

Monitoring-based Adaptive Overlay Streaming Media

Brian Chavez and Yan Chen

Abstract—

Streaming media delivery is of great importance for applications such as Video on Demand (VOD) and Internet conferencing. Traditional streaming media systems treat the underlying network as a best-effort black box, and apply various coding techniques or blindly transmit the data over multiple paths to improve the quality and reliability. In this paper, we design and implement an adaptive live streaming media system that leverages on the emerging overlay networks and scalable monitoring services which can provide real-time network congestion/failure information when it occurs. First, we examine the performance improvement with PlanetLab [1] global overlay networks and find that in many cases, overlay routing can effectively reduce the loss rate and/or increase the TCP throughput. The client of our streaming media system uses overlay routing to bypass faulty link(s) and reestablish new connection to the server in the order of seconds. Furthermore, we apply skip-free rewinding techniques so that the client perceive continuous and smooth media playback without disruption during the path switch. Deployment and experiments on PlanetLab testbed show that our system can adapt to network congestion in less than four seconds, and achieve skip-free media playback.

I. INTRODUCTION

The Internet has evolved to become a commercial infrastructure for service delivery. However, due to the IP addressing scheme, routing paradigm, and other historical reasons, the current Internet is not well suited for the purpose of service delivery. In response to these challenges, various forms of overlay networks, which introduce new functionality within the network near the edges, have been proposed and some deployed in the Internet. For instance, commercial Content Distribution Networks (CDN) have been deployed to bring content closer to the end users.

Many of the overlay applications and services have flexibility in choosing their communication paths and targets, they can benefit significantly from dynamic network distance prediction (e.g., latency and loss rate). Recently, some dynamic overlay monitoring systems are proposed and deployed [2], [3], which can detect path outages and periods of degraded performance within seconds.

Streaming media has many important applications, e.g., Video on Demand (VOD) and Internet conferencing. Meanwhile, streaming media has stringent network bandwidth/loss rate requirement to ensure its quality. In this

paper, we propose to an overlay streaming media system, which can dynamically switch the transmission path to bypass congestion/failures in the order of seconds, based on the feedback of overlay network monitoring system. Furthermore, we apply skip-free techniques so that the client perceives continuous and smooth media playback without disruption.

II. STREAMING MEDIA TECHNOLOGIES

A. TCP vs. UDP-based

It is a common view that neither TCP nor UDP are adequate for streaming media applications, and numerous transport protocols are proposed with alternative service models. Commercial products like Real Player and Media Player use RTSP protocol [4], which can be built on top of TCP, UDP or HTTP. By default, they usually use UDP for performance reasons.

However, in a recent revisit, Krasic *et al.* suggests that TCP is a viable and attractive choice for quality-adaptive video streaming applications, like VOD [5]. In fact, the Nullsoft Video (NSV) format [6], which is used by Winamp [7], only uses TCP for transport-level protocol. On the other hand, TCP is not applicable for purely-interactive applications, such as tele-conferencing and multiplayer gaming. Our implementation is based on the SHOUTcast server [8] and Winamp client. But our technique is applicable for both UDP and TCP-based streaming media.

B. Reliability Techniques

Most existing streaming media systems treat the network as a black box. The reliability is achieved through either transport/application-level coding or multi-path redundant transmission [9]. They also do not consider how to choose the multiple paths. We are one of the first to incorporate the feedback from network monitoring infrastructure to improve the reliability and throughput in a cost-effective manner.

III. DIVERSITY OF OVERLAY PATHS

In this section, we try to address the question: Can overlay routing indeed bypass the congestion/failures?

In [2], Anderson *et al.* studied the loss rate, latency and TCP throughput improvement on an overlay network of only 12 - 16 nodes, and found that about 5% of samples have significant improvement (i.e., loss rate reduction

by 0.05 and double throughput). However, there remain many open questions. For instance, it is not clear the percentage of lossy paths¹ that receive significant improvement. How many of them become non-lossy? How will the performance be for a larger overlay network, e.g., of 50 nodes? Furthermore, RON uses bi-directional performance measurements and take the half as uni-directional performance which is not accurate for asymmetric routing and asymmetric link performance [11]. We address these issues with our improved experiments below.

A. Measurement Methodology

We choose 51 Planet Lab hosts, each from a different organization as shown in Table I. All the international Planet Lab hosts are universities.

