A QoS Network Management System for Robust and Reliable Multimedia Services

S. Das, K. Yamada, H. Yu, S. S. Lee, and M. Gerla

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095-1596
{shanky, kenshin, heeyeoly, sslee, gerla}@cs.ucla.edu

Abstract. In this paper, we introduce a practical network management system with QoS for multimedia traffic. Today, multimedia applications have evolved significantly and have become an essential part of the Internet. In order to effectively support the newly emerging network traffic, the underlying network protocols need to be aware of the characteristics and demands of the traffic. The proposed network system is ready to address the varied characteristics of multimedia traffic and can assure a high degree of adherence to the quality of service demanded from it. Considering the need for reliable QoS services, the system is also equipped with fault tolerance capability by provisioning multiple QoS paths. Moreover, the system has been practically designed and implemented to provide "cost-effective" QoS support with respect to control overhead. It deploys measurement-based QoS path computation and call admission scheme which may be deemed ineffective for bursty multimedia traffic. However, results conclusively prove that provisioning multiple paths and utilizing them in parallel in our system not only provides high fault tolerance capability but also effectively accommodates multimedia traffic by relieving its burstiness with multiple paths. We present the architecture of the system and discuss the benefits gained for multimedia traffic.

1 Introduction

Recently, the Internet has become a significant medium for real-time data as demands for multimedia services become more and more pervasive. In order to accommodate such new types of traffic in the Internet, researchers have addressed various issues with respect to provisioning QoS. QoS provisioning issues consist of several building blocks such as *resource assurance*, *service differentiation*, and *routing strategies*. Among these, QoS routing as a routing strategy is essential since it executes computations to find QoS-feasible paths. [1,2] address a possible use of the Bellman-Ford algorithm in the link state (OSPF) routing environment for QoS routing. In addition, [3,4] showed specifications and feasibility of such QoS routing approaches. The development of the Multiprotocol Label Switching Protocol (MPLS) [5] and the increased acceptance of Traffic Engineering as an important component of routing protocols has accelerated the coming of age of QoS routing.

The original version of this chapter was revised: The copyright line was incorrect.This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-45812-8_28

K.C. Almeroth and M. Hasan (Eds.): MMNS 2002, LNCS 2496, pp. 1–11, 2002.

In this paper, we present a practical QoS network management system based on the proposed OSPF-based QoS architecture. This consists of OSPF with traffic engineering extensions running as the routing protocol, MPLS for explicitly routing packets, and a QoS path provisioning algorithm that also serves as a fundamental call admission control (CAC). The proposed network system is equipped with fault tolerance capabilities for unreliable network environment. This fault tolerance issue in QoS provisioning is considered as a very important aspect especially when mission-critical applications are running. This issue has been highlighted in [6,7]. Accordingly, a novel approach was recently introduced in [8] which deploys an effective algorithm to produce multiple QoS paths. Multiple QoS path provisioning also showed various benefits such as even network resource utilization and cost-effective link state information acquisition [9]. In addition, our proposed system in this paper provides statistical QoS guarantees. That is, it deploys measurement-based path computations and call admissions without any resource reservations. This helps in reducing the system complexity and achieving relatively fast QoS provisioning. It is well known that multimedia applications are likely to produce highly bursty traffic with hardly predictable characteristics [10,11]. This high burstiness keeps measurement-based approaches from properly performing. This is because the severe traffic fluctuation misleads the approaches into producing non-feasible QoS paths and making incorrect call admission decisions. The system that we propose carries out rather reliable call admissions and performs well since it provisions multiple QoS paths and spreads network traffic over them. Thus, it diffuses traffic burstiness and attenuates its negative impact.

Targeting such a fault-tolerant and reliable QoS-service architecture, we have implemented a QoS testbed. In this paper, we present the system architecture with the QoS routing approach for multiple path and show the fault tolerance capability with reliable multimedia services. In Section 2, the routing algorithms are briefly reviewed. Section 3 depicts the entire network system architecture, and Section 4 presents experiment results obtained with the implemented network system and demonstrates its effectiveness in representative traffic scenarios.

2 QoS Algorithms

The network system presented in this paper has QoS routing mechanisms which are ready to serve QoS applications in both conventional and fault-tolerant ways. The mechanisms serve as a simple call admission control (CAC). When a QoS application comes in and looks for its corresponding QoS services, it consults the underlying QoS routing algorithm (i.e., Q-OSPF with the enhanced routing algorithms in our case) for feasible paths. If no feasible path is found, the connection is rejected and the application exits. Thus, the QoS routing path computation algorithm not only provides the capability of finding QoS paths but also plays an important role in CAC. As previously mentioned and discussed in [8], we adopted the two different QoS routing algorithms in our system; the conventional single path algorithm and the newly introduced multiple path algorithm.

