Network Support for Mobile Multimedia Using a Self-adaptive Distributed Proxy

Zhuoqing Morley Mao
University of California at Berkeley
zmao@eecs.berkeley.edu

Hoi-Sheung Wilson So
University of California at Berkeley
so@eecs.berkeley.edu

Byunghoon Kang
University of California at Berkeley
hoon@eecs.berkeley.edu

ABSTRACT

Recent advancements in video and audio codec technologies (e.g., RealVideo [18]) make multimedia streaming possible across a wide range of network conditions. With an increasing trend of ubiquitous connectivity, more and more areas have overlapping coverage of multiple wired and wireless networks. Because the best network service changes as the user moves, to provide good multimedia application performance, the service needs to adapt to user movement as well as network and computational resource variations. For wireless multimedia applications, one must ensure smooth transitions when network connectivity changes. We argue that network adaptations for multimedia applications should be provided at the application layer with help from proxies in the network. The reasons are ease of programming, ease of deployment, better fault-tolerance, and greater scalability.

We propose a *self-adaptive distributed proxy system* that provides streaming multimedia service to mobile wireless clients. Our system intelligently adapts to the real-time network variations and hides handoff artifacts using application protocol specific knowledge whenever possible. It also uses application-independent techniques such as dynamic relocation of transcoders and automatic insertion of forward error correction and compression into the data transcoding path. We advocate a composable, relocatable transcoding data path consisting of a directed acyclic graph of *strongly-typed* operators to bridge any data format mismatch between the client and the data source. In this paper, we present the design, implementation, and evaluation of our system in the context of streaming video playback involving a series of transcoding proxies and a mobile client.

Keywords

adaptation, handoff, transcoding, mobile multimedia, application data path, distributed proxy

1. INTRODUCTION

Our work is motivated by the growing need to provide streaming multimedia services to any mobile client using any network and client software.

Given the increasing coverage of wireless networks, it is very common to find overlapping coverage of multiple wired and wireless networks. For instance, a mobile device may have access to one or more networks such as CDPD, Ethernet, and WaveLAN simultaneously. In addition to increasing wireless coverage, there is also a growing variety of mobile devices with varying capability, e.g., Pocket PCs with access to CDPD, laptops with access to Ricochet, and cell-phones with access to GPRS. Furthermore, multimedia services such as video-on-demand and video conferencing are rapidly gaining popularity.

Given these trends of increasing heterogeneity in terms of clients, codecs, networks, and services, there needs to be infrastructure support for multimedia services in the network to hide the access network and device heterogeneities as well as resource variations to simplify the deployment of streaming multimedia services. Moreover, when roaming, a mobile user may wish to transparently continue an ongoing multimedia service session across network changes or even across access devices changes.

We argue that roaming and adaptation support for multimedia streaming to heterogeneous clients should be provided at the application layer. Our solution is a self-adaptive distributed proxy system (Figure 1) that intelligently adapts to network variations, user mobility, and workload through passive monitoring of application performance to optimize the users' experience. It tries to avoid changes to legacy multimedia services and the underlying network. At the core of the proxy system is a middleware service, *Automatic Path Creation Service (APC)*, which provides a platform for building streaming multimedia playback to mobile clients.

The rest of the paper is organized as follows. We first examine related work. In Section 3, we illustrate our design goals and then describe the self-adaptive distributed proxy architecture in Section 4. The design and implementation of the Automatic Path Creation Service (APC) is described in Section 5. We then examine in detail our experience of how APC supports the construction of a multimedia playback application. We then summarize and conclude.

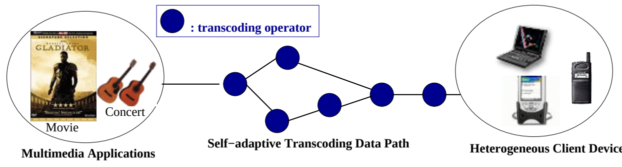

Figure 1: Distributed Proxy Architecture

2. RELATED WORK

2.1 Proxy Architecture

Our work is heavily influenced by the research in cluster based scalable network services [12]. Using the Transformation, Aggregation, Caching, and Customization (TACC) programming model, Fox et al. provided a platform to easily build new applications using service-specific modules such as filtering, down-sampling, and compression. However, TACC is limited to run on a single cluster of machines connected by a high speed system area network. The assignment of specific workers to individual machines depends only on the work load of the machines because the bandwidth within a SAN is abundant. However, one can greatly improve the service quality if the location of the computation with respect to the mobile user is taken into account.

Conductor [19] [28] provides a distributed framework for moving the complexity of network adaptation from the end-host application into the network. Each Conductor node can choose to run a set of adaptors such as schedulers, compressors, and decompressors to help applications adapt to the network path. One of the major differences between APC and Conductor is that APC takes a proactive approach in operator placement. Conductor sends a packet towards the destination to discover any Conductor-enabled nodes along the path. A planning algorithm then decides which adaptors should be deployed at each node. APC, however, places computation at a set of clusters strategically located inside the network which may not lie on the shortest path taken by an ordinary IP packet between the client and the server. In addition, the operators in APC are more general than the adaptors of Conductor in the sense that operators can have multiple inputs and multiple outputs which allows the creation of non-linear (parallel) paths as shown in Section 6. This capability is essential for supporting multimedia streams.

2.2 Handoff and Mobility Support

Our work has benefitted from research projects that study policy-based handoffs between networks (i.e., Vertical Handoffs [23]) and between cells in a single network (i.e., Horizontal Handoffs [20]). We extend their ideas to handoffs across end devices which encompass both of these types as well as handoffs of service sessions. A service session handoff occurs when the user changes the end-device used to access the network service during a service session. Traditionally, handoffs can be supported at the link, network, or transport layer. We argue that these solutions are not sufficient for our purposes due to two reasons. First, they do not address the issues of service session handoffs that involve more than simple redirection of data packets. Second, they require support from the infrastructure which may not be ubiquitous in today's heterogeneous Internet. Link layer handoff (e.g., WaveLAN) is limited to specific LANs with handoff support. Network layer solutions such as Mobile IP [16] require the deployment of a Home Agent to keep track of the current location of the mobile user. Transport layer solutions such as that in Snoeren's proposal [21] to add a new migrate option to TCP may not be compatible with existing firewalls.