Areas and Domains		# of hosts	
US (40)	.edu	33	
	.org	3	
	.net	2	
	.gov	1	
	.us	1	
Inter-national (11)	Europe (6)	France	1
		Sweden	1
		Denmark	1
		Germany	1
		UK	2
	Asia (2)	Taiwan	1
		Hong Kong	1
	Canada	2	
	Australia	1	

TABLE I

DISTRIBUTION OF PLANET LAB HOSTS FOR EXPERIMENTS.

We measure the loss rates between every pair of hosts. Our measurement consists of 300 trials, each of which lasts 300 msec. During a trial, each host sends a 40-byte UDP packet² to every other host. Usually the hosts will finish sending before the 300 msec trial is finished. For each path, the receiver counts the number of packets received out of 300 to calculate the loss rate.

To prevent any host from receiving too many packets simultaneously, each host sends packets to other hosts in a different random order. Furthermore, any single host uses a different permutation in each trial so that each destination has equal opportunity to be sent later in each trial, because when sending packets in a batch, the packets sent later are more likely to be dropped. Such random permutations are pre-generated by each host. To ensure that all hosts in the network take measurements at the same time, we set up sender and receiver daemons, then use a well-connected server to broadcast a “START” command.

¹We classify a path to be lossy if its loss rate exceeds 5%, which is the threshold between “tolerable loss” and “serious loss” as defined in [10].

²20-byte IP header + 8-byte UDP header + 12-byte data on sequence number and sending time.

Will the probing traffic itself cause losses? We did sensitivity analysis on sending frequency as shown in Fig. 1. All experiments were executed between 1am-3am PDT June 24, 2003, when most networks are free. The traffic rate from or to each host is $(51 - 1) \times \text{sending_freq} \times 40$ bytes/sec. The number of lossy paths does not change much when the sending rate varies, except when the sending rate is over 12.8Mbps, since many servers can not sustain that sending rate. We choose a 300 msec sending interval to collect loss rate statistics quickly but with moderate bandwidth consumption.

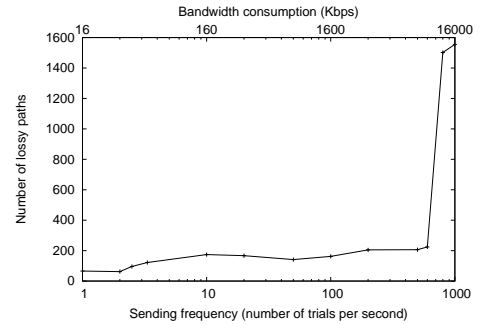


Fig. 1. Sensitivity test of sending frequency

From June 24 to June 27, 2003, we ran the experiments 100 times, mostly during peak hours 9am - 6pm PDT. Each experiment generates $51 \times 50 \times 300 = 765K$ UDP packets, totaling 76.5M packets for all experiments. We run the loss rate measurements three to four times every hour, and run the pair-wise traceroute for latency every two hours. Table II shows the loss rate distribution on all the paths of the 100 runs. About 96% of the paths are non-lossy. Among the lossy paths, most of the loss rates are less than 0.5. Though we try to choose stable nodes for experiments, about 25% of the lossy paths have 100% losses and are likely caused by node failures or other reachability problems.

loss rate	[0, 0.05]	lossy path [0.05, 1.0] (4.1%)				
		[0.05, 0.1]	[0.1, 0.3]	[0.3, 0.5]	[0.5, 1.0]	1.0
%	95.9%	15.2%	31.0%	23.9%	4.3%	25.6%

TABLE II

LOSS RATE DISTRIBUTION: LOSSY VS. NON-LOSSY AND THE SUB-PERCENTAGE OF LOSSY PATHS.

Next, we will show the performance improvement achieved through single-node relay on the overlay network. For each path ($src \rightarrow dest$) considered, we find the overlay path ($src \rightarrow relay \rightarrow dest$) that gives most performance improvement. The performance of overlay path is computed with the formulas below. lr stands for loss rate and tp stands for throughput.

$$lr_{overlay} = 1 - (1 - lr(src \rightarrow relay))(1 - lr(relay \rightarrow dest)) \quad (1)$$

$$tp_{overlay} = \min(tp(src \rightarrow relay), tp(relay \rightarrow dest)) \quad (2)$$

B. Loss Rate Improvement

In a total of $2550 \times 100 = 255,000$ path measurements, 10,980 (or 4.1%) are lossy. Among them, 5,705 paths (52.0%) have loss rate reduced by 0.05 or more and 3,084 paths (28.1%) change from lossy to non-lossy.

