User-specified Adaptive Scheduling in a Streaming Media Network

Michael Hicks, Adithya Nagarajan
Department of Computer Science
Univ. of Maryland, College Park, MD
mwh@cs.umd.edu, sadithya@cs.umd.edu

Robbert van Renesse
Department of Computer Science
Cornell Univ., Ithaca, NY
rvr@cs.cornell.edu

***Abstract*— In disaster and combat situations, mobile cameras and other sensors transmit real-time data, used by many operators or analysis tools. Unfortunately, in the face of limited, unreliable resources, and varying demands, not all users may be able to get the fidelity they require. This paper describes *MediaNet,* a distributed stream processing system designed with the above scenarios in mind. Unlike past approaches, MediaNet's users can intuitively specify how the system should adapt based on their individual needs. MediaNet uses both local and online global resource scheduling to improve user performance and network utilization, and adapts without requiring underlying support for resource reservations. Performance experiments show that our scheduling algorithm is reasonably fast, and that user performance and network utilization are both significantly improved.**

I. Introduction

Consider a dangerous setting, such as collapsed buildings caused by an earthquake or terrorist attack. Novel recording devices, such as cameras carried by Uninhabited Aerial Vehicles (UAVs) or by robots that crawl through rubble, may be deployed to explore the area. The output of these devices can be of interest to many operators. Operators may include rescue workers working in the rubble itself, people overseeing the work in a station somewhere, the press, or software that creates, say, a 3-dimensional model of the scene.

Different operators may require different views of the area, and may have different fidelity requirements or user priorities. Although the operators may work independently of one another, they share many resources, such as the recording devices themselves, compute servers, and networks. These resources have limited capacity, and so they must be allocated carefully. Without resource reservation, adaptivity is essential.

The conditions present in this disaster situation are not unique. That is, many applications consist of multiple operators interested in streaming data from multiple sources that must adapt to limited resources, potentially in application-specific ways. Examples include the exchange and aggregation of sensor reports [1], the distribution of media on a home network [2], the performing of reconnaissance and deployment in a military setting [3], and so on.

A number of projects have explored how to provide improved quality of service (QoS) for streaming media in resource-limited conditions. These systems place computations in the network, either within routers themselves (e.g., [4], [5], [6]) or at the application-level using an overlay network (e.g., [7], [8]), and employ system-determined, local adaptations, such as priority-based video frame dropping. While such adaptations impose little overhead, they can be inefficient because they do not take into account global information. Also, existing schemes typically do not consider user preferences in making QoS decisions.

To study whether these problems can be overcome, we are developing a system called *MediaNet* that takes a comprehensive view of streaming data delivery. MediaNet mainly differs from past approaches in two ways. First, rather than make QoS adaptation system-determined, MediaNet allows users to specify how it should adapt under overload conditions. Each user contributes a list of alternative specifications, and associates a utility value with each specification. To some users, color depth may be more important than frame rate, while for other users the preference may be the other way around. The primary goal of MediaNet is to maximize each user's utility.

Second, in addition to using local scheduling, MediaNet employs a *global scheduling service* to divide tasks and flows among network components. This global point of view has benefits to both fairness and performance, because the service can consider specifications from multiple users while accounting for priority and overall network efficiency; the challenge is to do this in a scalable manner. Different from other projects that use global schedulers (e.g., [5], [9], [2]), MediaNet's global scheduling service is continuously looking for improvements based on monitoring feedback. MediaNet employs a completely adaptive overlay network; it does not rely on resource reservations, and adapts to loads not under its control.

Experimental measurements with our prototype implementation are promising: users achieve better performance and the network is more efficiently utilized than without any or with only local adaptations. On the other hand, our system does exact a higher cost for its global adaptations, in terms of overhead and implementation complexity. We consider our work as a step to exploring how to synergistically apply adaptations from various levels in a scheduling hierarchy.

In this paper, we describe the MediaNet architecture (Section II) and our prototype implementation (Section III). We focus on the challenges of implementing a globally-

This work was funded in part by DARPA grant F30620-98-2-0198, DARPA/AFRL-IFGA grant F30602-99-1-0532, a grant under NASA's REE program administered by JPL, NSF-CISE grant 9703470, and the AFRL-IFGA Information Assurance Institute under grant AFOSR F49620-01-1-0312.

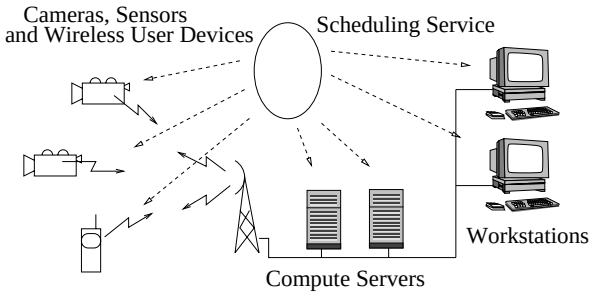


Fig. 1. MediaNet architecture.

reconfigurable stream processing system, and show experimental evidence of its costs and benefits (Section IV). We finish up with related work (Section V) and conclusions and future work (Section VI).

II. MediaNet Architecture

MediaNet's architecture defines a *computational network*, consisting of *compute nodes* and *network links*. These elements are responsible for receiving streaming data from various sources, computing on that data, and delivering it to the end-applications. As shown in Figure 1, compute nodes are highly heterogeneous, consisting of cameras, sensors, workstations, and compute servers; as such they have different computational power, available memory, hardware support for video operations, etc. Network links between nodes could be either wired or wireless; as such, the underlying network topology may change at run-time as components physically move around or new parts of the infrastructure are deployed.

The user's interface to this computational network is via a *global scheduling service*; the architecture leaves the implementation of this service abstract. The service is responsible for scheduling various flows and computations on the network, based on user specifications and the current state of the network. We describe these specifications next, followed by a discussion of scheduling.

A. Specifications

Users communicate their requirements to the global scheduling service using what we call a *continuous media network* (CMN), which is a directed acyclic graph (DAG) representing a computational dataflow. The job of the global scheduling service is to combine the CMNs of individual users into a single CMN, and then partition this CMN into subgraphs to be executed on the various compute nodes. The act of combining the CMNs and partitioning them among nodes takes into account issues of fairness, performance, and user-specified adaptation, as we describe later.

