A Cluster-Based QoS Testbed for Multimedia Communications

Hermann HELLWAGNER
Department of Information Technology, University Klagenfurt
9020 Klagenfurt, Austria
and
Erich KARGL[*]
Knapp Systemintegration GmbH
8700 Leoben, Austria

Abstract

This paper presents an inexpensive cluster-based QoS networking testbed that can be employed to "emulate" different networks for multimedia communication experiments. Such a network can be built using standard PC and Ethernet hardware and open-source software components, e.g., IP routing and traffic control available in recent Linux kernels as well as a Differentiated Services package built atop these building blocks. The testbed can flexibly be configured to model various link bandwidths as well as IP routers capable of classifying, queuing (with various disciplines), forwarding and/or dropping packets and shaping traffic. The QoS components and facilities of the testbed are introduced and initial performance analysis experiments and results are reported. A simple video streaming application under QoS control is presented to show the usefulness of the testbed.

Keywords: multimedia communications, quality of service, QoS, networking testbed, routing, Linux

1 Introduction

The amount of "multimedia" information available in distributed environments like the Internet or being used in broadcasting, telecooperation, or interactive networked environments is growing rapidly. Multimedia information like audio and video (A/V) streams, graphical animations, virtual reality experiences, or executable software components delivered over packet networks will become pervasive in the years to come. Applications in entertainment, education, e-commerce, information services, cooperative environments and many other areas are manifold and are expected to have a major impact on our everyday lives.

Multimedia data, in particular A/V streams, pose considerable requirements on all techniques and (hardware and software) components involved in capturing, coding, storing, searching, delivering, decoding, and presenting the data in an appropriate manner. Transport and presentation of A/V data need not only be correct, but also timely; i.e., real-time playout requirements must be met, despite potential impediments in the network like bursty traffic characteristics, low-bandwidth or/and lossy links or congestion along the delivery path.

As a consequence, providing quality of service (QoS) in packet networks has become a vivid research and development area in recent years. Several Internet QoS frameworks and protocols have been developed and suggested for standardization. Techniques include Integrated Services (IntServ) [4, 9] and the Resource Reservation Protocol (RSVP) [7, 11, 18], the Differentiated Services (DiffServ) framework [3, 8], fast packet-forwarding schemes like Multiprotocol Label Switching (MPLS) [16, 10], and resulting Constraint-Based Routing and Traffic Engineering mechanisms [17].

Currently, QoS support is, or is becoming, widely available in newer "enterprise" network components only, predominantly routers [14]. Access to these devices would be desirable for educational purposes, to provide students with hands-on experience with recent Internet QoS concepts and configurations. Moreover, experimental research into e.g. multimedia data delivery under QoS control would ideally be performed on a real-world QoS-capable network. In practice, though, such devices or network testbeds are typically *not* accessible for a small university department.

In order to address this deficiency, we set up and specifically configured a cluster of Linux machines to serve as a QoS network testbed for A/V data delivery. The cluster comprises A/V streaming server and client machines, several Linux routers, and a QoS-capable Ethernet switch. The routers implement QoS facilities using the network traffic control mechanisms of recent Linux kernels [1, 2, 13]. The routers can flexibly be configured to implement various QoS policies, based on different packet classification, queuing, for-

[*]Work was performed while the author was with the University Klagenfurt.

warding, and dropping schemes. Traffic can also be shaped to specific rates, in order to "emulate" different bandwidths available to A/V streams across the Internet.

The configuration and components of, and initial experiences with, this cluster-based QoS testbed are described in this paper. Section 2 outlines the basic configuration, while Section 3 discusses the components and QoS facilities in more detail. Section 4 gives preliminary performance results; Section 5 presents a simple video transfer experiment under QoS control. Related work is addressed in Section 6 and conclusions are given in Section 7.

2 Testbed Configuration

An example configuration of the QoS network testbed is depicted in Figure 1. The configuration can be changed or extended easily, as required for specific experiments.

The network shown comprises a single server and a single client machine, both of which are simple Linux PCs; a QoS-capable Summit4 Gigabit and Fast Ethernet switch by Extreme Networks, Inc.; and a couple of Linux Routers (LRs), each of which is equipped with multiple 100 Mbps Ethernet interfaces and IP routing and traffic control software [1, 13]. The LRs can thus be connected in a point-to-point fashion and the transfer rates among them can be controlled, to "emulate" networks with different numbers of hops, different link bandwidths along the server–client path, and alternative paths.