The single QoS path computation algorithm with multiple QoS constraints derives from the conventional Bellman-Ford algorithm as a breadth-first search algorithm minimizing the hop count and yet satisfying multiple QoS constraints. Each node in the network builds the link state database which contains all the recent link state advertisements from other nodes. With Q-OSPF, the topological database captures dynamically changing QoS information. The link state database accommodates all the QoS conditions, and we define each condition as a QoS metric and each link in the network is assumed to be associated with multiple QoS metrics which are properly measured and flooded by each node. Each of these QoS metrics has its own properties when operated upon in the path computation. The principal purpose of the path computation algorithm is to find the shortest (i.e., min-hop) path among those which have enough resources to satisfy given multiple QoS constraints, rather than the shortest path with respect to another cost metric (e.g., maximizing available bandwidth or minimizing end-to-end delay).

The proposed multiple QoS path algorithm is a heuristic solution. We do not limit ourselves to strictly "path disjoint" solutions. Rather, the algorithm searches for multiple, maximally disjoint paths (i.e., with the least overlap among each other) such that the failure of a link in any of the paths will still leave (with high probability) one or more of the other paths operational. The multiple path computation algorithm can then be derived from the single path computation algorithm with simple modifications. This multiple path computation algorithm produces incrementally a single path at each iteration rather than multiple paths at once. All the previously generated paths are kept into account in the next path computation. The detailed descriptions of the single and the multiple path algorithms are in [8].

3 System Architecture

The testbed consists of PCs running Linux, and all the QoS-capable features are embedded in the Linux kernel. Each of the machines has several modules running on it, namely the *link emulator, metric collector, OSPF daemon, MPLS forwarding* and the *applications*, and the entire system architecture is depicted in Fig. 1. The following sections describe in more detail each of these modules individually, explaining their functions and their implementation issues.

3.1 Link Emulator

The testbed that we implement uses the wired Ethernet LAN in the lab, as the physical communication medium. Since we want to emulate several scenarios with widely varying link capacities and propagation delays, we require a network emulator to do the job for us. We emulate the characteristics of bandwidth and delay over a link using `tc`, the Linux kernel traffic shaper and controller. `tc` can emulate a wide variety of policies with hierarchies of filters and class based queues on the outgoing interfaces. We require simple bandwidth and delay emulation

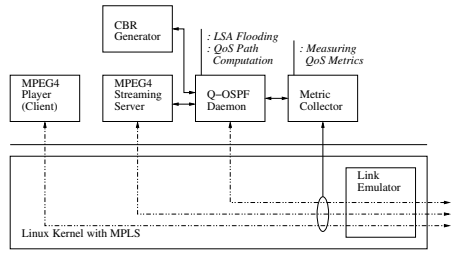

Fig. 1. The entire QoS system architecture for the experiment testbed.

which can be done quite easily. A simple "`tc qdisc add dev eth0 root tbf rate 10kbps latency 10ms`" emulates a 10 Kbps link with a latency of 10 ms on the `eth0` interface. Since we have bursty multimedia traffic, the buffer size is also an important factor, and after a few experiments, we settled on a value of 25 Kbits as a reasonable buffer to accommodate burstiness.

3.2 Metric Collector

To provide QoS, reserving bandwidth and bounding on delay for a connection, we require the knowledge of link characteristics at all times. This information is needed by the QoS allocation module which looks at the current usage of the network and figures out if the new request can be satisfied or not. The link metric collection module, therefore, is an integral part of any routing scheme implementation that provides QoS routing.

The OSPFD (OSPF daemon) implementation does not have any link characteristics measurement module. The metrics we are interested in are the available bandwidth in the link and the delay. One could think of other metrics also, such as queue length, loss probability, etc., but bandwidth and delay are the two most important metrics used in QoS routing. Thus, our design goal was to write a module to measure these two metrics and integrate this code with the existing OSPFD implementation.

