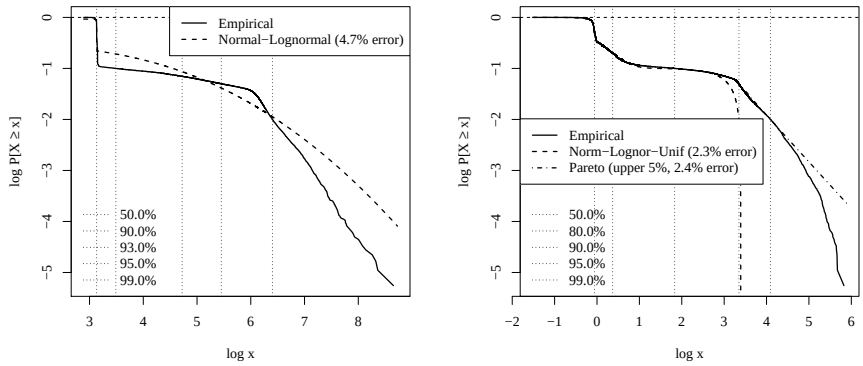
This hypothesis does however not apply to outgoing CLI_HSK due to the connection cap. Once `gtk-gnutella` reaches the preset amount of connections it does not attempt to establish new ones.

4.4.4 Session Size and Duration

The session size and duration models are reported in Table 4.10 and Figure 4.10. It is observed that the session duration statistic has a very complex ccdf, which cannot be modeled with only two distributions. This is the only reported model that uses a mixture of three distributions. Alternatively, the upper 5 % of the tail can be modeled with a Pareto distribution. The Pareto shape parameter

Statistic	Model	$E_{\%}$
Session size (bytes)	$0.69\,\mathrm{N}_X(x; 1356, 5.9) + 0.31\,\mathrm{LN}_X(x; 9.0, 3.17)$	4.7 %
Session duration (s)	$0.57\,\mathrm{N}_X(x; 0.85, 0.07)+$ $0.33\,\mathrm{LN}_X(x; 0.37, 0.96)+$ $0.10\,\mathrm{U}_X(x; 18.45, 2460)$	2.3 %
Session duration, upper 5 % (s)	$\mathrm{PA}_X(1.1, 1800, 1870.4)$	2.4 %

Table 4.10: Session size and duration models.



(a) Session size (bytes).

(b) Session duration (s).

Figure 4.10: Gnutella session size and duration.

$\alpha = 1.1$ implies that no upper bound exists for the mean session duration.

Most of the observed session sizes (64.8 %) lie in the range 1300–1400 bytes, 9.6 % are smaller than 1300 bytes and 25.6 % are larger than 1400 bytes.

4.4.5 Message Characteristics

In this section, message statistics are reported for each Gnutella message type. The message type UNKNOWN denotes messages with a valid Gnutella header, but with unrecognized message type. These messages are either experimental or corrupted. The message type ALL is used for statistics computed over all messages, irrespective of type. Only models for the aggregated message streams,

i. e., message type ALL are presented.

Table 4.11 shows interarrival times for messages received by the BTH ultrapeer and Table 4.12 shows interdeparture times for messages sent by the BTH ultrapeer. Although the PCAP timestamps have microsecond resolution [103], the times presented here have only 100 μs precision. This is due to memory limitations in the post-processing software.

Summing over the number of samples for each message type does not add up to the value shown in the number of samples for message type ALL. This is caused by the analysis software, which ignores messages that generate negative interarrival and interdeparture times. Negative times appear because the application flow reassembly handles several (typically more than a hundred) connections at the same time. On each connection the timestamp for arriving packets is monotonically increasing. However, the interarrival and interdeparture statistics are computed across all connections. To ensure monotonically increasing timestamps even in this case, new messages from arbitrary connections are stored in a buffer, where they are sorted by timestamp. The size of the buffer is limited to 500,000 entries due to memory management issues. By summing the entries in the "Mean" column in Table 4.19 it can be observed that, on average, there are 280 incoming and outgoing messages per second. This means that the buffer can store about 30 minutes of average traffic and much less during traffic bursts. If there are delayed messages due to TCP retransmissions or other events, they reach the buffer too late and are discarded.

The large interarrival and interdeparture times in handshake messages (CLI_HSK, SER_HSK, FIN_HSK) observed in Table 4.11 and Table 4.12 occur because once a servent reaches the preset amount of connections, it no longer accepts or attempts to open new connections until one or more of the existing connections is closed. This behavior also explains the large interarrival and interdeparture times for BYE messages.

It is interesting to see that interarrival times are exponentially distributed as shown in Table 4.13 and Figure 4.11. Analysis of the arrival process reveals that this is is not a pure Poisson process, but rather a compound Poisson process [129, 155, 156] since simultaneous message arrivals do occur. To understand why this happens, recall that before the messages can be extracted from the TCP flows, these flows pass through a decompression layer. Typically, a single TCP segment

Type	Max	Min	Mean	Median	Stddev	Samples
CLI_HSK	28.4591	0.0001	1.7246	1.1256	1.8644	551148
SER_HSK	5185.0490	0.0001	19.6294	0.2090	92.1849	48432
FIN_HSK	1118.9920	0.0001	5.3165	2.1942	19.4800	178783
PING	13.5871	0.0001	0.2762	0.1931	0.2726	3457169
PONG	2.2624	0.0001	0.1404	0.0979	0.1383	9086918
QUERY	1.4514	0.0001	0.0343	0.0240	0.0340	59010007
QUERY_HIT	19.2778	0.0001	0.1842	0.0976	0.2661	6932327
QRP	50.0632	0.0001	2.0475	1.0534	2.8707	478451
HSEP	1780.4420	0.0003	6.1560	4.3834	8.4758	154742
PUSH	40.1396	0.0001	0.0677	0.0405	0.1157	24934450
BYE	1119.5930	0.0001	5.9160	2.3591	22.3494	160695
VENDOR	30.8037	0.0001	0.4346	0.2207	0.5993	9669915
UNKNOWN	51576.8600	3.0680	2075.3190	6.9379	9298.3600	35
ALL	9.8299	0.0001	0.02436	0.0169	0.0243	114663084

Table 4.11: Message interarrival time statistics (s).

Type	Max	Min	Mean	Median	Stddev	Samples
CLI_HSK	5189.2340	0.0002	17.9655	0.1273	88.8506	52902
SER_HSK	28.4595	0.0003	1.7298	1.1287	1.8712	549456
FIN_HSK	5185.5150	0.0006	28.4784	0.3305	110.2372	33373
PING	20.5910	0.0001	1.3773	0.5077	2.1342	694550
PONG	2.7215	0.0001	0.1573	0.1012	0.1682	34639367
QUERY	12.1151	0.0001	0.0295	0.0003	0.0541	70066326
QUERY_HIT	19.2818	0.0001	0.2188	0.1285	0.2885	6309719
QRP	603.3599	0.0001	2.6350	0.0004	19.8572	680103
HSEP	358.3067	0.0001	2.5020	1.4089	5.8293	384084
PUSH	76.5303	0.0001	0.0429	0.0003	0.1713	38105019
BYE	3849.4550	0.0001	134.8121	77.2090	187.7784	7033
VENDOR	64.6689	0.0001	1.8253	1.1124	2.4838	525269
UNKNOWN	N/A	N/A	N/A	N/A	N/A	1
ALL	1.5450	0.0001	0.0178	0.0003	0.0353	152047214

Table 4.12: Message interdeparture time statistics (s).

DIR	Message	Model	$E_{\%}$
IN	`ALL`	$\mathrm{EXP}_X(x; 40.96)$	0.16 %
OUT (upper 26.1 %)	`ALL`	$0.261\,\mathrm{EXP}_X(x; 20.23)$ (see Table 4.14 for the body)	3.8 %

Table 4.13: Models for message interarrival and interdeparture times (s).

Interdeparture times	0.0001	0.0002	0.0003	0.0004	0.0005
Probability	0.024	0.515	0.155	0.033	0.012

Table 4.14: Probability mass points for message interdeparture times (s).

carries several Gnutella messages. All of them receive the same timestamp, since they traveled in bulk all the way from the source to the destination. Models for the bulk-size distributions are provided in Table 4.16 and Table 4.17.

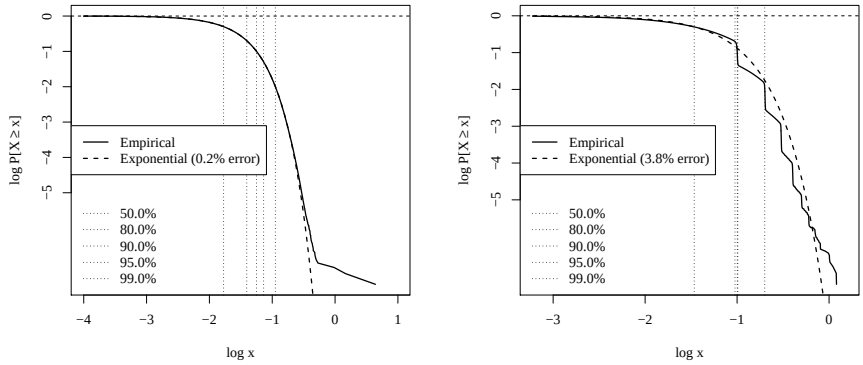
The appearance of the Poisson distribution can be explained by arguments similar to those considered in Section 4.4.3.

Message interdeparture times have an interesting distribution. As it can be observed in Table 4.14, approximately 73.9 % of the probability mass is clustered around the values 0.0001–0.0005. The remaining 26.1 % of the probability mass can be modeled by an exponential distribution ($\lambda = 20.23$) with 3.8 % error.

Table 4.15 shows the message size statistics for each Gnutella message type. In contrast to the other tables, messages are not classified by direction (incoming or outgoing). The rationale is that the message size is independent of message direction. It can be observed that, on average, QUERY_HIT and QRP messages have the largest size. They are closely followed by handshake messages, where the capability headers account for most of the data. It is interesting to notice that the maximum size of QUERY_HIT messages is 39 KB, which is an order of magnitude larger than the 4 KB specified in [106].

The model for the message bulk size is reported in Tables 4.16–4.17. Bulks of size 1–15 use 99.7 % of the probability mass. The remaining 0.3 % of the probability mass is modeled with a Pareto distribution.

The message duration statistic can be useful to infer waiting times at the application layer, when a message is transported in two or more TCP segments. The statistic is defined as the time difference between the first and last TCP



(a) Message interarrival times (s)

(b) Message interdeparture times (s), upper 26.1 %

Figure 4.11: Message interarrival and interdeparture times.

Type	Max	Min	Mean	Median	Stddev
CLI_HSK	696	22	336.91	328	65.69
SER_HSK	2835	23	386.83	369	145.69
FIN_HSK	505	23	107.92	76	88.55
PING	34	23	25.48	23	3.88
PONG	464	37	74.96	61	38.68
QUERY	376	26	70.17	55	46.40
QUERY_HIT	39161	58	590.28	358	1223.58
QRP	4124	29	608.60	540	596.70
HSEP	191	47	70.39	71	28.15
PUSH	49	49	49.00	49	0.00
BYE	148	35	40.02	37	15.84
VENDOR	177	31	36.45	33	19.51
UNKNOWN	43	23	23.53	23	3.24
ALL	39161	22	93.45	49	303.26

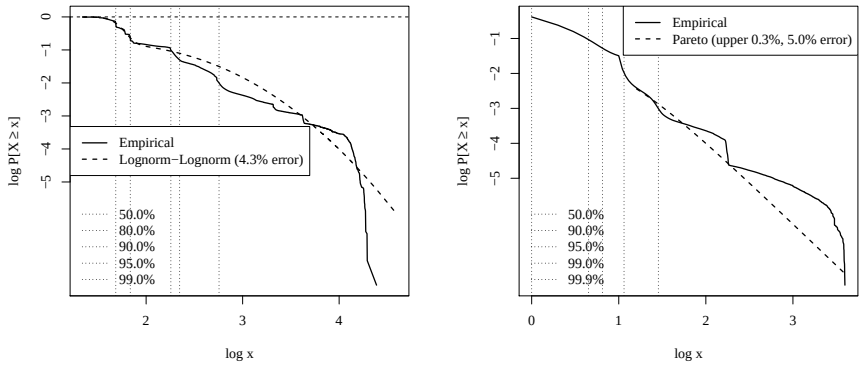
Table 4.15: Message size statistics (bytes).

DIR	Message	Model	$E_{\%}$
IN/OUT	`ALL`	$0.81\,\mathrm{LN}_X(x; 3.94, 0.23) + 0.19\,\mathrm{LN}_X(x; 5.14, 1.24)$	4.3 %
IN/OUT	Bulk size	$0.003\,\mathrm{PA}_X(x; 0.42, 15, 9.6)$	5.0 %

Table 4.16: Message size (bytes) and bulk size distribution.

Bulk size (messages)	1	2	3	4	5
Probability	0.586	0.173	0.082	0.049	0.031
Bulk size (messages)	6	7	8	9	10
Probability	0.020	0.012	0.008	0.005	0.003
Bulk size (messages)	11	12	13	14	15
Probability	0.019	0.005	0.002	0.001	0.001

Table 4.17: Probability mass points for message bulk size.



(a) Message size distribution

(b) Message bulk size distribution (upper 0.3 %)

Figure 4.12: Gnutella message size (bytes) and bulk distribution.

Type	Max	Min	Mean	Median	Stddev	Samples
CLI_HSK	349.3015	0	0.0308	0	1.0412	604072
SER_HSK	52.2645	0	0.0032	0	0.1350	597896
FIN_HSK	68.6295	0	0.0057	0	0.2838	212162
PING	251.2914	0	0.0273	0	0.6309	4151799
PONG	2355.8650	0	0.0077	0	0.5881	43727188
QUERY	2355.8650	0	0.0035	0	1.3271	129078986
QUERY_HIT	480.8159	0	0.0243	0	1.0260	13242329
QRP	753.1904	0	0.1883	0	1.6019	1158596
HSEP	74.0482	0	0.0017	0	0.2186	538834
PUSH	135.5155	0	0.0023	0	0.2017	63040718
BYE	148.7292	0	0.0386	0	0.5194	167726
VENDOR	391.3439	0	0.0117	0	0.2451	10195389
UNKNOWN	1.0418	0	0.2995	0	0.4294	38
ALL	2355.8650	0	0.0065	0	0.9968	266715733

Table 4.18: Message duration statistics (s).

segments that were used to transport the message. When a message uses only one TCP segment the time duration for that specific message is zero.

From the median column in Table 4.18 it can be observed that at least 50 % of the messages require just one TCP segment. The PONG and QUERY_HIT message rows contain extreme values for the maximum duration, namely 2355.9 seconds ($\approx$ 39 minutes). These values are most likely the result of malfunctioning or experimental Gnutella servents.

4.4.6 Transfer Rate Characteristics

This section reports on transfer rates in bytes per second and in messages per second for each Gnutella message types. All statistics are computed over 950,568 samples. The number of samples is equal to the time duration expressed in seconds (approximately 11 days) for the available measurement data. Models are reported only for aggregate message flows, *i. e.*, type ALL messages. As it can be observed in Table 4.21 both incoming and outgoing transfer rates are heavy-tailed. In terms of specific message types, QUERY and QUERY_HIT messages dominate incoming and outgoing streams, both in terms of average message rate and of average byte rates. This is expected since the Gnutella

system is used primarily for searching for files.

DIR	Max	Min	Mean	Median	Stddev
IN	6471	0	120.63	111	84
OUT	4164	0	159.96	153	61

Table 4.19: Gnutella (ALL) message rate (msg/s) statistics.

DIR	Max	Min	Mean	Median	Stddev
IN	1745341	0	12883	10113	24287
OUT	370825	0	13338	12062	7624

Table 4.20: Gnutella (ALL) byte rate (bytes/s) statistics.

DIR	Model	$E_{\%}$
IN	$0.76\,\mathrm{LN}_X(x; 9.26, 0.37)+$ $0.23\,\mathrm{PA}_X(x; 1.06, 0, 4003)$	5.2 %
OUT	$0.81\,\mathrm{LN}_X(x; 9.43, 0.39)+$ $0.19\,\mathrm{PA}_X(x; 0.63, 0, 3704)$	5.3 %

Table 4.21: Gnutella (ALL) byte rate (bytes/s) modeling results.

Table 4.24 provides the summary statistics for the IP byte rates. It is interesting to note that the mean and median IP byte rates are very similar to the corresponding statistics for Gnutella byte rates shown in Table 4.23. These values alone indicate that the compression of Gnutella messages does not yield large gains. However, if one takes into consideration the maximum and standard deviation values it can be observed that the compression removes much of the burstiness from the application layer, leading to smoother traffic patterns. This effect is visible if one compares Figure 4.14(a) to Figure 4.14(b).

Type	DIR	Max	Min	Mean	Median	Stddev
CLI_HSK	IN	12	0	0.58	0	0.79
CLI_HSK	OUT	30	0	0.06	0	0.56
SER_HSK	IN	20	0	0.05	0	0.48
SER_HSK	OUT	12	0	0.58	0	0.79
FIN_HSK	IN	9	0	0.19	0	0.46
FIN_HSK	OUT	18	0	0.04	0	0.34
PING	IN	72	0	3.64	3	1.94
PING	OUT	17	0	0.73	0	1.56
PONG	IN	130	0	9.56	9	4.33
PONG	OUT	433	0	36.44	36	19.12
QUERY	IN	347	0	62.08	60	19.64
QUERY	OUT	875	0	73.71	69	34.08
QUERY_HIT	IN	531	0	7.29	5	9.82
QUERY_HIT	OUT	272	0	6.64	5	7.39
QRP	IN	45	0	0.50	0	0.98
QRP	OUT	283	0	0.72	0	7.18
HSEP	IN	20	0	0.16	0	0.41
HSEP	OUT	23	0	0.40	0	0.68
PUSH	IN	1068	0	26.23	23	19.34
PUSH	OUT	4091	0	40.09	32	37.32
BYE	IN	40	0	0.17	0	0.43
BYE	OUT	118	0	0.01	0	0.15
VENDOR	IN	6385	0	10.17	1	76.17
VENDOR	OUT	24	0	0.55	0	0.80
UNKNOWN	IN	1	0	0.00	0	0.01
UNKNOWN	OUT	1	0	0.00	0	0.00

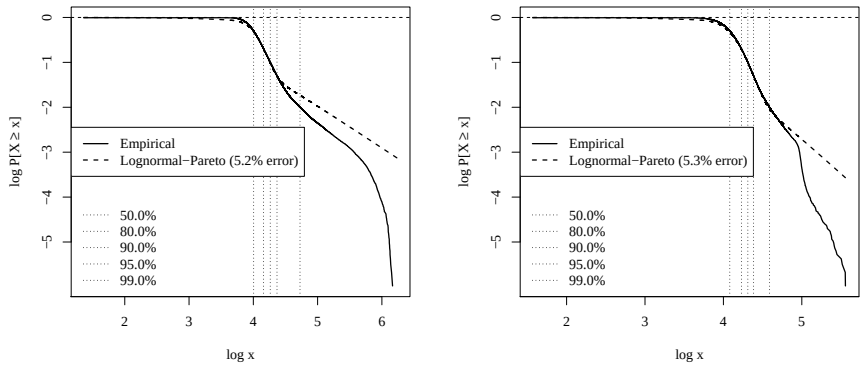
Table 4.22: Message rate (msg/s) statistics.

