Adaptivity for Improving Web Streaming Application Performance

Julie A. McCann
Department of Computing
Imperial College of Science, Technology and Medicine
London, UK
jamm@doc.ic.ac.uk

Introduction

We are seeing a rapid growth in multimedia applications that use the Internet. Examples of these are video on demand, video conferencing, Internet telephony and radio all requiring a high level of bandwidth. The ability to achieve high quality broadcast to users regardless of their numbers or location is becoming a challenge that many in industry and research are wishing to address. The aim is to provide a basic level of service that satisfies the receiver. For example audio and video applications serve their purpose and that interactive applications' data streams (e.g. voice with video) are synchronized. These impose quality of service (QoS) requirements on the networks in terms of throughput, packet loss, delays and jitter.

Without any form of QoS support, fluctuation in network performance can badly affect a multimedia application's performance and therefore it usefulness. These applications typically use the UDP transport protocol, which does not perform congestion control to better share resources. This means that is it possible for a resource hungry application to overload the network, starving other applications using the TCP protocol (which aims to share resources better). On the other hand there are also many problems caused by the TCP's congestion control mechanism, as it is not designed for streaming media.

There have been two main approaches to achieve better Internet performance; *guaranteed service* and *best-effort*[1]. Traditional techniques to improve bandwidth through reservation (i.e. a guaranteed level of service) are now seen as being too costly, bandwidth wasting and non-scalable. Therefore, adaptivity is an alternative solution that provides a best-effort service making use of intelligent network packet routing to intelligent application architectures rather than simply reserving an end-to-end channel. Adaptivity has the disadvantage in that it is not guaranteeing a level of service; rather it is providing a service as close to the requested level as possible. Further, in terms of overall network traffic and scalability etc. this is a better and fairer solution. Here, the QoS mechanism sits on top of the basic network synchronisation layers adapting application level functionality to suit what the network can provide. This chapter will discuss differing types of adaptive web systems presenting our Kendra system as an example. Initially we classify adaptive systems and their architecture. We then introduce Kendra and summarise some of our performance results. We will then critique Kendra based on other adaptive systems and discuss further work in this area.

Background - Improving Web Application Performance

Providing content from a single point to the user over the Internet is increasingly a non-viable activity. The bandwidth costs alone are one aspect, which have limited the growth of the Internet as a broadcast medium. When a content provider wishes to distribute their multimedia product, this raw video or audio is compressed using predetermined compression algorithms. This is the first technique used to improve the delivery of the stream as the compressed video consumes less network bandwidth in transmission than that of the raw data. The compressed object is then typically stored on the server's disk. Put simply, when a consumer requests this object, a streaming server will retrieve it, which is split into packets and sent out to the Internet. At this point packets may be dropped or may be delayed by network congestion. Once the packet send is successful i.e. a number of packets have reached the

[1] *Best-effort* is what the majority of IP Internet traffic is working with. Here the outcome is not guaranteed but all parties make a sincere attempt to do as best they can to deliver (e.g. datagrams). However with contention for services in the network some packets can be dropped therefore the deliverable is no longer guaranteed.

consumer, the packets are reassembled and decoded. Synchronization between video and audio is achieved by synchronisation mechanisms that are specific to the encoding used and are beyond the focus of this chapter. The streaming media server itself has been shown to impact greatly on the overall end-to-end performance. Such systems must be able to not only serve many users but also allow for VCR style operations such as pause, rewind and fast-forward. Typically they deploy real-time scheduling algorithms and consist of array disk systems, which provide for fault tolerance through parity and mirroring. Throughout this chapter we use the term 'server' to be synonymous with 'sender' and 'client' with 'receiver'. The following sections will look at the actual delivery of the object introducing reservation techniques and non-reservation techniques.

Reservation Techniques

Quality-of-Service (QoS) is the performance specification for a communications channel or communications system. It can be quantitatively indicated by performance parameters such as bit error ratio (BER) and message throughput rate. Resource Reservation Setup Protocol (RSVP) is an Internet protocol developed to enable the Internet to support a specified QoS. Using RSVP, an application can reserve resources along a route from source to destination. RSVP-enabled routers then prioritise packets and schedule them to fulfil the required QoS. There are a number of disadvantages with this method. Firstly it is client oriented and biased towards multicast topologies (see multicast later). It carries with it large overheads in that the admission control and policy management consumes large amounts of memory due to state information having to be stored and further maintained. Each time an update to state is required, refresh messages are sent out which further contributes to network traffic. This method also requires that RSVP be on each router, which adds to the large overheads required therefore limiting scalability. A further class of methods used to improve the stream's transmission have been proposed by Wu that have been termed *continuous media distribution services* (Wu et al 2001). Such services essentially aim to improve performance over a non reservation based *best-effort* Internet. They include, caching and content replication and multicast. Examples of these are discussed below.