APC does not exclude the use of existing techniques mentioned above. But in general, the APC service provides smooth handoffs across IP address changes at the application layer as described in Section *4.3.1*. Our approach requires no changes in the IP or the TCP layer. This is possible only because our system has complete control over the entire network path. The advantages of providing mobility support at the application layer include extensibility, ease of deployment and more intelligent handoff policies. By providing the necessary application-level gateways that translate between two network protocols (e.g., GSM-IP gateway), it is easy to extend our solution to different networks. Furthermore, user-defined handoff preferences can be easily specified in our system.

2.3 Channel Adaptation

The performance of distributed applications depends heavily on the conditions and characteristics of the communication channel. It is well known, for example, that any application built on top of an off-the-shelf TCP (e.g., TCP Reno) suffers poor performance when the communication path includes lossy wireless links. A good comparison of the effectiveness of different mechanisms to improve TCP performance over a heterogeneous wired/wireless network can be found in [4]. Two solutions, link layer retransmissions with TCP duplicate-ack suppression (Snoop) and split-TCP connection with selective acknowledgement (SPLIT-SACK) yield very good overall performance. While Snoop yields slightly better performance, it requires link layer support which may not be ubiquitous. The use of split-connections violates end-to-end TCP semantics and is usually considered undesirable. Fortunately, for our purposes, the data flow between the source of data and the user is already segmented into different transport connections and hence we are not bounded by the end-to-end semantics of TCP. Therefore, we can achieve near optimal performance using a split-connection approach by putting a proxy as close to the wired/wireless network boundary as possible, without requiring any support from the network.

For applications that use UDP or do Application Level Framing [8], the adaptation can be done at the application layer, at the proxy with the help of APC, or a combination of the two. Our architecture provides a general framework to allow automatic insertion of FEC or compression/decompression operators to adapt to the channel similar to Conductor [28] and transformer tunnels [24]. Bolot et al. in [6] [5] proposed to use FEC to protect delay-sensitive audio flows when sending over a lossy packet network. Poldolsky et al. in [17] have shown that using FEC can be beneficial even when taking into account the extra traffic generated by FEC. The idea of using FEC to protect against losses can also be extended to protect video flows [25].

3. DESIGN GOALS

- **Any-to-Any Communication:** Nowadays, mobile devices have a wide range of capabilities, e.g., in software, memory capacity, and display hardware. Transparent access to any existing service is an important goal.

- **Automated Data Format Adaptation:** The creation of a data path for the adaptation of service content to heterogeneous devices should be automated. Negotiation between devices and services for the purposes of selecting the device to receive the service content and for device handoffs should also occur without user intervention.

- **Transparency:** The proxy system must be as transparent to existing services and client applications as possible, incurring minimal changes without modifying the source code.

- **Seamless Handoffs across Networks:** During roaming, users may change their IP addresses or leave one network coverage area and enter a different one. It is therefore important to provide support for seamless handoffs across network changes while maintaining the current service session. When multiple networks are available, the selection decision should be based on the tradeoffs between the overhead and the quality of service. Cost and power considerations may also be factors and can be specified as user preferences.

- **Seamless Handoffs across Devices:** Mobile users may need to hand off a service session from one or more devices to another set of devices. Our service proxy system must support policy based, user preference driven handoffs across devices seamlessly—with minimal service disruption.

- **High Quality of Service:** Real-time multimedia applications desire low jitter, delay, and guaranteed bandwidth, thus the infrastructure must deploy various mechanisms to optimize performance.

4. SELF-ADAPTIVE DISTRIBUTED PROXY ARCHITECTURE

In this section, we describe the architecture of the distributed proxy system. Unlike the traditional proxy, which is commonly statically defined and manually configured, our system is defined and configured automatically. It dynamically adapts to resource variations and provides transparent client mobility support. The proxy system provides an application data path: a directed acyclic graph (DAG) of strongly-typed transcoding (e.g., GIF to JPEG converter) and optimization operators (e.g., FEC). The data path serves three purposes: adaptations to mismatched data formats, runtime adaptations to resource variations, and client mobility support. The creation of the data path is triggered when a client first initiates a connection to a service.

The data path imposes no modification to the server software. A slight change is nevertheless needed on the client side to intercept the communication between the client and the service. When the client first initiates a connection to the server, user input needs to be captured and relayed along with other necessary information to APC for path construction. Subsequently, client's control input for the service session are always redirected to the APC service for forwarding. This change can be achieved easily through a wrapper program without modifying the client software.

4.1 Data Format Adaptations

To achieve any-to-any communication between arbitrary client devices and multimedia services, the APC Service compiles a DAG of transcoding operators for data transformation to address any capability mismatches. The data path converts the original data flow between the client and the server to eliminate any format mismatch while optimizing the quality of service.

The path compilation process is an optimization problem with continuous feedback (Figure 2). Its objective is to provide a fault-tolerant data flow aiming for minimal delay, sufficient bandwidth, and low jitter. Our strategy combines the approaches of resource discovery, continuous passive resource monitoring, and proactive adaptation. The path compilation consists of two steps: logical path compilation and physical path creation.

4.1.1 *Logical Path Compilation*

A *logical path* consists of a DAG of operators with a single source and sink. To construct a logical path, the inputs to the APC must include the following information.

- information about the data source (i.e., multimedia service) and the data sink (i.e., client): location, data format, data rate, client's capabilities (e.g., display, memory, etc.).

