On the use of Flow Migration for Handling Short-term Overloads

Kuo-Tung Kuo Surapich Phuvoravan Bobby Bhattacharjee
Richard Jun La Mark Shayman Hyeong Soo Chang
University of Maryland, College Park, Maryland, USA

***Abstract*— In this work, we investigate *flow migration* as a mechanism to sustain QoS to network users during short-term overloads in the context of an MPLS IP network. We experiment with three different control techniques: static long-term optimal mapping of flows to LSPs; on-line locally optimal mapping of flows to LSPs at flow set-up time; and dynamic flow migration in response to transient congestion. These techniques are applicable over different timescales, have different run-time overheads, and require different levels of monitoring and control software inside the network. We present results both from detailed simulations and a complete implementation using software IP routers. We use Voice-over-IP as our test application, and show that if end-to-end quality is to be maintained during short unpredictable bursts of high load, then a fast-timescale control such as migration is required.**

I. Introduction

In this work, we investigate *flow migration* as a mechanism to provide QoS to users during short-term overloads. Such overloads occur when one site within the domain (or even a single document at a site) suddenly becomes hugely popular. An example of this is the so-called "slashdot-effect" that often besets hosts after they are cited by the popular Internet site `www.slashdot.org`. However, short-term overloads also occur during emergencies or during failures of different network components. As IP networks are used to support more QoS-aware applications, e.g. Voice-over-IP and media streaming, it is important to be able to maintain service guarantees even during such short-term overloads. In fact, commercial products are on the market that are intended to reroute VoIP traffic around transient congestion in order to maintain toll quality voice [1].

A fast timescale control such as flow migration will necessarily be a first-aid solution before the network is re-optimized by the slow-time scale control. Thus, in this paper, we concentrate solely on the period during the onset of congestion, and show that migration is useful in maintaining service quality for individual sessions even if high loads are introduced into the network over short intervals. Although the use of migration for dynamic load balancing is not new, to the best of our knowledge its use at a fast timescale (in our case 50 ms) has not been considered and implemented before.

In this paper, we evaluate the benefits of fast timescale control in the context of streaming media applications, specifically voice-over-IP and variable bit-rate video. We have implemented all of the MPLS, related protocols, and applications in a testbed and within a detailed packet-level simulation. Our basic approach is as follows: we set up a set of parallel Label Switched Paths (LSPs) between the ingress and egress nodes in our testbed, and run a set of VoIP and video sessions over these LSPs. After the testbed stabilizes, we introduce a perturbation, in the form of a configurable number of video sessions, onto one of the LSPs. We keep the total demand on the network below its capacity; however, the instantaneous demand on a link can be larger than its capacity temporarily due to imbalanced distribution of the load on the network.

The main contribution of this work is to show that a *fine timescale mechanism, such as flow migration, is required to maintain end-to-end QoS for individual flows during the onset of congestion*. As a part of our work, we also show that the run-time overhead of implementing flow migration, both in terms of monitoring overhead and reordered/dropped packets, is not prohibitively high. Clearly, there is an initial overhead of adding fine-grained monitoring and the flow-migration mechanisms into each network element; whether the gains from migration justify these initial costs depends on the guarantees provided by the network owner and on how often such unexpected congestion conditions occur. We believe this work is an effective first step towards evaluating both the benefits from and the cost of a fine-timescale control mechanism, such as monitoring.

The rest of this paper is organized as follows: In Section II, we present details of each of the three control mechanisms that we have implemented and tested. We describe our testbed and simulation environment in Section III, and present results from our experiments in Section IV. We conclude in Section V.

II. Control Algorithms

In this section, we briefly describe two (non-migration) algorithms – "Bernoulli Splitting" (BSplit), "Least Load Routing" (LLR) – and our migration scheme (MIG).

A. Non-migration Algorithms

Both non-migration algorithms apply a fixed admission control on arrival of a new call by assigning the calls to a set of provisioned LSPs. An accepted call will remain in a selected LSP by the admission control until it departs.

"Bernoulli Splitting" is based on an optimal off-line analysis that assumes that the average traffic rate between each ingress-egress pair is known. BSplit is a randomized policy such that each new arriving call is dispatched among LSPs according to a split rule (probability distribution). The computation of

BSplit can be obtained from, for example, Altman et al.'s work [4]. It ignores the fast timescale traffic variations and makes decisions only based on slow timescale information.