Caching

Caching decreases latency, network- and server-load by storing a copy of the object closer to (or even at) the client. Web content can be quite time varying as servers offer a variety of content, some of which is generated on the fly (i.e. they not static Webpages). Furthermore, the very interconnectivity of the Web reduces locality of reference. That is, a user can fetch a document from a company web site in the UK and then follow a link for the same company but it is being delivered from the USA. Caching a resource close to a client is a useful way of spreading server load, as well as reducing network latency and load. It is especially applicable to the delivery of resources, e.g. video which is relatively immutable once published.

The idea of caching data to improve performance is a subject, which has been well researched in the contexts of file-systems and database systems. It is now common practice for the Web architecture community caching; see (Bestavros et. al. 96, Breslau et. al. 98, Rizzo et. al. 98, Williams et. al. 96). This is in terms of adding client caches to store data that is accessed by that client frequently during a given session (per-user caching), or proxy caches which store documents from the original site on behalf of the client (per-site caching). The proxy cache has the advantage that it can serve a number of clients and is ideally located at data centres, points of presence, next to firewalls or next to routers. Perhaps the most well known proxy caches are Harvest (Chankhunthod et. al. 96) and its commercial successor Squid (Wessels 96). These provide inter-cache cooperation and enable a hierarchy of caches to be built throughout the world, therefore exploiting both the temporal and spatial locality of requests. Consequently, the performance improvement gained by introducing a cache is through satisfying requests directly from the cache instead of generating traffic to and from the server.

To work effectively, the footprint of a cache should cover a large population of users. This is to increase the likelihood that two or more users will request the same resource that can then be returned at least once from the local cache. Also the cache server should be kept as local to the end-user as possible – ideally within the local area network of the organisation – so as not to flood expensive or low-bandwidth long distance links with traffic. As far back as 1993, Danzig et al showed that by providing caches in a hierarchical arrangement (or mesh) network, bandwidth could be reduced up to 42% (Chankhunthod et. al. 96).

Multicast Technology
Multicast allows data to be sent to a set of distributed servers on an MBone (Multicast Backbone) network. The server can broadcast a message to many recipients simultaneously, which makes it an efficient technology for large amounts of data. That is, many recipients share the same data source, unlike using the traditional Internet, which requires separate connections for each source-destination pair. This means that just one set of packets is transmitted for all the destinations as opposed to many two-way transmissions of data being required between multiple sites. Further, the TCP/IP protocol divides messages into packets and sends each packet independently, potentially travelling along different routes to their destination. This means that they can arrive in any order and with sizable delays between the first and last packets. In addition, each recipient of the data requires that separate packets be sent from the source to the destination. This works fine for static information, such as text and graphics, but is inefficient for real-time audio and video. With an MBone network, a single packet can have multiple destinations and is not split up until the last possible moment. This leads to more efficient transmission rates and also ensures that packets reach multiple destinations at roughly the same time. There are many barriers to the deployment of multicast within the network due to scaling problems, network management, support for error control and congestion control, and these led the way for application level multicast. Here network service providers (e.g. ISPs) create their own multicast networks and allow them to be interconnected to other multicast networks in a peering relationship. This is sometimes called *content backbones* or CDNs.

Content Delivery Networks (CDNs)
CDNs are closely tied to caching and multicast technologies. This is an alliance of network backbone/broadband providers and ISPs providing a distributed multi-tier broadcast network. Their principal purpose is to place content to the edge of the network i.e. closer to the consumer. It is essentially a network of networks that provide a high-bandwidth backbone. Partners are strategically located throughout a given geographical location to provide reliability and locality of access through redundancy, i.e. content is mirrored.

In such architectures content is replicated onto multiple servers. The decision to send a particular piece of content to a particular server is carried out based on demand. That is, the content is adaptively routed to the topographically closest or least busy server in the CDN. Monitoring the status of broadcasts as well as server and network loads, using user location information and user access patterns, drives adaptivity. Here, user clicks on the link of a particular piece of content and intelligent routing software resolves the URL pointing to the best replication of that content. Further, such systems can adapt to constraints that a particular client may put on the delivery. For example a firewall may prevent UDP transmission and therefore such systems must adapt the content stream to TCP or HTTP format. Such architectures have the advantage that they are in the position to move to a multicast format when it is deemed to be cost-effective in terms of demand for this service.