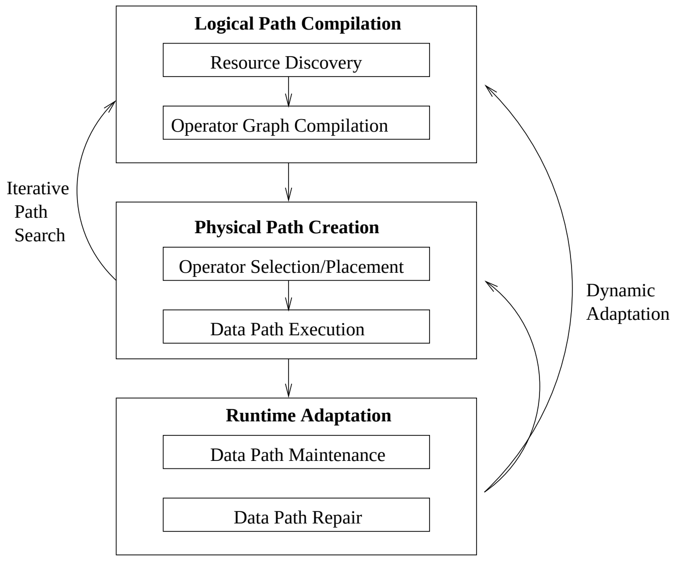

Figure 2: **Iterative Path Execution Process.**

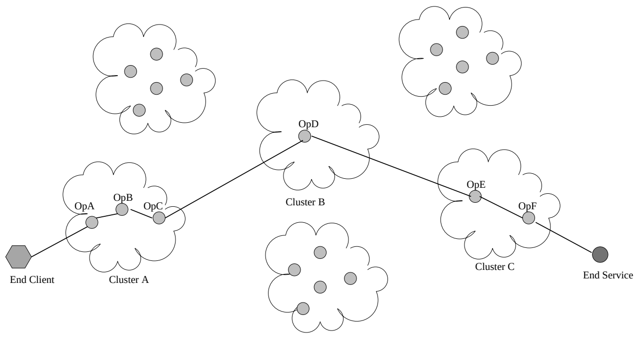

Figure 3: **An Example Wide Area Path: each circle denotes an operator; the connecting lines indicate data flows. A data path may span across multiple clusters in the wide area.**

- available locations for executing operators, i.e., APC service clusters.

- available operator software and existing operator instances for reuse,

- operators' execution requirements: restrictions on placement, needed software or hardware environment, operator properties such as input/output rate, external data source, network protocols,

- current network topology and load, and computational server load.

Some of this information can be automatically discovered by APC or obtained from other infrastructure services such as Service Discovery Services [9] . To obtain network and server load information, we propose using a wide-area monitoring service (e.g., [27]) that continuously gathers information to identify network bottleneck links and overloaded servers.

We showed in [15] that the logical path search problem can be mapped to a shortest path graph search problem. The optimization criteria are application-specific, e.g., data throughput, jitter, delay, video/audio quality metrics. Some additional consideration factors include operator instantiation latency and other path setup overhead.

4.1.2 Physical Path Creation

After the logical graph of operators is selected, their physical locations for execution are determined next by APC. Short-lived operators are created on demand and their running locations are determined. Long-lived operators in the paths are chosen from existing operator instances. A *physical path* (Figure 3) is a logical path with each operator's location determined.

Operator placement in the wide area is an optimization problem that maximizes application performance while not exceeding the available resource. Our operator placement strategies take into account network, server load information, the properties of operators and data flow, as well as workload characteristics. Each operator is benchmarked to determine its CPU and bandwidth requirement for both input and output. We define *operator execution criteria* to be those requirements in addition to other software and hardware needs. APC obtains a global view of the available computational clusters and estimates the network distances between them through active probing and passive monitoring.

Given the client's location and the requested service, APC locates the closest unloaded server to the client. We propose a greedy approach for placing operators in the path. Operators in the path are placed such that the end to end delay is minimized and operator execution criteria are satisfied. In general, operators are placed on a single cluster with the minimal aggregate distance to the data source and sink if sufficient computational power and bandwidth are available at that cluster. If that is impossible, unloaded clusters near the source and sink with good network interconnectivity are selected. Adjacent operators in the transcoding data path with high data rate are either colocated on the same machine or placed on machines in a high-speed LAN. A compression operator is inserted between operators with a high data rate when the data is compressible.

If the client uses wireless access, then all operators should be placed in the wired network. The last-hop operator is placed close to the wireless access. This arrangement increases the effectiveness of FEC encoding because the losses are only due to wireless link errors. To scale with increasing data path users, operator placement avoids creating any bottlenecks on both clusters and network links using load-balancing heuristics.

For a given logical path, APC may not find a physical path with acceptable performance due to resource constraints, thus APC may go back to the logical path compilation step illustrated by the arrow labeled "Iterative Path Search" in Figure 2.

4.2 Proactive Network and Server Load Adaptation

4.2.1 Adaptation to Resource Variations

The path execution process is iterative. It has a life cycle shown in Figure 2. Once a path has been constructed, it is modified during runtime given the feedback from the *monitoring agents* colocated with each operator. These monitoring agents passively measure at each operator to gather data on throughput, delay variation, and any other application-specific metrics to discover any performance anomalies. If the network or server performance degrades below an application-specific threshold, the agent immediately notifies the APC service for path adaptation.

Since the current Internet does not yet provide guaranteed end-to-end QoS, the goal of the runtime network adaptation performed by the APC service is to achieve service quality through proactive operator adjustment, strategic placement and dynamic relocation of computation. We claim that application-level adaptation is more flexible, extensible, and well-controlled. The potential concern is added complexity and performance overhead. Our mechanism hides this overhead as much as possible. For instance, when operators are relocated, the original data flow is switched only after the new data flow has been initiated to hide the data path migration latency. Furthermore, the proxy system is easy to deploy, because it requires essentially no modification to the client or service software. The adaptation mechanism can be easily customized given that the optimization goals may differ for different classes of multimedia applications. For instance, minimizing delay is more important for interactive applications like video-conferencing than for video-on-demand.

The proxy system deploys both application-specific and application-independent adaptation mechanisms.

- **Application-specific Adaptation:** APC makes use of application-specific information or techniques such as control channels and knowledge of application protocols and data.

  - **Application-specific control:** The application provides mechanisms for dynamic adjustment but needs external assistance. As an example, a codec may be error-resilient, but needs to be notified of the current error rate through a control channel. Another example would be dynamically adjusting the source encoding rate of video given feedback about the network conditions. In this case, APC is responsible for monitoring the resource changes and providing information to the applications to enable dynamic adaptation.
  - **Application protocols:** APC uses application protocol specific knowledge whenever possible. For instance, the Real Player reports bandwidth information back to the Real Server using the RTSP protocol so that the server can choose the data stream with the proper encoding rate. We observe from our experiments that the bandwidth information is not very accurate or timely. To improve performance, the proxy system parses the stream and replaces the RTSP control messages with more accurate bandwidth information to make applications more network-aware.
  - **Application data knowledge:** For example, differential protection of data based on type and semantics (e.g., audio vs. video) is very important in the case where the available bandwidth is not sufficient to support both.