The next non-migration algorithm, "Least Load Routing", on the other hand, relies on the load information that is updated frequently capturing the fast timescale traffic variation. It employs the admission control, which is simply given as follows. Whenever a call arrives into a particular ingress node, the controller in the node finds which LSP has the least load among the LSPs that the ingress node is using. The new call is simply joined into the least loaded LSP. If there is a tie, we break this with a random assignment with equal probabilities among LSPs. Since we consider aggregate rather than per-LSP queueing, the load for an LSP is defined to be the maximum load among the links it traverses. In our case, each LSP has a single bottleneck link, and each ingress node gets the current load information by a feedback packet from the Label Switching Routers (LSRs) at the head end of the bottleneck link for each LSP that the ingress node is using.

B. Migration Algorithm

We next describe our on-line centralized migration controller that is built upon the Bernoulli Splitting controller. See Alanyali and Hajek's work [2] for an analysis of "asymptotic" optimality of LLR, BSplit, and LLR with a migration for a long term (normalized) average cost under certain assumptions of the cost function. However, it has been also shown that LLR is not necessarily optimal for finite arrival rates and BSplit has a higher overflow rate than LLR [3] in certain cases. In fact, this behavior has been observed in our experiments. It is worth mentioning that our migration controller is modular in that it can be used on top of any call-level traffic balancing scheme, e.g., LLR and BSplit.

The algorithm has three configurable parameters: the *migration threshold* and two different *safety margin*. The idea is that the controller is triggered whenever a link utilization exceeds the migration threshold; in this case, it attempts to migrate flows to reduce the utilization of the bottleneck link to the migration threshold minus the safety margin (SM_{FROM}). In doing so, it will not cause other link utilizations to exceed certain level specified by the safety margin (SM_{TO}). The algorithm is described as follows:

1) Find the link with maximum utilization that exceeds the migration threshold. If there is no such link, skip the remaining steps. If such a link exists, denote the link by j^*. Determine the set $\mathcal{L}^*$ of LSPs using link j^*.
2) For each LSP $l \in \mathcal{L}^*$, identify its alternate LSPs and calculate the aggregate of their available bandwidth, denoted by $\mathcal{B}_l$. Let y_l be the bandwidth utilized by LSP $l \in \mathcal{L}^*$ on link j^*. Determine the LSP $\tilde{l} \in \mathcal{L}^*$ that has the largest $\min\{y_l, \mathcal{B}_l\}$, i.e.,
$$\tilde{l} = \arg\max_{l \in \mathcal{L}^*} (\min\{y_l, \mathcal{B}_l\}) \ .$$
Compute the number of flows that can be accommodated by the alternate LSPs of $\tilde{l}$. Denote it by z.
3) Determine the number of flows x that need to be migrated from the LSPs in $\mathcal{L}^*$ in order to bring the load on link j^* to the threshold minus the safety margin.
4) Request the ingress LSR to migrate $\min\{x, z\}$ number of flows from LSP $\tilde{l}$ to its alternate LSPs proportional to their available bandwidth.
5) If $x > z$, remove $\tilde{l}$ from $\mathcal{L}^*$ and go back to step 2. Otherwise, go back to step 1.

Here we implicitly assumed that the algorithm terminates after reducing the utilization of the congested link to below the migration threshold minus the safety margin. However, if the network is severely congested, the algorithm may run out of the LSPs in $\mathcal{L}^*$ after exhausting all the available bandwidth on their alternate LSPs, in which case the algorithm fails to relieve the congested link below the migration threshold minus the safety margin (because it is not possible) and moves to the next possible congested link.

III. Implementation

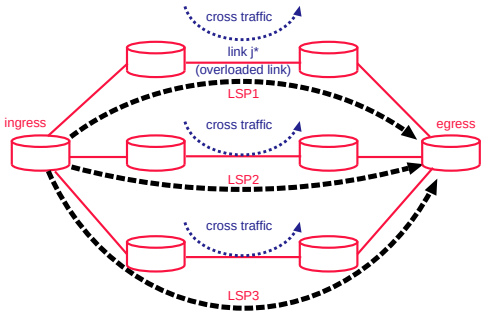

Fig. 1. Experimental Topology.