In a *best-effort* Internet there are further techniques that can be used to improve performance of web streaming applications that are end-to-end solutions, which do not require the network to support QoS. Wu describes this as being *application-layer QoS control* (Wu et al 2001). That is, the application has a control element that adapts the video/audio bit streams according to network conditions in terms of the amount of bandwidth available and required QoS as requested by the user. Using adaptivity at the application level means that the network resources as a whole are better utilised compared with reservation approaches such as RSVP in which a single application can reserve a large element of the bandwidth starving other applications. Wu describes the adaptive technique as a *rate control* method that attempts to minimise the possibility of congestion by matching the rate of the stream to the available network bandwidth. The following sections introduce such a system - Kendra.

Adaptivity and Kendra
Here we introduce our Kendra adaptive content delivery architecture. It consists of a multi-cast group of Kendra servers with caches at the edge of the network. Adaptivity is used throughout this architecture to improve performance of delivery through both locality of access benefits and adapting to fluctuation bandwidth policies. The Kendra work presented here is described in more detail in (McCann 2000, McCann et. al. 2000).

The initial objective of this project was to test the hypothesis that metadata would not only increase searching effectiveness on the Internet, but also provide optimisation parameters in data distribution and delivery systems. To this end we looked at content, metadata, representation and standards.

Instead of developing a new metadata standard we adapted the Dublin Core (Weibel et. al. 98). We were one of the first projects to use the Dublin Core with Warwick frameworks and were involved with the development of this standard during the project. However a more up-to-date solution could be the use of XML. We built two systems that used our metadata sets. The first system was an adaptive delivery mechanism that monitored Internet performance and made decisions on how best to deliver the data given the resources available. The second system allowed us to examine Internet caching. Both systems demonstrated that the use of metadata to drive adaptivity aids Internet performance significantly.

Kendra Project Demonstrator

The Kendra system delivers audio on demand over the Internet, figure 1. Two types of client are supported: those dedicated to the Kendra application (for example publicly accessible jukeboxes) and general-purpose PCs. Once a user finds a particular piece of music, they then request it using the Kendra client. The request is then matched against all the servers, which contain the data within the Kendra service group.

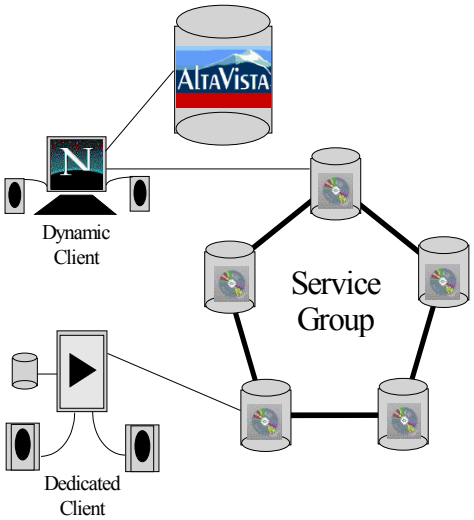

Figure 1. Kendra Architecture

For a client running on a general-purpose PC (Dynamic Client in figure 1), resource discovery is a two-stage process: it must first locate and bind to the service. For example, in the diagram, this is shown by its use of a Web browser and a search index, while a dedicated client is statically bound to the service. Once the service has been located, a client must specify the resource it desires. The Warwick Framework provides a convenient means of representing both the resources' descriptive metadata (for discovery purposes) *and* any other metadata relating to the resource, such as its formats, their temporal characteristics and even applets for rendering these formats at the client.

Metadata can help find a particular resource and it can also aid in resource delivery in several ways:

- By describing the *resource* in order to avoid delivery of subsequently unwanted data. Mixed-media data transfer can consume significant amounts of time and network resources.
- By describing the *client* resources required to present the resource, for example image-format, resolution, etc.
- By describing the properties required of the *network* when the resource has real-time deadlines, for example network bandwidth, maximum tolerable jitter, etc.

In a distributed mixed-media environment, such metadata is said to perform a quality of service (QoS) management function. Low QoS is perceptible to the user as unwanted delays and distortions to the resource being accessed. Such environments often allow the client to specify lower bounds on QoS degradation; termed the end-to-end QoS (Kerherve et. al. 96). Then client and server negotiate in order to allocate resources to satisfy the client's requirements. Re-negotiation between system components can occur when either the client requirements or the system conditions change. To detect the latter,

QoS must be monitored so that the client can be notified of a violation of its requirements and initiate re-negotiation.

Metadata for resource delivery describes the interaction between the characteristics of the transfer medium and the data being delivered. These characteristics include the capabilities of the client that is rendering the data, the server that is supplying it, and the network that is transferring it. Before resource transfer takes place, the client and server perform QoS negotiation to find the best match between the various components of the transfer characteristic. During resource transfer, QoS monitoring ensures that the negotiated conditions still remain. If they do not, re-negotiation or some predefined degradation occurs. QoS monitoring can also restore the originally negotiated conditions if the source of the degradation is transient. In the Kendra system we have focused on the delivery of audio data, as the human ear is highly sensitive to the disruption in audio delivery and is therefore a good demonstrator of our concepts.