  For this class of adaptation, APC sets up the necessary control channels for these mechanisms during path compilation. The mechanisms are dynamically triggered based on performance monitoring. We aim to overcome the application-specific complexities while still making use of them for resource adaptation.

- **Application-independent Adaptation:** APC takes advantage of the fact the proxy system has control over the entire network path of the data flow. The following mechanisms are dynamically deployed based on per application hop performance monitoring. Measurement information may contain fluctuations; therefore, the proxy system uses hysteresis to avoid instability and carefully considers the tradeoff between the adaptation overhead and possible improvement in quality of service.

- **Dynamic operator relocation** moves operators to avoid bottleneck servers and network links. Transcoders are usually easy to relocate, because they are a special class of programs with only soft state. To hide relocation latency, data buffering and prefetching are used, as well as prestarting the new data stream before switching over.
- **Dynamic insertion of computation** is another technique used to adapt to resource variations. For instance, if the network between two operators has a high error rate or becomes lossy, forward error correction (FEC) is automatically inserted to increase goodput. FEC also varies the amount of redundancy based on the measured packet loss rate and error rate.

4.2.2 Data Path Failure Recovery

Besides adapting to resource changes, another aspect of adaptation is quickly reacting to any path failures. APC provides recovery mechanisms for both operator and network failures. Path failure is detected through active probing and passive monitoring. Active probing occurs between APC and each operator's monitoring agent to detect failures at both the process and machine level. Passive monitoring is also used by taking advantage of the continuous data flow during execution. Any interruption of data flow can be an indication of a potential failure. Each operator can time out depending on the application-specific expected latency of receiving data from the previous operator in the chain. Furthermore, upon catching an I/O exception or getting an error message when an operator attempts to read or write, it can deduce that its connecting operators have failed.

Restarting a failed operator during a live data path is necessary in our recovery mechanism. From our implementation experience, transcoding operators have only soft-state and are restartable. MPEG decoders, for example, can restart in the middle of a video stream [26].

To achieve fast fault recovery, we use the following protocol to minimize the amount of path down-time and its impact on the end-users. *Partial path repair* is always attempted before rebuilding the entire path. There is usually never a need to tear down the entire path to rebuild it as long as the control path is resilient to failures. APC has built-in redundant control paths for each data path: between operators in the data path and between operators and APC.

The following discussion of path repair applies to both single and multiple operator failure. First, if failure occurs only at the process level, the failed operator is restarted on the original node. If that is unsuccessful, a new physical path is constructed to relocate the failed operator to a different processor. APC always attempts to reuse existing operators to avoid loading and initialization overhead. If none are found, the physical path construction process is repeated to find an optimized location to restart the failed operator. The connections between the failed operator and its neighboring operators are reestablished to resume the data flow. To minimize the amount of data lost, as soon as the path component failure is detected, the data flow is halted (i.e., buffered) to minimize loss of data. To achieve full reliability, lost data need to be retransmitted by the application. If the newly found physical path fails again, the logical path is rebuilt.

4.3 Client Mobility Support

4.3.1 Handoffs across networks/devices

During a service session, roaming users may experience changing network connectivity and new networks may become available for access. The APC service automatically manages handoffs to new networks. The discovery of new network interfaces or network disconnectivity occurs without user intervention. A light-weight *client proxy* that executes on the client device monitors the availability of network interfaces and signals APC when changes occur. As new devices become available, service session handoff occurs automatically.

The handoff decision across networks or devices is policy-driven. The best network connection is selected based on cost, connectivity, handoff overhead, and network characteristics (e.g., bandwidth, error rate.) The handoff overhead is weighed against the potential improvement in quality of service. During handoff, the data stream is temporarily interrupted. To minimize the disruption to the users, the proxy system proactively caches, prefetches, and retransmits data.

4.3.2 General mobility support

As users roam farther away from the last hop of the proxy chain, the performance may start to degrade due to the increasing network distance between the client and proxy. APC adapts to the changes by modifying the data path using techniques mentioned in the Section 4.2.

5. DESIGN AND IMPLEMENTATION OF APC

A distributed middleware service, *Automatic Path Creation Service* (APC) [15] is responsible for automatically creating, executing, and maintaining the data path. APC is implemented as a cluster-based, wide-area service. A bootstrapping mechanism is used for a client to first contact the APC service. It also serves to load-balance across APC service instances.

5.1 Bootstrapping and Load-balancing Mechanisms

This scheme has two alternatives: the BGP anycast mechanism (used by Inktomi) and the DNS redirection technique (used by Akamai). In the first scheme, a well-known "phantom" IP address (e.g., 1.2.3.4) is associated with the APC service. All the APC service clusters form a virtual AS. A redirector router is located in each APC service cluster, and it advertises BGP routes to this phantom address. The redirector is aware of all APC service instances in the network. When a client sends its first path creation request, the nearest redirector by the hop count metric intercepts the request and returns the physical IP address of the colocated APC cluster service. This address can be that of a transport level switch for the cluster. The switch forwards the request to a least-loaded service instance in the cluster. Notice that the APC service cluster found in this way is by the hop count metric which in some cases may not give the best application performance. To improve this scheme, the redirectors can exchange load information of the APC service clusters and monitor their network delay among each other. The redirector returns to the client the address of the least loaded APC cluster with a small network distance.

The second scheme uses modified local DNS servers which contain information about the current load of and network distance to the APC service clusters. When a client sends a request to the well-known APC service domain name, the local DNS server translates the name to the least loaded APC service cluster within close network proximity.

5.2 Performance Evaluation

To achieve scalability, the APC service consists of multiple service clusters in the wide area. Within each local-area cluster, multiple APC service instances exist to guarantee scalability and fault-tolerance. Their persistent state is kept in a fault-tolerant, distributed data structure (e.g., [13]).

APC uses an asynchronous, event-driven programming model rather than the traditional thread-based approach for better scalability and more graceful degradation in case of an overload. Over the wide area, a soft-state based signaling protocol is used to communicate across clusters and to maintain liveness information.

Our prototype implementation uses the Ninja cluster platform [11]. Table 5.2 shows good performance of a single APC service instance for a path consisting of 4 operators on 2 nodes with 200 paths continuously being created and torn down in the background. The associated path application is accessing an mp3 streaming Jukebox service using a GSM cell-phone. The measurements are performed on a local area cluster of 400MHz Pentium-II machines each with 256MB of main memory and 512KB of processor cache with a Gigabit Ethernet connection.