A Windows NT-based PC denoted as management station (MS) allows to configure and monitor the Summit4 switch. Furthermore, this machine and an additional Linux-based lab PC connected to the switch, are used to inject traffic into the network that competes with the server–client data stream for link bandwidth. All machines are off-the-shelf, low-cost PCs equipped with 450 MHz Pentium II processors.

Notice that the client machine and LR2, LR3 and LR4 reside in subnetworks that can only be reached via LR1. The other machines have standard class-B IP addresses and are linked into the university network via the Summit4 switch.

3 QoS Components and Facilities

The basic QoS components in the network are the Extreme Networks Summit4 switch and the Linux routers that implement traffic control.

Summit4

The Summit4 is a Gigabit Ethernet (6 ports) and Fast Ethernet (16 auto-negotiating ports) switch, supporting most of the features one can expect from a state-of-the-art switch. Examples include switching and IP

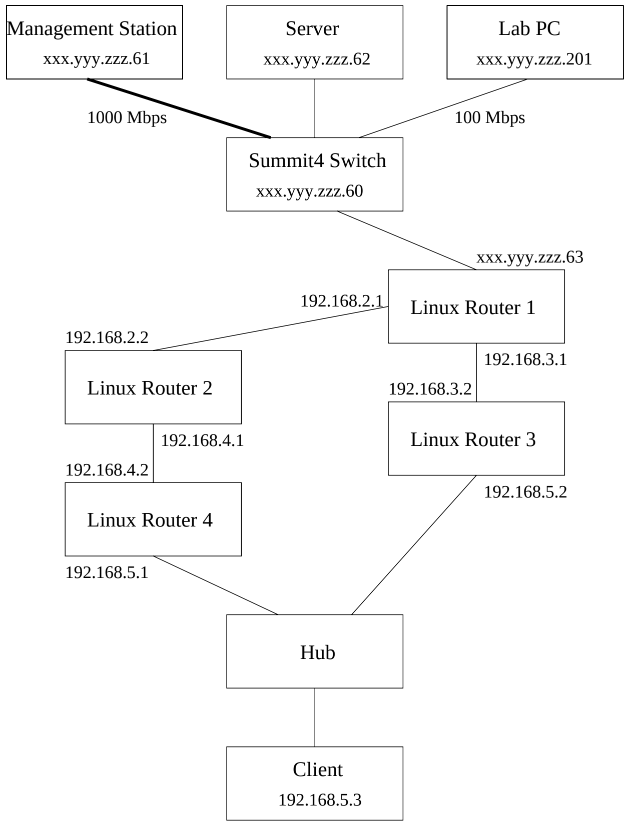

Figure 1: Example configuration of the QoS testbed

routing, support of a wide range of standardized protocols, load sharing, definition of VLANs, and convenient configuration and management software. In addition, the Summit4 has interesting provisions for QoS [12] that make it well suited for our testbed.

QoS mechanisms in the Summit4 are based on QoS profiles and traffic classification. A QoS profile consists of the name of the profile, a minimum and a maximum bandwidth, and a priority. The minimum and maximum values are percentages of the available link bandwidth (i.e., 100 or 1000 Mbps). Thus, the data transfer rate provided can be adjusted in increments of 1 Mbps on a 100 Mbps port, for instance. During data transfer, the switch tries to assure the minimum bandwidth and limits the traffic to the given maximum. The priority of the profile (low, normal, medium, or high) corresponds to one of the four separate hardware queues which are available for every port on the Summit4.

Traffic can be classified according to a number of criteria, including IP source and destination addresses, TCP and UDP source and destination ports, as well as packet priority (IEEE 802.1p), physical source port, MAC addresses and VLAN names. For our purposes, IP information-based classification and QoS mechanisms are the most important. Traffic categories defined using these criteria are finally mapped to QoS profiles.