Each node in a CMN represents an operation mapping zero or more input *frames* (stream-specific packets of data such as video frames, audio clips, etc.) to zero or more output frames. Operations can be simple, e.g., data forwarding, frame prioritizing, and frame dropping; or they can be more complex, e.g., video frame cropping, changing the resolution or color depth, "picture-in-picture" effects, compression, encryption, etc. We also need operations to receive input from and send output to components external to the DAG, to perform I/O with devices like video cameras and players. The global scheduling service takes into account the bandwidth, latency, and processing requirements of operations.

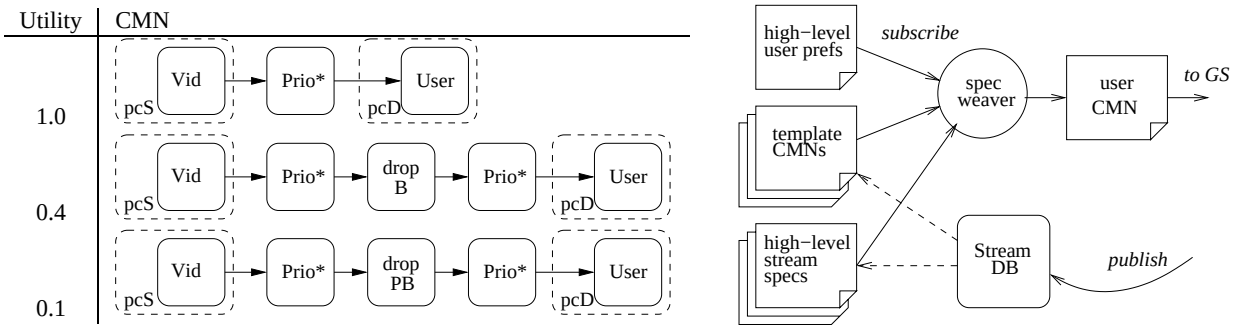
Operations have a number of associated attributes. One important attribute is the *interval* which indicates the minimum time between operations on subsequent frames (i.e., the inverse of the rate). For better performance, operations can either process input frames immediately, or they can be forced to execute at the specified intervals on queued data. In either case, the interval effectively specifies a soft real-time constraint on the processing of frames; if frames arrive faster than the specified interval, or if the node cannot process them at that interval (perhaps because of downstream congestion), then either backpressure must be applied to the incoming flow or frames must be dropped. How to handle this situation adaptively is considered in the next subsection.

A CMN node can be fixed at a certain location in the actual network (e.g., to indicate the network location of a particular video source), or left unspecified. Moreover, a node can be considered *optional*, meaning that it is only inserted between mandatory nodes when the CMN is scheduled across multiple compute nodes. Operations can maintain internal soft-state and need not actually operate on packets. Requiring *soft-state* is important for allowing operations to relocate during a reconfiguration.

A user specification is a list of CMNs, where each CMN's relative preference is indicated by a corresponding *utility value*, which is a real number between 0 and 1, where 1 means most desirable. In our implementation, we encode user specifications, and consequently CMNs, as XML documents. An example specification is shown in Figure 2(a), depicted graphically, where the user specifies three CMNs, having decreasing utility. In each CMN, an MPEG video stream originates at location `pcS`, the frames are prioritized for intelligent dropping by the optional (as indicated by the `*`) `Prio` operation, and they are finally delivered to the user's player on `pcD`. In the second CMN, the frame rate is reduced by proactively dropping B frames, while in the third CMN the P frames are dropped as well.[1] The MediaNet scheduler can decide which of these specifications to run, and where to run the operations with unspecified locations.

We do not expect users will author CMNs directly, but rather provide higher-level preferences, such as the general adaptation methodology and the streams of interest. For example, a user might specify (in some declarative format) "I want MPEG stream X from location Y, and I want to adapt using frame dropping." A *weaving* tool, which is part of the global scheduling service, would index these preferences into a database that contains template CMNs and stream specifications. The template CMNs are basically like the

[1] In MPEG streams, I frames are essentially JPEG images, while P frames and B frames exploit temporal locality, including "deltas" from adjacent frames. P frames rely on the most temporally-recent P or I frame, and B frames rely on the prior I or P frame, and the next appearing I or P frame. Therefore, I frames are more important than P frames, while B frames are the least important.



(a) Example User CMN (b) MediaNet User API

Fig. 2. User specification APIs

CMNs we have shown, but without any stream-specific data, like the stream location, resource usage characteristics, etc., while the stream specifications would include this missing information. The weaver then merges the template and the stream specification of the requested movie together to create an almost-complete CMN; only the utility values have not been filled in. This idea is shown in Figure 2(b).

The weaver should set utility values to share resources among users of potentially differing priority fairly. Utility values have both *relative* and *absolute* effect. That is, a user's alternative specifications are prioritized relatively by the ordering of their utilities, while the particular magnitude of a utility value relates globally to the utility values of other users. For example, a higher priority user might have the same specification as in Figure 2(a), but have utility values 1.0, 0.2, and 0.1, respectively. When resources became limited, this user would be forced to adapt only after a user having the utilities assigned in Figure 2(a). We expect to report on the implementation of this aspect of the MediaNet architecture in future work; in the meantime, our implementation assumes utilities are set fairly by hand.

B. Scheduling

Once a user provides a specification, the global scheduling service schedules it on the network in conjunction with all existing user specifications. Some schedules generated by our prototype implementation are depicted in Figure 6. Here we have combined five user specifications of equal priority, each varying from that in Figure 2(a) only in the user and video locations, and scheduled them on a network (shown in Figure 3). In Figure 6 the `v1` and `v2` nodes correspond to the video sources, the `Pr` nodes correspond to the frame priority-setting `Prio` operations, the `dB` node corresponds to the `drop B` operation. The empty circles are *send* and *receive* operations inserted by the global scheduling service to transport data between nodes.

The quality of a schedule can be evaluated by the provided per-user utility, and the network-wide utilization, in terms of CPU, memory, and bandwidth usage. Schedule evaluation in an absolute sense is difficult because the scheduling problem is almost certainly NP-complete, so generating an optimal schedule for comparison is not feasible. Therefore, we must assess schedules manually (if possible), or compare them with schedules from different algorithms.