For bandwidth metric collection, we opted to simply use the log file maintained in `/proc/net/dev/` which contains information about the number of packets and bytes received and transmitted on each interface. By examining this file at regular intervals, we calculate the bandwidth used on the each outgoing interface. Delay metric collection is done using the 'ping' utility to send ping probes to the other side of the link and collect the delay value. The metric collection code sends ping message and collects bandwidth values from the `/proc/net/dev` log file, every time interval (typically 10 ms). The values collected are exponentially averaged to smooth out the fluctuations.

3.3 Q-OSPF Daemon

To propagate QoS metrics among all routers in the domain, we need to use an Interior Gateway Protocol (IGP). OSPF is one of the major IGPs and significant

researches have been recently made on OSPF with traffic engineering extensions. We selected the open source OSPF daemon (OSPFD) [12,13] to implement our QoS routing scheme. [14] defines Opaque LSA for OSPF nodes to distribute user-specific information. Likewise, we define our specific Opaque LSA entries by assigning new type values in the Opaque LSA format shown in Fig. 2.

When OSPFD runs at routers, it tries to find its neighbor nodes by sending HELLO messages. After establishing neighbor relationship, OSPFD asks the metric measurement module to calculate the QoS metrics of the established link. OSPFD gives as input the local interface address and the neighbor router interface address to the metric measurement module and generates the opaque LSA for each interface. The LSA contains the available bandwidth and queuing delay metric obtained from the metric measurement module. LSA update frequency is currently restricted to MinLSInterval (5 seconds). In our implementation, we followed this restriction, but we feel that the LSA update interval is an important parameter for the correctness of our algorithms because if there are too frequent changes to the generated LSA, there will be miscalculations during the QoS path computation due to the flooding delay.

In addition to LSA flooding, OSPFD exchanges router LSAs to build a full network topology. Router LSAs originate at each router and contain information about all router links such as interface addresses and neighbor addresses. We bind the link metrics that we are interested in, viz. bandwidth and delay to the opaque LSA specified by the link interface address. Thus, we introduce the pointer from each link of a router LSA to the corresponding opaque LSA as shown in Fig. 3. Whenever OSPFD receives router LSAs or opaque LSAs, it just updates the link pointer to point to the new traffic metrics reported by that link.

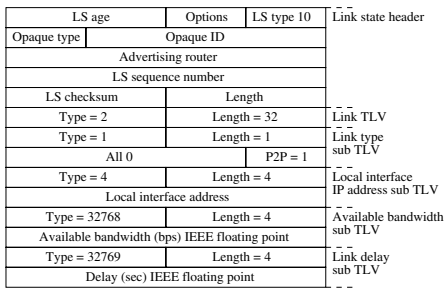

Fig. 2. Opaque LSA format.

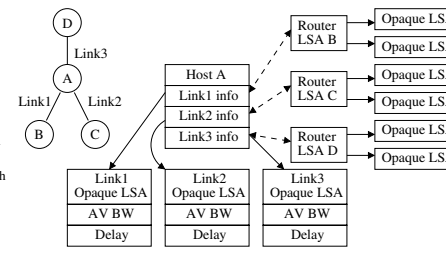

Fig. 3. Router LSAs.

3.4 MPLS

One of the key assumptions for Q-OSPF to be effective is the capability of setting up explicit paths for all packets of a stream to use. Thus, we need to pin down the

path that a connection uses. One of the main advantages of MPLS is its efficient support of explicit routing through the use of Label Switched Paths (LSPs). With destination-based forwarding as in the conventional datagram networks, explicit routing is usually provided by attaching to each packet the network-layer address of each node along the explicit path. This approach makes the overhead in the packet prohibitively expensive. This was the primary motivation behind deploying MPLS at the lower layer instead of using something like IP source-routing or application level forwarding.

3.5 Applications

An integral part to the whole process of making a testbed was the development of applications running on top of this architecture. These applications request QoS-constrained routes, generate traffic, and in turn change the QoS metrics of the network. We used the open source *Darwin Streaming Server* distributed by Apple [15] to stream MPEG-4 files using RTSP over RTP. The open source `mp4player` from the MPEG4IP project [16] was used to play these files over the network. Modifications to the source code involved extending the capabilities of the server and the player for streaming and playing over multiple paths, and designing an interface for the server to interact with Q-OSPF daemon and request paths to the client. For the experiments involving CBR traffic, we used a home-brewed traffic generator which generates UDP traffic at a given rate in either single path or multiple path mode.