Type	DIR	Max	Min	Mean	Median	Stddev
CLI_HSK	IN	4126	0	187	0	258
CLI_HSK	OUT	14519	0	27	0	273
SER_HSK	IN	12507	0	31	0	289
SER_HSK	OUT	4001	0	212	0	306
FIN_HSK	IN	982	0	15	0	42
FIN_HSK	OUT	4474	0	9	0	94
PING	IN	1665	0	92	92	50
PING	OUT	503	0	19	0	45
PONG	IN	17043	0	1213	1173	541
PONG	OUT	26050	0	2235	2162	1179
QUERY	IN	24101	0	4441	4317	1426
QUERY	OUT	46424	0	5088	4702	2511
QUERY_HIT	IN	1736791	0	4868	1912	23917
QUERY_HIT	OUT	360235	0	3355	1837	5229
QRP	IN	47340	0	389	0	1408
QRP	OUT	152820	0	353	0	3660
HSEP	IN	940	0	8	0	21
HSEP	OUT	2185	0	32	0	58
PUSH	IN	52332	0	1285	1127	948
PUSH	OUT	200459	0	1964	1568	1829
BYE	IN	1720	0	6	0	16
BYE	OUT	4956	0	1	0	11
VENDOR	IN	210702	0	347	33	2514
VENDOR	OUT	2197	0	44	0	81
UNKNOWN	IN	23	0	0	0	0.1
UNKNOWN	OUT	43	0	0	0	0.1
ALL	IN	1745341	0	12883	10113	24287
ALL	OUT	370825	0	13338	12062	7624

Table 4.23: Message byte rate (bytes/s) statistics.

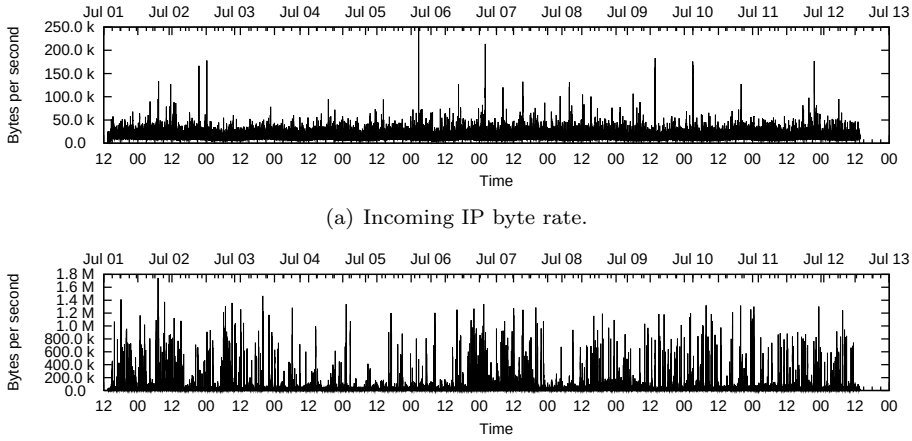
DIR	Max	Min	Mean	Median	Stddev
IN	249522	0	11536	10961	4075
OUT	176986	0	12668	12037	5722

Table 4.24: IP layer byte rate (bytes/s) statistics.



(a) Incoming (b) Outgoing

Figure 4.13: Gnutella (ALL) byte rates (bytes/s) models.



(a) Incoming IP byte rate.

(b) Incoming Gnutella byte rate (message type ALL).

Figure 4.14: Comparison of compressed and decompressed traffic.

4.5 Summary

In this chapter we examined the characteristics of Gnutella traffic. An 11-days long Gnutella link-layer packet trace collected at BTH was systematically decoded and analyzed. We extracted several traffic characteristics and constructed statistical models for some of them. The emphasis for the characteristics has been on accuracy and detail, while for the traffic models the emphasis has been on analytical tractability and ease of simulation. To the author's best knowledge this is the first work on Gnutella that presents statistics down to message level.

The results show that incoming requests to open a session follow a Poisson distribution. Incoming messages of mixed types can be described by a compound Poisson distribution. Mixture distribution models for message transfer rates include a heavy-tailed component.

Chapter 5

Overlay Routing Protocol

In this chapter we present the current implementation of the ORP framework and the associated performance results. ORP consists of two protocols: the Route Discovery Protocol (RDP) and the Route Management Protocol (RMP).

RDP is used to find network paths subject to various QoS constraints [46, 47]. To achieve this goal, RDP uses a form of selective diffusion in which a node that receives a path request forwards the request only on outgoing links that do not violate the QoS constraints. Eventually, the request arrives at the destination node if there is at least one path satisfying the constraints. At that point, a reply message containing information about the complete path is sent back to the requesting node. RDP is based on ideas presented in [91, 157, 158].

The purpose of RMP is to alleviate changes in the path QoS metrics, due to node and traffic dynamics. This is done through a combination of path restoration and optimization algorithms for traffic flow allocation on bifurcated paths. The purpose of the flow allocation is to spread the demand on multiple paths towards the destination [86]. The design of RMP is influenced by ideas presented in [159, 160].

In Section 5.1 we give a brief overview of fundamental elements of QoS routing. Based on this, we discuss in Section 5.2 a number of assumptions used in the design of RDP and RMP. The protocol specification and the performance results for RDP and RMP are presented in Section 5.3 and Section 5.4, respectively.

5.1 Elements of QoS Routing

In QoS networks every link and every node has a state described by specific QoS metrics. The *link state* can consist of available bandwidth, delay and cost whereas the *node state* can be a combination of available memory, CPU utilization and harddisk storage. The link state and the node state may be considered separately or they may be combined. The focus in the reminder of this thesis is on link state.

Recall from Section 1.1 that routing is the process of finding a path between two hosts in a network. In QoS routing, the path must be selected such thatQoS metrics of interest stay within specific bounds. The routing process relies on a *routing algorithm* for computing constrained paths and on a *routing protocol* for distributing state information.

There are three basic forms of storing state information: *local state*, *global state* and *aggregated (partial) state* [161].

When a node keeps local state, it maintains information about its outgoing links only. No information about the rest of the network is available.

A global state is the combination of local states for all nodes in a graph. Global states are imprecise (*i. e.*, they are merely approximations of the global state) due to non-negligible delay in propagating information about local states. When the network size grows, the imprecision grows as well. This makes it hard to maintain an accurate picture about resource availability in the network and it has severe impact on QoS routing.

Aggregated state aims to solve scalability issues in large networks. The basic idea is to group together adjacent nodes into a single *logical node*. The local state of a logical node is the aggregation of local states for physical nodes that are part of the logical node. Similar to the case of global state, this leads to imprecision that grows with the amount of state information aggregated in the logical node.

Imprecision, also called uncertainty [162], is not generated by aggregation only. Other sources of uncertainty are network dynamics (churn), information kept secret by ISPs due to business reasons, as well as approximate state information due to systematic or random errors in the measurement process [162]. An interesting solution suggested for mitigating these problems is to replace

the deterministic state information metrics with random variables. In this case, the routing algorithm must be changed such as to select feasible paths on a probabilistic basis, with the result that the selected paths are those most likely to satisfy the QoS constraints [163, 164]. However, a non-trivial problem with this approach lies in the estimation of the probability distributions for state variables [165].

There are three different classes of routing strategies, each corresponding roughly to one form of maintaining state information: *source routing*, (*flat*) *distributed routing* and *hierarchical routing* [166].

In source routing the nodes are required to keep global state and the feasible path is computed at the source node. The main advantage in this case is that route computation is performed in a centralized fashion, avoiding some of the problems associated with distributed computation. The centralized computation can guarantee loop-free routes. One disadvantage of source routing is because of the requirement to maintain global state. In a network where the QoS metrics often change, this requires large communication overhead in order to keep the state information updated. Additionally, due to the propagation delay, the state information may become stale before reaching the destination. This leads to imprecise state information, as explained above. Furthermore, depending on the network size and the number of paths to compute, the source routing algorithm can result in very high computational overhead [166].

Distributed routing, typically, also relies on nodes maintaining global states, but the path computation is performed in a distributed fashion. This diminishes computational overhead and also allows concurrent computation of multiple routes in search for a feasible path. Distributed computation suffers from problems related to distributed state snapshot, deadlock and loop occurrence [161]. Additionally, when global state is maintained, distributed routing shares with source routing the problems related to imprecise state information.

Some suggestions on using flooding-based algorithms, require nodes to maintain local state only [91, 167]. This mitigates problems related to imprecise state information. However, flooding-based algorithms tend to generate large volumes of traffic compared to the other forms of routing.

In hierarchical routing, the network is divided into groups of nodes and the state information is aggregated for the nodes participating in a group. With

this form of aggregation, a group appears as a logical node. One node in the group is designated *leader* or *border node* and acts as a gateway for the communication with other logical nodes. Each group can in turn be divided into smaller groups. Using this form of recursion, several hierarchical levels can be created. Nodes maintain global state information for peers within a group and aggregated state information about the other groups. The major advantage of hierarchical routing is scalability [159, 166]. In particular, since nodes maintain aggregated state information there is less state information to be transmitted to other nodes, hence less communication overhead. For the same reason, there is also less computational overhead. However, each level of hierarchy induces additional uncertainty in the state information. This problem becomes more difficult when several QoS metrics must be aggregated, since for some topologies there can be no meaningful way to combine the metrics [161]. Some solutions for topology aggregation are presented in [168, 169].

The networks considered here rely on end-nodes. Since end-nodes are under the control of their users, they tend to be an unreliable infrastructure. By this, we mean that end-nodes can be turned off by their users, effectively removing them from the network. This type of node churn is similar to the topology dynamics occurring in mobile ad-hoc networks, when stations move out of radio range. Routing protocols that handle topology dynamics can be classified as *proactive* or *reactive* protocols.

Proactive protocols, such as destination sequence distance vector (DSDV) periodically update the routing tables [170]. In contrast, reactive protocols (*e. g.*, dynamic source routing (DSR) and ad-hoc on-demand distance vector (AODV)) update the routing tables only when routes need to be created or adjusted due to changes to topology [170]. Proactive protocols are in general better at providing QoS guarantees for real-time traffic such as multimedia. Their disadvantage lies in the traffic volume overhead generated by the protocol itself. Reactive protocols scale better than proactive protocols, but will experience higher latency when setting up a new route [170].

5.2 Design Assumptions

In this section are discussed a number of assumptions that have influenced the design of RDP and RMP. These assumptions pertain to the environments in which the protocols are running and to the type of media being routed.

The first assumption is that ORP is executed on end-nodes consisting of off-the-shelf hardware (*e. g.*, PC, Macintosh) running a general purpose operating system (*e. g.*, Linux-based operating systems, Mac OS X, Microsoft Windows). ORP runs in unprivileged mode accessing peripheral devices through standard calls to the operating system API.

ORP requires that nodes interested in performing QoS routing form an application-layer overlay. The overlay may be structured (*i. e.*, a DHT) or unstructured. The only requirements for it are the ability to forward messages and to address individual nodes through some form of universally unique identifier (UUID) [114]. In the simulations we assumed that ORP runs on top of a Gnutella-like topology.

The type of services considered for the QoS layer are currently restricted to those that require interactive and non-interactive live unicast multimedia streams only.

Multimedia stream refers to a stream containing audio, video, text (*e. g.*, subtitles or Text-TV), control data (*e. g.*, synchronization data), or a combination thereof. If an application chooses to use several media streams (*e. g.*, one stream per media type), the QoS routing protocol treats them independently of each other and assumes that the application is capable on its own of performing synchronization or any other type of stream merging processing.

The multimedia streams within the scope of ORP are of unicast type (one-to-one). Multicast streams (one-to-many) are not withing the scope of this thesis. Furthermore, the streams are considered to be live, which means that the receiver is not willing to wait until the whole stream data is received, but would rather start watching and listening to the contents as soon as enough data is available for rendering.

Interactive multimedia streams refers to streams generated by user interaction as in a video conference or a VoIP call. Conversely, non-interactive multimedia streams do not involve any interaction between users as is the case

of IPTV or music streaming.

Applications using ORP request overlay paths from the node where they are running to specific destinations, along with constraints attached to each path. The path is discovered using RDP as described in Section 5.3.

It is assumed that each node is capable of estimating the available host resources (*e. g.,* RAM, storage) as well as link properties (*e. g.,* residual bandwidth, round-trip time (RTT), loss rate) to its one-hop neighbors in the overlay. The amount of available host resources can be obtained using calls to the operating system API. Link properties can be estimated using active measurements [171–175]. Nodes are expected to exchange this information using RMP, as described in Section 5.4.

Furthermore, it is assumed that ORP-enabled software cannot interfere with resources used by processes outside ORP's scope. In other words, ORP cannot perform resource reservation other than on residual resources, which are resources currently unused by other applications running simultaneously on the node. Consequently, it is expected that the volume of available resources will fluctuate.

Large fluctuations can drive the node into resource starvation. During resource starvation the node is unable to honor some or all of the QoS constraints. This type of events can lead to degradation in the quality of rendered media (*e. g.,* MPEG frames that are lost, garbled, or arrive too late). Applications may be able to tolerate quality degradation for very short periods of time or even recover from brief degradation by using forward error correction (FEC) codes or retransmissions. However, prolonged quality degradation may eventually lead to user dissatisfaction with the quality of the service. Each node must therefore carefully monitor the link properties to each of its immediate neighbors. If resource starvation is detected, or anticipated, then a new feasible path should be found and traffic re-routed on it. It is clear that the latency experienced in obtaining measurement results is in fact an upper bound on how fast ORP can react to changes. Estimating the effect of the upper bound on the performance of ORP has not been studied yet, but is planned as an item for future work.

5.3 Route Discovery Protocol

RDP is a distributed routing protocol relying on local state information. This architectural choice is motivated because of:

i) overlay networks with a large number of nodes,

ii) unreliable end-nodes as infrastructure,

iii) topology dynamics.

To quantify the impact of the first factor, consider the Gnutella network and the Kademlia-based [34] DHT used by Azureus[1]. Both systems are good candidates for running ORP. Recent measurements indicate that these systems have more than one million concurrent peers [176, 177]. The memory requirement to store the complete network topology makes it impractical to maintain global state in each node. If global state cannot be maintained, then source routing is not a viable alternative. The computational overhead associated with path selection in topologies of this size is yet another argument against using source routing.

The second factor implies that elements critical for the correct behavior of the protocol should not depend on single nodes. For example, in hierarchical routing, if the border node leaves the overlay, the hosts represented by the logical node are cut-off from the rest of the network. This can be counteracted by a leader-election protocol at the cost of increased complexity. Taking into account this issue as well as the problem of state aggregation, it was decided to leave out hierarchical routing from ORP's architecture.

The third factor, topology dynamics, refers to node churn or to significant changes in the state information. Two types of latencies come into play when information about these events must be disseminated throughout the network. The first type of latency is the time duration required to detect that such an event has occurred, which is directly related to the measurement method used. The second type of latency is the one-way delay to nodes receiving the event information. When the sum of these two latencies grows, the probability of nodes receiving stale information increases. There is nothing that can be done in the

[1] Azureus is a very popular BitTorrent client.

case of the first latency, other than changing the measurement method. However, one completely avoid the second type of latency if distributed routing with local state is used, as explained in Section 5.1. In this case, path computation is achieved by selective diffusion over the feasible paths in the network.

Obviously, selective diffusion is a form of flooding and as such it comes with a cost in terms of bandwidth overhead. The benefit of using flooding can hardly motivate the cost in a network where the topology changes slowly. However, when the dynamics become more aggressive the benefit to cost ratio increases, providing a more compelling argument in favor of flooding. This is the case of P2P networks, which are typical environments with aggressive traffic dynamics, as it was shown in Chapter 4 in Gnutella's case.

5.3.1 Protocol Elements

All ORP messages[2] start with the generic header shown in Figure 5.1. Field

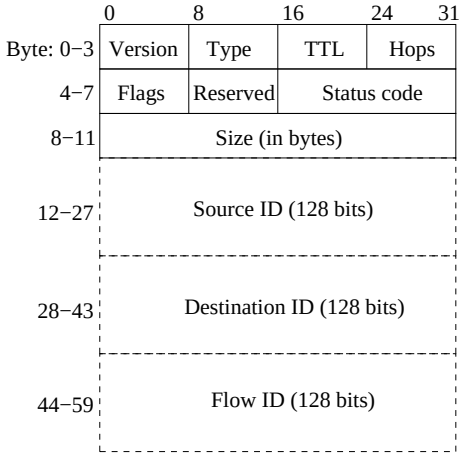


Figure 5.1: ORP generic packet header.

values in the packet header are arranged in network byte order. The following elements are included in the ORP packet header:

[2]The terms *message* and *packet* are treated as synonyms in the thesis.

Version ORP protocol version. At the moment of writing the protocol is at version 1.

Type ORP packet type, showed below.

Field value	Packet type
0	reserved
1	control packet (CP)
2	acknowledgement packet (AP)
3	data packet (DP)
4	used by RMP

TTL time-to-live, denoting how many overlay hops the packet is allowed to travel.

Hops indicates the amount of links the packet already has passed. If the value in the Hops field equals the value in the TTL field the packet is dropped.

Flags bitfield arranged as |0|0|E|0|D|C|B|R|, where 0 denotes unused bits.

- E indicates that the node is leaving the overlay and all routes associated with *Source Id* should be rerouted or deleted.
- D indicates that the path associated with the *Flow Id* should be deleted.
- C denotes a route change.
- B denotes a bidirectional route request.
- R indicates a redundant AP.

Reserved For future use.

Status code Used to exchange status codes among nodes.

Size Packet size in bytes excluding the generic header.

Source ID UUID denoting the source node of the packet[3], also abbreviated as *SrcID*.

[3]ORP UUIDs are defined as specified in [114].

Destination ID UUID denoting the destination node of the packet, also abbreviated as *DstID*.

Flow ID UUID of the flow to which this packet belongs, also abbreviated as *FlowID*.