The following simple example illustrates QoS profile specification and IP address-based traffic classification. The commands modify the QoS profile QP3 of a given (1000 Mbps) port in such a way that the minimum bandwidth equals 1% and the maximum bandwidth 2% of the link bandwidth and the priority becomes medium. Furthermore, traffic originating from the IP address xxx.yyy.zzz.61 (the MS in our testbed) is shaped according to profile QP3, thus restricted to 20 Mbps:

```
config qosprofile QP3 minbw 1% maxbw 2%
  priority medium
config ipqos add UDP xxx.yyy.zzz.61 / 32 QP3
config ipqos add TCP xxx.yyy.zzz.61 / 32 QP3
config ipqos add OTH xxx.yyy.zzz.61 / 32 QP3
```

Linux Routers

Recent Linux kernels and additional user-space programs provide IP routing, network traffic control, and DiffServ facilities. The LRs in our testbed run kernel version 2.2.14 and the `iproute` programs provided by [13]. A major challenge in employing this software is that its documentation is still preliminary and incomplete. A more complete picture and documentation of the Linux network traffic control and QoS software is given by [1, 2]. The source code and configuration information of a prototype DiffServ package built atop the traffic control code, are available there as well.

Linux network traffic control operates on the output side of the path which network packets take through a Linux node (router), i.e., at the point where packets are queued on the output interface. Among other things, traffic control can decide whether packets are queued or dropped, can reorder packets according to their priorities or delay them to limit the rate of the outgoing traffic.

The Linux QoS model given by this traffic control code consists of queuing disciplines, classes, and filters. Each network device has a queuing discipline associated with it. The queuing discipline is the fundamental QoS concept under Linux because it generally controls how packets enqueued on the device are treated, e.g., limited to a specified maximum data rate using a token bucket filter (TBF) or simply sent in FIFO order.

A queuing discipline may contain one or more classes. Each class facilitates different treatment for the data flow; for instance, one class might be given priority over the others, another class could limit the data transfer rate to 5 Mbps. Classes in turn may make use of queuing disciplines to take care of storing the packets in the output queues; see [1, 2] and the example given below.

The building blocks that differentiate traffic into classes are called filters. Filters can be combined arbitrarily with queuing disciplines and classes; multiple filters may map traffic to the same class.

When QoS is based on IP information, as in our testbed, filters are usually supported by rules that map traffic e.g. with given IP source and destination addresses to so-called "realms". These are further mapped to classes by the filters, as shown in the example below.

The example illustrates how the entire traffic from/to the server of the testbed (node xxx.yyy.zzz.62) and the additional lab PC (node xxx.yyy.zzz.201) traversing the network interface *eth2* (address 192.168.3.1) of LR1 is limited to 128 kbps:

```
# Queuing discipline (QD) 1 with CBQ
tc qdisc add dev eth2 root handle 1:
  cbq bandwidth 100Mbit
# Class 1 associated with QD 1
tc class add dev eth2 parent 1:0 classid 1:1
  cbq bandwidth 100Mbit rate 100Mbit
# Class 5 with rate 128 kbps
tc class add dev eth2 parent 1:1 classid 1:5
  cbq bandwidth 100Mbit rate 128kbit
# New QD used by class 5 with 128-kbps TBF
tc qdisc add dev eth2 parent 1:5
  tbf rate 128kbit buffer 10Kb/8 limit 15Kb
# Filter to map traffic 'realm' 1 to class 5
tc filter add dev eth2 parent 1:0 protocol ip
  prio 10 route to 1 classid 1:5
# IP rules to map traffic to 'realm' 1
ip rule add from xxx.yyy.zzz.62/32 realms 1/1
ip rule add to xxx.yyy.zzz.62/32 realms 1/1
ip rule add from xxx.yyy.zzz.201/32 realms 1/1
ip rule add to xxx.yyy.zzz.201/32 realms 1/1
```

4 Performance

A number of performance experiments were performed to measure the raw performance, the QoS overhead, and the efficacy of the QoS mechanisms.

The raw performance was assessed using a microbenchmark that floods packets over a TCP connection from the server to the client. Throughput is measured using a one-way version of this program, while round-trip delay is measured with a ping-pong variant of the program; i.e., the server waits for the reply packet from the client before sending the next one.

Throughput

Figure 2 shows the throughput results of a TCP connection along the path server–Summit4–LR1–LR3–client, under different traffic and QoS conditions.

The curve 'QoS mechanisms' denotes the experiment *with* QoS mechanisms being active in the LRs and *without* competing traffic. Throughput of the server–client data stream achieves up to about 79 Mbps, which is remarkable for software routing on a (slightly dated) PC and packets smaller than 1 kByte. The sawtooth shape of this curve (as well as of the other curves) is attributed to TCP buffering effects; with the `TCP_NODELAY` option, the curve becomes smoother, but achieves a maximum throughput of about 70 Mbps only.