In the figure, the global scheduling service has exploited the commonality among the specifications sharing the same source, creating a multicast-like effect, but generalized to CMNs. Moreover, the service avoids wasted bandwidth by locating bandwidth-reducing computations as far upstream as possible; in Figure 6(c), video 2's `dB` node is scheduled at `pc2` rather `pc3`, which is connected to the congested link; this avoids wasting bandwidth across link `L2`. Though not shown here, user CMNs of different utility values can be combined. For example, combining the utility 1.0 CMN of one user with the 0.4 CMN of another (assuming a shared source) would result in a split path on the way to the destinations, with the `drop B` operation only operating on one arm of the split; the result is a sort of RTP-style *mixer* [10] providing a local resource adaptation on a shared stream.

While other architectures with similar global scheduling services either set up only the initial computational flow [2], or reschedule very rarely (such as when compute nodes or network links fail), MediaNet's global scheduling service operates *on-line*, performing continuous scheduling. As such, the service needs regular reports of current conditions, including changes to link and CPU/memory loads, and changes to topology. Because of delays in detecting and reporting changing information, changes to the schedule necessarily occur on the order of seconds. To mitigate these delays, user specifications can employ local adaptations, like intelligent packet dropping or upstream backpressuring.

III. Implementation

The MediaNet architecture leaves the details of the global scheduling service unspecified, admitting the possibility of a variety of implementations. Our current prototype implements the most straightforward approach: a single *global scheduler* (GS) computes a CMN subgraph for each node and sends it to a *local scheduler* (LS) running there. The LS implements the CMN and periodically reports local resource usage to the GS, which can periodically recompute and redistribute its schedules, as necessary. This approach has the benefit that since the scheduler can consider the entire network and all of its users, it can likely achieve better fairness and performance, but at the cost of scalability. Conversely, a completely dis-

tributed approach would improve scalability but likely degrade performance.

We believe that the best approach will be to use a hierarchy of GSs, each responsible for sub-components of the network and combined user CMNs. The users will provide their specifications to a top-level GS, which will aggregate all of the specifications and disseminate partitions of them to its child schedulers. These will do likewise, until ultimately a single CMN is provided to the LS for implementation on a compute node. Conversely, each LS will report its available resources to its parent GS, which will report aggregated resource amounts to its parent, and so on. Moreover, the hierarchy will be best created on-the-fly, depending on the size of the network, or its current state. For small networks (e.g. 5-15 nodes, with 5-10 users, as might be expected in the motivated disaster situation), a single GS will likely be ideal, while for larger networks, more hierarchy will reduce the system-wide effects of reconfiguration, reduce monitoring overhead, etc.

In this section, we describe our global scheduling algorithm, the implementation of the local schedulers, our monitoring methodology, and our reconfiguration protocol. The aspects of this implementation should extend naturally to a hierarchical arrangement; we leave such an extension to future work.

A. Global Scheduling

Though space restrictions prevent us from fully describing our scheduling algorithm, we summarize its key features; complete details can be found in [11]. Our framework is not tied to this particular algorithm; other approaches such as [2] and [12] could be adapted.

The challenge with any on-line scheduling algorithm is that it must be fast, while still arriving at a good schedule. To do this, we used a couple of techniques. First, rather than consider an entire search space (such as assignments of user operations to nodes, or the combinations of users' CMNs at different utility levels), we break a space into more coarse-grained pieces, and make locally optimal decisions. For example, we schedule each derived multicast tree one at a time, rather than consider all possible combinations at once. Similarly, rather than consider all possible user-utility combinations, we first find a single, optimal utility for all users, and then optimize individual user utilities.

To insert send and receive operations between user operations, the GS must choose the path they should follow; we do this using a shortest-path computation based on maximum bottleneck bandwidth. To arrive at the optimal path, we could recompute the shortest paths between user operations for every possible assignment, but this would be too expensive. We could conversely do it once at the outset of scheduling, but this would fail to discover alternate paths (among other deficiencies). We compromise by calculating the shortest paths before scheduling each derived multicast tree, which effectively exploits alternate paths. Alternatively, we could use a multi-path algorithm such as the k-best paths algorithm in [13].

Our prototype GS is written in C, consisting of about 10,000 lines of code. For the experiment presented in Section IV-C, we measured GS running times of between 1 ms and 90 ms, with the longer running times for the cases when the network was more loaded, and thus more possibilities were considered. Much of the running time is due to 'constant factors' in our implementation that we have yet to tune.

B. Local Scheduling

A LS is responsible for implementing the CMN provided by the GS. To do this, it first creates data structures that represent the CMN nodes, and sorts them topologically based on their data-flow. Next, it uses deadlines to ensure that operations are run as soon as possible after the prescribed interval, following the topological ordering. When operations are data-driven the LS simply runs the operations when frames arrive. The LSs are written in the type-safe systems language Cyclone [14], essentially a safe C, comprising roughly 13,000 lines of code.

To allow legacy applications to use MediaNet seamlessly, MediaNet's transport protocol, implemented by its inserted send and receive operations, needs to meet the API expected by the application. For example, if the application uses TCP to receive its data, then MediaNet must not only connect to that application via TCP on its last hop, but it must also ensure that data is delivered to the receiver reliably, in order, and without duplication. A UDP-based application would impose fewer requirements. MediaNet should support a variety of transport protocols between send and receive operations to maximize the performance of the system while still meeting the minimal requirements of the application. For expedience, our prototype implementation uses TCP exclusively; we plan to support other transport protocols, such as UDP, RTP [10] over UDP, and possibly others.

One benefit of using TCP within MediaNet is that it readily communicates bandwidth limitations, mitigating the need for external available bandwidth detection facilities. In particular, when the TCP send buffer fills up, the application receives an `EWOULDBLOCK` error and therefore queues its frames until more bandwidth is available. Once the application queue is filled, the consequent action depends on the application semantics. For streams that can tolerate dropped frames, like video streams, MediaNet will start dropping frames based on priority. User-supplied operations are used to set the priority (see Figure 2), supporting local adaptation. If a stream cannot tolerate lost data, then MediaNet will exert backpressure to the sending application to effectively throttle its rate (until a reconfiguration can take place). Choosing a reasonable queue size is important for reconfigurations, and we mention it further below.