RDP uses both a TTL field and a hops field. This is in contrast with, *e. g.*, IP, where only TTL is available. The reason for using both fields is that the hops field can relay distance information, which can be useful for selecting a proper TTL value. Consider for example a network where each node can be reached in N hops or less. In the absence of any information, nodes use TTL=N. Suppose that a node v_0 forwarding control traffic learns from the hops field that another node v_1 is $M < N$ hops away. Based on this information, if node v_0 wants to open a route to v_1 it can set the TTL field to M in order to reduce the flooding, thus saving bandwidth and CPU utilization.

The use of 128-bit identifiers is mandated by the UUID specification [114]. Even if direct use of UUIDs is not desired, a 128-bit large field offers the following advantages:

- IPv6 addresses can be mapped directly on this field,
- IPv4 addresses can be mapped on this field, provided padding is used, or by using IPv6 mapping,
- addressing used by other systems can be directly mapped on this fields if the address length is equal or smaller than 128 bits, as in the case of Gnutella, or mapped by truncating the address space when it exceeds 128 bits.

RDP uses two different kinds of packets: control packets (CPs) and acknowledgement packets (APs).

A CP begins with the generic header followed by a data structure called *QoS map*, as shown in Figure 5.2. The QoS map starts with the flow demand, *i. e.*, with the QoS constraints for the requested path. ORP currently supports two type of QoS constraints: *minimum bandwidth* specified in kilobytes per second and the *maximum path delay*, specified in milliseconds. We plan to integrate additional constraint types in future ORP versions. The *timestamp*,

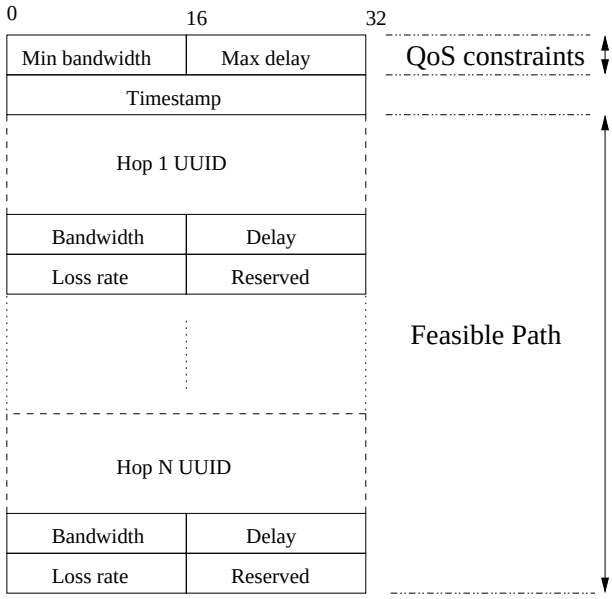


Figure 5.2: QoS map.

in Coordinated Universal Time (UTC) format, indicates the time when the QoS map was sent to the next hop.

Following the path QoS constraints comes the *feasible path* explored so far by the CP in question. Each node that forwards the CP appends an entry to the feasible path. The entry consists of the UUID of the downstream node and a set of QoS metrics associated with the link on which the packet is forwarded. Metrics currently supported by ORP are *bandwidth* (expressed in kilobytes per second), *delay* (expressed in milliseconds) and packet *loss rate*. The packet loss rate is a fraction with the accuracy $1/(2^{16}-1)$. A loss rate of 0 indicates that no packets are lost whereas a loss rate of $2^{16}-1$ denotes that all packets are lost. The use of the last field is not defined yet. The manner in which the QoS metrics are computed is not within the scope of the thesis, as stated in Section 5.2.

When the destination node receives a CP it assembles an AP by copying the triple (SrcID, DstID, FlowID) and feasible path from the CP. Then, the AP is sent back to the source node over the reverse feasible path[4]. The purpose of APs is to inform nodes on the feasible path that a complete route to the destination has been found.

After a route has been established between two nodes, the source node can start sending data. Data (payload) is transported in data packets (DPs). DPs using the same FlowID are said to form a *flow*. The actual format of these packets is left open. The only requirement is that they begin with the generic header. The simulations presented in the reminder of this chapter are concerned with control traffic (*i. e.*, CPs and APs) and do not include DPs.

Each node maintains a number of flow relays (FRs). A FR is an abstract data type associated with a single flow or a group of flows (*flow bundle*) sharing common characteristics (*e. g.*, the same QoS constraints). The information in the FRs is updated by CPs and APs associated with the flows and by QoS measurements performed by the node in question.

At each node a list of active CPs is maintained. Each entry in the list consists of a copy of the triple (SrcID, DstID, FlowID) from the corresponding CP. A CP is considered active from the time it is forwarded towards the destination until a corresponding AP is received or a timeout occurs. Further, a timer T_{out} is associated with every entry in the list. When the timer expires the corresponding entry is removed from the list.

5.3.2 Path Discovery Procedure

When a node in the overlay wants to open a route to another overlay node it assembles a CP with the desired QoS constraints. The requesting node, also called *source node*, sends the CP to all adjacent nodes connected by links satisfying the QoS constraints. If at least one feasible link is found, the CP is added to the list of active CPs and a timer is started accordingly. If after T_{out} seconds no information is received, the CP is considered lost and it is removed from the active CP list.

[4]Traveling on the reverse feasible path between node v_1 and node v_N means traveling in the opposite direction on the feasible path (*i. e.*, over hops $v_N, v_{N-1}, \ldots, v_1$).

Each node that forwards the CP computes the value of T_{out} by the following formula:

$$T_{out} = 0.2 \times (TTL - Hops). \tag{5.1}$$

Initially, $(TTL - Hops)$ was multiplied by 2 instead of 0.2 in order to obtain a conservative estimate of the round-trip time from the node in question to the destination node. However, it was observed that the T_{out} values were excessively large, keeping links blocked for unnecessarily long durations of time. This problem occurs because in the Equation 5.1 it is implicitly assumed that the delay of each link is one second, when in fact the link delays are much smaller. Several tests were performed, in which the multiplying factor was successively lowered. Based on the results, it was decided to scale down the T_{out} values by a factor of 1/10. The advantage of the new equation is that link bandwidth is freed up much faster, decreasing the call blocking. The disadvantage is that in some very few cases, the timer expires before the corresponding AP is received.

Each node receiving a CP checks whether its node UUID is matching an entry in the feasible path of the CP or not. A matching entry means that the CP has entered a loop and causes the CP to be disregarded.

If no matching node UUID entry is found and at least one feasible link exists, the received CP is added to the list of active CPs. For each feasible link found, the adjacent node UUID (denoted by Hop UUID in Figure 5.2) and the QoS statistics of the link are appended to a copy of the received CP. The modified CP is then forwarded over the link in question. This process is performed for each link, except for the one connected to the node that sent the original packet.

If no feasible link exists, the CP is dropped and no further actions are taken. The receiving and forwarding process is repeated at several nodes until one or more CPs reach the *destination node*, or all CPs are dropped by intermediate nodes. If all CPs are lost, the nodes on the feasible path eventually experience T_{out} timeouts and thus are able to free any reserved resources.

The first CP that arrives to the destination node is used to obtain the feasible path between source and destination. This policy favours minimum delay feasible paths. However, the downside of this approach is that it can create hot-spots in the network, *i. e.*, congested links. Investigating policies for load-balancing is an item for future work.

Upon receiving the CP, the destination node creates a flow relay (FR) for

packets corresponding to the FlowID in the CP and sends an AP back to the source node over the reverse feasible path. If the received CP indicates that the source node wishes bidirectional communication, then the destination node begins immediately a route discovery process towards the source node, using the same QoS constraints specified in the CP. This feature can be useful for example for VoIP, where the path in each direction uses the same QoS constraints.

All subsequent CPs that arrive to the destination node are used to construct corresponding APs. These APs are marked as redundant. The manner in which the redundant APs are treated depends on the particular overlay policies. If the overlay policies favor backup paths or multipath routing, the redundant APs are treated just as regular APs and forwarded to the source node on the reverse feasible path. Otherwise, redundant APs are dropped.

Each node receiving an AP checks whether the triple (SrcId, DstId, FlowID) is matching an entry in the list of active CPs or not. If a matching entry is found, the node either creates a FR or adds the flow to an existing flow bundle corresponding to a FR. Further, the CP entry is removed from the active CP list and the AP is forwarded to the next node on the reverse feasible path. If no matching entry is found, the AP is dropped silently. As mentioned before, the manner in which redundant APs are treated depends upon the overlay policy. In the reported experiments the redundant APs are dropped.

The first AP to arrive at the source node signals that a feasible path has been set up and the application can begin sending DPs. A feasible path can be torn down by a CP with the delete (D) flag set.

Chen and Nahrstedt [91, 161] provide worst-case complexity results for the time and communication overhead required to establish a QoS path. They assume that each link requires unit time to transport a packet. The path discovery procedure uses one round-trip to establish a path. For a path length l, the time complexity is $\mathrm{O}(2l)$. In the case of RDP, the path length is bounded by the TTL value. Hence, RDP's time complexity for one QoS request is $\mathrm{O}(2\,\mathrm{TTL})$.

In the case of computation overhead it should be noted that each CP is sent at most once over a link. Thus, the number of CPs corresponding to a QoS request is bounded by the number E of links in the network. Assuming that redundant APs are dropped, one AP travels each link on the feasible path. Chen and Nahrstedt [91, 161] count this as a different packet each time it arrives at

a new link. Consequently, the communication complexity for one QoS request is on the order $\mathrm{O}(E + \mathrm{TTL})$.

5.3.3 Implementation

To evaluate the performance of RDP the open-source simulation environment OMNeT++ is used [178]. OMNeT++ is an object-oriented, modular discrete-event simulation environment with an embeddable simulation kernel.

An OMNeT++ simulation is built out of hierarchically nested modules, which is ideal for an object-oriented approach. Modules communicate with each other by means of messages and these messages may contain data of arbitrary length. Messages are transported through gates and over channels. A node maintains an arbitrary amount of gates and different gates are connected with channels. The topology of a network, in terms of gates, channels and modules, is defined in the Network Description (NED) language [178].

The simulator includes two different modules: the `ORP` module and the `DATACENTER` module. The `ORP` module implements the RDP protocol and the `DATACENTER` module collects the simulation statistics. These statistics can be easily written to files with help of dedicated classes provided by the OMNeT++ framework.

OMNeT++ allows arbitrary parameters to be defined in an external initialization file, which can be loaded in the simulation at any time. This allows the user to control the behaviour of the simulation without having to recompile the source code. The parameters available in the initialisation file are:

- TTL value of the packets,
- destination node to which a route is opened,
- delay and bandwidth QoS constraints used for route requests,
- session arrival rate and session duration.

The destination node parameter can be a node identifier or a discrete probability distribution used to randomly select a node.

5.3.4 Simulator Validation

The simulation model for RDP was primarily developed for assessing the performance of the protocol under different workloads. Ideally, a simulator should be validated against an existing system or against a mathematical model [132]. Since RDP is a prototype for a new system, the first form of validation cannot be applied. Also, due to the complexity of the protocol it is difficult to construct an accurate mathematical model. This is however left for future work. In order to circumvent these hurdles and still provide some validation, a number of scenarios with known correct outcome, called sanity tests, were considered:

i) one feasible path only,

ii) no feasible paths,

iii) multiple feasible paths.

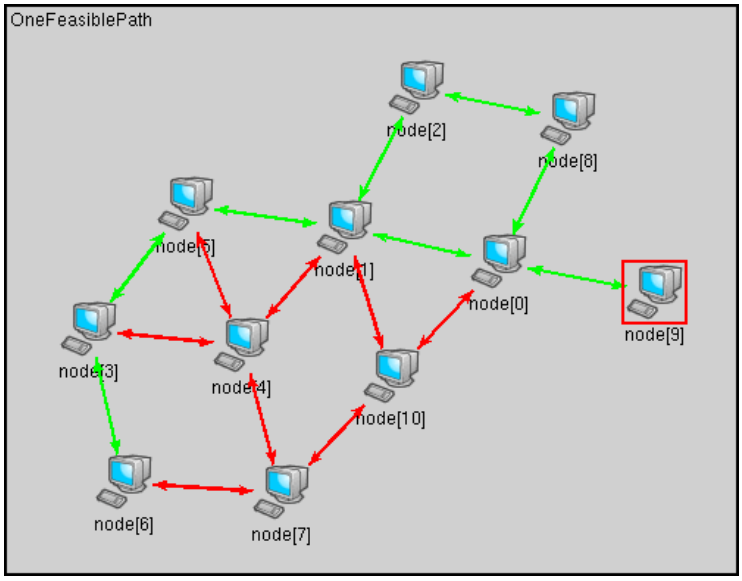


Figure 5.3: Topology for validation of RDP simulator.

Link-state class	Capacity (Kbps)	Delay (ms)
1	64	1000
2	500	5
3	500	35

Table 5.1: Topology parameters for validation of RDP simulator.

The sanity tests are executed in a network with 11 nodes, shown in Figure 5.3. The network with the corresponding scenarios, is small enough to allow inspection of each event occurring in the simulator. In each scenario, the goal is to find paths from node 0 to node 6, such that the maximum end-to-end delay does not exceed 100 ms and the available bandwidth is at least 64 Kbps. The path computation is initiated from node 0. For each link, the link state is deterministically set to one of the link-state classes shown in Table 5.1. The links are error-free. It should be noted that links with link-state class 1 are always infeasible due to excessive delay. For class 2 and 3, the path feasibility depends on its length.

In the first scenario, links have either link-state class 1 or link state class 2. In Figure 5.3, links with link state class 1 are colored red and links with link-state class 2 are colored green. If correctly implemented, RDP finds a single feasible path $0 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 6$. CPs traveling over the path $0 \rightarrow 8 \rightarrow 2 \rightarrow 1$ are dropped when reaching node 1. This happens because node 1 has already received a CP from node 0, since a CP requires only 5 ms over the path $0 \rightarrow 1$ compared to 15 ms over the path $0 \rightarrow 8 \rightarrow 2 \rightarrow 1$.

In the second scenario, the network shown in Figure 5.3 was modified by changing the state of the links $(6, 3)$ and $(3, 6)$ to link-state class 1. In this case, a valid RDP implementation finds no feasible path between node 0 and node 6.

The network shown in Figure 5.3 was changed once more for the third scenario. In this scenario, all links in the network have link-state class 2, with the exception of the links $(1, 4)$, $(4, 1)$, $(4, 7)$ and $(7, 4)$, which have link-state class 3. These changes prevent node 4 from being used as intermediate node on the path. A valid RDP implementation finds two feasible paths: $0 \rightarrow 1 \rightarrow 5 \rightarrow 3 \rightarrow 6$ and $0 \rightarrow 10 \rightarrow 7 \rightarrow 6$.

The sanity tests above are based on modifying the delay of various links. Similar tests were performed with regards to the link bandwidth and error rate. Additional validation tests were performed on the output of the simulator, *e. g.*, controlling that measured bitrates are proportional to the number of exchanged messages.

5.3.5 Experiment Setup

The purpose of the experiments is to evaluate the performance of RDP for different workloads. The experiments are divided into two sets. In the first set of experiments, the network size is increased from 10 to 1000 nodes and various performance metrics are computed. In the second set of experiments, the same metrics are observed when the network utilization increases in a network of 100 nodes. The network size in the two experiments was limited by the amount of memory required. In particular, high network utilization results in high memory usage in the simulator.

In the experiments the focus was entirely on bandwidth reservations. The term *QoS session* is used to denote a request for a directed path with a constraint on minimum available bandwidth. Each session has an associated session duration, which specifies the life length of the path. If a path is successfully established, the amount of bandwidth specified by the path constraint is reserved for the entire session duration. The links in these experiments are error-free and no churn occurs.

During simulation data has been collected for the following metrics:

call blocking ratio: ratio between the number of infeasible QoS sessions and the total number of QoS sessions arrived at the network,

low-TTL blocking ratio: ratio between infeasible sessions due to low TTL value and the total number of infeasible sessions,

bandwidth utilization: average number of bytes per second due to RDP control information,

bandwidth overhead: ratio between the average number of RDP bytes per second and network capacity (*i. e.*, the aggregated volume of every link in the network),

path stretch: average value for the ratio between feasible path length to shortest path length.

The call blocking ratio metric is a good indicator of the overall success or failure of the protocol in establishing QoS paths. When the call blocking ratio is high, it is important to diagnose the root cause of the failures. In this context, the low-TTL blocking ratio metric can help in deciding if failures occur because bandwidth requests cannot be satisfied or because CPs are hindered in finding a feasible due to the TTL value used.

The bandwidth metrics provide information about the cost of using RDP expressed in terms of protocol overhead. The utilization metric can be used to compare the required bandwidth per second with that used by other protocols. On the other hand, the overhead metric reflects how much of the entire network capacity is dedicated to RDP.

Finally, path stretch is also a way of estimating the cost of using RDP. When the path stretch is higher than 1, a penalty is payed in terms of bandwidth and delay. The bandwidth penalty has to do with reserved bandwidth on additional links compared to the case of the shortest-path. Since more links are used, the one-way delay of packets sent over feasible paths increases as well.

In the first set of experiments, the sessions arrive at the network according to a Poisson distribution with parameter $\lambda = 1$. The session duration is drawn from a generalized Pareto distribution with mean 180 seconds. This value reflects the average duration of a voice conversation [29, 179, 180]. The Pareto distribution is used to model the effect of long-tails and heavy-tails. The existence of long-tails in the distribution of session duration in wide-area networks is supported by empirical evidence from several studies [179, 181, 182].

The bandwidth requested for a QoS session is uniformly distributed over three different ranges: 16–64 Kbps, 128–512 Kbps and 1–2 Mbps. These ranges roughly represent low-to-high quality audio, streaming music or low quality video, and high quality video, respectively. The network topologies are created with the same settings as described in Section 3.4. The bandwidth in the topologies generated with BRITE is uniformly distributed between 10 Kbps and 10 Mbps. The upper bound of the distribution represents an average value for the maximum connection capacity offered to residential users by ISPs in Swe-

den[5]. Links with less than 10 Kbps are not considered useful. The link delay constraint is set to 10000 seconds. This value is several orders of magnitude higher than the link delay, thus the delay constraint is always satisfied.

Preliminary results [183] indicate that RDP's performance is strongly influenced by the TTL value in use. Consequently, the performance of the protocol is compared for the following TTL values: 4, 8 and 250. For brevity, these values are denoted by TTL=4, TTL=8, and TTL=250, respectively.

In the second set of experiments, the metrics are observed in terms of increasing network utilization. The network utilization, ρ, is defined as [29, 181]

$$\rho = \frac{\lambda TQH}{\sum_{e \in \mathcal{E}} b_e} \tag{5.2}$$

where T is the average session duration, Q is the average amount of QoS (bandwidth) requested, H is the average path length across all node pairs, and b_e is the available bandwidth on link e.

The session duration is selected from a generalized Pareto distribution with mean 180 seconds, as in the first set of the experiments. Additionally, these experiments include Pareto distributed sessions with a mean of 600 seconds. This value was selected to explore the effect of longer sessions on the performance of the protocol.