4 Experiments

We performed two kinds of experiments for this paper. The first set of experiments involves comparing the performance of Q-OSPF with respect to the throughput of multimedia traffic using single paths and using multiple paths. The second set involves comparing the performance of Q-OSPF with respect to fault tolerance and robustness of multimedia connections using single path and using multiple path. Fig. 4 shows the network topology for the experiments. 9 nodes (i.e., routers) are connected directly to each other through 1.5 Mbps links.

We performed experiments in certain representative scenarios where multiple path is expected to give better results as well as in totally general scenarios.

1. The first experiment involved sending six streams of multimedia traffic from QOS6 to QOS5. There are three disjoint paths available that satisfy the bandwidth constraints, QOS6 - QOS2 - QOS1 - QOS5, QOS6 - QOS7 - QOS4 - QOS5, QOS6 - QOS9 - QOS8 - QOS5. Let us name them *QPath1*, *QPath2* and *QPath3* respectively. We run the experiment in both *Single Path* and *Multi Path* mode. In single path mode, each stream uses one of the three QPaths, while in multi path mode, each stream uses all the three QPaths. Results are in Fig. 5 and Fig. 6.

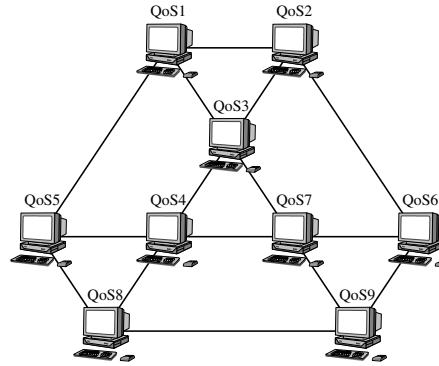

Fig. 4. The QoS testbed topology

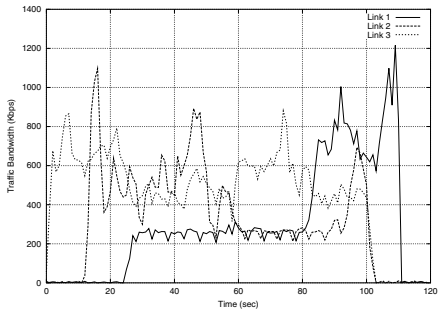

Fig. 5. Throughput profile on the three QPaths: single path

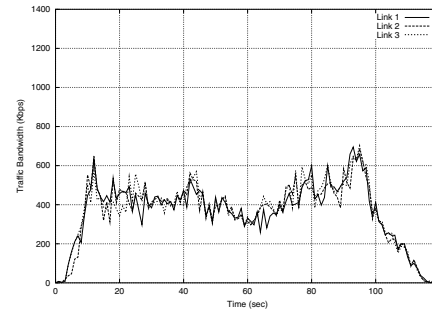

Fig. 6. Throughput profile on the three QPaths: multi path

2. The second experiment involved sending three CBR streams of 800 Kbps first on the three QPaths, and then introducing the six multimedia streams later. In the single path case, the calls get accepted and follow the same paths as before, two streams each sharing one QPath, but in this case due to burstiness, the total throughput on each path frequently goes above the 1.5 Mbps limit, and thus there are losses. One the other hand, with multi path mode, the traffic on all the paths is much smoother, and thus the total traffic never goes above 1.5 Mbps on any link and thus there are no losses. This scenario demonstrates the benefits on multi path with respect to smoothening the bursty traffic. Results are in Fig. 7 and Fig. 8.
3. The last experiment in this set was a totally general scenario, in which random connections were set up, and the traffic characteristics were measured. We first perform the experiment for well behaved CBR traffic in the single path mode and then perform it with bursty multimedia traffic first using single path mode and then multiple path.