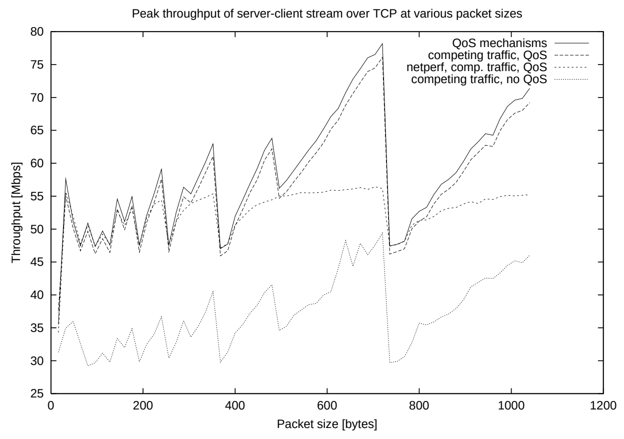

Figure 2: Throughput of server–client stream over TCP, *without/with* QoS mechanisms in the LRs enabled and *without/with* competing traffic

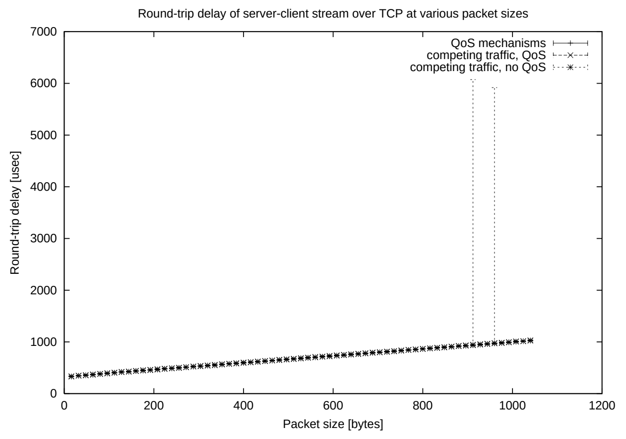

Figure 3: Round-trip delay of server–client stream over TCP, *without/with* QoS mechanisms in the LRs enabled and *without/with* competing traffic

The specific QoS mechanisms used in these experiments comprise TBFs in LR1 and in LR3 that shape the competing traffic (if existing) down to 1 Mbps. The throughput curves *without* the QoS mechanisms being enabled in the LRs are almost identical to those *with* QoS; the former are therefore not shown in the figure. We determined the QoS overhead in these experiments to be less than 1%.

The curve 'competing traffic, no QoS' shows how the throughput on the server–client path degrades when other traffic emerges on the network. In this case, 20 Mbps additional traffic flows along the path MS–Summit4–LR1–LR3–client. This traffic is generated by a large file transfer from the MS to the client and is restricted to a rate of 20 Mbps by the Summit4 switch; see the Summit4 example in Section 3. The figure reveals that the server–client data stream is degraded by roughly 20 Mbps if the QoS provisions in the LRs are inactive.

The picture changes when the QoS mechanisms in LR1 and LR3 (the 1 Mbps TBFs for the competing traffic) are enabled. The results are depicted by the curve 'competing traffic, QoS' in Figure 2. The throughput on the server–client path is raised close to the original level.

For comparison purposes, we ran the well-known `netperf` benchmark. With competing traffic and QoS enabled, throughput achieved a peak of about 56 Mbps, as also shown in Figure 2. The differences to our results are primarily attributed to the different benchmarking approaches; `netperf` does more extensive measurements, including overheads that our microbenchmark does not consider.

Delays

We have performed a similar series of experiments to assess the round-trip delays as well as the connection setup and teardown delays. The results confirm the efficacy of the traffic control (QoS) software mechanisms, similar to the throughput results reported above.

As an example, Figure 3 shows the round-trip delay results corresponding to the above experiments. In rare cases (not reproducible in a deterministic way), extreme delays occurred in the competing-traffic, no-QoS scenario, as depicted in the figure; these anomalies were not observed with QoS mechanisms in place.

5 Video Streaming Experiment

The usefulness of the QoS network testbed for multimedia communication is demonstrated in the following experiment.

Experimental Setup

We use the MPEG video/audio player described in [6] and available from [5] to stream video data from the server to the client. Although this player is rather dated (it uses MPEG-1 video still, for example), it is useful for QoS experiments since the video server and the audio server can be located on different machines and deliver their data over different connections to the client. Thus, QoS control can be selectively applied to the A and V streams, with potentially interesting results.