In this section, we describe our implementations and experimental setup. We have both simulated and implemented the entire range of MPLS protocols, and each of the control algorithms. We used the topology shown in Fig. 1 for both our simulations and our implementations. The capacity of the links shared by the LSPs between the ingress and egress points and the cross traffic is 90 Mbps. Although the topology we have used is simple, it is enough to demonstrate the benefits of our proposed algorithm. The three parallel LSPs in our topology allow us to understand the benefits of finer timescale migration. We begin with a description of our implementations, and then describe our traffic models and experimental scenarios.

A. Testbed Implementation

In this section, we describe our testbed, traffic generators, and different controllers. We also present details about the implementation of the migration controller. We have implemented the flow migration protocol over NISTswitch [9] software IP routers running on FreeBSD 3.3. We use RSVP-TE [5] for LSP signaling and to set QoS parameters. At each node, we use Alt-Q [6] to provide guaranteed per-packet service.

1) Monitoring: Each first-hop LSR runs a monitoring program that uses the ALTQ/CBQ [10] module to monitor the congested link. The monitoring program queries Alt-Q CBQ module for current link utilization and packet loss rate every monitoring interval. Link utilization information is sent to the

ingress LSR, and we record the packet loss information for our performance evaluation. Each feedback packet is 52 bytes; this includes 2 bytes for LSP identification, 8 bytes for current link utilization, and 42-bytes for packet headers.

2) Traffic Generators: The voice call generators and the migration controllers are all implemented within the ingress LSR. In our implementation, all three controllers are integrated into a single user-level program; the voice calls require more stringent timing control and are generated using a FreeBSD kernel thread. See section IV-A for traffic model details.

a) Voice traffic generator: The voice calls have Poisson arrival rate, and exponential call holding times. As mentioned before, we have to use kernel thread to generate voice packets in order to minimize system call overhead. Kernel thread is also required to implement the fast timescale controllers on the ingress node. The user-level program is used only to provide the kernel threads with streams of random numbers.

b) Cross traffic generator: The cross traffic generator is located at a different node not shown on in Fig. 1. It injects three independent aggregated 40 Mbps (on average) video flows to each of the LSPs. Each aggregated video flow consists of 100 (on average) random 10-minute (on average) segments of traces from the movie *Star Wars*.

3) Controller Implementations: We have implemented the three controllers within the ingress LSR. The LLR and migration controllers receive feedback messages from the the network and may schedule (or migrate) voice calls onto different LSPs. The Bernoulli Splitting controller only performs static call scheduling and ignores all feedback.

Migration controller details: The ingress LSR kernel maintains a flow table structure that maps an active flow id to its currently scheduled LSP. In our testbed, we use the destination port as the flow id. The migration controller collects link utilization feedback from the first-hop LSRs and makes a migration decision every monitoring interval. If the feedback results in a migration decision, the controller makes a system call, and "migrates" flows by changing the assigned LSPs of target flows. In essence the LSP id of this flow is changed in the flow table structure. This is explained next.

At the ingress router we maintain a linked list of flows traversing each LSP. When a new flow arrives at the ingress, it is linked to the head of the chosen LSP. When a set of flows have to be migrated amongst LSPs, we choose a sublist S, starting at the head of the flow list, such that the amount of bandwidth in S is less than the amount of bandwidth that can be safely migrated, and no more flows can be added to S without violating the migration bandwidth limit. This sublist S is then linked to the *tail* of the new LSP list. By attaching to the tail of the new LSP list, we decrease the probability that a flow is migrated multiple times.

Note that the migration procedure here requires $O(C/f)$ work, where C is the capacity of a LSP, and f is the size of a flow. This turns out to be optimal in our case, since all the migrated flows are of only one capacity. Note that in the general case, if the flow capacities are not equal, then choosing an optimal set of flows to migrate is equivalent to the NP-Complete Subset-Sum (or Knapsack) problem.

B. Simulation Implementation

We have implemented the topology in Fig. 1 within the `ns2` simulator. MPLS capability is already available in standard `ns`, and we have added the additional functionality such as multipath routing.

We use 400 byte video and voice packets in all simulations. Each link has a buffer size of 24K bytes (this is approximately twice the bandwidth-delay product for a 90 Mbps link with 1 ms propagation delay). Each of the first-hop LSRs monitors the utilization of its congested link every 50 ms. We have also implemented the controllers as described in the testbed implementation.