Table 5.2 shows that the response time for a path creation is less than 500ms. The path creation time consists of logical path compilation, physical path creation, and path instantiation time. Users typically do not care about how long it takes for the service session to terminate. In this case, it takes less than 300ms to terminate the operators. The recovery time is that of a single failed operator. The recovery mechanism involves relocating the operator on a different processor and repairing the connections. Although it takes 400ms to recover, buffering is used to reduce the perturbation to the user. The path scalability measure denotes that using two processors, we can transcode for 32 conversations (for mp3 to GSM conversion) with good performance.

Table 1: Performance of a single APC service instance for operating on 4-operator-paths on 2 nodes.

Logical and physical path creation time:	264ms
Path instantiation time:	215ms
Total path construction latency:	479ms
Path teardown time:	289ms
Path recovery from one failed operator:	402ms
Data rate:	64kbps
Path scalability:	32 concurrent paths

6. SAMPLE PATH APPLICATION

To validate the design of our self-adaptive proxy system, we built a service for real-time multimedia stream playback to a wireless client. Here we discuss our experience and show how the proxy system facilitates building such an application.

The example application is delivering an MPEG-1 video/audio stream to a mobile user using a laptop with both WaveLAN and Ethernet interfaces. The client is only capable of playing RealVideo and RealAudio format.

6.1 Data Format Adaptation

In this scenario, because the end client cannot play any MPEG stream, we must provide a service to transcode MPEG to RealVideo on the fly. Four operators are needed for the data format adaptation. The first operator, MPEG Demultiplexer (mpegdemux), is a simple program which separates an integrated MPEG Audio/Video stream into separate MPEG-1 video and MPEG-1 layer 3 (mp3) audio streams. The second operator, MPEG-1 video decoder (mpegdec) is used to decode MPEG-1 video stream into raw YUV video frames. Both mpegdemux and mpegdec are derived from a code base produced at Berkeley [7]. The third operator is a simple wrapper around the popular command line mp3 decoder called mpg123. It takes in mp3 audio and outputs sound in PCM format. The creation of these operators is very easy, not only because their source code is freely available, but also because they have a simple I/O model using either the UNIX style standard I/O or files.

The last operator needed for format conversion is a RealVideo encoder. It is developed using a toolkit called RealProducer [18]. The toolkit includes a video and audio encoder library in binary format together with the source code of some sample programs to invoke the library. We wrapped the video/audio codec in an operator which takes in an audio stream of PCM samples and a video stream of YUV frames, and produces a combined A/V stream in RealVideo format. Even though the source code of the codec was not available, the conversion of the library into an operator is relatively straightforward.

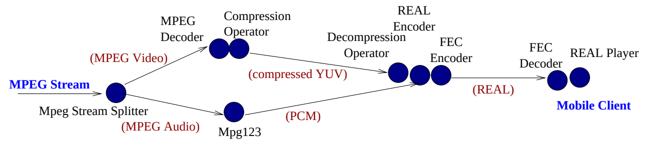

Figure 4: Example Transcoding Path: each circle denotes an operator. Operators separated by arrows may be located on separate physical machines.

Once the input and output types of the operators are encoded properly, APC automatically finds a transcoding data path as shown in Figure 4. Depending on the bandwidth available, a compression operator may be inserted between MPEGDecoder and RealEncoder to reduce the data rate. Similarly, an FEC encoder and decoder pair may be inserted if the path between the RealEncoder and the client includes a wireless link.

6.2 Mobility Adaptation

6.2.1 *Network Address Changes Detection*

The ability to detect network address changes provides an excellent opportunity for optimization for better network utilization. For example, if we know that an application is running on a laptop computer that is about to lose its wired network connectivity and later be reconnected to a wireless network of lower bandwidth, we can inform the streaming media server to send lower-quality version of the same stream immediately. One technique we use is promptly detecting the insertion of a PCMCIA network card. Once a card is inserted, a hardware interrupt is generated. As a result, Linux executes a pre-defined shell script (/etc/pcmcia/network). We modified this script such that every time a network card is inserted, its hardware address is used to lookup a table of known cards. The script contains information about a list of known cards that the user specifies. In addition to setting the IP address and gateway of the card, the script also checks whether the card is a wireless or wired network interface card (NIC). A routing metric is associated with each card so that when a high speed NIC (e.g., Fast Ethernet) and a lower speed one (e.g., Digital RoamAbout) are both available, we pick the one with a higher speed. The insertion of a wireless NIC is a good hint that the user is about to roam. Conversely, the insertion of a wired NIC indicates that the user intends to stay in physical proximity of a certain area. As soon as the card is inserted and configured, it sends a signal to the APC client-side network

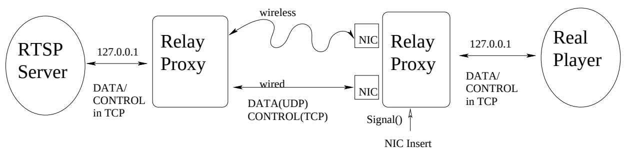

Figure 5: Redirection Proxies (Relay Proxies) for TCP and UDP Connections

monitor along with the speed and type (wireless vs. wired) of the NIC. The client-side network monitor subsequently makes use of the address change information and interacts the redirection proxy in preparation for a handoff.

6.2.2 Redirection Proxy

One objective of our system is to allow transparent service handoffs when the user is roaming. There has been a range of solutions proposed by the research community. Mobile IP [3] is one example that works at the network layer, completely transparent to the transport and application layer. Added TCP options proposed by [22] work at the transport layer. MSOCKS [14] works at the session layer by replacing the regular TCP/IP socket layer. Our approach is entirely at the application layer. We deploy TCP and UDP proxies which terminate a connection on one side and reconnect to the desired destination on the other. By adding a level of indirection between the two parties of communications, the client can roam without affecting the operations of the client or the server.