C. Monitoring

For adaptive reconfiguration to be profitable, the GS must be reasonably well-informed of changes to the network, particularly those to its topology and to the loads on nodes and links. We have been focusing on bandwidth limitations in our experiments, and therefore on available bandwidth reporting; we have yet to implement a CPU monitor.

Available bandwidth detection is an ongoing area of research with no clear, general solutions as yet [15]. In particular,

various techniques trade off accuracy, overhead, and measurement time. For example, packet-pair-based estimates [15], [16], [17] can quickly predict available bandwidth with extremely low overhead (just a few packets), but only reliably so for single hop links [18], [16], including wireless links [16]. On the other hand, Jain and Dovrolis' approach using one-way delays [18] works for multi-hop paths with reasonably low overhead (on the order of a few hundred packets), but the estimation time is typically between 10–30s and is often within a couple Mbps of the "actual" value. These limits to accuracy and speed constrain the time scales and magnitude of the changes made by the global scheduling service.

In our implementation, each LS notes how much data is sent and dropped (at the application level) for a given link, and sends this information periodically to the GS. These reports provide a low-overhead, highly relevant way of assessing the available bandwidth. It is low-overhead because the information is piggy-backed on the actual stream being sent, and it is most relevant because it directly reports the value of interest: how much data can a node send across a particular link using the appropriate transport protocol?

We observe that each link report will indicate that either the link can support the bandwidth imposed on it, or it cannot. That is, either all the data intended to be sent was sent, or else frames were dropped or backpressure was applied. For the latter case, the GS knows that the link is at peak capacity, so it sets its estimate to the reported sent bandwidth (for broadcast links, reports must be aggregated). In the former case, it knows the capacity is *at least* the reported amount.

Unfortunately, this approach only provides an upper bound on the available bandwidth when a link is overloaded; this is the main drawback of the technique. To compensate, if the GS does not receive a link peak capacity report for some time, it assumes that additional bandwidth might be available and so begins "creeping" its bandwidth estimate for that link at regular intervals in the spirit of TCP's additive increase. The net effect is that eventually a reconfiguration will take place that uses the assumed-available bandwidth; if the estimation is incorrect then the new configuration will fail and another will take place to compensate. We currently increase the estimate by a constant $w = 3\%$ each second beginning $t = 5$ seconds after a peak capacity report. The choice of values for w and t essentially determines how rapidly the GS tries to find unused bandwidth.

There is a tension between monitoring and accuracy: the more frequently that monitoring information is sent, the more accurate the GS network model will be, but the more overhead there will be on network links. To reduce traffic but maintain accuracy, each LS sends non-peak reports only when the reported bandwidth increases by $\Delta = 10\%$. Peak reports are sent every $r = 1$ seconds. We have found that this approach (rather than sending *all* reports every $r = 1$ seconds) reduces monitoring traffic by roughly 75% in our experiments.

As future work, we plan to incorporate other forms of feedback and estimation into our link estimates to improve the accuracy of the GS's network view. For example, Jain and Dovrolis' technique of finding an *increasing trend* in the latencies of sent packets could be incorporated into our measurements to determine an upper bound *before* a link becomes overloaded. In general, we wish to associate "confidence" measures with link bandwidth estimates, so that mostly estimated links are not weighted as highly as those with recent measurements during scheduling. We also imagine that "link profiles" could be used to estimate unmeasured links based on past usage. Finally, we could consider testing unmeasured links after a new schedule is determined but before it is used to configure the network.

D. Reconfiguration

When a global reconfiguration is initiated, it should take effect quickly and safely, without negatively impacting perceived user quality. We do this by defining a protocol that allows old and new configurations to run in parallel until the old configuration can be removed. We use a number of mechanisms to ensure the old configuration is removed as quickly as possible, while preserving the application's expected stream semantics.

The protocol works as follows. Whenever the GS calculates a new schedule, it sends a new CMN to each LS. The LS schedules this CMN immediately upon receipt, in parallel with its existing configuration. So that the two configurations do not interfere, the GS assigns different TCP port numbers to its inserted send and receive operations. As such, these operations will establish connections, but connections to the video source and receiver outside of MediaNet (which are still using the same ports) will be delayed until they are closed by the old configuration. Next, all video source applications are notified that a reconfiguration has taken place (we do this using out-of-band TCP data from the downstream MediaNet node). They each close their connections to MediaNet and reconnect, this time connecting to the new configuration. In the meantime, the old configuration will continue to forward any data it has toward the destination; when a LS's old queues are flushed the old configuration is removed. When the last bit of old data is sent to the video receiver, the new configuration will be able to connect to it and forward its data.[2]

Using this protocol, we minimize the time during which the video source and receiver are disconnected from MediaNet; for our experiments this time averages 1 ms, which is far less than a typical video inter-frame interval of 33 ms. Even so, to reduce the total switching time we must reduce the time that the old configuration stays connected to the receiving application; during this time, frames from the new configuration queue while waiting to connect to the receiver. In the case that frames can be dropped, we reduce the old configuration's lifetime by quickly clearing its application queues via priority-based frame dropping, as described next. Otherwise, the queues must clear naturally; this suggests that when frames cannot be dropped, application queue lengths should be relatively short, so as to permit quicker reconfigurations.

We initiate frame dropping in two ways. First, we tie together the queue lengths of connections using the same link, so that higher priority new packets can force the dropping

[2]Our API assumes that applications will be able to reconnect or be disconnected as described; we have used proxies to support legacy applications.

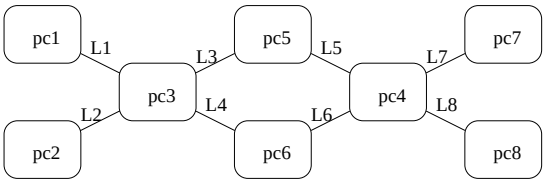


Fig. 3. Experimental Topology on Emulab.

of lower-priority old ones sharing the same link. Second, for those cases in which the old and new paths are not shared, we set a drop timer (currently going off every 0.5 s) that proactively drops increasingly higher priority frames from the old queues. Using these methods, our average reconfiguration time in the larger experiment in Section IV-C is 0.3 s, with a maximum time of about 1.1 s; these times are easily within the buffering window of most video players.