C. Throughput Improvement

We do not have root access on PlanetLab machines, thus we cannot deploy lightweight bandwidth measurement tools such as [12], [13] to simultaneously measure the bandwidth among the 2,550 paths. Instead, we use the formula from [2] as below to estimate the TCP throughput.

$$throughput = \frac{\sqrt{1.5}}{rtt \times \sqrt{loss_rate}} \quad (3)$$

where rtt is the round-trip time and $loss_rate$ is the unidirectional loss rate of the path. It provides a good estimation of the path throughput for a wide range of loss rate [14].

Among the 255,000 path measurements, 60,320 paths (24%) have non-zero loss rate, and thus computable throughput. When the loss rate is 1.0, the path is disconnected, so we set the throughput to be zero. Note that most paths with small bandwidth has non-zero loss rates. So this subset covers most of the paths of which the throughput can be improved.

Since we only consider a subset of overlay paths (24%), the performance improvement below is a conservative estimation. Still, the results are very encouraging. Among the 60,320 path measurements, 32,939 (54.6%) paths have throughput improved, and 13,734 (22.8%) paths have throughput doubled or more.

IV. MONITORING-BASED ADAPTIVE OVERLAY STREAMING MEDIA

In addition to streaming media clients and server, the system is composed of a set of overlay nodes, e.g., the CDN. For the rest of the paper, we refer “overlay nodes” as “nodes”. These nodes and the server are controlled by an Overlay Network Operation Center (ONOC), which instruments some of the nodes to continuously measure some paths for a complete map of network conditions among the nodes [3].

Normally a client directly connects to a server for streaming media content. It also registers the path and sets up a trigger for path performance warnings at ONOC. When the path incurs congestion/failure, ONOC detects that either through passive monitoring by the client or through some active probing instrumented by ONOC. Then ONOC searches for an alternative overlay path to bypass the faulty path, and sends that to the client if such path exists. The client tears down the current connection, sets up a new connection via overlay node(s), and attempts to concatenate the new streams with the old one for skip-free effect. The event driven diagram is shown in Fig. 2.

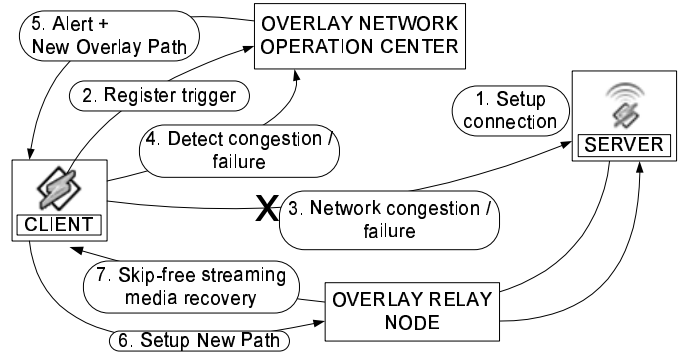


Fig. 2. Event-driven diagram of monitoring-based adaptive overlay media streaming

Given millions of clients, we cannot monitor the network distance for each of them. We group the client by the autonomous system (AS) (or more sophisticated technique like [15]) and assume that there is an overlay node (referred as proxy) in the same AS of the client and experiences similar congestion/losses as the client does. If congestion is in the last mile of the client, any scheme based on path diversity will not work. For simplicity, we assume that the client and its overlay proxy are the same node.

Next, we will discuss the techniques for each step of adaptation.

A. Detection of Congestions

In our implementation, we have the client passively monitor the throughput every 200-300 msec. Use exponential-weighted moving average (EWMA) for better stability. If the smoothed throughput drops below certain threshold (e.g., 50% of streaming bitrate), we assume the congestion occurs.

B. Recovery Path Selection with Dual Metrics

The path is selected based on two metrics: throughput and latency, based on a similar techniques in [16]. Assume that we use a scalable overlay monitoring system like [3], the up-to-date throughput and latencies between every pair of nodes are available at ONOC, which will select the recovery path for the client.