Application layer mobility support is well suited for our purposes because in APC, all operators of a path are set up and maintained by the APC service. Because the locations of all operators are known to APC, there is no need for the redirection proxies to understand the application protocol spoken by the application. The destination of the redirection can be specified by APC, which is constantly monitoring the location of the client. In fact, redirection at the application level is ideal for two reasons. First, because APC relies only on application level redirection proxies for service handoff, there is no need for any changes in the network infrastructures in which the client roams. This increases the service coverage area — an important step toward ubiquitous deployment. For example, Mobile IP has problems that the user's home network must provide a Home Agent to support Mobile IP. The second reason that application level redirection is ideal is that APC is directly involved in monitoring the location of the clients and redirecting data flow, APC now has the knowledge and flexibility to relocate computation within the network to reduce delay, increase throughput, and provide a higher level of quality of service.

In APC, we introduce a special type of operator called the *redirection proxy*. A redirection proxy is a very light-weight operator whose only purpose is to listen on a TCP or UDP socket and forward data to another outgoing socket. The source address of the incoming socket and the destination address of the outgoing socket are specified by APC and may change during the lifetime of a path. To allow uninterrupted conversation between an unmodified server and an unmodified client when the mobile client roams to a different network and obtains a different address, we need to set up a path consisting of two redirection proxies as shown in Figure 5.

The server redirection proxy resides on any machine with a fixed IP address. Its job is to provide a fixed point of correspondence from the point of view of the server. A client-side proxy sits on the same machine as the client application. The client-side proxy binds to a well known port on the loop back address 127.0.0.1 so that its IP address remains valid across IP address changes during roaming. The connection between the client application and the client-side proxy stays constant; so does the one between the server application and the server-side proxy. On the client side, a small program constantly monitors for available network interfaces and IP level connectivity as described in Section *6.2.1*.

When APC detects that the network address of the client has changed through the signal from the client-side network monitor, it notifies the client-side proxy to reconnect to the server-side proxy using the newly obtained IP as the source address. As soon as the new connection is established, data from the client to the server is sent over the new connection. The server-side proxy then immediately starts sending and receiving data through the new connection. If the transport used is UDP, this approach would work perfectly, but TCP requires reliable, in-order, and duplicate-free semantics. There is a race condition due to fact that the opening of the new connection and the closing of the old happen asynchronously. Once data is sent through the socket interface, there is no way for the redirection proxy to check the receiving status of the data sent. When the new connection is established, it is unclear if all the data sent over the old connection has been delivered. Our solution is to introduce a 3-way handshake at the application layer between the redirection proxies every time the IP address of the client changes. When the new connection is established, the client-side proxy sends a message (over the new connection) telling the server-side proxy the number of bytes it has received since the beginning of the old connection. The server-side proxy also sends a message telling the client-side the number of bytes received since the beginning of the old connection. After that, each side resumes the transmission of data starting from the offset specified by the other side. To achieve this, both proxies must buffer a certain amount of data. Fortunately, this amount of data is bounded by the size of the TCP send buffer, which can be queried by calling getsockopt() with the option name SO_SNDBUF on most operating systems. Thus, our approach does not violate the TCP semantics and yet provides transparent transport session handoff at the application layer.

6.3 Wireless Channel Adaptation

6.3.1 Differential Treatment of Control/Audio/Video

RealServer serves data to a client using a variety of application and transport protocols. Below we describe a common setting where a single TCP connection is used as the transport protocol and how APC optimizes it. RealServer and RealPlayer can use the Realtime Streaming Protocol (RTSP) [10] for controlling playback, recording, and getting multimedia clip information. RTSP is very similar to HTTP in that both define how the client requests data from the server and negotiates options with the server. Actual data (e.g., HTML, JPEG, etc) are embedded in the HTTP streams in binary format. RTSP also embeds video and audio data in the form of interleaved RTP frames among blocks of RTSP control data. The difference between HTTP and RTSP is that the RTSP client does not need to request each individual video and audio frame. The server encapsulates audio samples and video frames in RTP frames. The audio and video frames are then interleaved in between the RTSP control data sent to the client.

From a performance perspective, streaming multimedia over TCP is a very bad design choice, especially over wireless links. The congestion control mechanism of TCP decreases the sending window by at least half on each data packet loss which causes the bandwidth to fluctuate too much and increases jitter. In addition, TCP enforces reliability and in-order delivery which means that if a single packet is dropped, all subsequent packets received cannot be played back until the lost packet is retransmitted and received. When TCP runs over a wireless link, it mistakes wireless losses as congestion losses, causing the congestion window to shrink roughly proportional to $\frac{1}{\sqrt{lossrate}}$. To tackle these problems, APC differentiates application data from control information and selectively applies FEC or ARQ for error control. Control information is usually reliable and not very delay-sensitive. Therefore, control information is sent over TCP. For audio and video data in RTP format, UDP is a better choice for reasons of real-time requirements. We therefore apply FEC to RTP frames to increase their loss resilience.

6.3.2 *Forward Error Correction (FEC)*

FEC is a well-known technique for protecting data over lossy links. The use of FEC to protect data can replace traditional techniques such as retransmission (ARQ) for unreliable data. FEC is suitable for streaming multimedia data because retransmission can increase the delay of certain packets beyond their playout point and render them useless to the receiver. APC automatically inserts FEC for unreliable multimedia streams transmitted over a wireless link. In our example scenario, the link between the two redirection proxies changes from wired to wireless. At the time of change, APC quickly detects the change and inserts an FEC-encoding and an FEC-decoding operator to compensate for wireless losses to protect delay-sensitive data.

The FEC encoding that we have implemented in APC is a simple parity scheme. More effective FEC schemes can be implemented for different channels, but the design of FEC schemes is outside the scope of this paper. Packets are sent in groups, with the last packet of the group being a parity packet. Because the packets can have different lengths, the parity packet is calculated as if all packets were of a length equal to the longest packet of the group. Each RTP data packet is sent with an FEC header of the format shown in Fig. 6. Each FEC packet is sent with an FEC Parity header of the format shown in Fig. 7.

Group Seq Num 2 bytes	Group Size 1 byte	Seq Num 1 byte	RTP Packet Payload variable length

Figure 6: Data Packet Header

Group Seq Num 2 bytes	Group Size 1 byte	Seq Num=0 1 byte	Length of the packets in the group	XOR of all other packets payload in the group

Figure 7: FEC Parity Packet Header

The group sequence number identifies the group to which the packet belongs. Group size indicates the number of packets including the parity packet in each group. Sequence number field is the sequence number of a packet within a group starting at 1 for data packets. The parity packet always has a sequence number of 0. The packet header of the FEC packet has an array which stores the size of each packet in the group. This information is necessary in the reconstruction of a lost packet.