IV. Results

In experiments that follow, there are two types of traffic in the same (Preferred Traffic) MPLS class. We view the video traffic as uncontrollable cross traffic and consider migration only applied to the voice calls. We choose to migrate the voice calls because they are all 64 Kbps CBR streams. This allows us to better predict bandwidth consumption once a flow has been migrated. Thus, even with a relatively fine-grained control action such as migration, we do not suffer from instability problems.

Next we describe the particulars of each traffic class, and present the different experimental scenarios in Sections IV-B and IV-C. We present results from two different types of experiments: first we quantify the benefit from different controllers in terms of loss rate, impact on voice codecs, etc. Next we quantify the overhead that is required to implement each type of control, and analyze their performance as the monitoring granularity is varied.

A. Traffic Classes and Types

We model voice calls as constant 64 Kbps bit rate streams. The voice calls have Poisson arrival with rate λ and exponentially distributed duration time with mean of $1/\mu$ (set to 3 min for all experiments). The voice call process can be viewed as an $M/M/\infty$ queueing system. In the steady state, the number of the voice calls in the system is Poisson distributed with mean λ/μ. Thus, the average bandwidth consumption of voice traffic is $64\lambda/\mu$ Kbps. In our experiments, we fix the arrival rate λ at 10 calls/second. This leads to 115.2 Mbps average load due to the voice calls (38.4 Mbps on each LSP).

For video traffic, we use an MPEG-1 encoding of the Star Wars movie trace as our video data sequence. We generate many different video streams as follows: we start with two-hour length of Star Wars data sequence. We consider the sequence as a circular list and use random starting points in this list for each stream. The method to generate the video streams above is common in video traffic modeling literature, e.g., in [7].

The video streams consume, on average, 400Kbps, and have Poisson arrival rate with a mean of 5/9 arrivals per second. The duration of each video stream is exponentially distributed with a mean of 3 minutes.

B. Experimental Method

LSP carries 40Mbps of video traffic. Since the average traffic load on all three LSPs is exactly the same in the first phase, the off-line optimization problem is easy to solve, and the Bernoulli splitting controller can (optimally) assign each incoming voice call to an LSP chosen uniformly at random. Similarly, the LLR controller can also assign incoming flows to the least-loaded LSP and reach long-term optimality.

After the controllers reach steady state in the first phase, we start the second phase of the experiments. We introduce a perturbation at only one of the LSPs (the first LSP in our experiments) to make the system temporarily imbalanced and congested. Note that the perturbation does not exceed the *total* system capacity, but the load on the link shared by the LSP and cross traffic may exceed the link capacity in some cases.

We construct the perturbation with new video streams, using two parameters. They are the number of new streams (perturbation size: N_p), and a time interval over which these N_p video streams arrive (perturbation interval: Δt). In our experiments, we vary N_p from 20 to 50 video streams, corresponding to 8 to 20 Mbps of new traffic over intervals of 50 ms to 5 seconds.

In the rest of this section, we present results from both our implementation and our simulations. We begin with effect of different controllers on the loss rate at the bottleneck links.

C. Drop rate at bottleneck links

In this experiment, the first phase lasts for 24 seconds. At the end of the first phase, 40 additional video streams are introduced in the cross traffic for LSP 1. The stream arrivals are distributed uniformly at random over 50 ms interval.

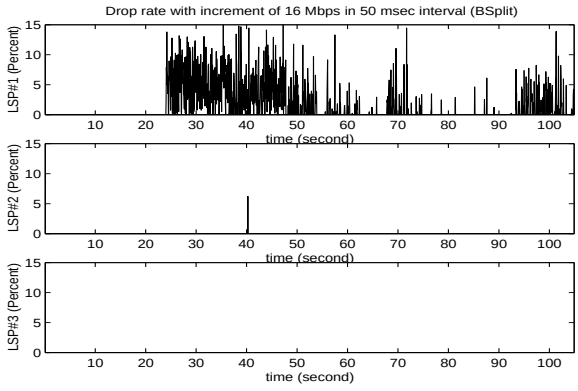

Fig. 2. Testbed results: the drop rates with BSplit controller.

In Fig. 2, we plot the drop rate in our testbed over each 50 ms period when the BSplit controller is used. The figure shows the losses at each LSP for both the voice and the video traffic. We show the analogous loss plots for the LLR controller in Fig. 3 and the migration controller in Fig. 4. We repeated the same experiment in our simulator, and the outcomes are very similar to those of testbed.