The amount of bandwidth requested by each QoS is defined exactly as in the case of the first set of experiments.

The network utilization variable, ρ, was allowed to assume values from the set $\{0.1, 0.25, 0.50, 0.75, 1.00\}$. This is accomplished by correspondingly adjusting the arrival rate variable, λ, in Equation 5.2, while the remaining variable were kept fixed.

For the second set of experiments a single BRITE topology is used, which consists of 100 nodes. The topology is generated with the configuration parameters described for the first set of experiments.

The TTL value is set to 8, based on results from the first set of experiments. Redundant APs are dropped.

Both sets of experiments are executed 30 times for each parameter combi-

[5]This was estimating by comparing the current offers from ISPs in Sweden.

Parameter	Assigned value
Session arrival	Poisson with parameter $\lambda = 1$
Session duration	Generalized Pareto with mean 180 s
Requested bandwidth ranges	Uniform, 16–64 Kbps, 128–512 Kbps, 1–2 Mbps
Link bandwidth	Uniform, 10–10000 Kbps
Link delay	10000 s
TTL	4, 8, 250 hops

Table 5.2: Parameters for the first set of experiments.

Parameter	Assigned value
Network utilization, ρ	$0.1, 0.25, 0.50, 0.75, 1.00$
Session duration	Generalized Pareto with mean 180 s and 600 s
Requested bandwidth ranges	Uniform, 16–64 Kbps, 128–512 Kbps and 1–2 Mbps
Link bandwidth	Uniform, 10–10000 Kbps
Link delay	10000 s
TTL	8 hops

Table 5.3: Parameters for the second set of experiments.

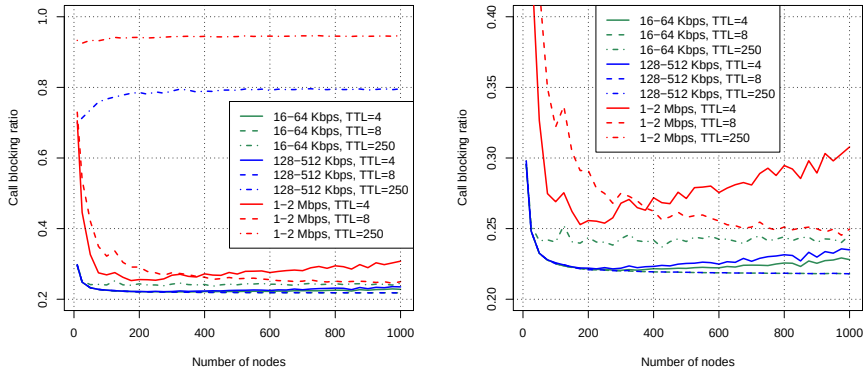
nation. This allows for the construction of average values with corresponding 95 %-confidence intervals. Each run lasts for a duration of 3600 seconds simulated time preceded by a warmup period of 1000 seconds.

The parameters for each set of experiments are summarized in Tables 5.2 and Table 5.3, respectively.

5.3.6 Performance Results

Figure 5.4 shows the changes in the call blocking ratio as a function of the network size. The functions plotted in the graph are grouped according to bandwidth range and TTL value. The bandwidth ranges corresponding to 16–64 Kbps, 128–512 Kbps and 1–2 Mbps range are colored green, blue and red, respectively. For each bandwidth range, when the TTL=4, the curve is drawn with a solid line, with a dashed line in case of TTL=8, and with alternating dots and dashes when TTL=250.

It is observed in Figure 5.4(a) that 128–512 Kbps sessions and 1–2 Mbps



(a) Call blocking ratio.

(b) Call blocking ratio with zoom.

Figure 5.4: Call blocking ratio.

sessions, both with TTL equal to 250, experience call blocking ratio in the range of 0.8–0.9. This effectively dwarfs the remaining curves. The zoomed view in Figure 5.4(b) makes it easier to distinguish these curves.

A session is blocked for one of the following three reasons:

no-path blocking: there is no path connecting the source node with the destination node,

QoS blocking: a connecting path exists, but it does not satisfy the QoS constraints,

low-TTL blocking: the TTL value is too low to allow any CP to reach the destination.

The BRITE topologies used for experiments are strongly-connected, which means that at least one connecting path exists for each node pair. Hence, sessions are blocked either due to QoS blocking or due to low-TTL blocking.

The plots shown in Figure 5.5 indicate that the cause for the high blocking ratio observed in Figure 5.4(a) is QoS blocking. The curves in Figure 5.5

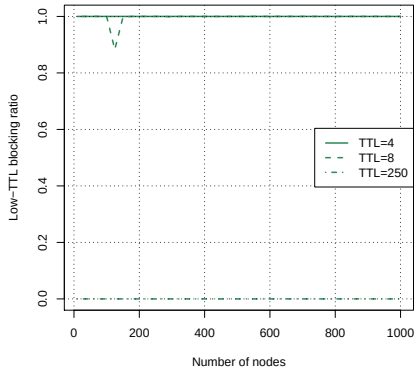
represent the ratio of low-TTL blocked sessions to the total number of blocked sessions. This ratio is zero when the TTL value is equal to 250, implying that QoS blocking is solely responsible for call blocking. Additionally, this shows that in every topology every node can be reached in 250 hops or less.

Returning to Figure 5.4 it is observed that the curves corresponding to 1–2 Mbps sessions with TTL=4 and TTL=8 experience high call blocking ratio for very small network sizes. The call blocking ratio decreases abruptly when the network size increases. The explanation is that, in small networks of 10–50 nodes, there is simply not enough bandwidth to accommodate the QoS sessions. This assertion is corroborated by Figure 5.5(c). For larger networks, the low-TTL blocking ratio increases. This happens very fast for sessions with TTL=4, because for large networks the number of unreachable destinations increases. This also explains the slight increasing trend in call blocking ratio for networks larger than 400 nodes.

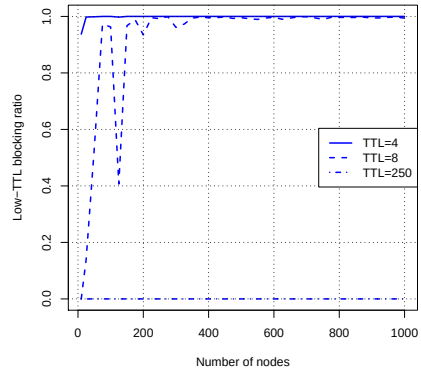
Figure 5.4(b) shows that the call blocking ratio for 16–64 Kbps sessions and 128-512 Kbps sessions changes less dramatically. Since these sessions require a much smaller amount of bandwidth, there is a higher probability of finding feasible paths of short length.

Finally, it can be observed that call blocking ratio does not decrease below 0.23. The reason for this behavior has to do with the link bandwidth distribution. Recall that link bandwidth is uniformly distributed with a lower bound on 10 Kbps. Yet, QoS sessions require 16 Kbps or more. Consequently, several links cannot accommodate any QoS sessions at all. Additional contributing factors are the TTL value and the session duration. The TTL value prevents the protocol from finding potential routes extending beyond the TTL horizon. The session duration keeps link bandwidth reserved. In several cases, the remaining amount of free bandwidth is too small to allow additional sessions over the same link.

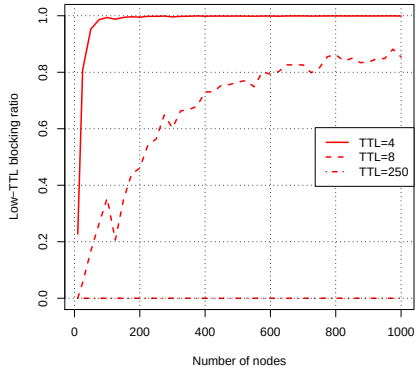
As mentioned before, 95 % confidence intervals have been computed for all metrics of interest. The intervals are very narrow for all metrics, implying that the mean value estimates are quite accurate. As an example, the 95 % confidence intervals for 128–512 Kbps session with TTL=8 and for 1-2 Mbps sessions with TTL=250 are shown in Figure 5.6. The confidence intervals are represented by vertical black lines, with the length corresponding to the width of the confidence



(a) 16–64 Kbps.

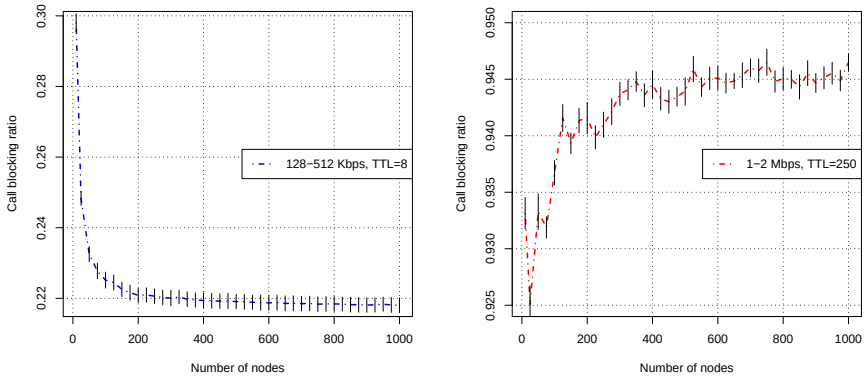


(b) 128–512 Kbps.



(c) 1–2 Mbps.

Figure 5.5: Low-TTL blocking ratio.



(a) 95 % confidence interval.

(b) 95 % confidence interval.

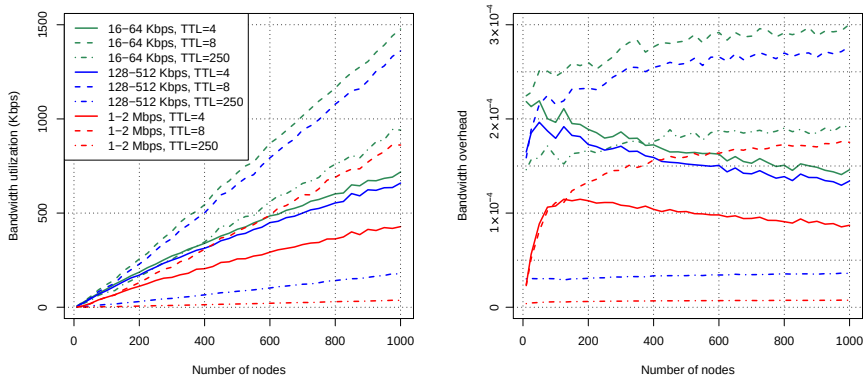
Figure 5.6: Call blocking ratio with confidence intervals.

interval. It can be observed that all confidence intervals are within the range of 0.005. The intervals are so small, that in fact it was required to zoom in the plot to make them visible. If they would be drawn on top of the curves in Figure 5.4 they would not show at all. This is a recurring observation for all metrics. Hence, additional confidence interval plots are not shown.

As stated in Section 5.3, RDP's foremost disadvantage is its cost in terms of bandwidth. Figure 5.7(a) shows the RDP bandwidth utilization in Kbps, as a form of quantitative estimate of the cost. However, Figure 5.7(b) showing the bandwidth utilization normalized by network capacity, is a better indication of how much capacity is wasted by the path discovery process[6].

The upper bound on RDP's bandwidth utilization is decided by two factors: the number of feasible links in the network and the TTL value used. In Figure 5.7 it can be observed that 128–512 Kbps sessions and 1–2 Mbps sessions with TTL=250 consume the least amount of bandwidth. This happens because most sessions of this type experience QoS blocking as it can be observed in Figure 5.4(a) and in Figure 5.5.

[6]The legend shown in Figure 5.7(a) applies to Figure 5.7(b) as well.



(a) Utilization. (b) Overhead.

Figure 5.7: RDP bandwidth.

RDP's bandwidth overhead shows a slow growing trend for networks with 200 nodes or more, with the exception of sessions with TTL=4. These sessions experience an increasing blocking probability with growing network size because more and more destination nodes are beyond the 4-hops TTL horizon. The largest bandwidth overhead occurs in the case of 16–64 Kbps sessions with TTL=8 and shows that roughly 0.03 % of the network capacity is used to provide QoS routing.

Figure 5.8 shows the path stretch metric. In general, it is desired to limit the path stretch because it can increase the traffic delay and the call blocking ratio. When the path stretch is greater than one, the corresponding paths are on average longer than the shortest paths. This means that traffic along these paths tends to incur a higher delay than if transported along shortest paths. When a feasible path is found, RDP reserves bandwidth on each link along the path. When the paths are long the capacity is reduced on many links for the duration of the QoS session. This increases the call blocking ratio for subsequent sessions. As expected, the path stretch depends on the TTL value used. It can be observed that the QoS paths are at most 3.3 times longer then the corresponding shortest paths.

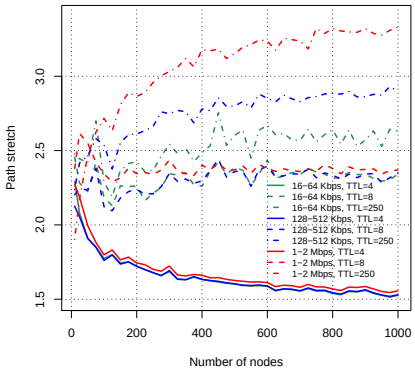
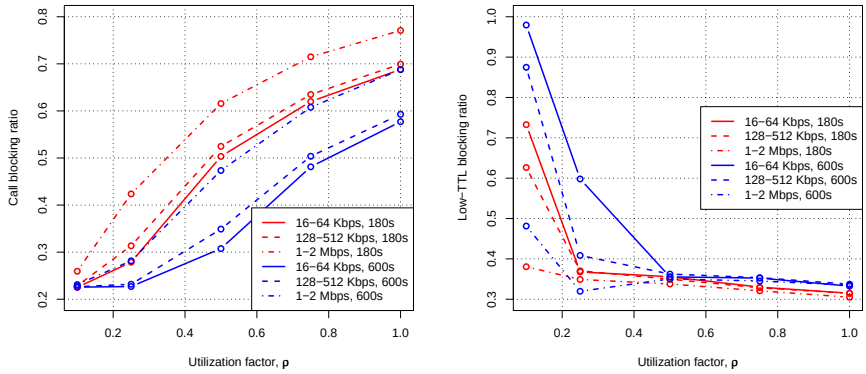


Figure 5.8: Path stretch.

This concludes the first set of experiments. Based on the results it was decided to use a TTL value of 8 for the second set of experiments. The main reasons for this decision is the combination of low call blocking ratio and medium path stretch.

For the second set of experiments, the same metrics as in the first set are observed while the network utilization increases. The blue color is used for sessions with mean duration of 180 seconds, while red color denotes sessions with mean duration of 600 seconds. Plots for 16-64 Kbps sessions are drawn with solid lines, those for 128–512 Kbps are drawn with dashed lines, and 1-2 Mbps session plots use alternating dots and dashes. Each hollow circle indicates the simulated utilization factor pertaining to the value on the y-axis.

The call blocking ratio and the low-TTL blocking ratio are shown side by side in Figure 5.9. The first thing to notice in Figure 5.9(a) is that 180 seconds sessions consistently experience higher call blocking ratio than 600 seconds sessions. The explanation is found in Equation 5.2. This equation is used to compute the utilization factor, ρ, by adjusting the arrival rate, λ, while the other parameters are kept fixed. For a given ρ value, the arrival rate must be higher for short sessions than for long sessions. A higher arrival rate implies that more feasible paths must be found. This leads to higher bandwidth overhead



(a) Call blocking ratio.

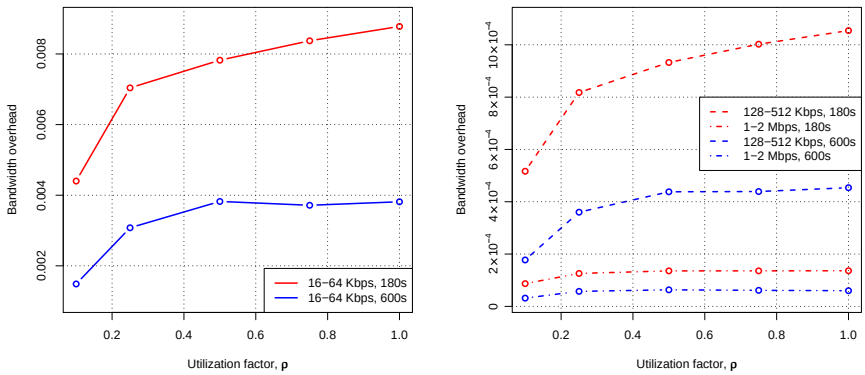
(b) Low-TTL blocking ratio.

Figure 5.9: Call blocking.

since more CPs are in the network, as it can be observed in Figure 5.10. Since less network capacity is available in this case than in the case of low arrival rate, the call blocking ratio is higher. Further support for this assertion is found in Figure 5.9(b), where it can be observed that, when the utilization factor exceeds 0.25, most call blocking is due to QoS blocking.

Two different graphs are used in Figure 5.10 to show the RDP bandwidth overhead. This is done because in the case of a single graph, the high bandwidth overhead of 16-64 Kbps sessions would make it hard to distinguish the bandwidth overhead of the remaining sessions. Indeed, for 16-64 Kbps sessions the bandwidth overhead is 40–80 times higher than that of 1–2 Mbps sessions.

Figure 5.11 shows the path stretch metric as a function of the utilization factor. The figure shows that when the utilization factor is below 0.75 the path stretch is higher for 180 second sessions. When the path stretch increases beyond 0.75 the situation is reversed. Unfortunately, there are currently no results available for intermediate ρ values between 0.75 and 1. In their absence, the hypothesis is that, at high network utilization, the session duration has more influence over the path stretch. Long sessions keep the links occupied forcing RDP to search longer feasible paths.



(a) 16–64 Kbps. (b) 128–512 Kbps and 1–2 Mbps.

Figure 5.10: RDP bandwidth overhead.

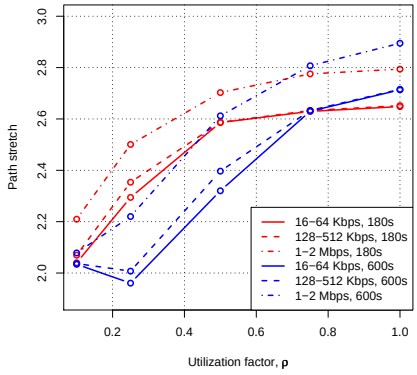


Figure 5.11: Path stretch.

5.4 Route Maintenance Protocol

The paths established with RDP must satisfy the QoS constraints (*i. e.*, the flow demand) in the face of resource fluctuations, as explained in Section 5.2. Applications are expected to cope with resource starvation during short durations of time. However, prolonged resource outage can have a serious adverse impact on the QoE. When resource starvation is detected it is necessary to re-route the flows such that the QoS constraints can still be satisfied. RMP combines path restoration and flow allocation to achieve this goal. In this thesis the focus is on bandwidth constraints, but the protocol as such has support for different types of QoS constraints.