The packet payload of a parity packet is simply the XOR of all packet payloads in the group. Because XOR'ing with 0 has no effect on the result, we are able to calculate the parity of a shorter packet simply by XOR'ing the packet with the parity packet buffer over the length of the packet. Hence there is no need to find out about the length of each packet ahead of time.

This simple FEC encoding scheme can recover exactly one lost packet in each group. If the packet lost is the FEC packet, no recovery is necessary. Otherwise, we can recover a lost packet as follows. When a packet is received, its sequence number is remembered. In addition, its payload is XOR'ed with a parity buffer initialized to be zero for each group of packets. When exactly one packet is lost in a particular group, the parity buffer should contain the payload of the missing packet.

To achieve better performance, we can apply different levels of FEC for audio and video data. The amount of FEC overhead is controlled by the group size parameter. The smaller the group size, the more resilient the coding is to losses. However, it also increases the overhead. From experimenting with the code used by RealPlayer, it seems that their audio codec is extremely error-resilient. At a 50% drop rate, the audio is still intelligible but the video becomes very distorted and pauses quite often. Therefore, it makes sense to use a smaller group size for video data than for audio data.

6.4 RTSP Parsing

To separate the control data from audio and video sent so that APC can apply selective FEC on audio and video data, we implemented a simple RTSP parser. The syntax of RTSP is very similar to that of HTTP/1.1. It is a request-response protocol. Each request or response starts with a header which ends with a blank line (i.e., a pair of CR and LF characters). The header is followed by an optional body. RTSP can encapsulate any binary data in it. For our example scenario, RTP frames are encapsulated in RTSP. The encapsulated binary data in RTSP begins with a header with the character '$' followed by a 1-byte channel ID, and 2 bytes of encapsulated payload length in network byte-order. An embedded binary data block can appear between any two RTSP requests/responses blocks.

APC parses all data sent by the server into 3 separate streams: control data (RTSP), video (RTP) and audio (RTP). Each control data block is prepended with a 2-byte length field to make the parsing of the control data easier on the client-side proxy which has to merge the 3 streams back into one. Control data is sent over TCP immediately with a 2-byte header for each RTSP request/response block. Video and audio frames are sent as complete UDP datagrams without any additional header. Therefore, there are a total of three streams: one TCP stream for RTSP control information, two separate UDP streams for audio and video.

Upon receiving data from the server-side proxy, the client-side proxy merges the three streams together and sends the result to the client over a single TCP connection. The client-side proxy issues a select() system call on all 3 sockets. Because RTSP requires that the embedded data only appear in between RTSP blocks, the client-side proxy must buffer up any partial RTSP block and prevent it from being sent to the client. RTP frames received from the UDP packets have well-defined boundaries and hence can be forwarded directly to the client as soon as they arrive.

6.5 Evaluation

When the wireless network card is inserted and the Ethernet card is removed, handoff occurs immediately. No noticeable delay and loss of data occur at the client. The RealPlayer continues playing without any pause or gap. This is due to the effective network adaptation and the fact that RealPlayer buffers a few seconds of data. In Figure 8, we compare the video with and without the above-mentioned optimizations after the client switches to 1.5Mb/sec shared WaveLAN network. This clearly demonstrates the benefit of network adaptation provided by the proxy system.

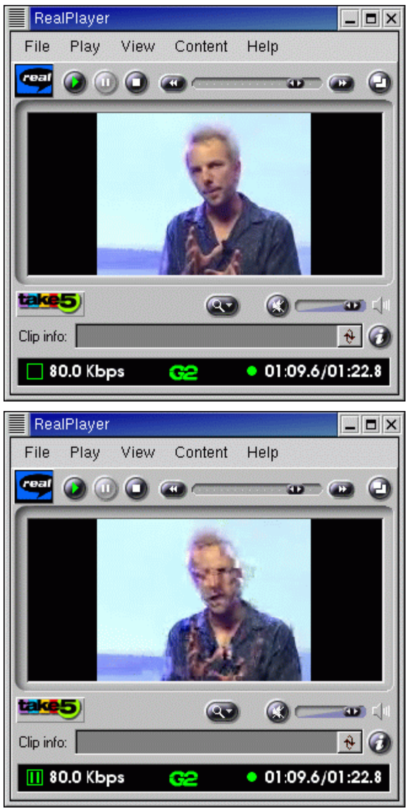

Figure 8: Evaluation: benefit of FEC over wireless link with 20% uniform error rate.

7. LESSONS AND FUTURE WORK

7.1 Creating Operators

Retrofitting legacy programs into operators is not an easy task because the source code is often unavailable. For example, a legacy program may be designed to use file I/O which must be converted to use socket I/O. Fortunately, a wide range of run-time linking tricks can be used to redirect I/O. Also, to improve performance, if two operators run on the same machine, they can be manipulated to communicate through shared memory instead of sockets as follows. When a legacy program (operator) is loaded at run time, we link it with a modified C runtime library so that all socket-related calls are trapped. To redirect file I/O to the network, all file I/O calls are remapped to socket calls. Mapping socket calls to shared memory I/O operations works in a similar way: if it is determined that a socket is connected to another socket on the local machine, then the modified runtime creates a piece of shared memory to be used for I/O between the sender and the receiver. Using shared memory avoids the overhead of going through the TCP-stack. From our experience, many of these I/O optimizations are possible only when each operator reads from the input and writes to the output sequentially. But most audio/video encoders and decoders work in this fashion.

Encoding all relevant properties of an operator in a machine readable form is also a difficult problem. It is obvious that the input and output types of each operator must be encoded in the description for APC to create a logical path. From our experience, we must also benchmark each operator to obtain a profile of the resource requirement of each operator in terms of network I/O, file I/O, memory, and CPU. Given our focus on streaming multimedia applications, CPU and network I/O performance profiles are likely to be the most critical factors in the placement decisions. In addition, the output of an operator should be measured to find out the amount of entropy contained in the data. This information is necessary to avoid making mistakes such as compressing data that are already compressed or encrypted. Each output stream should also be marked as “soft restartable” or not. A soft-restartable stream output can be decoded by the next operator even if the beginning of the stream is not available.