The experiment reported here focuses on streaming an MPEG-1 video from the server to the client via LR1 and LR3. A small video, with 128x96 resolution, an average of 586 bytes/frame, and a rate of 141 kbps is used (about 6 minutes play time); the default frame sequence is IBBPBBPBBPBB.

The video stream may be impeded by competing traffic (a large file transfer, i.e. FTP) flowing from the lab PC to LR3, thus taking the same route across LR1. In addition, both streams may be subject to a bandwidth limitation imposed by LR1 and their QoS may be controlled by LR1.

As described in detail in [6], the video player uses a sophisticated software feedback mechanism to smooth the presentation of the video. The user may specify a desired display frame rate, which is 30 fps by default. If the network between server and client (player) cannot provide enough effective bandwidth to support the desired display frame rate, frames will arrive too late and will have to be dropped by the player. The client feeds back this information (i.e., the QoS delivered by the network and observed by the player) to the server.

The video server may then "intelligently" drop excess frames at the source to adapt the stream to the network QoS. That is, the server does not even retrieve excess frames from storage and tries to space dropped frames evenly throughout the video stream. For instance, every other B frame could be skipped if the delivered network QoS is close to the QoS desired by the user (the display frame rate). If the observed QoS is far below the desired QoS, the player may (linearly) decrease the display frame rate and feed back its updated requirements (and the observed network QoS) to the server.

This technique iteratively adjusts the display frame rate to the constraint imposed by the effective network bandwidth, while maintaining a smooth playout of the video. Random frame dropping at the bottleneck in the video delivery pipeline is thus avoided. The feedback mechanism however is slightly conservative in that it does not fully exploit the effective network bandwidth (in equilibrium).

We performed a number of video streaming experiments without and with competing traffic (FTP). A fundamental bandwidth limitation of 128 kbps was imposed on the link LR1–LR3 by a TBF in LR1; see the Linux Router example in Section 3. Different scenarios were obtained by varying the bandwidths granted to the competing flows on this 128 kbps link. The default feedback interval of 2000 ms was used for the server–player interaction.

Results

The results differ widely in terms of perceptual quality of the video presentations. For example, the video is played out at the client with acceptable quality if the full 128 kbps are available to the video stream. If competing traffic from the lab PC to LR3 is injected that consumes part of the 128 kbps, video presentation quality degrades significantly, down to a few frames per second.

To assess the results in quantitative terms, the adjusted display frame rate (resulting from the client-server feedback mechanism) is used as a metric. Table 1 shows some of the results. Each adjusted display frame rate value shown in the table represents the median of five video presentations.

If the full 128 kbps bandwidth is available to the 141 kbps video stream, the display frame rate is adjusted to 19.5 fps. This is acceptable in terms of visual quality but seems to leave some of the link bandwidth unused. In fact, the average effective bandwidth consumed by the video stream in this case is about 116 kbps only; the resulting frame pattern is IB-PB-PB-PB-.

Table 1: Adjusted display frame rate for video stream over 128 kbps link (LR1–LR3), *without/with* competing traffic (FTP)

Available bandwidth, traffic classes in LR1 (video : FTP)	Video traffic only	Video *and* FTP traffic
128 kbps (no QoS)	19.5 fps	3.5 fps
28 : 100 kbps	–	3 fps
64 : 64 kbps	–	7 fps
100 : 28 kbps	–	13.5 fps

When the competing FTP traffic emerges on the unregulated 128 kbps link, the display frame rate decreases to a final 3.5 fps, due to the (conservative) feedback technique. The FTP traffic acquires as much as 90 kbps bandwidth, leaving the video only about 38 kbps; almost all non-I frames are skipped in the video stream.

Disabling the client–server software feedback does not improve the situation, since most of the frames arrive too late and have to be dropped by the video decoder or the display controller; the frame rate severely degrades to <1 fps.

In order to enhance the visual quality of the video session, the QoS mechanisms (traffic control) in LR1 were employed to guarantee bandwidth for the video stream. The results are shown in the last two rows of Table 1 where the two data streams were separated into different traffic classes. With a guaranteed 64 kbps and 100 kbps bandwidth, the video plays out at an adjusted frame rate of 7 fps and 13.5 fps, respectively. The second row confirms the result of the unregulated, congested 128 kbps link case above.