The factors that influence the design of RDP are also responsible for the design of RMP. They were introduced in Section 5.3:

i) overlay networks with a large number of nodes,

ii) unreliable end-nodes as infrastructure,

iii) topology dynamics.

The use of traditional routing algorithms based on link state or distance vectors in an environment with a large number of nodes is problematic at best [159]. In particular, the reliable flooding required by link-state algorithms presents serious scalability issues. If the link-state updates are triggered by changes in the link-state information in an environment with unreliable network infrastructure and aggressive topology dynamics, the scalability problem becomes even more serious. On the other hand, if the updates are periodic, nodes are likely to act upon stale information. In the same type of environment, distance-vectors algorithms suffer from problems related to routing loops and explosion in the size of the routing table when multiple QoS constraints are maintained [159, 184].

To alleviate the scalability problem it is necessary to drastically reduce the communication requirements. This entails to reducing the number of source-destination pairs for which link state or distance vectors must be maintained. Simultaneously, care must be taken such that QoS information is maintained fresh.

If one considers the typical applications of unicast QoS routing (*e. g.*, VoIP, videoconferencing) it becomes clear that maintaining information about all

source-destination pairs in each node is unnecessary. For example, most people have voice conversations with a limited number of other people, who add up to just a tiny fraction of the entire population. In the VoIP scenario, this implies that a source needs to maintain only topology information that describes potential paths to its preferred destination nodes.

The solution proposed here assumes that source nodes establish QoS paths using RDP. Neighbouring nodes share link-state information about established paths among themselves. Contrary to the approach used by traditional link-state protocols where complete topology information is flooded over the whole network, the solution presented here exchanges specific topology information by using selective diffusion. When an intermediate node on the path detects that the QoS constraints of its outgoing link can no longer be maintained, it enters *restoration mode*. A node in restoration mode computes a number of backup paths based on the available link-state information. Traffic flows over the broken path are reallocated to the backup paths using the techniques explained in Chapter 3. Additionally, the intermediate node sends a message to the source of each affected flow, informing it that backup paths are maintained until the corresponding source node finds a new QoS path to the destination. In the case when a link fails at the source node, the same actions are taken as in the case of an intermediate node. Destination nodes are not required to do anything other than to exchange link-state information.

5.4.1 Protocol Description

RMP relies on two main components: an algorithm for distributing link-state information and an optimization algorithm for flow allocation.

Link-state information is distributed using the *link vector* algorithm proposed by Behrens and Garcia-Luna-Aceves [159]. The difference between a link-state algorithm and a link vector algorithm is that the link-state algorithm is require to broadcast complete topology information. When a link vector algorithm is used, a node uses selective diffusion to disseminate link-state information pertaining only to its preferred paths. This reduces the communication overhead associated with traditional link-state algorithms.

In the case of RMP, the preferred paths are setup using RDP. The QoS information provided by RDP ensures that a node has link-state information

about each link in each of its preferred paths. The set of links belonging to a node's preferred paths is called the *source graph* of that node. Nodes exchange source graph information with their neighbours. Additionally, nodes have link-state information about their own outgoing links. The topology information known to a node consists of its own links, its own source graph and the source graphs reported by its neighbours [159].

Nodes report incremental source graph information to their neighbours. Obviously, when a node joins the overlay it receives complete source graphs from its neighbours. Beyond that, information is transmitted only if the link-state changes, *i. e.*, triggered updates.

There are two type of updates: *add* update and *delete* update. In RMP parlance they are called LS_ADD and LS_DEL update, respectively. The prefix LS is an abbreviation for link-state. A node sends an LS_ADD update when a new link is added to its source graph or when updated link-state information is available for a link in its source graph. A LS_DEL update is sent when a node discards a link from its source graph.

A RMP message begin with the generic ORP header shown in Figure 5.1. The type field in the header uses a value of 4. The generic header can be followed by multiple LS_ADD and LS_DEL updates. The format of a sequence of link-state updates is shown in Figure 5.12.

The bandwidth, delay and loss rate fields have the same meaning as in the case of RDP. A value of one in the type field indicates a LS_ADD update and a value of two indicates a LS_DEL update, respectively. The link source UUID denotes the head node of a link. Similarly, the link destination UUID denotes the tail node of a link.

Nodes receiving link-state updates must be guarded against the possibility of processing stale information. Therefore, each node maintains a sequence number counter that is updated by changes occurring to its topology table. Only the head node of a link is allowed to change the sequence number and the link-state information (*i. e.*, the QoS metrics) associated with the link in question. When a node receives a link-state update for a link already available in its topology table, it compares the sequence number of the update with the sequence number stored in the topology table. If the sequence number of the update is higher, the contents of the topology table are updated with the information from the

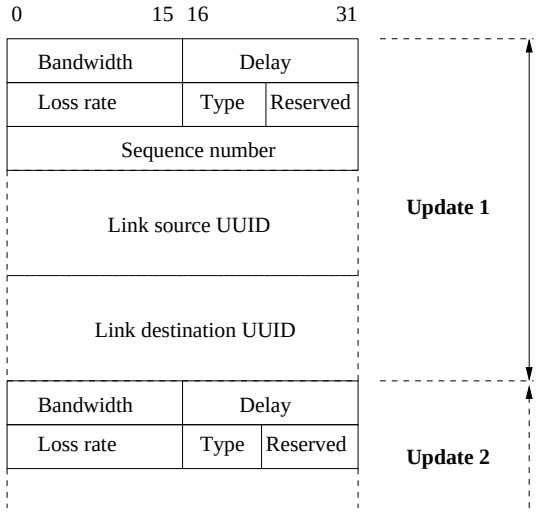


Figure 5.12: Sequence of link-state updates.

update. Otherwise, the update is discarded.

For each link stored in the topology table there is also information maintained about the set of neighbours that sent LS_ADD updates concerning that particular link. The node that maintains the topology table is part of the neighbour set if the link is in its source graph. When a LS_DEL update is received, the sender is removed from the set of neighbours. If the set of neighbours becomes empty, the corresponding link is deleted from the topology table.

A link is deleted automatically from the topology table if a received LS_DEL message has a sequence number higher than the one stored in the table. This is an indication that the link head node is unable to reach the link tail node.

When a link is removed from the topology table, it is stored temporary in a list together with an age variable. The age variable is a conservative estimate of the time it takes for an update to propagate throughout the network and indicates how long the deleted link is kept in the list. This protects against stale LS_ADD updates for the deleted link and against wrapped sequence numbers [49, 159].

Upon receiving a stale update, the receiver uses information from its topology table to prepare an update for sender. If the stale information describes a link present in the receiver's source graph, the prepared update is of type LS_ADD. Otherwise, a LS_DEL update is prepared to inform the sender that the link in question is not in the receiver's source graph.

A node enters restoration mode when a path is broken, *i. e.*, when one or several links are deleted from the source graph or when their updated state information makes it impossible to satisfy the path QoS constraints. In restoration mode the following actions are taken:

i) an AP is sent to the source node of each affected flow, with the status code set to indicate path error,

ii) Yen's KSP algorithm is executed to find the K backup paths to the destinations affected by the topology updates,

iii) the corresponding flow demands and the backup paths are used to construct a PAP-MLPF optimization problem as described in Section 3.2.2,

iv) the simplex method is used to solve the PAP-MLPF problem as described in Section 3.1 and Section 3.3,

v) if the simplex method is successful, the links on the new paths are added to the source graph; otherwise the affected flows are dropped (*i. e.*, packets belonging to them are not forwarded further).

In restoration mode, RMP exploits path diversity in order to keep traffic flowing when the main route is failing. A strongly-connected topology (*e. g.*, a graph with high outdegree) is a key element for creating high path diversity. Path diversity is also dependent upon the number of constraints. In particular, finding several backup paths can be more difficult if several QoS constraints must be satisfied simultaneously. The focus here is on bandwidth management, but it is theoretically possible to replace Yen's algorithm with the Self-Adaptive Multiple Constraints Routing Algorithm (SAMCRA), which allows for path selection with multiple constraints [55, 185].

As stated above, if the simplex method fails, then path failure occurs and the node drops the concerned flows. An alternative approach is to drop one flow

demand at a time and rerun the simplex method. This can improve the success ratio for the flow allocation, albeit at a higher computational cost. Only the first approach is used in the implementation reported here.

The following actions [159] are taken by a node if its topology table is updated in response to a received RMP message:

i) a new source graph is constructed using the updated information,

ii) the new source graph is compared to the old source graph and based on their differences a set of LS_ADD and LS_DEL updates is created,

iii) the set of updates from the previous step is used to construct a RMP messages that is sent to all neighbours,

iv) aged links are removed from the deleted links list,

v) the sequence number counter is incremented.

The time complexity of the link vector algorithm after a single link change is $\mathrm{O}(n)$, where n is the number of nodes affected by the change. The upper bound for n is given by the length of the longest path in the network. The communication complexity $\mathrm{O}(E)$ is asymptotic in the number of links in the network [159].

5.4.2 Implementation and Validation

The OMNeT++ simulator [178] environment is used to test the performance of RMP under various levels of churn. The simulator consists of two modules: `ORPNODE` and `WORLD`.

The `ORPNODE` module is the actual RMP implementation. It should be noted that the module is as close to a real implementation as it can be done in OMNeT++. This statement applies to the datatypes used, the function calls, the message format and the optimization algorithms. In fact, the simulator is linked with the `liboptim` software library presented in Chapter 3. If the message-passing layer in OMNeT++ is replaced with a TCP/IP-based layer, then minimal changes to `ORPNODE` are required in order to make it work over a real network. Suggestions for enabling network communication in OMNeT++ can be found in [186].

Given that RMP depends on RDP for obtaining preferred paths, it should ideally use the RDP implementation described in Section 5.3.3. However, empirical evidence from the experiments with RDP suggested that this approach is likely to result in extremely long simulation times. Instead, the RMP simulator uses a bootstrap phase to setup preferred routes.

The bootstrap phase is implemented in the `WORLD` module. This phase is initiated with a set of randomly generated flow demands and with a BRITE topology object. For each flow demand, a path is computed with Yen's KSP algorithm, where $K = 5$. The reason for using Yen's KSP is to induce path stretch as observed in the performance results for RDP. The bandwidth is adjusted for each link on every 5SP[7] in order to satisfy the flow demand. This concludes the bootstrap phase.

It should also be noted that because RDP is not used, when a node enters restoration mode no AP message is sent to the source node of each broken flow.

In addition to the bootstrap phase, the `WORLD` module is responsible for generating link churn. Each connected pair of nodes shares a pair of links, with each link going in the opposite direction of the other. Every link pair in the network has an associated session duration variable, which is used to schedule a session termination event. The value for the session duration is drawn from a user-configurable probability distribution. When the session termination event occurs, the node pair is disconnected. Nodes are assigned a session arrival variable, also drawn from a user-selectable probability distribution. The session arrival variable is used to schedule a session arrival event. Furthermore, each node has a fixed degree, which is an upper bound on the number of nodes it can be connected with. A node can be in three states: full, active or idle. A full node has all its links in use and ignores any session arrivals. Active nodes have one or more unused links, while idle nodes are completely disconnected from the network. When a session arrival event occurs, the node in question is selected as destination node. If idle nodes are available, the simulator randomly selects one of them as source node. Otherwise, an active node is selected. If only full nodes are available, the session arrival is ignored. When the simulator is able to find both a source and a destination node, two links are created: one from the source node to the destination node and another one in the opposite direction.

[7]Using the notation from Chapter 3, a 5SP denotes a path computed with Yen's KSP algorithm for $K = 5$.

The link pair is assigned a session duration and a corresponding session duration event is scheduled.

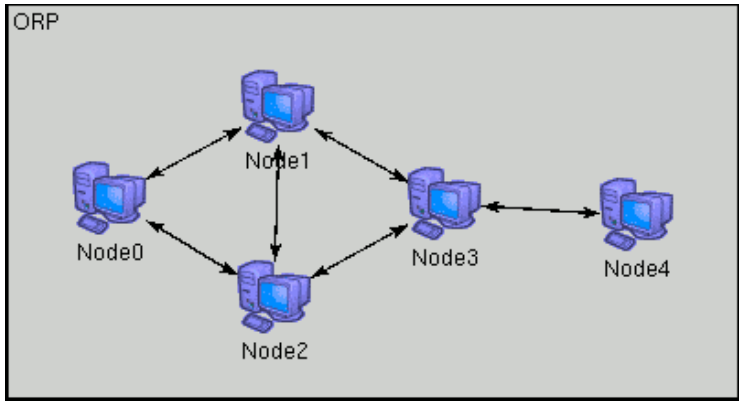


Figure 5.13: Topology for RMP simulator validation.

Validation methods similar to those used for RDP are used in the case of RMP as well. The topology shown in Figure 5.13 has been extensively used since it is simple enough both for calculating the outcome for the validation tests and also for inspecting each event that occurs in the simulator. The tests performed cover the following aspects of the protocol:

- Yen's KSP algorithm,
- flow allocation,
- link vector algorithm.

The `liboptim` implementation of Yen's KSP algorithm was executed to compute up to the first seven shortest paths (7SPs), from each node to the remaining nodes shown in Figure 5.13. For example, when the algorithm is searching paths from from node 0 to node 4, only the following four paths must be found if the algorithm works correctly:

i) $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$,

ii) $0 \rightarrow 2 \rightarrow 3 \rightarrow 4$,

iii) $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$,

iv) $0 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 4$.

In one of the flow allocation tests the bandwidth is forced to zero on all links except for the links on the path $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$, where each link is assigned enough bandwidth to satisfy the demand from node 0 to node 4. A correct implementation allocates the entire demand to the path $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$, when no other demands exist. In another test, bandwidth is allocated to the paths $0 \rightarrow 1 \rightarrow 3 \rightarrow 4$ and $0 \rightarrow 2 \rightarrow 3 \rightarrow 4$, such that the demand from node 0 to node 4 can be evenly distributed. Error conditions are tested also, for example, by setting the bandwidth on the link $(3, 4)$ to a value below that of the demand from node 0 to node 4. In this case no path from node 0 to node 4 can be found.

The link vector algorithm is tested by removing specific links and tracing the updates generated. Also, an important test is to turn off the churn completely after a number a link changes occurs. A valid implementation converges to a stable state in these conditions.

Although each test described here validates only small aspects of the protocol, together they provide an indication that the protocol as a whole works as intended.

5.4.3 Experiment Setup

The purpose of the experiments is to evaluate RMP's performance for different levels of churn. In the experiments we focus entirely on bandwidth reservations. A network topology with 100 nodes and a maximum of 780 links is used for all experiments. The links in these experiments are error-free.

The simulation time is divided into non-overlapping intervals that are one minute long each. Within each interval, the simulator keeps track of the flows that are affected by link churn. A path affected by churn is referred to as a *broken path*. If the flow over a broken path can be allocated to a set of backup paths, the path status is set to restored. Otherwise, the path status is set to failure. A link that is down during consecutive minute intervals contributes

only during the first interval to the number of path failures or to the number of restored paths.

We denote by p_t the total number of preferred paths, by p_r the number of restored paths, and finally by p_f the number of path failures. The relation $0 \leq p_r + p_f \leq p_t$ always holds. At the beginning of each interval the variables p_r and p_d are reset to zero.

During simulation the following averages are computed:

path failure ratio: ratio between the number of path failures and the total number of preferred paths in the network, p_f/p_t,

restored paths ratio: ratio between the number of restored paths and the number of broken paths, $p_r/(p_r + p_f)$ for $p_r + p_f > 0$[8],

bandwidth utilization: average number of bytes per second due to RMP control information,

bandwidth overhead: ratio between the average number of RDP bytes per second and network capacity (*i. e.*, the aggregated volume of every link in the network),

There are 50 random flow demands in each simulation, *i. e.*, $p_t = 50$. This value of p_t provides an acceptable trade-off between link utilization and the time required to run the simulations. The flow demand bandwidth is uniformly distributed over three different ranges: 16–164 Kbps, 128–512 Kbps and 1–2 Mbps, as in the case of RDP. The source and destination node are selected randomly as described in Section 3.3.

In these experiments, link bandwidth is interpreted as residual capacity after bandwidth is reserved on preferred paths. The residual capacity determines the amount of path diversity within the network. Here, the residual capacity is exponentially distributed with mean value equal to the maximum bandwidth demand multiplied by an integer scaling factor. For example, for the bandwidth range 1–2 Mbps and a scaling factor of 2, the link bandwidth is exponentially distributed with mean value 4 Mbps. This means that, on the average, each link in the network can accommodate two flows or more.

[8]Intervals where $p_r + p_f = 0$ are not used in computing the average.

The following scaling factors are used: 1, 2, 3, 4, and 5. The use of exponential distribution with mean value based on the maximum bandwidth demand results in a good mix of links with very little bandwidth as well as links with lots of residual capacity. Using the upper bound of the bandwidth range is a matter of preference. In fact, any value within the bandwidth range can be selected and the mean link bandwidth is scaled accordingly. The residual network capacity increases proportionally with the integer multiple value.

Four different levels of churn are simulated: one based on the Gnutella session duration model from Table 4.10, and the remaining three based on exponential session duration with mean 10 seconds, 30 seconds and 300 seconds respectively. The last three types of churn are referred to as exponential churn for the remainder of the thesis. They are denoted by Exp(T), where T is mean session duration.

The four types of churn correspond roughly to the following scenarios:

Gnutella churn: general purpose P2P overlay network,

Exp(300 s): a more reliable general purpose network,

Exp(30 s): wireless network with slowly moving stations,

Exp(10 s): wireless network with rapid moving stations.

The Gnutella mean session duration is approximately 130 seconds, since in the model equation from Table 4.10

$$0.57\,\mathrm{N}_X(x; 0.85, 0.07) + 0.33\,\mathrm{LN}_X(x; 0.37, 0.96) + 0.10\,\mathrm{U}_X(x; 18.45, 2460)$$

the largest contribution comes from the uniform distribution term. The session interarrival time for Gnutella churn is modeled by the equation $\mathrm{LN}_X(x; 0.71, 1.08)$, as shown in Table 4.8 on page 101. From [138], the expected value for a random variable X following the lognormal distribution $\mathrm{LN}_X(x; \mu, \sigma)$ is $\mathrm{E}[X] = e^{\mu+\sigma^2/2}$. Hence, the Gnutella mean session interarrival time is 3.6 seconds.

For the remaining cases of churn, the Gnutella mean session interarrival time with an exponential distribution time is used.

RMP is configured to use 3, 5, and 7 backup paths, respectively. A higher number of backup paths increases the chances for successful flow allocation in situations with low residual network capacity.