C. Skip-free (Lossless) Streaming Media Recovery

To bypass network congestion or failures, the client needs to tear down the current connection, switch to the overlay path and reestablish the connection to the server. For live streaming media or when the server is broadcasting the media to multiple clients, the reconnected client may lose part of the data. In this section, we discuss the protocol and implementations for continuous (skip-free) media playback.

We add a buffering layer at the server and an overlay layer at the client to work with legacy client and server

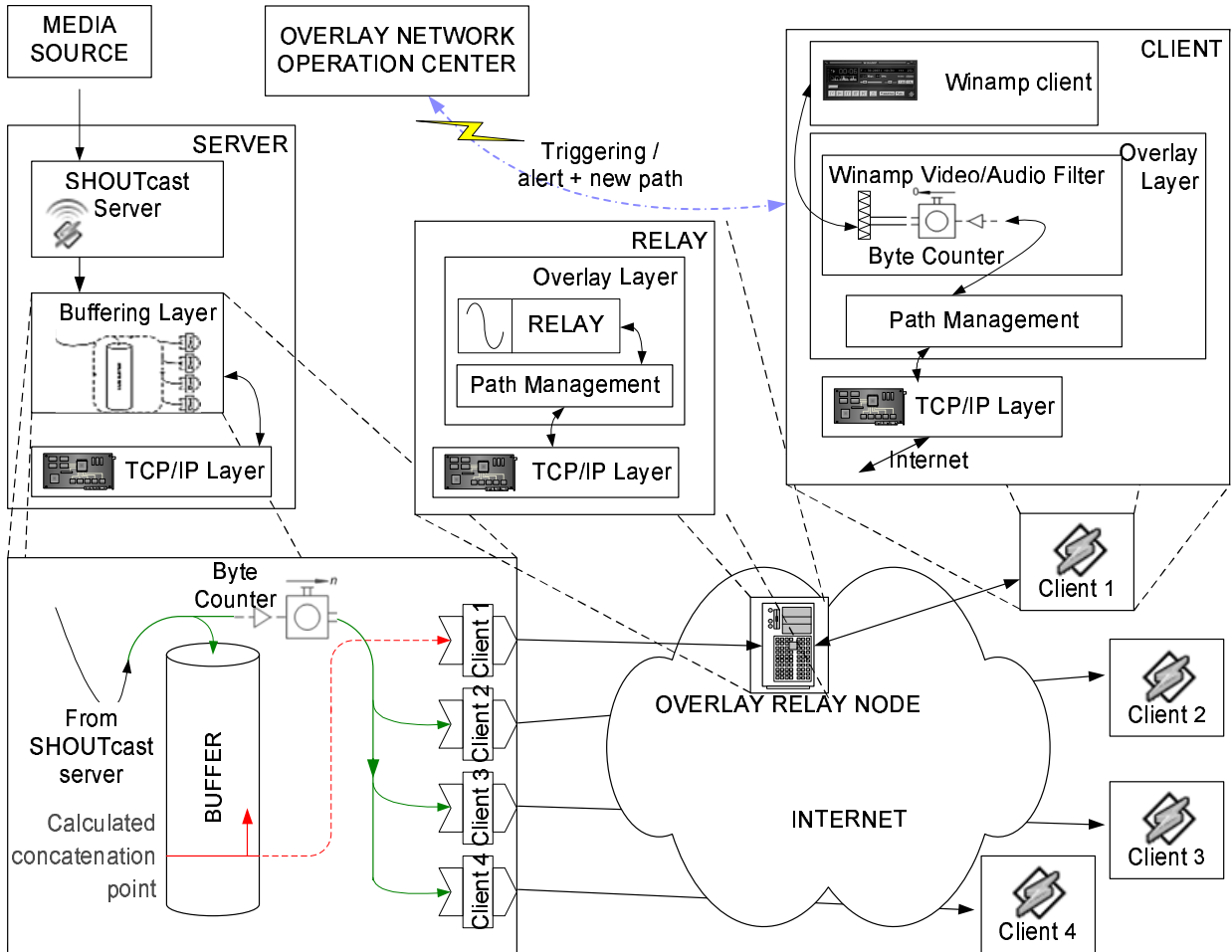


Fig. 3. Architecture of monitoring-based adaptive overlay media streaming

softwares. The architecture is shown in Fig. 3. Our implementation is built on Winamp [7] client and SHOUTcast [8] media server software. But it can be extended for any TCP or even UDP based streaming media protocols.