For stability, global reconfigurations are initiated at most once per *reconfiguration window* w, currently with $w = 5$ seconds. The larger this window, the less adaptive, but the more stable the system. We are currently experimenting with different kinds of windows for limiting reconfigurations, based on the quality of the network model, rather than on a fixed timeout.

IV. Experiments

In this section, we present experiments that measure MediaNet while delivering an MPEG video stream under various topologies and load conditions. We show that MediaNet consistently delivers good performance and efficient network utilization by effectively utilizing redundant paths, by exploiting commonality in user specifications, and by carefully locating CMN operations.

A. Configuration

The experiments were performed on *Emulab* [19], configured to use the topology shown in Figure 3. Each node is a 850MHz Intel Pentium III Processor running RedHat Linux 7.1, having 512MB RAM and 5 Intel EtherExpress Pro 10/100Mbps Ethernet cards. Emulab supports "dynamic events scheduling" to dynamically inject traffic shaping events, implemented by Dummynet nodes [20]; we use this to increase and decrease the available bandwidth on various links during our experiments. In all experiments we ran a LS on every node and the GS on `pc3`.

For the source video, we loop an MPEG video stream where I frames appear twice per second, P frames 8 times per second, and B frames 20 times per second, with average sizes 13500 B, 7625 B, and 2850 B, respectively. This video requires about 145 KB/s to send at its full rate, about 88 KB/s to send only I and P frames, and about 27 KB/s to send only I frames.

B. Exploiting Global Adaptation

To demonstrate the benefit of local adaptation under network load, and then the added benefit of global adaptation, we compare four different configurations:

- The *no adaptation* configuration consists of streaming the data at the desired play rate, oblivious to network conditions. We implement this with the MediaNet LSs only.
- The *priority-based frame dropping* configuration consists of tagging P, B, and I frames with successively higher priority, and during overload the lowest priority frames are dropped. This approximates some past approaches on intelligent frame dropping [4], [21].
- The *proactive frame dropping* configuration also consists of intelligently dropping video frames during overload. In this case, the LS observes when frames start getting dropped for a particular link, and then adapts by proactively dropping *all* B frames, and then later *all* P frames. When dropping frames, the LS will occasionally attempt to improve the configuration; i.e., if it is dropping P and B frames, it will try just dropping B frames. This configuration approximates past approaches to intelligent, in-network frame dropping, as well as end-to-end layered approaches [22] (where each frame type essentially defines a layer). In particular, the path of the data never changes, just what data is sent along that path. For this experiment, we implement this approach by using the GS but preventing it from choosing alternate paths.
- Finally, the *global adaptation* configuration uses MediaNet's GS with the user specification depicted in Figure 2.

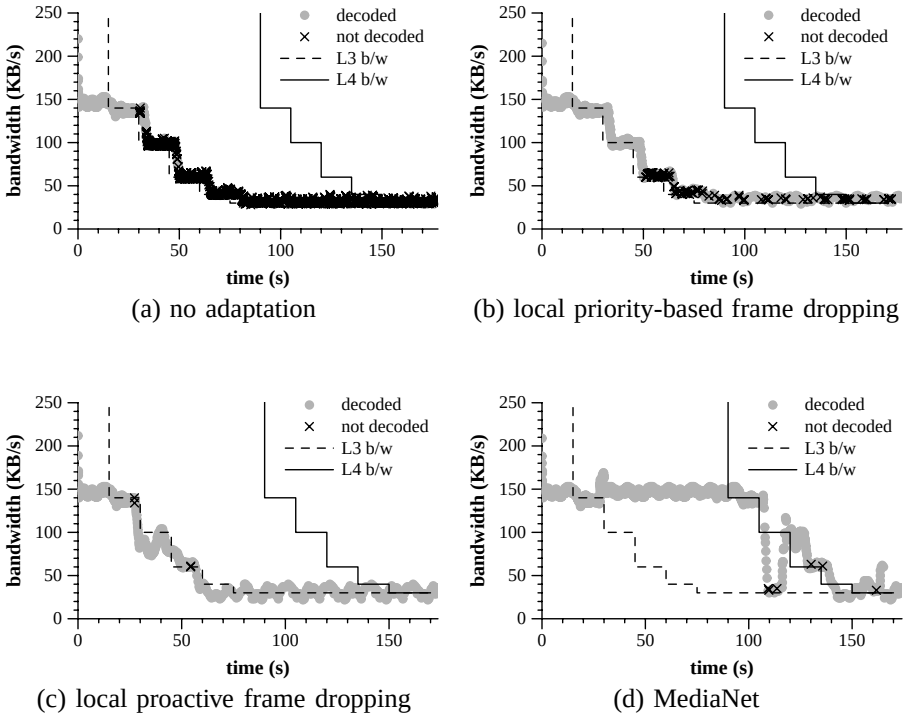
For each configuration, we ran an experiment that uses the diamond portion of our topology, with a single video sender on `pc3` and a receiver on `pc4` The experiment measures the video player's performance, in terms of the received bandwidth and the decodable frames, as we lower both link `L3`'s and `L4`'s available bandwidth over time.

Each of the graphs shown in this section has the same format. Each light gray circle in the figure is a correctly-decoded frame, while each black $\times$ is an incorrectly decoded one. The figure plots time versus bandwidth, so the x-location is the time the frame is received, and the y-location is the bandwidth seen by the player at that time (aggregated over the previous second). The available bandwidth, as set by Dummynet, is shown as dashed and/or solid lines. Dropped frames are not shown.

Figure 4(a) shows the no adaptation case. At the start, the route to the receiver is fixed along L3, and as the available bandwidth on the link drops the video quality degrades. The application cannot decode the majority of the received frames because temporally important frames (I and P frames) are being dropped. During playback, each undecodable frame manifests as a "glitch" noticeable by the user. In this case, the large and constant clumping of glitches is quite disruptive.

Figure 4(b) shows the priority-based dropping case. In this case, playback improves when dropping B frames, but remains poor under highly loaded conditions. Until roughly time 50, the player can decode all of its received frames, but after that a large fraction of frames cannot be decoded properly. By this time we are only ending I or P frames, so any dropped P frame could prevent downstream P frames from being decoded.

In contrast, when using local adaptation along the same path, the performance improves significantly, as shown in



(a) no adaptation (b) local priority-based frame dropping

(c) local proactive frame dropping (d) MediaNet

Fig. 4. User-perceived performance under diminishing bandwidth for various adaptivity schemes.