6 Related Work

The Drexel University Network Testbed (DNT) [14] is most closely related to the QoS testbed presented in this paper, both in terms of its scale, motivation, and the tools used. DNT may also be used for educational purposes and serve as a vehicle for inexpensive and quick experiments with various real-world networking scenarios, including e.g. satellite-based IP communication (which requires to "emulate" long packet transmission delays). Whereas DNT uses the more complete DiffServ prototype of [1, 2], our testbed also includes a QoS-capable Ethernet switch, providing for higher networking performance if required as well as the opportunity for students to get hands-on experience with a modern QoS-configurable network device.

On a much larger scale, the QBone [15] is a networking testbed for Internet2 QoS technology, with an initial focus on the DiffServ architecture. While QBone and our testbed do not have much in common, QBone tools like e.g. traffic generators (if publicly available) could be adopted to enrich our infrastructure.

7 Conclusions

We have described a simple, but versatile cluster-based QoS networking testbed that can be employed to "emulate" different networks for multimedia communication experiments. The network was built using standard PC and Ethernet hardware and open-source software components, i.e., IP routing and traffic control available in recent Linux kernels and provided as separate packages. The testbed can be flexibly configured to model various link bandwidths as well as IP routers capable of classifying, queuing (with various disciplines), forwarding and/or dropping packets and shaping traffic. We have performed initial performance analysis experiments and results for the testbed and demonstrated its usefulness in a simple video streaming application.

We plan to use the testbed in a project on quality-adaptive video transfer over heterogeneous networks, where the video flow is subject to adaptation by network nodes (routers), depending e.g. on the bandwidth available on an outbound link or the current load on the router. Video adaptation in this sense may mean that low-priority enhancement layers of a layer-encoded video may be dropped by the routers, whereas the base layer needs to be delivered with high quality. The QoS mechanisms and versatility of such a testbed will be important for closely modeling real-world multimedia networking scenarios.

References

[1] W. Almesberger. Linux Network Traffic Control – Implementation Overview. Technical report, EPFL ICA, April 1999. http://icawww1.epfl.ch/linux-diffserv/.

[2] W. Almesberger, J. H. Salim, and A. Kuznetsov. Differentiated Services on Linux. Technical report, EPFL ICA, June 1999. http://icawww1.epfl.ch/linux-diffserv/.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services (RFC 2475), December 1998.

[4] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture (RFC 1633), June 1994.

[5] S. Cen. Internet-Based Distributed Real-Time MPEG Video Audio Player Version 2.0. Distributed Systems Research Group, Dept. of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, Portland, Oregon. http://www.cse.ogi.edu/DISC/projects/synthetix/Player/, June 1998.

[6] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A Distributed Real-Time MPEG Video Audio Player. In *Proceedings 5th Int'l. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95))*, November 1995.

[7] R. Braden (ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification (RFC 2205), September 1997.

[8] IETF DiffServ Working Group. Differentiated Services (diffserv) Charter. http://www.ietf.org/html.charters/diffserv-charter.html, 2001.

[9] IETF IntServ Working Group. Integrated Services (intserv) Charter. http://www.ietf.org/html.charters/intserv-charter.html, 2000.

[10] IETF MPLS Working Group. Multiprotocol Label Switching (mpls) Charter. http://www.ietf.org/html.charters/mpls-charter.html, 2001.

[11] IETF RSVP Working Group. Resource Reservation Setup Protocol (rsvp) Charter. http://www.ietf.org/html.charters/rsvp-charter.html, 2001.

[12] Extreme Networks Inc. ExtremeWare Software User Guide. Software Version 6.1. http://www.extremenetworks.com/support/documentation/ExtremeWare6_1.zip, April 2000.

[13] A. Kuznetsov. IP Command Reference. Institute for Nuclear Research, Moscow. ftp://ftp.inr.ac.ru/ip-routing/iproute2.current.tar.gz, 1999.

[14] D.T. McWherter, J. Sevy, and W.C. Regli. Building an IP Network Quality-of-Service Testbed. *IEEE Internet Computing*, 4(4):65–73, August 2000.

[15] QBone Website, May 2001. http://qbone.internet2.edu/.

[16] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture (RFC 3031), January 2001.

[17] X. Xiao and L.M. Ni. Internet QoS: A Big Picture. *IEEE Network*, 13(2):8–18, 1999.

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 5:8–18, September 1993.