When a client sends a connection request to the streaming media server, the media server responds with a server header indicating the server type, build, etc. The buffering layer at the server keeps track of the current number of bytes broadcasted and inserts that information into the server header. The overlay layer at the client also records the current number of bytes broadcasted. This is to ensure that both the server and the client have a common reference point as the absolute byte offset from the beginning of the media, then the client starts counting the number of bytes it receives.

During normal video streaming playback, both the server and the client know the number of bytes broadcasted. The data flows directly from the SHOUTcast server to the client. Meanwhile, the most recent data streamed are cached in the buffering layer of the server.

When a path change occurs, the client sets up a new path via overlay node(s) to connect to the streaming media server. When reconnected, the filtering layer of client issues a “reset” signal along with the clients last known number of bytes the server broadcasted (denoted

as $client_count$). The server then uses the current number of bytes broadcasted ($server_count$) to determine the amount of broadcast data that the client is missing ($server_count - client_count$). If the amount of missing data is no larger than the buffer size, the server resumes data transmission from exactly the point where the connection dropped before, and the client will have perfect concatenation with the old media streams.

During the path change, the Winamp client is disconnected from the server, but it will continue the playback with data in its own buffer. If it receives new streams from the reestablished connection before running out of the buffer, the client will perceive continuous skip-free media playback. The path change and stream concatenation are transparent to the user.

The minimum size of buffers on the server and client for skip-free playback depends on the streaming bitrate and the adaptation time defined as the period from the moment that congestion/failure occurs to successful stream concatenation. The maximum streaming bitrate for DSL/cable modem is 450 Kbps [17]. As shown in Sec. VI-A, the adaptation time is about 10-20 seconds. Thus a buffer of 1.2 MB is sufficient. Even if the missing data is bigger than the server buffer size, the server can still start from the beginning of the buffer to alleviate the

losses.

1) *Implementation Details:* Each normal client has a data queue. There is a main thread reading data from the SHOUTcast server and enqueueing the data to each queue. Every client also has a dequeuing thread which dequeues the data and sends it to the client. When disconnected, the client removes the queue and exit the dequeuing thread.

For each reconnected client, a separate rewind thread (denoted as the dotted curve in the buffering layer at Fig. 3) will be spawned to calculate the byte offset of the streaming data where the client left off (the concatenation point in the buffer as shown in Fig. 3), and transmit the data from the buffer. The sending rate of the rewind thread for the reconnected client is higher than that of those normal clients so that the reconnected client can refill its buffer quickly. From our experiments, the catch up time is negligible to the path switch time.

When the reconnected client catches up with normal clients, i.e., the rewind thread reaches the same index of the streaming data as the main thread, the main thread will take over the client, and the rewind thread can exit. To avoid the synchronization overhead between these two threads, we have the main thread enqueue the reconnected client as well (denote the starting index as reconnection point). Since the catch up time is negligible (in the order of seconds), we assume that the queue of reconnected client can hold the transmitted data of the main thread in that period. Then the rewind thread only needs to transmit the data from the concatenation point to the reconnection point. After that, it becomes a normal dequeuing thread, transmitting data from the queue to the client.

We can also apply this technique to application-level media multicast. So the edge overlay nodes use skip-free technique for their clients. Presumably, this technique is also applicable to wireless streaming media when the handoff occurs.

V. EVALUATION METHODOLOGY

We implemented the client overlay layer in 2200 lines of C# code and the server buffering layer and overlay relay layer in Java, with 1100 lines and 900 lines of code, respectively.

We deploy our system on Planet Lab [1], a global network testbed. For most of our experiments, the SHOUTcast server is placed at UC San Diego, the ONOC is at Stanford University, the overlay relay node is at HP Lab of Palo Alto, CA, and the Winamp client is at UC Berkeley. The client is an Intel PIII/500MHz Windows XP machine with 256MB RAM on 100Mbps switched Ethernet. All other hosts are Planet Lab nodes, 1.0GHz-1.8GHz Linux machines with 512MB-883MB RAM.

The bottleneck is introduced by using a Packeteer® PacketShaper [18]. The streaming bitrate is about 600 Kbps and less than the normal available bandwidth between client and server. During streaming, we set the bandwidth limit from SHOUTcast server to Winamp client as 76 Kbps.