Figure 4(c). The few glitches that occur are as a result of a sudden drop in available bandwidth, and due to attempts to obtain a better configuration when no resources are available. By dropping all B and/or P frames, we avoid dropping frames that could lead to temporal glitches.

While the proactive frame-dropping adaptation significantly improves playback along a congested path, it fails to use alternative paths that could further improve playback. In contrast, MediaNet's global scheduler reconfigures the network to utilize redundant paths.[3] Figure 4(d) shows how MediaNet's GS reroutes traffic through `pc6` when `L3` becomes congested at roughly time 30, utilizing the idle `L4`. Later, `L4`'s bandwidth is reduced as well, which causes MediaNet to drop frames until it reaches the same level as the local case.

A number of times in this experiment, the GS optimistically assumes that more bandwidth is available on unmaximized links and attempts to improve the total utility. At time 105 when link L4's bandwidth drops, it tries to reroute the flow through link L3. However, L3 has even lower available bandwidth, and so after the reconfiguration window expires (here set to 5 seconds), the GS returns the configuration to link L4, at utility 0.4 (dropping B frames). Similar failed attempts occur at times 120 and 155. Our user configuration mitigates the negative effects of such reconfigurations by intelligently dropping frames until the network is reconfigured. Ideally we could prevent these spurious configurations without becoming so conservative so as to degenerate to local adaptation only; possible approaches are discussed in Section III-C.

[3]Redundant paths occur frequently in the wide area [23], and mobile hosts often have multiple networks available, e.g., many laptops have cellular, 802.11b, and Ethernet.

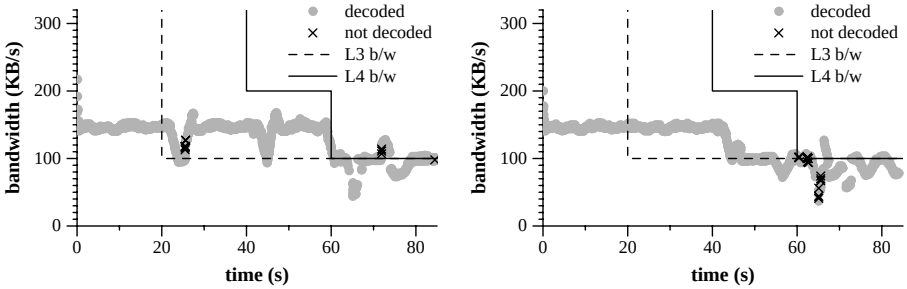
We should emphasize that MediaNet's contribution is not simply multi-path routing or local adaptation, since each has been explored in prior contexts. Rather, MediaNet's global scheduling service encapsulates a more general way of performing adaptation on a network-wide basis, based on individually-specified adaptation preferences and metrics. In so doing, it in effect employs both local adaptations (i.e., proactive frame dropping) and global adaptations (i.e., path rerouting), among others, to meet the needs of users and the network.

C. Multi-user Sharing

To examine how resources are shared among users, we configured MediaNet with two video sources and five clients. Video `v1` on `pc1` has three clients: users `user1` on `pc5`, `user3` on `pc7`, and `user5` on `pc8`; video `v2` on `pc2` has two clients: users `user2` on `pc4` and `user4` on `pc7`. Each user specification varies from the one shown in Figure 2 in the specification of the video source and user locations.

If all links are fully available, the GS assigns the operations as shown in Figure 6(a). The unlabeled operations are TCP sends and receives, and the `Pr` operations assign frame priorities for intelligent dropping. In combining the five user specifications, the GS has essentially created two multicast dissemination trees, and uses `L3` for the `v1` stream and `L4` for `v2`.

The performance as seen at the two sets of receivers is shown in Figure 5. At time 20, the bandwidth on link L3 is reduced to 100 KB/s, and so the GS reconfigures the network to be as in Figure 6(b) where all flows go along link L4 so as to maintain utility 1.0 for all users. At time 40, the bandwidth on link L4 is dropped to 200 KB/s, making it impossible to carry



(a) users 1, 3, 5 (b) users 2, 4

Fig. 5. User-perceived performance for multiple user scenario.

both streams along that link. As such, the GS reconfigures to be as in Figure 6(c), in which `v2` is sent along link L3 with its B frames dropped (as indicated by the `dB` node on `pc2`), using the utility 0.4 CMN, while `v1` goes along link L4 at utility 1.0 for all users. Notice that the GS has scheduled the `dB` (dropping B frames) node *at the source* `pc2` rather than at the node connected to the congested link, for better network utilization. At time 60, L4's bandwidth also drops to 100 KB/s, which results in all flows now operating at utility 0.4 (not shown in Figure 6). This configuration is essentially the same as the unloaded configuration in Figure 6(a) but with `dB` nodes on both of the video source hosts.

During the run, the GS guesses that additional bandwidth might be available on various links, and so attempts to improve the configuration. This occurs at time 50 (to improve to Figure 6(a)), but fails and reverts back at time 55. A similar attempt is made at time 80 (to go up to Figure 6(c)).

V. Related Work

Although distributed multi-media research has been popular for decades, the idea of multi-media processing in the network was first inspired by the problems of digital video broadcasting in heterogeneous networks [24], [25]. The goal of providing adaptive QoS for streaming data is shared by a number of systems, including *Active Networks* applications [26], [6], [4], QoS middleware substrates [21], [27], [28], and application-layer in-network processors [29], [7], [30]. Other projects have targeted the dissemination to mobile, wireless workstations, such as Quasar [31] and Odyssey [32]. None of these systems focuses on sharing resources among many users with differing adaptation preferences, though adaptivity mechanisms and resource models are quite relevant.

A few systems have considered efficient stream adaptations shared among many users. Layered multicast [22], [6], [33] shares resources efficiently among many users, and Degas [8] contains decentralized protocols for task distribution and load balancing of streaming data operations. Layered multicast layers are coarse-grained abstractions, however, and do not support more "computational adaptations" like transcoding. Degas similarly fails to account for user preferences in scheduling adaptations.

MediaNet shares some mechanisms with certain overlay networks (e.g., RON [34]) that, in addition to constructing a flexible virtual network on top of physical networks, can provide improved network performance via alternative paths in conjunction with bandwidth probing and failure detection [34], [23], [35]. To date, these systems have not been concerned with QoS (i.e., real-time constraints) of streaming data, or sharing of resources among many users.