7.2 Fault Tolerance

Building and maintaining a fault-tolerant data path requires making both the APC service and the individual operators fault tolerant. The fault tolerance of the APC service itself is due to its design and its implementation on clusters. The fault tolerance of the individual transcoders is much more difficult to guarantee. People have proposed heavy weight mechanisms such as process pairs, checkpointing, proof-carrying code, and formal verification techniques to combat the general problem of making software systems robust. However, video and audio codecs often can start encoding or decoding from mid-stream which makes guaranteeing fault-tolerance an easier task.

Most commercial video and audio codecs provide “entry points” into a compressed stream in which a decoder can start decoding. For example, MPEG video streams contain frames divided into Groups of Pictures (GOP). Each GOP can be decoded independently. Therefore, to make a data path fault tolerant, we do not need to restore the state of a crashed decoder operator explicitly. Once crashed, a new instance of the decoder can be started. It should be able to decode starting at the next GOP. In MPEG [1] video, a GOP is on the order of 15 frames. Given a frame rate of about 30 frames per second, the expected wait for the next GOP should be less than a second. While not ideal, such a design greatly simplifies crash recovery. We are investigating addition techniques such as caching and retransmission after an operator failure to reduce this gap. As another example, H.263+ [2] video source encodes the video by sending only those blocks that have changed from previous frame. There is also a refresh message that the receiver can send to the sender to trigger a full-frame update. Once a decoder has crashed, the new instance of the decoder can send a refresh request to the sender, and the decoder now has enough information to decode the video. Similarly, any encoder that takes in raw uncompressed data can start encoding at any time. However, after an operator has crashed and restarted, audio and video synchronization can become a problem. For future work, we plan to use the timestamps embedded in video and audio streams to re-synchronize the two after a crash.

8. CONCLUSION

By passively monitoring and proactively adapting to network variations using both application-independent and application-specific techniques, we have demonstrated a pragmatic approach to provide good performance for streaming multimedia applications for mobile wireless clients. These techniques include dynamic insertion of optimization operators, strategic placement and dynamic relocation of computations, proactive caching, prefetching, and retransmission of data. Since the self-adaptive distributed proxy system has control over the network paths, it can easily deploy these mechanisms. We have shown that implementing network adaptation at the application level is flexible, extensible, and facilitates adapting existing multimedia services to wireless clients.

9. ACKNOWLEDGEMENT

We are grateful to Adam Costello for his suggestions to greatly improve the writing of this paper. We thank Professor Anthony Joseph, and the anonymous reviewers for useful comments. We also thank Samson Cheung for helpful discussion on MPEG codecs.

10. ADDITIONAL AUTHORS

Additional authors: Randy H. Katz (University of California at Berkeley, email: `randy@eecs.berkeley.edu`)

11. REFERENCES

[1] *ISO/IEC 11172-1 11172-2 11172-3 11172-4 11172-5 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s Part 1 to Part 5.*

[2] *ITU-T Recommendation H.263+.*

[3] C. P. A. Myles, D. B. Johnson. A mobile host protocol supporting route optimization and authentication. *IEEE Journal of Selected Areas in Communication*, 13(5):839–849, June 1995.

[4] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[5] J. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive fec-based error control for internet telephony. In *Proc. of IEEE INFOCOM'99*, pages 1453–1460, March 1999.

[6] J.-C. Bolot and A. Vega-Garcia. The case for FEC-based error control for packet audio in the Internet. *ACM Multimedia Systems*, 1997.

[7] Berkeley Multimedia Research Center. http://bmrc.berkeley.edu/frame/research/mpeg.

[8] D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 200–208, Philadelphia, PA, 1990.

[9] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An Architecture for a Secure Service Discovery Service. In *Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*, August 1999.

[10] H. S. et al. Rtsp: Real time stream protocol. *RFC 2326*, April 1998.

[11] S. D. G. et. al. The MultiSpace: an Evolutionary Platform for Infrastructural Services. In *Usenix Annual Technical Conference, June, 1999, Monterey, CA.*

[12] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Symposium on Operating Systems Principles*, pages 78–91, 1997.

[13] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *Proceedings of OSDI 2000*.

[14] D. Maltx and P. Bhagwat. Msocks: An architecture for transparent layer mobility. In *IEEE Infocom '98*, March 1998.

[15] Z. M. Mao and R. H. Katz. Achieving service portability in iceberg. In *Proceedings of IEEE GlobeCom 2000, Workshop on Service Portability*, March 2000.

[16] C. Perkins. IP mobility support. Technical Report Internet Draft, 1994.

[17] M. Podolsky, C. Romer, and S. McCanne. Simulation of fec-based error control for packet audio on the internet. In *Proc. of IEEE Infocom'98, San Francisco, CA*, March 1998.

[18] Real.com. *Working with RealProducer 8 codecs*, June 28, 2000.

[19] P. Reiher, M. Y. R. Guy, and A. Rudenko. Automated planning for open architectures. In *Openarch 2000 short paper*, March 2000.

[20] S. Seshan. Low-latency handoff for cellular data networks. Technical Report CSD-96-899, 1996.

[21] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proc. 6th International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[22] Snoren and Balakrishnan. An end-to-end approach to host mobility. In *Mobicom 2000*.

[23] M. Stemm and R. H. Katz. Vertical handoffs in wireless overlay networks. *ACM Mobile Networking (MONET), Special Issue on Mobile Networking in the Internet*, winter 1998.

[24] P. Sudame and B. R. Badrinath. Transformer tunnels: A framework for providing route-specific adaptations. In *Proc. of the USENIX Technical Conf.*, 1998.

[25] W. Tan and A. Zakhor. Multicast transmission of scalable video using receiver-driven hierarchical fec. In *Proc. Packet Video'99, New York*, April 1999.

[26] S. M. Weiss. Switching facilities in mpeg-2: Necessary but not sufficient. In *Proceedings SMPTE Advanced Television and Electronic Imaging Conference, San Francisco, CA*, pages 44–70, February 1995.

[27] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 1999.

[28] M. Yarvis, A.-I. A. Wang, A. Rudenko, P. Reiher, and G. J. Popek. Conductor: Distributed adaptation for complex networks. Technical Report UCLA Technical Report: CSD-TR990042, 1999.