Each experiment is executed 30 times for each parameter combination. This allows the construction of average values with corresponding 95 % confidence intervals. As in the case of RMP, the confidence intervals are very narrow, implying that the estimated average values are statistically sound. For ratio metrics in particular, the width of a 95 %-confidence interval is in the range 0.001.

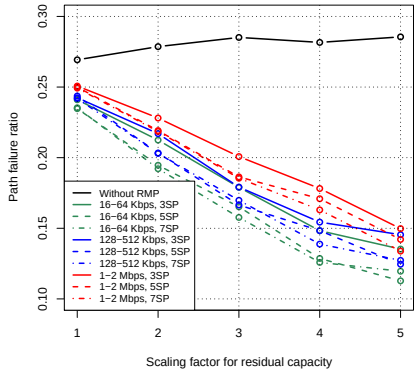
In each run the duration of simulated time is 3600 seconds. Analysis of simulation output by means of Welch's procedure [132, 140] revealed that RMP has a longer initial transient than RDP. A 2000 second warmup period was required for removing the transient, which is twice as long as the warmup period used in the case of RDP.

5.4.4 Performance Results

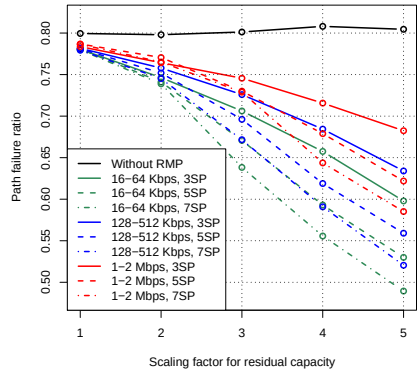
Figure 5.14 shows the performance of path restoration for each type of churn. The solid black line at the top of each subfigure indicates the ratio of path failures to the total number of paths ($p_t = 50$) when no path restoration is in use. It can be observed that the shorter the mean session duration is, the higher the path failure ratio becomes. The minimum path failure ratio occurs in the case of exponential churn with 300 seconds mean session duration. The maximum path failure ratio is registered for scenarios with Exp(30 s) and Exp(10 s) churn. These two scenario types show similar path failure ratio because 20 seconds difference in the mean session duration is not enough to affect significantly more links in the preferred paths.

Colored lines are used to plot the path failure ratio when RMP is used. Lower path failure ratio values indicate higher RMP success in restoring paths. The colors green, blue and red are used for curves belonging to bandwidth ranges 16–64 Kbps, 128–512 Kbps and 1–2 Mbps, respectively. Solid lines denote scenarios with 3SPs, dashed lines are used for scenarios with 5SPs and and alternating dots and dashes are used for scenarios with 7SPs.

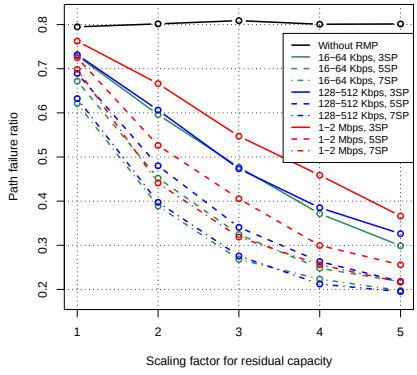
In all four cases of churn, the path failure ratio decreases when RMP is used. Clearly, RMP's success is directly proportional to the amount of residual capacity. In terms of reduced path failure ratio, the largest gains are registered for Exp(30 s) scenarios. In these scenarios, the path failure ratio is very high in the absence of RMP, which means that there is a lot of room for improvement.



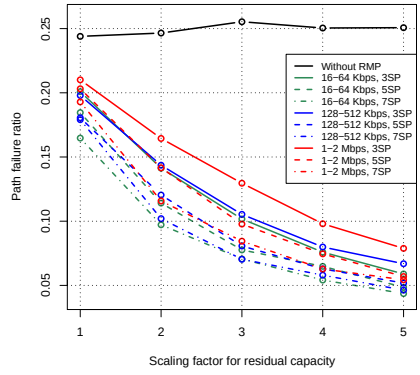
(a) Gnutella churn



(b) Exp(10 s) churn



(c) Exp(30 s) churn



(d) Exp(300 s) churn

Figure 5.14: RMP path restoration.

At the same time, the churn is less aggressive than in the case of Exp(10 s) scenarios. Hence, path diversity is less affected and backup paths are more stable.

Using more backup paths (*i. e.* , 5SPs or 7SPs) shows most gain in the case of Exp(30 s) scenarios for bandwidth range 1–2 Mbps. With less aggressive churn, the usefulness of additional backup paths decreases.

A complementary view of RMP's path restoration success is shown in Figure 5.15 in terms of the ratio of number of restored paths to the number of broken paths. RMP's largest success in restoring broken paths is experienced in the case of 16-64 Kbps scenarios, in each scenario. In Figure 5.15(c) for scaling factor 4 and 5, the restored paths ratio for 128-512 Kbps scenarios with 7SPs is slightly higher than the restored path ratio for 16-64 Kbps scenarios with 7SPs. These minimal differences are due to the random selection of source and destination node for each flow demand.

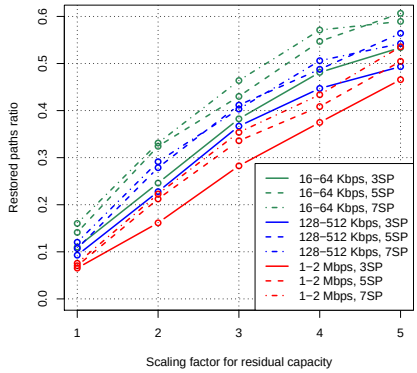
The worst path restored ratio is observed in the case of 1–2 Mbps scenarios with 3SPs. This is consistent across all subfigures in Figure 5.15 and is a direct result of lack of path diversity. In order to allocated 1–2 Mbps flow to a small number of paths, those particular paths must have high residual capacity. For the topologies used, it is more likely to find a large number of paths with low residual capacity that together can accommodate the 1-2 Mbps broken flows.

As in case of RDP, the cost for using RMP is assessed in terms of bandwidth utilization and bandwidth overhead, as shown in Figure 5.16 and Figure 5.17, respectively.

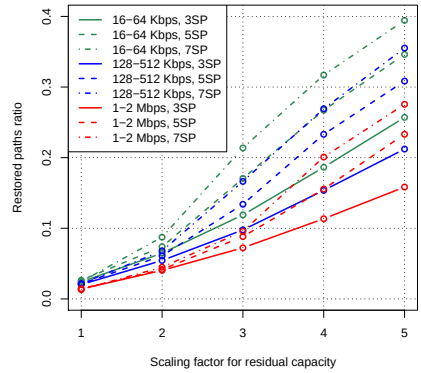
Since RMP's link vector algorithm uses triggered updates, it is no surprise that the highest bandwidth utilization occurs in the case of aggressive churn as it can be observed in Figure 5.16(b) and Figure 5.16(c).

It should be recalled from Section 5.4.1 that RMP sends updates in response to changes in the topology table. The topology table stored at a node consists of the node's outgoing links, its source graph and the source graphs of its neighbours. It should be clear that, with increasing number of backup paths, the number of links stored in the topology table grows as well. Consequently, changes to links in the network is more likely to trigger the node to send update messages. This behavior is observed in Figure 5.16 for 5SPs and 7SPs scenarios.

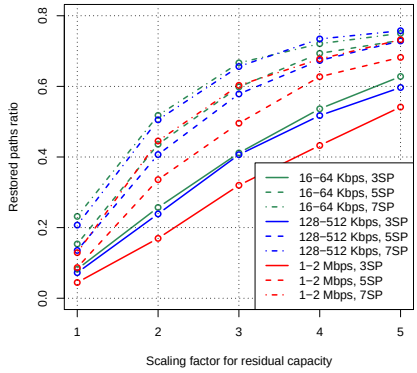
The bandwidth overhead shown in Figure 5.17 is computed by dividing the



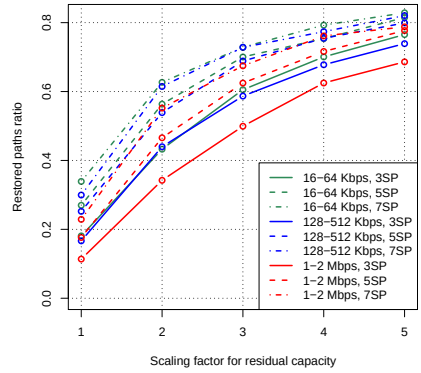
(a) Gnutella churn



(b) Exp(10 s) churn

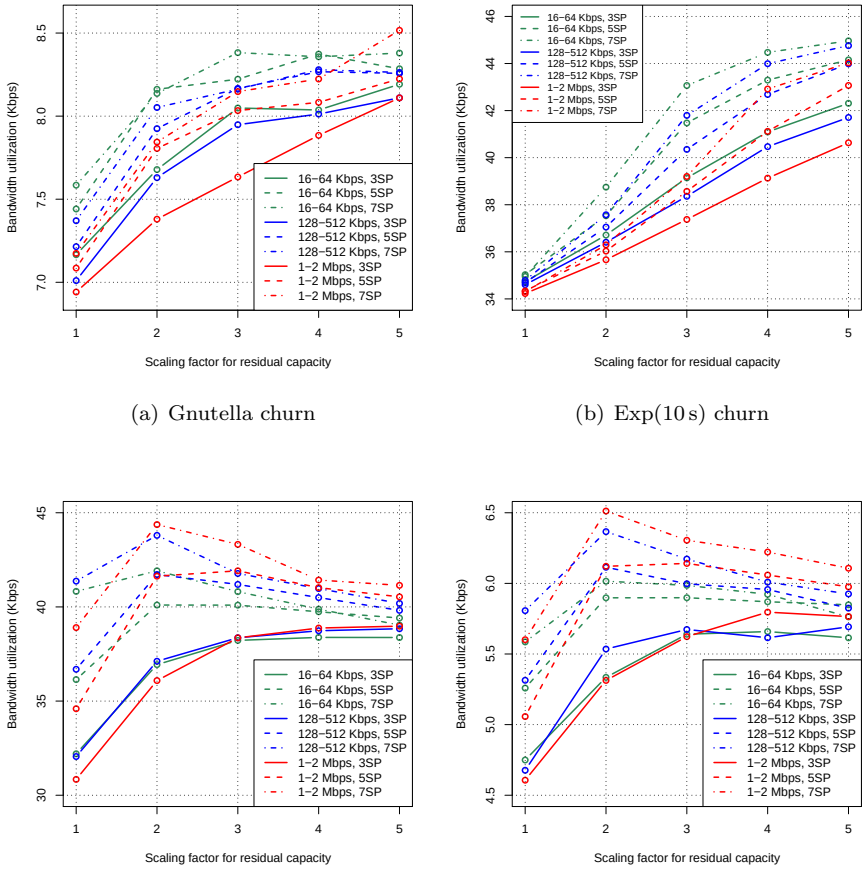


(c) Exp(30 s) churn



(d) Exp(300 s) churn

Figure 5.15: RMP restored paths ratio.



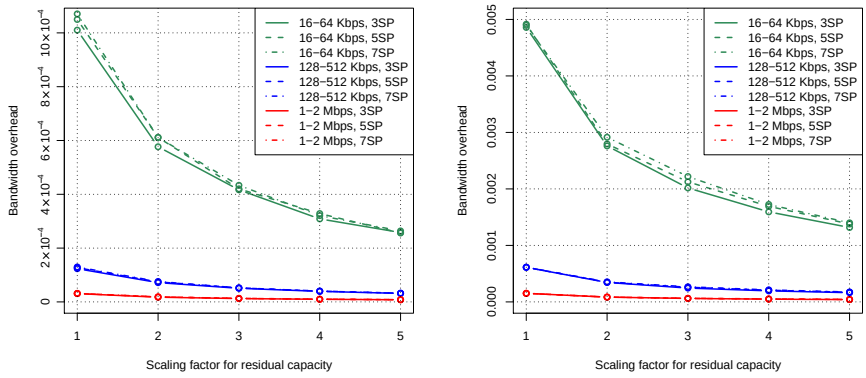
(a) Gnutella churn

(b) Exp(10 s) churn

(c) Exp(30 s) churn

(d) Exp(300 s) churn

Figure 5.16: RMP bandwidth utilization.



(a) Gnutella churn

(b) Exp(10 s) churn

(c) Exp(30 s) churn

(d) Exp(300 s) churn

Figure 5.17: RMP bandwidth overhead.

bandwidth utilization with the network capacity, *i. e.*, the sum of residual capacity and used capacity. In contrast to RDP's bandwidth overhead, the RMP bandwidth overhead is biased. For example, the 16–64 Kbps scenarios dominate in each of the churn scenarios. This happens because the amount of average residual capacity per link is determined by the product between the scaling factor in use and the upper bound of the bandwidth range. Hence, the network capacity for 16–64 Kbps scenarios is always lower than for the other two bandwidth ranges. Comparison of bandwidth overhead is fair only for graphs within the same bandwidth range.

For each bandwidth range, the largest bandwidth overhead occurs in 7SPs scenarios with Exp(30 s) churn when the scaling factor is equal to one: 0.6 % in the case of 16–64 Kbps scenarios, 0.07 % in the case of 128–512 Kbps scenarios, and 0.02 % in the case of 1–2 Mbps scenarios. RMP is less successful in establishing backup paths for Exp(10 s) scenarios. This explains why Exp(10 s) scenarios show lower bandwidth overhead than Exp(30 s) scenarios. The lowest bandwidth overhead occurs in the case of Exp(300 s) scenarios where it is approximately one magnitude lower than in the other scenarios.

5.5 Summary

At the beginning of this chapter a brief overview of QoS routing was provided. This was followed by a discussion about the design assumptions used in the design of ORP. The discussion included the type of underlying environment and the targeted multimedia applications. The remaining part of the chapter described RDP and RMP, the simulations performed for each protocol, and the performance results obtained from the simulations.

RDP is used to select a path between two nodes subject to a number of QoS constraints. Results from the RDP simulations indicate that the protocol can find bandwidth-constrained paths with no more than 0.03 % overhead in a network with 1000 nodes when TTL=8. The call blocking ratio depends on the amount of available bandwidth in the network, but it is also sensitive to the TTL value in use. When the utilization factor increases, the bandwidth overhead increases as well, especially in the case of 16–64 Kbps flow demands. Also, as the utilization increases, the call blocking ratio becomes increasingly

sensitive to the amount of available bandwidth in the network.

RMP is used to restore RDP paths when the original paths are broken. This includes the case when the path QoS constraints can no longer be satisfied. A link vector algorithm is used to share link state information among neighbouring nodes. When path needs to be restored, Yen's KSP is used to compute backup paths and the affected flows are allocated to them. RMP can be quite efficient in restoring broken paths provided enough residual capacity is available. For example in Exp(30 s) scenarios, in spite of aggressive churn, RMP is able to restore up to 40 % of broken paths used for transporting 1–2 Mbps flows, with approximately 0.02 % bandwidth overhead.

The worst-case cost in the case of RDP, namely 0.9 % bandwidth overhead, occurs for 16–64 Kbps flow demands with 180 seconds mean session duration when the utilization factor is one. For RMP, the worst-case cost, 0.6 %, occurs in Exp(30 s) scenarios with 16-64 Kbps flow demands, 7SPs and scaling factor one for residual capacity. Assuming an additive cost when RDP and RMP are being used together, the worst-case cost is estimated to be as high as 1.5 % protocol overhead in terms of bandwidth.

Chapter 6

Conclusions and Future Work

Wide availability of computing resources at the edge of the network has lead to the appearance of services implemented in overlay networks. These applications often utilize end-node resources as infrastructure rather than relying on dedicated resources. This approach greatly reduces the deployment effort and the cost for the service in question.

The dissertation addresses the problem of unicast QoS routing in overlay networks. More precisely, the emphasis is on methods for providing a QoS-aware service on top of IP's best-effort service, with minimal changes to existing Internet infrastructure.

6.1 Contributions of the Thesis

The ORP framework for QoS routing is proposed along with a set of design assumptions on the environment and multimedia application for which ORP can be used. The framework consists of two protocols: Route Discovery Protocol (RDP) and Route Management Protocol (RMP). RDP is used to find paths subject to a number of QoS constraints and RMP's task is to maintain them when churn occurs. The design and implementation of both protocols is presented and their performance is evaluated through an extensive simulation study. The study is focused on several parameters that capture the success of finding and maintaining QoS paths along with the cost in terms of bandwidth

overhead for using these protocols.

A Gnutella study was performed in order to gain a better understanding of the dynamics that occur in an overlay network. The study resulted in highly detailed statistical models and characteristics for Gnutella traffic crossing an ultrapeer. The models for session arrival and session duration were later used to generate churn in the ORP simulations. An additional contribution is the design and implementation of a flexible software library for P2P traffic decoding, based on `tcptrace`.

An important part of RMP's operation involves solving linear optimization problems for flow allocation. A software library for solving network flow problems was designed, based on the GLPK implementation of the simplex method and interior point method. The software library implementation, called `liboptim`, can be relatively easy extended to include additional flow problems and algorithms.

The `liboptim` library was essential for the implementation of a performance testbed for network flow algorithms. The testbed was used for performance measurements of the simplex method and interior point method as implemented by GLPK. Based on these measurements the simplex method was selected for use in RMP.

6.2 Future Directions and Research

The results of the performance study reported in the previous chapter indicate that ORP is a viable solution for QoS routing. The intention is to run similar tests in a more realistic environment, such as PlanetLab. However, a realistic PlanetLab implementation requires that ORP is extended to include two important elements: an overlay network that organizes nodes and transports ORP messages and the ability to measure link-state variables (*e. g.*, available bandwidth, delay and loss).

ORP's initial design envisioned the Gnutella network as the overlay of choice for ORP. In fact, the CPs and APs used by RDP can be directly mapped on Gnutella's QUERY and QUERY_HIT messages. For DPs and RMP link-state updates, either a new message type can be created or a existing message type (*e. g.*, QUERY_HIT) can be extended to include ORP data. However, ORP can

use any overlay that can forward message and address individual nodes.

Several methods for active measurement [171–175] exist already. It is important that these methods are evaluated and a subset suitable for ORP is selected.

There is a certain latency involved in obtaining link-state information via active measurements, as it was explained in Section 5.2. An interesting item for future work involves estimating the extent of this latency, its effect on ORP's ability to react to changes in the link-state, and finding methods to reduce it.

RDP's current path-selection policy is to use the path determined by the first CP arriving at the destination node. This policy favours paths with short one-way delay but can create hot-spots in the network. Future research should therefore also focus on designing policies for load-balancing in order to improve the utilization of network resources.