With the static (BSplit) controller, drops on LSP1 persist after the perturbation arrives. This is because the static controller still assigns incoming voice calls with 1/3 probability to each LSP even though the load on the system is no longer

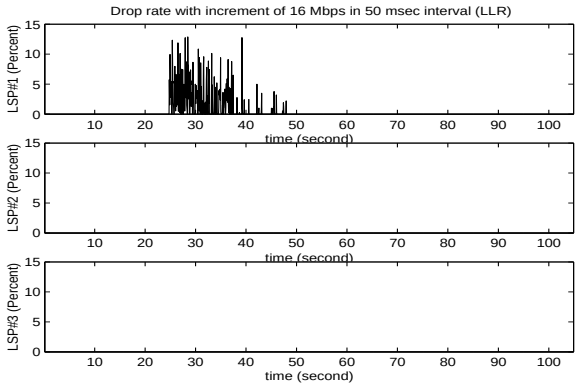

Fig. 3. Testbed results: the drop rates with LLR controller.

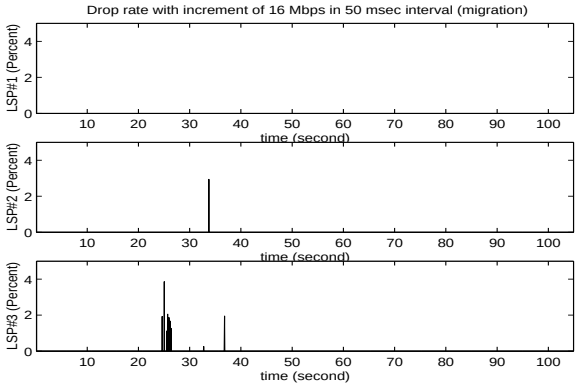

Fig. 4. Testbed results: the drop rates with migration controller (100% migration threshold and 2% safety margin).

symmetric. Unlike the static controller, the least-loaded routing (LLR) controller is able to adjust the link loads by assigning new voice calls to the least-loaded LSP. Thus, using LLR, the drop rate on LSP1 continuously decreases until it reaches 0 (at time 48.5 seconds). This is because the LLR controller is notified of the congestion on LSP1 and it reassigns incoming voice calls to the other LSPs. However, the LLR controller cannot reallocate the calls already scheduled on LSP1, and the situation there improves only as the voice calls depart at the rate of 64Kbps every 300ms.

The migration controller reacts immediately (to be precise, within 50 ms) and does not suffer prolonged losses on LSP1. There are a few random losses (on the *other* LSPs in this case) because the migration controller is not able to predict the fluctuations in the video traffic. In this experiment, the migration controller migrated 1,853 flows from LSP 0 over the first 25 seconds after the onset of congestion. Overall, there were 5504 migrations in the system over this period. This relatively large number of migrations is due to the bursty nature of the video cross-traffic. The total amount of bandwidth consumed due to the 1,471 monitoring packets was 24.98 Kbps (8.33 Kbps per feedback link).

D. End-to-end Quality

In the last section we showed that a migration-based controller can effectively reduce packet loss rates during short-term overloads. In this section, we analyze the end-to-end

performance of the VoIP flows, and show that the losses incurred by the other schemes do, in fact, translate to poor end-to-end performance. Further, we show that the small amount of random losses due to migration does not affect the quality of the VoIP calls. Specifically, we analyze two different measures of quality: runs of dropped packets (loss run length), and correlation between drops.

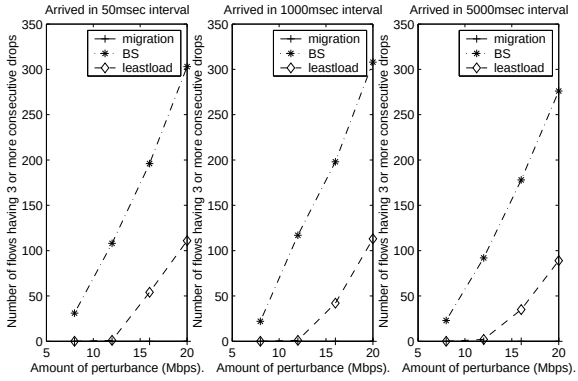

Fig. 5. Number of voice calls with loss run length greater than 150 milliseconds.