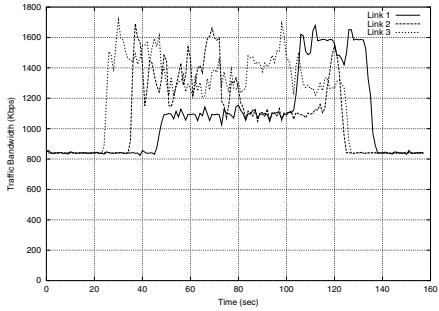

Fig. 7. Throughput profile on the three QPaths: single path

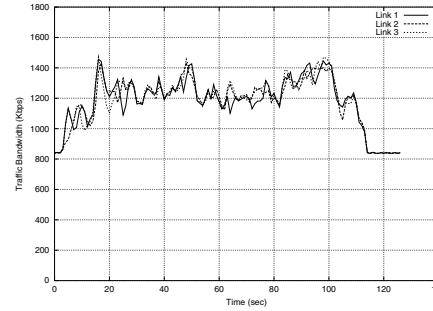

Fig. 8. Throughput profile on the three QPaths: multi path

Owing to performance bounds of the multimedia server and the CPUs of the nodes, we reduced the bandwidth emulated on each link to 1 Mbps. The total number of connections generated in both CBR and multimedia case was 28. These connections were generated using randomly chosen source and destination from the 9 QoS routers. The measured traffic characteristics are the usual throughput profile on the links. Fig. 9 shows the throughput characteristics on each link when CBR traffic is in the network and Q-OSPF is in single path mode. Fig. 10 shows the throughput characteristics when multimedia traffic (of the same average bandwidth as the CBR in the previous experiment) is in the network and Q-OSPF is in single path mode. We can see that with multimedia the performance of Q-OSPF is visibly worse and the traffic frequently goes above the threshold of 1 Mbps. Fig. 11 shows the throughput characteristics when multiple paths are provisioned. Here we see that on each link the burstiness is much less and the overall QoS constraints are being satisfied. The throughput is below the 1 Mbps limit. Thus, we clearly see the benefits of using multiple paths even in a completely random general scenario.

The second set of experiments examines the fault tolerance capability by provisioning multiple QoS paths between source and destination and spreading packets over the multiple paths. The network topology is the same as in Fig. 4. Here again we experiment with two extreme traffic scenarios, well behaved CBR traffic and bursty multimedia traffic. The pairs of source and destination nodes for the connections are randomly selected, and the link failures occur also randomly during the execution of the experiments. For the link failure model, we used a exponential distribution such that the average link down time is equal to 5 seconds. The number of multiple paths requested is kept at 3. The statistics which we collect during this experiment is the amount of time for which each connection is down. A connection is said to be down when none of its packets are reaching the destination for 2 seconds.

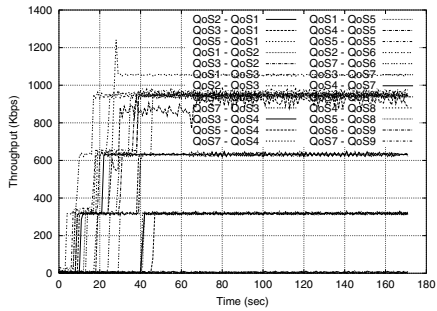

Fig. 9. Throughput profile on every link: CBR: Single Path

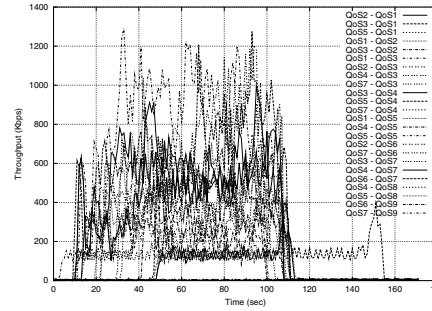

Fig. 10. Throughput profile on every link: Multimedia: Single Path

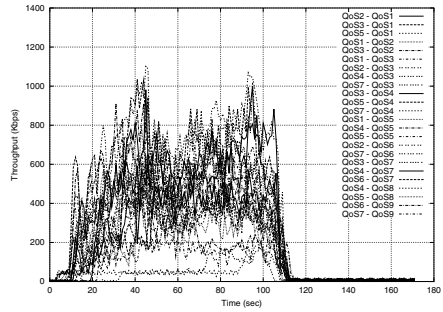

Fig. 11. Throughput profile on every link: Multimedia: Multi Path

The duration of the experiments was 10 minutes each. Fig. 12 and Fig. 13 show the connection down-time percentage when single path or multiple paths are provisioned for the connections for both the CBR traffic case and the multimedia case. The result shows that multiple path provisioning has lower connection down rate as expected. We can see that the benefits although almost equal are slightly lesser than in the CBR traffic case. The reason is that a lot of multimedia calls do not get 3 multiple paths due to the burstiness of existing connections which fill up the pipe. Thus, we show that we have greater fault tolerance for multimedia traffic when we provision multiple QoS paths compared to single QoS path. Note that the experiments are slightly different in spirit from the simulation experiments in [8]. In [8], the authors assume that a connection gets dropped when all its paths have one or more faults, and does not come up automatically when the fault gets repaired. However, the experiments have enough similarity to give similar results.