An alternative approach to adaptive QoS is *reservation-based* QoS, in which resources like CPU and bandwidth are allocated for applications in advance [36], [37], [38]. The drawbacks of reservations are that underlying support is not widely available, and allocated resources can be underutilized, resulting in inefficiency. A number of systems looked at application-specific scheduling in reservation-capable environments, for example, the OMEGA end-system architecture [39], [40].

A number of systems share our goal of supporting user-specified, adaptive streaming data applications, including CANS [9], Conductor [41], Darwin [5], End-to-end Media Paths [2], Ninja [42], PATHS [43], and [12]. Central to all of these systems is the notion of paths of stream transformers that must be scheduled on the network, and the presence of a centralized *plan manager* to schedule paths across the network, similar to MediaNet's GS. However, these systems only use the plan manager at initialization or rarely, while MediaNet's GS runs continuously. Less attention has been paid to exploring fast, on-line scheduling algorithms that are nonetheless effective, which would be needed in a scalable on-line system. As such, these systems do not take advantage of path-based, user-specified adaptation. In addition, plan managers appear to consider scheduling only for a particular application or flow, as opposed to the combination of many or all existing applications or flows, and therefore miss opportunities to improve both per-user and network performance, for example, by aggregation and/or re-routing.

The full paper contains additional comparisons to related work [11].

VI. Conclusions

MediaNet is an architecture for user-specified, globally-adaptive QoS in distributed streaming data applications. It has two clear benefits. First, adaptations are user-specified, rather than system-determined. Second, MediaNet's global and local scheduling approach results in both global and local adaptations applied among all user flows; our experiments demonstrate better user and system performance in three ways:

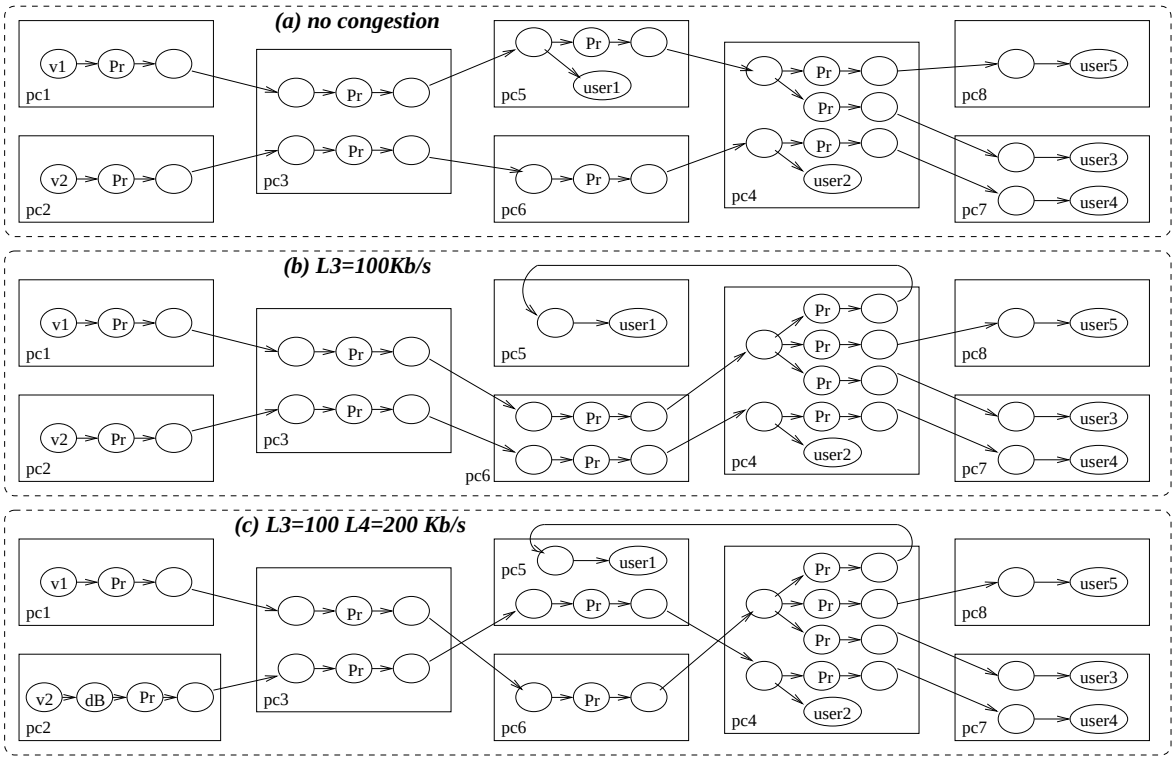


Fig. 6. Scheduling under varying conditions for multiple users.

1) The GS aggregates users' continuous media networks, removing redundancy in a multicast-like fashion.
2) It utilizes redundant resources, such as alternative, unloaded routing paths.
3) It adapts proactively to prevent wasted resources, for example by dropping frames close to the source when there is downstream congestion.

While our work is a promising first step, many interesting directions remain. To understand the scalability of the architecture, we plan to explore a hierarchical implementation of the global scheduling service, described in Section III. In addition, we intend to examine practical hindrances to growth—such as monitoring message overhead, GS running times, and network instability—to understand possible tradeoffs. As mentioned earlier (Section III-C), we are interested in employing additional monitoring techniques and better heuristics for weighing and aggregating information. Finally, we plan to explore automated means for setting or scaling user utilities to systematically ensure fair sharing of resources.

We have just scratched the surface of MediaNet's possibilities by experimenting with only network-limited (i.e., video plus frame dropping) applications. We believe that MediaNet's generality will be quite useful when considering CPU-limited cases; for example, when streaming data to embedded devices, or while performing computationally-intense transformations, such as digital facial recognition or motion analysis.

Acknowledgments

Thanks to Cyclone development team members Greg Morrisett and Dan Grossman for their rapid response to our Cyclone-related problems. Thanks also to Scott Nettles, Jonathan T. Moore, Bobby Bhattacharjee, and the anonymous reviewers for helpful comments on earlier versions of this paper.