Loss Run Length Analysis: Consecutive drops are potentially the most limiting factor in designing VoIP networks [8]. Many codecs can compensate for losses as long as the loss run length is less than 150 ms [8]. In our experiments, a loss period of 150 ms corresponds to three consecutive dropped packets for the same flow. In Fig. 5, we plot the number of voice calls that have more than three consecutive drops. Over the lifetime of these (simulation) results, there were 1700 voice calls active in the system. Under this consecutive drop criteria for quality, over 18% of the calls are affected using BSplit controller, and about 7% of the calls are affected using the LLR controller (with perturbation size of 20Mbps and any perturbation interval). The study in [8] also points out that losses greater than 2% also affect end-to-end quality of VoIP calls. In our experiment it turns out that the affected call numbers under 2 % packet loss criteria are almost exactly the same as the 150 ms of loss criteria. In contrast, the migration controller is able to provide quality service to almost all calls for both quality criteria: this is particularly important since this result shows that not only does migration perform well on average, it is useful on a per-flow basis.

E. Timescales

In the previous sections, we based our performance evaluation on 50 ms monitoring granularity. Since timescale plays an important role for an adapting controller to react to short-term overloads, we next study the performance of LLR and migration controllers under different timescales and overloads in our testbed.

Fig. 6 shows the drop rates on congested links for LLR and migration controllers with six different timescales ranging from 50 ms to 2 seconds. In the figure, we show experiments with two different perturbations (40 video flows arriving over 1 second, and 50 video flows arriving over 50 ms). As

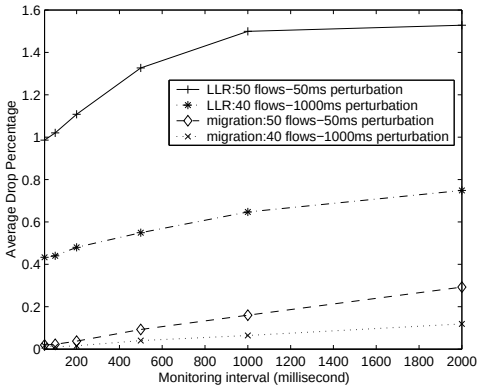

Fig. 6. Testbed results: the drop rates for LLR and migration controller under different timescales and overloads.

expected, for both controllers, more precise data due to the faster monitoring reduces the drop rate. Note that over all monitoring periods, migration performs better than LLR. This is because once LLR detects an overload, it requires between 25-30 seconds (depending on the call departure rate) to reduce the congestion on an LSP, while the migration controller is able to shift multiple flows instantaneously.

V. CONCLUSION

In this paper, we have studied flow migration as an emergency mechanism to sustain QoS during short periods of heavy load, specifically the onset of congestion. Our contributions in this work are twofold: we have shown that a fast timescale control, such as migration, is necessary to maintain end-to-end application quality for sensitive applications such as Voice-over-IP. We have also presented a detailed schematic implementation of flow migration for MPLS networks, and have shown how migration can be efficiently implemented such that the probability that the same flow being repeatedly migrated is minimized.

REFERENCES

[1] www.kagoor.com/Kagoor_VoIPplatform_whitepaper.pdf.

[2] M. Alanyali and B. Hajek. Analysis of simple algorithms for dynamic load balancing. Mathematics of Operations Research, vol. 22, no. 4, pp. 840–871, 1997.

[3] M. Alanyali and B. Hajek. On large deviations in load sharing networks. Ann. Appl. Probab., vol. 8, no. 1, pp. 67–97, 1998.

[4] E. Altman, T. Basar, T. Jimenez, and N. Shimkin. Competitive routing in networks with polynomial cost. In *Proceedings of Infocom*, 2000.

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

[6] Kenjiro Cho. Sony Computer Science Laboratories. ALTQ: Alternate Queueing for BSD UNIX. http://www.csl.sony.co.jp/ kjc/software.html.

[7] D. P. Heyman and T. V. Lakshman. Source models for VBR broadcast-video traffic. *IEEE/ACM Transactions on Networking*, 4(1):40–48, 1996.

[8] Keith Kim, Petros Mouchtaris, Sunil Samtani, Rajesh Talpade, and Larry Wong. QoS provisioning for VoIP in bandwidth broker architecture: A simulation approach. In *SCS WMC01*, January 2001.

[9] NISTswitch. http://www.antd.nist.gov/itg/nistswitch/.

[10] Floyd S. and Jacobson V. Link-sharing and resource management models for packet networks. In IEEE/ACM Transactions on Networking, Vol. 3 No. 4, pp. 365-386, August 1995.