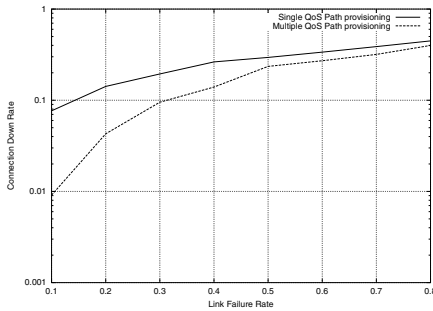

Fig. 12. Connection down rate as a function of link failure rates: CBR

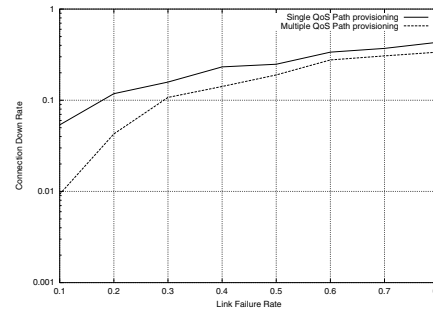

Fig. 13. Connection down rate as a function of link failure rates: Multimedia

5 Conclusion

We proposed a QoS network management system in which QoS provisioning mechanisms are measurement-based and multiple QoS paths are computed. With the practical system implementation and experiments, we discussed the interaction of measurement-based call admission and bursty multimedia traffic. The experiment results verified that the bursty nature of multimedia traffic degrades the performance of measurement-based approaches. However, when multiple QoS paths are provisioned, the burstiness becomes diffused and its negative impact is lessened as the results showed. We also evaluated the fault tolerance capability of the network and found that the multiple paths provide a high degree of robustness in servicing QoS applications. In conclusion, provisioning multiple paths during call admission not only attenuates the impact of the burstiness but also provides highly robust QoS services.

References

1. R. Guerin, A. Orda, and D. Williams. QoS Routing Mechanisms and OSPF Extensions. In *Proc. of Global Internet (Globecom)*, Phoenix, Arizona, November 1997. 1
2. Dirceu Cavendish and Mario Gerla. Internet QoS Routing using the Bellman-Ford Algorithm. In *IFIP Conference on High Performance Networking*, 1998. 1
3. G. Apostolopoulos, S. Kama, D. Williams, R. Guerin, A. Orda, and T. Przygienda. QoS Routing Mechanisms and OSPF Extensions. Request for Comments 2676, Internet Engineering Task Force, August 1999. 1
4. Alex Dubrovsky, Mario Gerla, Scott Seongwook Lee, and Dirceu Cavendish. Internet QoS Routing with IP Telephony and TCP Traffic. In *Proc. of ICC*, June 2000. 1
5. E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. Request for Comments 3031, Internet Engineering Task Force, January 2001. 1

6. Shigang Chen and Klara Nahrstedt. An Overview of Quality of Service Routing for Next-Generation High-Speed Networks: Problems and Solutions. 12(6):64–79, November 1998. 2
7. Henning Schulzrinne. Keynote: Quality of Service - 20 Years Old and Ready to Get a Job? *Lecture Notes in Computer Science*, 2092:1, June 2001. International Workshop on Quality of Service (IWQoS). 2
8. Scott Seongwook Lee and Mario Gerla. Fault Tolerance and Load Balancing in QoS Provisioning with Multiple MPLS Paths. *Lecture Notes in Computer Science*, 2092:155–, 2001. International Workshop on Quality of Service (IWQoS). 2, 2, 3, 9, 9
9. Scott Seongwook Lee and Giovanni Pau. Hierarchical Approach for Low Cost and Fast QoS Provisioning. In *Proc. of IEEE Global Communications Conference (GLOBECOM)*, November 2001. 2
10. S. Floyd. Comments on Measurement-based Admissions Control for Controlled-load Services, 1996. 2
11. Lee Breslau, Sugih Jamin, and Scott Shenker. Comments on the Performance of Measurement-Based Admission Control Algorithms. In *INFOCOM (3)*, pages 1233–1242, 2000. 2
12. J. Moy. OSPF Version 2. Request for Comments 2328, Internet Engineering Task Force, April 1998. 5
13. John T. Moy. OSPFD Routing Software Resources. http://www.ospf.org. 5
14. R. Coltun. The OSPF Opaque LSA Option. Technical report. 5
15. Apple Computer. The Darwin Streaming Server. http://www.opensource.apple.com/projects/streaming. 6
16. Bill May David Mackie and Alix M. Franquet. The MPEG4-IP Project. http://mpeg4ip.sourceforge.net. 6