REFERENCES

[1] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. S. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *Proceedings of the Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[2] A. Nakao, L. Peterson, and A. Bavier, "Constructing end-to-end paths for playing media objects," in *Proceedings of the IEEE Conference on Open Architectures (OPENARCH)*, Apr. 2001.

[3] "JBI - Joint Battlespace Infosphere," http://www.rl.af.mil/programs/jbi/default.cfm.

[4] S. Bhattacharjee, K. Calvert, and E. Zegura, "On Active Networking and congestion," College of Computing, Georgia Tech, Tech. Rep. GIT-CC-96-02, 1996.

[5] P. Chandra, A. Fisher, C. Kosak, T. S. E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang, "Darwin: Customizable resource management for value-added network services," in *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, Oct. 1998.

[6] R. Ramanujan and K. Thurber, "An active network-based design of a QoS adaptive video multicast service," in *Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.

[7] E. Amir, S. McCanne, and Z. Hui, "An application level video gateway," in *Proceedings of the Third ACM International Multimedia Conference and Exhibition (MULTIMEDIA)*, Nov. 1995.

[8] W. Ooi, R. van Renesse, and B. Smith, "Design and implementation of programmable media gateways," in *Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video*, June 2000.

[9] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti, "CANS: Composable, adaptive network services infrastructure," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2001.

[10] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Internet RFC 1889, 1996.

[11] M. Hicks, A. Nagarajan, and R. van Renesse, "User-specified adaptive scheduling in a streaming media network," Department of Computer Science, University of Maryland, Tech. Rep. CS-TR-4430, Jan. 2003, available at http://www.cs.umd.edu/~mwh/papers/arch-extended.pdf.

[12] S. Choi, J. Turner, and T. Wolf, "Configuring sessions in programmable networks," in *Proceedings of the IEEE INFOCOM Conference*, Apr. 2001.

[13] S. Lee and C. Wu, "A k-best paths algorithm for highly reliable communication networks," *IEICE Transactions on Communications*, vol. E82B, no. 4, Apr. 1999.

[14] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang, "Cyclone: A safe dialect of C," in *Proceedings of the USENIX Annual Technical Conference*, June 2002.

[15] J. Curtis and A. McGregor, "Review of bandwidth estimation techniques," in *New Zealand Computer Science Research Students' Conference*, vol. 8, no. 3, Apr. 2001.

[16] B. Atkin and K. P. Birman, "Evaluation of an adaptive transport protocol," in *Proceedings of the IEEE INFOCOM Conference*, April 2003.

[17] N. Hu and P. Steenkiste, "Estimating available bandwidth using packet pair probing," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-02-166, Sept. 2002.

[18] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," in *Proceedings of the ACM SIGCOMM Conference*, Aug. 2002.

[19] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Bard, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[20] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, Jan. 1997.

[21] D. Karr, C. Rodrigues, J. Loyall, R. Schantz, Y. Krishnamurthy, I. Pyarali, and D. Schmidt, "Application of the QuO quality-of-service framework to a distributed video application," in *Proceedings of the International Symposium on Distributed Objects and Applications*, Sept. 2001.

[22] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast," in *Proceedings of the ACM SIGCOMM Conference*, Aug. 1996.

[23] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of Internet path selection," in *Proceedings of the ACM SIGCOMM Conference*, September 1999.

[24] T. Turletti and J. Bolot, "Issues with multicast video distribution in heterogeneous packet networks," in *Proceedings of the Packet Video Workshop*, Sept. 1994, pp. F3.1–3.4.

[25] J. C. Pasquale, G. C. Polyzos, E. W. Anderson, and V. P. Kompella, "Filter propagation in dissemination trees: Trading off bandwidth and processing in continuous media networks," *Lecture Notes in Computer Science*, vol. 846, 1994.

[26] D. L. Tennenhouse and D. J. Wetherall, "Towards an Active Network architecture," *ACM Computer Communication Review*, vol. 26, no. 2, Apr. 1996.

[27] B. Li and K. Nahrstedt, "Dynamic reconfiguration for complex multimedia applications," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, June 1999, pp. 165–170.

[28] ——, "QualProbes: Middleware QoS profiling services for configuring adaptive applications," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware)*, 2000.

[29] N. Yeadon, A. Mauthe, D. Hutchison, and F. Garcia, "QoS filters: Addressing the heterogeneity gap," *Lecture Notes in Computer Science*, vol. 1045, 1996.

[30] E. Amir, S. McCanne, and R. Katz, "An Active Service framework and its application to real-time multimedia transcoding," in *Proceedings of the ACM SIGCOMM Conference*, Sept. 1998.

[31] J. Inouye, S. Cen, C. Pu, and J. Walpole, "System support for mobile multimedia applications," in *Proceedings of the Workshop on Network and Operating System Support for Digital Audio and Video*, May 1997.

[32] B. Noble and M. Satyanarayanan, "Experience with adaptive mobile applications in Odyssey," *Mobile Networks and Applications*, vol. 4, 1999.

[33] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet," in *Proceedings of the ACM SIGCOMM Conference*, 1999.

[34] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2001.

[35] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *Proceedings of the ACM SIGCOMM Conference*, 2002.

[36] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," Internet RFC 1633, 1994.

[37] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," Internet RFC 2475, 1998.

[38] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (rsvp)," IETF, Tech. Rep. RFC 2205, September 1997.

[39] K. Nahrstedt and J. M. Smith, "Design, implementation and experiences of the omega end-point architecture," Department of Computer and Information Science, the University of Pennsylvania, Tech. Rep. MS-CIS-95-22, 1995.

[40] ——, "The QoS broker," *IEEE Multimedia*, vol. 2, no. 1, 1995.

[41] M. Yavis, P. Reiher, and G. J. Popek, "Conductor: A framework for distributed adaptation," in *Proceedings of the IEEE Workshop on the Hot Topics in Operating Systems (HOTOS)*, Mar. 1999.

[42] S. Gribble, M. Welsh, R. Van Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z. Mao, S. Ross, and B. Zhao, "The Ninja architecture for robust internet-scale systems and services," *Computer Networks*, vol. 35, no. 4, Mar. 2001.

[43] J. M. Bjørndalen, O. J. Anshus, T. Larsen, L. Bongo, and B. Vinter, "Scalable processing and communication performance in a multi-media related context," in *Proceedings of the IEEE EUROMICRO Conference*, Sept. 2002.