All experiments presented here focus on a single QoS metric: bandwidth. Additional experiments should be performed to evaluate ORP's performance when several QoS metrics are used. For RMP, this requires that Yen's KSP algorithm is replaced by SAMCRA or a similar algorithm.

Appendix A

Acronyms

ccdf	complementary cumulative distribution function
cdf	cumulative distribution function
cedf	complementary empirical distribution function
edf	empirical distribution function
pdf	probability density function
AODV	ad-hoc on-demand distance vector
AP	acknowledgement packet
API	application programming interface
AS	autonomous system
BFS	breadth-first-search
BGP	Border Gateway Protocol
BTH	Blekinge Institute of Technology
CLVL	controlled-loss virtual link
CP	control packet
CPU	central processing unit
DFS	depth-first-search
DHT	distributed hash table
DiffServ	Differentiated Services
DP	data packet
DSDV	destination sequence distance vector
DSR	dynamic source routing
EDA	exploratory data analysis
FEC	forward error correction
FIFO	first in first out

FR flow relay

GCC GNU Compiler Collection

GLPK GNU Linear Programming Toolkit

GSL GNU Scientific Library

GT-ITM Georgia Tech - Internetwork Topology Models

GUID globally unique identifier

GWC Gnutella Web Cache

HSEP Horizon Size Estimation Protocol

HTTP Hypertext Transfer Protocol

IP Internet Protocol

IPM interior point method

ISP Internet service provider

IPTV IP Television

IntServ Integrated Services

KSP K shortest paths

LN leaf node

MCP multi-constrained path

MCOP multi-constrained optimal path

ML maximum likelihood

MOS mean opinion score

NAT Network Address Translation

NED Network Description

ORP Overlay Routing Protocol

OSPF Open Shortest Path First

P2P peer-to-peer

PAP pure allocation problem

PAP-MLPF PAP with modified link-path formulation

PCAP packet capture

PIT probability integral transform

PSO particle swarm optimization

QoE quality of experience

QoS quality of service

QRON QoS-aware routing protocol for overlay networks

QRP Query Routing Protocol

QSON QoS overlay network

RDP Route Discovery Protocol

RIP Routing Information Protocol

RMP Route Management Protocol

RON Resilient Overlay Network

ROVER Routing in Overlay Networks

RSVP Resource Reservation Protocol

RTT round-trip time

SAMCRA Self-Adaptive Multiple Constraints Routing Algorithm

SLA service level agreement

SSH Secure Shell

STL Standard Template Library

SVD singular value decomposition

TCP Transmission Control Protocol

TTL time-to-live

UDP User Datagram Protocol

UHC UDP Host Cache

UMTS Universal Mobile Telecommunications System

UTC Coordinated Universal Time

UUID universally unique identifier

UP ultrapeer

VoD video on demand

VoIP voice over IP

Appendix B

Notation

B.1 Graph Theory

NOTATION	DEFINITION	PAGE
$\mathcal{G}$	shorthand notation for a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$	17
$\mathcal{V}$	set of vertices (nodes)	17
V	number of vertices in $\mathcal{V}$	17
$\mathcal{E}$	set of edges (links)	17
E	number of edges in $\mathcal{E}$	17
(u, v)	edge connecting nodes u and v	17
$P(u, v)$	path from vertex u to vertex v	19
$P^*(u, v)$	optimal path (*e. g.*, shortest path)	29
L	characteristic path length	19
C	clustering coefficient	19
λ	Euclidean distance between two nodes in the graph	20
Λ	maximum distance between two nodes in the graph	20
d_v	degree of vertex v	18
r_v	rank of vertex v	19

B.2 Probability and Statistics

NOTATION	DEFINITION	PAGE
θ	distribution parameter	81
$\hat{\theta}$	point estimate of θ	87
$f_X(x;\theta)$	pdf for the random variable X	81
$F_X(x;\theta)$	cdf for the random variable X	81
$\overline{F_X}(x;\theta)$	ccdf for the random variable X	82
$\mathcal{F}_{\mathcal{X}}(x;\theta)$	edf for the random variable X	86
$\overline{\mathcal{F}_{\mathcal{X}}}(x;\theta)$	cedf for the random variable X	86
X_n	nth element of a random sample	82
$X_{(n)}$	nth order statistic	83
$\mathrm{E}[X]$	expected value of the random variable X	83
$\mathrm{Var}[X]$	variance of the random variable X	87
$\hat{\mu} = \bar{X}$	sample mean (point estimate)	83
$\hat{\sigma}$	sample standard deviation (point estimate)	83

Appendix C

Probability Distributions

This is a very short review of the pdfs and cdfs for distributions used in this thesis. The review is based on information presented in [132, 187, 188].

C.1 Uniform Distribution, U[a,b]

$$F_X(x;a,b) = \begin{cases} 0 & \text{if } x < a \\ \dfrac{x-a}{b-a} & \text{if } a \le x \le b \\ 1 & \text{if } b < x \end{cases} \tag{C.1}$$

$$f_X(x;a,b) = \begin{cases} \dfrac{1}{b-a} & \text{if } a \le x \le b \\ 0 & \text{otherwise} \end{cases} \tag{C.2}$$

The special case, $U[0,1]$ is equivalent to

$$F_X(x;0,1) = \begin{cases} 0 & \text{if } x < a \\ x & \text{if } 0 \le x \le 1 \\ 1 & \text{if } 1 < x \end{cases} \tag{C.3}$$

$$f_X(x;0,1) = \begin{cases} 1 & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{C.4}$$

C.2 Poisson Distribution, PO[λ]

$$F_X(x;\lambda) = \begin{cases} 0 & \text{if } x < 0 \\ e^{-\lambda} \sum_{i=0}^{[x]} \frac{\lambda^i}{i!} & \text{otherwise} \end{cases} \tag{C.5}$$

where $[x]$ is the largest integer such that $[x] \leq x$.

$$f_X(x;\lambda) = \begin{cases} \frac{e^{-\lambda}\,\lambda^x}{x!} & \text{if } x \in \mathbb{N} \\ 0 & \text{otherwise} \end{cases} \tag{C.6}$$

C.3 Exponential Distribution, EXP[λ]

$$F_X(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & \text{if } 0 \leq x \\ 0 & \text{otherwise} \end{cases} \tag{C.7}$$

$$f_X(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{if } 0 \leq x \\ 0 & \text{otherwise} \end{cases} \tag{C.8}$$

C.4 Normal Distribution, N[μ, σ^2]

$$f_X(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \quad \text{for all } x \in \mathbb{R} \tag{C.9}$$

There is no closed form available for $F_X(x;\mu,\sigma^2)$. The values must be estimated numerically [132].

C.5 Lognormal Distribution, LN[μ, σ^2]

$$f_X(x;\mu,\sigma^2) = \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(\ln[x]-\mu)^2}{2\sigma^2}\right) \quad \text{for all } x \in \mathbb{R} \tag{C.10}$$

There is no closed form available for $F_X(x;\mu,\sigma^2)$. The values must be estimated numerically [132].

C.6 Pareto Distributions

The classical Pareto distribution, used for example in [136], is defined as

$$F_X(x; a, \kappa) = 1 - \left(\frac{\kappa}{x}\right)^a \tag{C.11}$$

where a is the shape parameter and κ is the location parameter. The location parameter is actually the lower bound of x. This is called a Pareto distribution of the first kind [188]. The corresponding probability density function is

$$f_X(x; a, \kappa) = a\kappa^a x^{-a-1} \tag{C.12}$$

The symbol a is used instead of α to avoid confusion with the shape parameter in the generalized Pareto distribution.

A generalized Pareto distribution [149] is defined as

$$F_X(x; \alpha, \kappa, \beta) = 1 - \left[1 + \frac{\alpha(x-\kappa)}{\beta}\right]^{-\frac{1}{\alpha}} \tag{C.13}$$

where α is the shape parameter, κ is the location parameter, and β is the scale parameter. The corresponding density function is

$$f_X(x; \alpha, \kappa, \beta) = \frac{1}{\beta}\left[1 + \frac{\alpha(x-\kappa)}{\beta}\right]^{-\frac{1}{\alpha}-1} \tag{C.14}$$

Clearly, for $\beta = \alpha\kappa$ and $\alpha = 1/a$, the generalized Pareto distribution is equivalent to the classical Pareto distribution.

In [189], the authors define the bounded Pareto density function

$$f_X(x) = \frac{a\kappa x^{-a-1}}{1 - (\kappa/K)^a} \quad k \le x \le K \tag{C.15}$$

with the probability distribution function

$$F_X(x) = \frac{\kappa x^{-a}}{(\kappa/K)^a - 1} \quad k \le x \le K. \tag{C.16}$$

where K is the upper bound of x. In contrast with the previous two Pareto distributions, the bound Pareto distribution is not a heavy-tail distribution. The distribution shows high variability if $k \ll K$, but its moments are finite.

Bibliography

[1] Google Inc., "Youtube," 2008. [Online]. Available: http://www.youtube.com

[2] Skype Communications, "Skype," Mar. 2006. [Online]. Available: http://www.skype.com

[3] Z. Wang, *Internet QoS: Architectures and Mechanisms for Quality of Service*. San Francisco, CA, USA: Morgan Kaufman Publishers, 2000, ISBN: 1-55860-608-4.

[4] Cisco Systems, *Internetworking Technologies Handbook*. Cisco Press, 2003, ch. Quality of Service (QoS). [Online]. Available: http://www.cisco.com/en/US/docs/internetworking/technology/handbook/QoS% .html

[5] R. Braden, D. D. Clark, and S. Shenker, *RFC 1633: Integrated Services in the Internet Architecture: an Overview*, IETF, Jun. 1994, category: Informational. [Online]. Available: http://www.ietf.org/ietf/rfc1633.txt

[6] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network*, vol. 7, no. 5, pp. 8–18, Sep. 1993.

[7] J. Wroclawski, *RFC 2210: The Use of RSVP with IETF Integrated Services*, IETF, Sep. 1997, category: Standards Track. [Online]. Available: http://www.ietf.org/ietf/rfc2210.txt

[8] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area Internet traffic patterns and characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, Nov. 1997.

[9] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the Sprint IP backbone," *IEEE Network*, vol. 17, no. 6, pp. 6–16, Nov. 2003.

[10] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, Dec. 1998. [Online]. Available: http://www.ietf.org/ietf/rfc2475.txt

[11] C. Bouras and A. Sevasti, "Service level agreements for DiffServ-based services' provisioning," *Journal of Computer Networks*, vol. 28, no. 4, pp. 285–302, Nov. 2005.

[12] T. Hoßfeld, P. Tran-Gia, and M. Fiedler, "Quantification of quality of experience for edge-based applications," in *Proceedings of ITC*, Ottawa, Canada, Jun. 2007, pp. 361–373.

[13] S. Winkler and C. Faller, "Perceived audiovisual quality of low-bitrate multimedia content," *IEEE Transactions on Multimedia*, vol. 8, no. 5, pp. 973–980, Oct. 2006.

[14] L. Ding, Z. Lin, A. Radwan, M. S. El-Hennawey, and R. A. Goubran, "Non-intrusive single ended speech quality assessment in VoIP," *Journal of Speech Communication*, vol. 49, pp. 477–489, Apr. 2007.

[15] V. Grancharov, D. Y. Zhao, J. Lindblom, and W. B. Kleijn, "Low-complexity, nonintrusive speech quality assessment," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 6, pp. 1948–1956, Nov. 2006.

[16] T. M. O'Neil, "Quality of experience and quality of service for IP video conferencing," Polycom Video Communications, Milpitas, CA, USA, White paper, 2002.

[17] G. J. Armitage, "Revisiting IP QOS," *ACM SIGCOMM Computer Communications Review*, vol. 33, no. 5, pp. 81–88, Oct. 2003.

[18] G. Bell, "Failure to thrive: QoS and the culture of operational networking," in *Proceedings ot the ACM SIGCOMM Workshops*, Karlsruhe, Germany, Aug. 2003, pp. 115–120.

[19] L. Burgsthaler, K. Dolzer, C. Hauser, J. Jähnert, S. Junghans, C. Macián, and W. Payer, "Beyond technology: The missing pieces for QoS success," in *Proceedings ot the ACM SIGCOMM Workshops*, Karlsruhe, Germany, Aug. 2003, pp. 121–130.

[20] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*, 2nd ed. San Francisco, CA, USA: Morgan Kaufman Publishers, 2000, ISBN: 1-55860-514-2.

[21] G. Malkin, *RFC 2453: RIP Version 2*, IETF, Nov. 1998, category: Standards Track. [Online]. Available: http://www.ietf.org/ietf/rfc2453.txt

[22] J. Moy, *RFC 2328: OSPF Version 2*, IETF, Apr. 1998. [Online]. Available: http://www.ietf.org/ietf/rfc2328.txt

[23] Y. Rekhter, T. Li, and S. Hares, *RFC 4271: A Border Gateway Protocol 4 (BGP-4)*, IETF, Jan. 2006. [Online]. Available: http://www.ietf.org/ietf/rfc4271.txt

[24] C. Huitema, *Routing in the Internet*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2000, ISBN: 0-13-022647-5.

[25] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Boston, MA, USA: Addison Wesley Longman, 2001, ISBN: 0-201-47711-4.

[26] D. G. Andersen, "Resilient overlay networks," Master's thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 2001.

[27] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in a cooperative environment," in *Proceeding of IPTPS*, Berkeley, CA, USA, Feb. 2003.

[28] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous streaming multicast in application-layer overlay networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 91–106, Jan. 2004.

[29] Z. Li and P. Mohapatra, "QRON: QoS-aware routing in overlay networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 29–40, Jan. 2004.

[30] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz, "OverQoS: An overlay based architecture for enhancing Internet QoS," in *Proceedings of NSDI*, San Francisco, CA, USA, Mar. 2004.

[31] D. Ilie and A. Popescu, "A framework for overlay QoS routing," in *Proceedings of 4th Euro-FGI Workshop*, Ghent, Belgium, May 2007.

[32] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001, pp. 149–160.

[33] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proceedings of ACM SIGCOMM*, Pittsburgh, PA, USA, Aug. 2002, pp. 73–88.

[34] P. Maymounkov and D. Maziéres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proceedings of IPTPS*, Berkeley, CA, USA, Mar. 2002.

[35] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a common API for structured peer-to-peer overlays," in *Proceedings of IPTPS*, Berkeley, CA, USA, Feb. 2003.

[36] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, no. 10, Oct. 2000. [Online]. Available: http://www.firstmonday.org/issues/issue5_10/adar/index.html

[37] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, pp. 1243–1248, Dec. 1968. [Online]. Available: http://www.sciencemag.org/cgi/content/full/162/3859/1243

[38] J. Ritter, *Why Gnutella Can't Scale. No, Really.*, Feb. 2001. [Online]. Available: http://www.darkridge.com/~jpr5/doc/gnutella.html

[39] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of MMCN*, San Jose, CA, USA, Jan. 2002.

[40] S. Sen and J. Wang, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, Marseille, France, Nov. 2002, pp. 137–150.

[41] N. B. Azzouna and F. Guillemin, "Experimental analysis of the impact of peer-to-peer applications on traffic in commercial IP networks," *European Transactions on Telecommunications: Special Issue on P2P Networking and P2P Services*, 2004.

[42] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "File-sharing in the internet: A characterization of P2P traffic in the backbone," University of California, Riverside, Tech. Rep., Nov. 2003.

[43] ——, "Is P2P dying or just hiding?" in *Proceedings of IEEE Globecom*, Dallas, TX, USA, Dec. 2004.

[44] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic," in *Proceedings of IMC*, Taormina, Sicily, Italy, Oct. 2004.

[45] L. Lao, S. S. Gokhale, and J.-H. Cui, "Distributed QoS routing for backbone overlay networks," in *Proceedings of IFIP Networking*, Coimbra, Portugal, May 2006.

[46] D. Ilie, "Overlay routing protocol (ORP)," Dec. 2004, unpublished architecture and design document.

[47] K. De Vogeleer, "QoS routing in overlay networks," Master's thesis, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, Jun. 2007, MEE07:24.

[48] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont, MA, USA: Athena Scientific, 1997, ISBN: 1-886529-01-9.

[49] D. P. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1991, ISBN: 0-13-200916-1.

[50] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Mineola, NY, USA: Dover Publications, 1998, ISBN: 0-486-40258-4.

[51] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: The MIT Press, 2001, ISBN: 0-262-53196-8.

[52] T. Bu and D. Towsley, "On distinguishing between Internet power law topology generators," in *Proceedings of IEEE INFOCOM*, Amherst, MA, USA, Jun. 2002.

[53] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

[54] S. Jin and A. Bestavros, "Small-world characteristics of Internet topologies and implications on multicast scaling," *Journal of Computer Networks*, vol. 50, no. 5, pp. 648–666, Apr. 2006.

[55] F. A. Kuipers, "Quality of service routing in the internet: Theory, complexity and algorithms," Ph.D. dissertation, Delft University, Delft, The Netherlands, 2004, ISBN: 90-407-2523-3.

[56] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[57] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for Internet topology," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 770–783, Dec. 1997.

[58] "`GT-ITM`: Georgia Tech internetwork topology models," 1997. [Online]. Available: http://www.cc.gatech.edu/projects/gtitm

[59] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of SIGCOMM*, Cambridge, MA, USA, Aug. 1999, pp. 251–262.

[60] A. Medina, I. Matta, and J. Byers, "On the origin of power laws in Internet topologies," *ACM SIGCOMM Computer Communications Review*, vol. 30, no. 2, pp. 18–28, Apr. 2000.

[61] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power laws and the AS-level Internet topology," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, pp. 514–524, Aug. 2003.

[62] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, Oct. 1999.

[63] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal topology generation from a user's perspective," Boston University, Boston, MA, USA, Tech. Rep. BUCS-TR-20001-03, Apr. 2001. [Online]. Available: http://www.cs.bu.edu/brite

[64] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theories, Algorithms, and Applications.* Upper Saddle River, NJ, USA: Prentice Hall, 1993, ISBN: 0-13-617549-X.

[65] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completness.* New York, NY, USA: W. H. Freeman and Company, 1979, ISBN: 0-7167-1045-5.

[66] D. G. Luenberger, *Linear and Nonlinear Programming.* Norwell, MA, USA: Kluwer Academic Publishers, 2004, ISBN: 1-4020-7593-6.

[67] R. W. Hamming, *Numerical Methods for Scientists and Engineers.* Mineola, NY, USA: Dover Publications, 1986, ISBN: 0-486-65241-6.

[68] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: The Johns Hopkins University Press, 1996, ISBN: 0-8018-5414-8.

[69] Y. Donoso and R. Fabregat, *Multi-Objective Optimization in Computer Networks Using Metaheuristics.* Boca Raton, FL, USA: Auerbach Publications, 2007, ISBN: 0-8493-8084-7.