The baseline for comparison is the client-server streaming media without monitoring-based adaptation. Thus when congestion occurs, the Winamp client will gradually run out of buffer, and eventually stall the media playback. In comparison, our monitoring-based approach will adapt to the congestion. There are two metrics: 1) the adaptation time defined before, and 2) successful skip-free continuous playback.

VI. PERFORMANCE RESULTS

A. Adaptation time

The adaptation time breakdown are as follows.

- 1) Detection of congestions: the time from introducing the bottleneck link via PacketShaper to when the client detects the congestion is less than one second.
- 2) Client reports the congestion to the ONOC: less than one second.
- 3) ONOC triggers the warning and figure out alternative path, send to the client. We run the throughput calculation and path finding on the tomography-based overlay monitoring system [3] for a 50-node overlay network on Planet Lab, on average it takes about one second.
- 4) Client tears down old connection, sets up new connection via overlay node and gets the new media data concatenated: on average 0.6 second.

Total adaptation time: less than four seconds. When the latency between client and overlay relay node or between overlay relay node and server is bigger, we get larger adaptation time. But conservatively speaking, the adaptation time should be 10-20 seconds.

B. Effectiveness of skip-free

Current experimental settings show perfect concatenation of streaming media. We also do the experiments on a few other Planet Lab nodes, and find that as long as there are enough available bandwidth on the overlay path and reasonable adaptation time, we achieve the skip-free concatenation.

VII. CONCLUSIONS

In this paper, we first examine the diversity of Internet end-to-end paths and find that in many cases, overlay routing is effective on reducing loss rates and/or increasing throughput. Then we design and implement an adaptive streaming media system which leverages on scalable overlay monitoring services to bypass congestion/failures within seconds. Furthermore, skip-free rewinding techniques are applied so that client perceive continuous and smooth media playback during the path switch.

VIII. ACKNOWLEDGEMENT

This work is supported by SUPERB (Summer Undergraduate Program in Engineering at Berkeley) program.

REFERENCES

- [1] Planet Lab, "<http://www.planet-lab.org/>," .
- [2] D. G. Andersen et al., "Resilient overlay networks," in *Proc. of ACM SOSP*, 2001.
- [3] Y. Chen, D. Bindel, and R. H. Katz, "Tomography-based overlay network monitoring," in *ACM SIGCOMM Internet Measurement Conference*, 2003.
- [4] IETF, "RTSP protocol," <http://www.rtsp.org/>.
- [5] C. Krasic, K. Li, and J. Walpole, "The case for streaming multimedia with tcp," in *the 8th International Workshop on Interactive Distributed Multimedia Systems (iDMS)*, 2001.
- [6] M. Melason, "Description of the Nullsoft Video (NSV) format," <http://www.pcisys.net/~melanson/codecs/nsv-format.txt>.
- [7] Nullsoft, "Winamp," <http://www.winamp.com/>.
- [8] Nullsoft, "Shoutcast," <http://www.shoutcast.com/>.
- [9] T. Nguyen and A. Zakhor, "Path diversity with forward error correction (PDF) system for packet switched networks," in *IEEE INFOCOM*, 2003.
- [10] Y. Zhang et al., "On the constancy of Internet path properties," in *Proc. of SIGCOMM Internet Measurement Workshop 2001*.
- [11] K. Claffy, H.-W. Braun, and G. Polyzos, "Measurement considerations for assessing unidirectional latencies," *Journal of Inter-networking*, 1993.
- [12] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Sprobe: A fast tool for measuring bottleneck bandwidth in uncooperative environments," <http://sprobe.cs.washington.edu/>.
- [13] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *ACM SIGCOMM*, 2000.
- [14] J. Padhye, V. Firoiu, D. Towsley, and J. Krusoe, "Modeling TCP throughput: A simple model and its empirical validation," in *ACM SIGCOMM*, 1998.
- [15] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *Proc. of SIGCOMM '2000*.
- [16] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *ACM SIGCOMM*, 2001.
- [17] RealOne Player, "Pre-processing and encoding: Making the most of your bandwidth," <http://service.real.com/learnnav/ppth1.html>.
- [18] Workgroup Solutions, "PacketShaper: Bandwidth Control and IP Management," <http://www.bandwidth-management.org/>.