[70] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks.* San Francisco, CA, USA: Morgan Kaufman Publishers, 2004, ISBN: 0-12-557189-5.

[71] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, Sep. 2003.

[72] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence.* Chichester, West Sussex, England: John Wiley & Sons, 2006, ISBN: 0-470-09191-6.

[73] M. Sipser, *Introduction to the Theory of Computation.* Boston, MA, USA: PWS Publishing Company, 1997, ISBN: 981-240-226-8.

[74] S. A. Cook, "An overview of computational complexity," *Communications of the ACM*, vol. 26, no. 6, pp. 400–408, Jun. 1983.

[75] Z. Wang and J. Crowfort, "Quality-of-service routing for supporting multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, Sep. 1996.

[76] F. A. Kuipers and P. F. A. Van Mieghem, "Conditions that impact the complexity of QoS routing," *IEEE/ACM Transactions on Networking*, vol. 13, no. 4, pp. 717–730, Aug. 2005.

[77] S. A. Vavasis, *Nonlinear Optimization: Complexity Issues.* Oxford, UK: Oxford University Press, 1991, ISBN: 0-19-507208-1.

[78] M. Avriel, *Nonlinear Programming: Analysis and Methods.* Mineola, NY, USA: Dover Publications, 2003, ISBN: 0-486-43227-0.

[79] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, Jul. 1971.

[80] E. Q. V. Martins and M. M. B. Pascoal, "A new implementation of Yen's ranking loopless paths algorithms," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 121–133, 2003.

[81] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[82] R. L. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ, USA: Princeton University Press, 1962, ISBN: 0-691-07962-5.

[83] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[84] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987.

[85] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Transactions on Algorithms*, vol. 3, no. 4, pp. 45:1–45:19, 2007.

[86] D. Ilie, "Optimization algorithms with applications to unicast QoS routing in overlay networks," Blekinge Institute of Technology, Karlskrona, Sweden, Research Report 2007:09, Sep. 2007, ISSN: 1103-1581.

[87] J. Nocedal and J. S. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer Science+Business Media, 2006, ISBN: 0-387-30303-0.

[88] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, Nov. 1992.

[89] P. Van Mieghem and F. A. Kuipers, "Concepts of exact QoS routing algorithms," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 851–864, Oct. 2004.

[90] F. Kuipers, P. Van Mieghem, T. Korkmaz, and M. Krunz, "An overview of constraint-based path selection algorithms for QoS routing," *IEEE Communications Magazine*, vol. 40, no. 2, pp. 50–55, Dec. 2002.

[91] S. Chen and K. Nahrstedt, "Distributed quality-of-service routing in high-speed networks based on selective probing," in *Proceedings of LCN*, Lowell, MA, USA, Oct. 1998, pp. 80–89.

[92] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, Dec. 1952.

[93] W. H. Press, S. A. Teukolsky, W. T. Vettering, and B. P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*, 2nd ed. Cambridge, UK: Cambridge University Press, 2002, ISBN: 0-521-75033-4.

[94] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, jan 1998.

[95] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, *GNU Scientific Library Reference Manual*, 2nd ed. Bristol, UK: Network Theory Limited, 2006, ISBN: 0-9541617-3-4. [Online]. Available: http://www.gnu.org/software/gsl

[96] GNU Linux, "random, urandom — kernel random number source devices," in *Linux Programmer's Manual*, Jan. 2008, RANDOM(4). [Online]. Available: http://www.kernel.org/doc/man-pages

[97] A. Makhorin, *GNU Linear Programming Kit: Reference Manual version 4.24*, Moscow Aviation Institute, Moscow, Russia, Nov. 2007. [Online]. Available: http://www.gnu.org/software/glpk

[98] GNU Linux, "proc — process information pseudo-filesystem," in *Linux Programmer's Manual*, Jan. 2008, PROC(5). [Online]. Available: http://www.kernel.org/doc/man-pages

[99] ——, "getrusage — get resource usage," in *Linux Programmer's Manual*, Jan. 2008, GETRUSAGE(2). [Online]. Available: http://www.kernel.org/doc/man-pages

[100] ——, "gettimeofday, settimeofday — get / set time," in *Linux Programmer's Manual*, 2008, GETTIMEOFDAY(2).

[101] N. M. Josuttis, *The C++ Standard Library: A Tutorial and Reference*. Boston, MA, USA: Addison Wesley, 1999, ISBN: 0-201-37926-0.

[102] S. Meyers, *Effective STL*. Boston, MA, USA: Addison Wesley, 2001, ISBN: 0-201-74962-9.

[103] L. Torvalds, "`do_gettimeofday()`," Linux 2.6.14 Kernel Sources, 2005, arch/i386/kernel/time.c.

[104] D. Ilie, "Gnutella network traffic: Measurements and characteristics," Licentiate Dissertation, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, Apr. 2006, ISBN: 91-7295-084-6.

[105] D. Ilie and A. Popescu, "Statistical models for Gnutella signaling traffic," *Journal of Computer Networks*, vol. 51, no. 17, pp. 4816–4835, Dec. 2007.

[106] T. Klingberg and R. Manfredi, *Gnutella 0.6*, The Gnutella Developer Forum (GDF), Jun. 2002. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Development

[107] A. Singla and C. Rohrs, *Ultrapeers: Another Step Towards Gnutella Scalability*, 1st ed., Lime Wire LLC, Nov. 2002. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Development

[108] A. A. Fisk, *Gnutella Dynamic Query Protocol*, 0th ed., LimeWire LLC, May 2003. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/search/Dynamic Querying

[109] P. Verdy, "Gnutella topology," The Gnutella Developer Forum (GDF), Jan. 2006, http://groups.yahoo.com/group/the_gdf/message/22187.

[110] Gnucleus, "Gnutella web cache," Jun. 2006, http://www.gnucleus.com/gwebcache.

[111] GDF, "Gnutella protocol development," Dec. 2005, http://www.the-gdf.org.

[112] R. Manfredi, *Gnutella Traffic Compression*, The Gnutella Developer Forum (GDF), Jan. 2003. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/Gnet Compression

[113] J.-l. Gailly and M. Adler, "zlib," Dec. 2005, http://www.gzip.org/zlib.

[114] P. Leach, M. Mealling, and R. Salz, *RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace*, Jul. 2005, category: Standards Track. [Online]. Available: http://www.ietf.org/ietf/rfc4122.txt

[115] C. Rohrs, *SACHRIFC: Simple Flow Control for Gnutella*, Lime Wire LLC, Mar. 2002.

[116] ——, *Query Routing for the Gnutella Network*, 1st ed., Lime Wire LLC, May 2002. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Development

[117] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communication of the ACM*, vol. Volume 13, no. Number 7, pp. p. 422–426, July 1970, iSSN:0001-0782.

[118] A. A. Fisk, *Gnutella Ultrapeer Query Routing*, 0th ed., Lime Wire LLC, May 2003. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Proposals/Working Proposals/search/Ultrapeer QRP

[119] T. Schürger, *Horizon size estimation on the Gnutella network v0.2*, Mar. 2004. [Online]. Available: http://www.menden.org/gnutella/hsep.html

[120] F. Michaut and F. Lepage, "Application-oriented network metrology: Metrics and active measurement tools," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 2–24, 2005.

[121] D. Ilie, D. Erman, A. Popescu, and A. A. Nilsson, "Traffic measurements of P2P systems," in *Proceedings of SNCNW*, Karlstad, Sweden, Nov. 2004, pp. 25–29.

[122] V. Jacobsen, C. Leres, and S. McCanne, "Tcpdump/libpcap," http://www.tcpdump.org, Aug. 2005.

[123] S. Ostermann, "Tcptrace," http://www.tcptrace.org, Aug. 2005.

[124] D. Ilie, D. Erman, A. Popescu, and A. A. Nilsson, "Measurement and analysis of Gnutella signaling traffic," in *Proceedings of IPSI*, Stockholm, Sweden, Sep. 2004.

[125] D. Erman, "Bittorrent traffic measurements and models," Licentiate Dissertation, Blekinge Institute of Technology (BTH), Karlskrona, Sweden, Oct. 2005, iSBN: 91-7295-071-4.

[126] D. Erman, D. Ilie, and A. Popescu, "BitTorrent session characteristics and models," in *Proceedings of HET-NETs*, D. Kouvatsos, Ed., Ilkley, West Yorkshire, UK, Jul. 2005, pp. P30/1–P30/10.

[127] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated: The Implementation.* Boston, MA, USA: Addison Wesley, 1995, ISBN: 0-201-63354-X.

[128] V. Paxson, "Empirically derived analytic models for wide-area tcp connections," *IEEE/ACM Transactions on Networking*, vol. 2, no. 4, pp. 316–336, Aug. 1994.

[129] L. Kleinrock, *Queueing Systems Volume 1: Theory.* Hoboken, NJ, USA: John Wiley & Sons, 1975, ISBN: 0-471-49110-1.

[130] R. B. D'Agostino and M. A. Stephens, *Goodness-of-Fit Techniques.* New York, NY, USA: Marcel Dekker, Inc., 1986, ISBN: 0-8247-7487-6.

[131] J. Maindonald and J. Braun, *Data Analysis and Graphics using R: An Example-based Approach.* Cambridge, UK: Cambridge University Press, 2003, ISBN: 0-521-81336-0.

[132] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 3rd ed. New York, NY, USA: McGraw-Hill, 2000, ISBN: 0-07-059292-6.

[133] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S-PLUS*, 3rd ed. New York, NY, USA: Springer-Verlag, 1999, ISBN: 0-387-98825-4.

[134] M. P. Wand, "Data-based choice of histogram bin width," *The American Statistician*, vol. 51, pp. 59–64, 1997.

[135] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation.* Hoboken, NJ, USA: John Wiley & Sons, 2000, ch. 1: Self-Similar Network Traffic: An Overview, pp. 1–38, ISBN: 0-471-31974-0.

[136] V. Paxson and S. Floyd, "Wide area traffic: The failure of the poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, Jun. 1995.

[137] R. J. Larsen and M. L. Marx, *An Introduction to Mathematical Statistics and Its Applications*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1986, ISBN: 0-13-487174-X.

[138] A. M. Mood, F. A. Graybill, and D. C. Boes, *Introduction to the Theory of Statistics*, 3rd ed. New York, NY, USA: McGraw-Hill, 1974, ISBN: 0-07-085465-3.

[139] D. C. Montgomery and G. C. Runger, *Applied Statistics and Probability for Engineers*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, 1999, ISBN: 0-471-17027-5.

[140] J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2001, ISBN: 0-13-088702-1.

[141] J. Beran, *Statistics for Long-Memory Processes*. New York, NY, USA: Chapman & Hall, 1994, ISBN: 0-412-04901-5.

[142] D. Ilie, D. Erman, and A. Popescu, "Transfer rate models for Gnutella signaling traffic," in *Proceedings of ICIW*, Guadeloupe, French Caribbean, Feb. 2006.

[143] F. N. David and N. L. Johnson, "The probability integral transform when the variable is discontinuous," *Biometrika*, vol. 37, no. 1–2, pp. 42–49, Jun. 1950.

[144] D. M. Titterington, A. F. M. Smith, and U. E. Makov, *Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons, 1985, ISBN: 0-471-90763-4.

[145] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2005, ISBN 3-900051-07-0
http://www.R-project.org.

[146] J. C. Lagarias, J. A. Reeds, M. H. Wright, and W. P. E., "Convergence properties of the Nelder-Mead simplex algorithm in low dimensions," *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 112–147, 1998.

[147] J. H. Mathews and K. K. Funk, *Numerical Methods using Matlab*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2004, ch. 8: Numerical Optimization, pp. 430–436, ISBN: 0-13-065248-2.

[148] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," Northwestern University, Evanston, IL, USA, Tech. Rep. NAM-08, May 1994.

[149] S. Coles, *An Introduction to Statistical Modeling of Extreme Values*, ser. Springer Series in Statistics. London, UK: Springer-Verlag, 2001, ISBN: 1-85233-459-2.

[150] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," Deparment of Computer Science and Engineering, University of Washington, Seattle, WA, USA, Tech. Rep. UW-CSE-01-06-02, Jul. 2001.

[151] N. Brownlee and K. C. Claffy, "Understanding internet traffic streams: Dragonflies and tortoises," *IEEE Communications Magazine*, pp. 110–117, Oct. 2002.

[152] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, *Nonlinear Estimation and Classification*, ser. Lecture Notes in Statistics. New York, NY, USA: Springer-Verlag, 2003, vol. 171, ch. Internet Traffic Tends Toward Poisson and Independent as the Load Increases, pp. 83–110, ISBN: 0-387-95471-6.

[153] J. Cao and K. Ramanan, "A Poisson limit for buffer overflow probabilities," in *Proceedings of IEEE Infocom*, no. 1, Jun. 2002, pp. 994–1003.

[154] K. Sriram and W. Whitt, "Characterizing superposition arrival processes in packet multiplexers for voice and data," *IEEE Journal on Selected Areas in Communications*, vol. SAC-4, no. 6, pp. 833–846, Sep. 1986.

[155] D. R. Cox and H. D. Miller, *The Theory of Stochastic Processes.* London, UK: Chapman & Hall, 1965, ISBN: 0-412-15170-7.

[156] S. M. Ross, *Applied Probability Models With Optimization Applications*, ser. Dover Books On Mathematics. Mineola, NY, USA: Dover Publications, 1992, ISBN: 0-486-67314-6.

[157] E. Gelenbe, M. Gellman, R. Lent, P. Lei, and P. Su, "Autonoumous smart routing for network QoS," in *Proceedings of ICAC*, New York, NY, May 2004, pp. 232–239.

[158] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu, "Cognitive packet networks: QoS and performance," in *Proceedings of IEEE MASCOTS*, Ft. Worth, TX, USA, Oct. 2002, pp. 3–12.

[159] J. Behrens and J. J. Garcia-Luna-Aceves, "Distributed, scalable routing based on link-state vectors," in *Proceedings of SIGCOMM*, London, UK, Aug. 1994, pp. 136–147.

[160] J. J. Garcia-Luna-Aceves, "Loop-free routing using diffusing computations," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, pp. 130–141, Feb. 1993.

[161] S. Chen, "Routing support for providing guaranteed end-to-end quality-of-service," Ph.D. dissertation, Engineering College of the University of Illinois, Urbana, IL, USA, 1999.

[162] D. H. Lorenz, "QoS routing and partitioning in networks with per-link performance-dependent costs," Ph.D. dissertation, Israel Institute of Technology, Haifa, Israel, 2004.

[163] R. Guérin and A. Orda, "QoS-based routing in networks with inaccurate information: Theory and algorithms," in *Proceedings of INFOCOM*, vol. 1, Kobe, Japan, Apr. 1997, pp. 75–83.

[164] D. H. Lorenz and A. Orda, "QoS routing in networks with uncertain parameters," *IEEE/ACM Transactions on Networking*, vol. 6, no. 6, pp. 768–778, Dec. 1998.

[165] S. Chen and K. Nahrstedt, "Distributed QoS routing with imprecise state information," in *Proceedings of ICCCN*, Lafayette, LA, USA, Oct. 1998.

[166] ——, "An overview of quality of service routing for the next generation high-speed networks: Problems and solutions," *IEEE Network*, vol. 12, no. 6, pp. 64–79, Nov. 1998.

[167] E. Gelenbe, R. Lent, and A. Nunez, "Self-aware networks and QoS," in *Proceedings of the IEEE*, vol. 92, Sep. 2004, pp. 1478–1489.

[168] W. C. Lee, "Topology aggregation for hierarchical routing in ATM networks," *ACM SIGCOMM Computer Communications Review*, vol. 25, no. 2, pp. 82–92, Apr. 1995.

[169] K.-S. Lui, K. Nahrstedt, and S. Chen, "Routing with topology aggregation in delay-bandwith sensitive networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 1, pp. 17–29, Feb. 2004.

[170] J. Schiller, *Mobile Communications*, 2nd ed. Boston, MA, USA: Addison Wesley, 2003, ISBN: 0-321-12381-6.

[171] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proceedings of IMW*, Marseille, France, Nov. 2002.

[172] K. G. Anagnostakis, M. Greenwald, and R. S. Ryger, "cing: Measuring network-internal delays using only existing infrastructure," in *Proceedings of INFOCOM*, San Francisco, CA, USA, Apr. 2003.

[173] R. Prasad, C. Dovrolis, M. Murray, and K. C. Claffy, "Bandwidth estimation: Metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, no. 6, pp. 27–35, Nov. 2003.

[174] J. Sommers, P. Barford, and W. Willinger, "Laboratory-based calibration of available bandwidth estimation tools," *Microprocessors and Microsystems*, vol. 31, no. 4, pp. 225–235, Jun. 2007.

[175] J. Sommers, P. Barford, N. Duffield, and A. Ron, "A geometric approach to improving active packet loss measurement," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 307–320, Apr. 2008.

[176] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson, "Profiling a million user DHT," in *Proceedings of IMC*, San Diego, CA, USA, Oct. 2007.

[177] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern P2P file-sharing systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 267–280, Apr. 2008.

[178] A. Varga, "`OMNeT++`," Mar. 2006. [Online]. Available: http://www.omnetpp.org

[179] V. A. Bolotin, "Modeling call holding time distributions for CCS network design and performance analysis," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 3, pp. 433–438, Apr. 1994.

[180] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *Proceedings of ICNP*, Atlanta, GA, USA, Oct. 1997.

[181] A. A. Shaikh, "Efficient dynamic routing in wide-area networks," Ph.D. dissertation, University of Michigan, Ann Arbor, MI, USA, 1999.

[182] S. Floyd and V. Paxon, "Difficulties in simulating the internet," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 392–403, Aug. 2001.

[183] K. De Vogeleer, D. Ilie, and A. Popescu, "Constrained-path discovery by selective diffusion," in *Proceedings of HET-NETs*, Karlskrona, Sweden, Feb. 2008.

[184] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, no. 1, pp. 95–116, Apr. 1984.

[185] F. A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem, "Performance evaluation of constrained-based path selection algorithms," *IEEE Network*, vol. 18, no. 5, pp. 16–23, Sep. 2004.

[186] C. P. Mayer and T. Gamer, "Integrating real world applications into OMNeT++," Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2, Feb. 2008, ISSN: 1613-849X.

[187] S. Ghahramani, *Fundamentals of Probability with Stochastic Processes*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2005, ISBN: 0-13-145340-8.

[188] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions, Vol. 1*, 2nd ed., ser. Wiley Series in Probability and Mathematical Statistics. Hoboken, NJ, USA: John Wiley & Sons, 1994, ISBN: 0-471-58495-9.

[189] M. Harchol-Balter, M. Crovella, and C. Murta, "On choosing a task assignment policy for a distributed server system," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 204–228, Nov. 1999.