Adaptive Delivery in Kendra

To recap, after data discovery the requesting client binds to one server and audio is delivered to the user from it. The server chosen is, firstly, the one which contains the requested audio, and beyond that it is the server that has been first to reply to the request. This is a simple way of ensuring that the server which is virtually closest to the client, or which has a less heavy load, is the one chosen to deliver the stream. Figure 3 shows the very basic client user-interface as seen by the user. Typical of rate-based adaptive Web application systems, while audio data is being delivered, the client monitors the network's bandwidth performance by looking at the data arrival rate in the client buffers. If the performance meets a lower threshold, the client then indicates that it wishes to switch to a less bandwidth hungry means of data delivery. Alternatively, when an improvement in network performance is perceived, the client indicates that it would like the quality of the data to be improved and requests a higher QoS is required i.e. the use of a bandwidth hungry delivery method. The delivery methods are graded by how much bandwidth they save, which is offset by how they affect the quality of the audio being played. The aim is to achieve automatic adaptation in which the user perceives continuous musical sound without any silence due to either decreased bandwidth or the switchover between delivery mechanisms. This is described in more detail below.

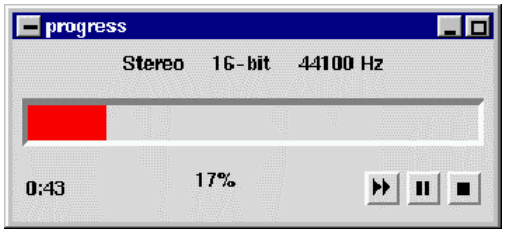

Figure 3: Client user-interface

For this application, the nature of the interaction between client and service is a stream (or flow in the terminology of (ISO, 1995)). The semantics of this interaction requires the component managing the source (the session) to transmit a continuous stream of objects (in this case, octets) to the sink, subject to flow control specified by the component managing the sink (the client). Essentially, the flow control specifies the *volume* of a 'leaky bucket' (Turner, 1986) into which data pours from the stream and which then drains into the audio device. In other words, the *volume* of the bucket represents a 'budget' up to which the stream source may transmit freely. When it has exhausted its budget (after a high threshold is reached), a further attempt to pour data into the stream will block the component driving the source (analogous to buffering). That is, the data is arriving at a rate faster than the client can 'consume' or drain (i.e. play the music). The second flow-controlling parameter denotes a low watermark on the bucket. When data drains below this level, the sink refreshes the source's budget, allowing it to fill the bucket once more. If the bucket is not filled immediately the user will experience a loss of sound. This, we describe as a '*disaster'*.

Adaptation in Kendra

Adaptive computer systems consist of three stages: change detection, adaptation agreement and action. On detecting a change requiring some form of adaptation, a control message must be sent to the user application so that it can safely initialise the next stage – i.e. agree to adapt. The system must avoid a disaster (i.e. the bucket becoming empty), which results in a period of silence. A very simple method of

adaptation is that the client can detect transient loss of bandwidth and then can decide to increase the buffer size and accommodate this short-term bandwidth loss. However, this strategy, while undemanding, is unsustainable where there are persistent adverse conditions. This may involve adaptation, which requires increased use of the CPUs (especially the server's) to transmit less data. Transmitting less data may be achieved by dropping samples (e.g., switching to Mono) or by compressing the stream, or a combination of the two. In our implementation, we selected a number of encoding methods of varying compression strengths. A system-wide map maintains a list of available compressors, legal combinations of them and the order of increasing compression. The compression algorithms used in our experiments are Mono-ADPCM, ADPCM[2], u-law[3] and 16-bit. These are dynamically loaded as required into client and server components. The Kendra low-level architecture and operation is described in more detail in (McCann).

The performance of an adaptation model like Kendra is a measurement of how well it maximises resource utilisation while minimising problems i.e. maximum QoS with the minimum number and duration of silence periods. In the Kendra model, the decision to adapt is influenced by a number of factors. However, finding such an optimal strategy is non-trivial and can be framed as a combinatorial optimisation problem (that of allocation) which is NP-hard. The factors affecting the level of sensitivity are the amount of data sampled to estimate current bandwidth and throughput. Other elements affect how optimistic or pessimistic the system is. These are: the buffer size, the disaster threshold which activates adaptivity, and the threshold level of improved throughput which needs to be observed before the system will adapt up to a better quality of service. These factors are listed in table 1, with their respective default values used for our experiments.

Particular to the Kendra model, the factor affecting the level of sensitivity is the amount of data sampled to estimate both bandwidth and throughput. These parameters are currently configured manually, although the possibility of the system managing these factors itself is considered a direction for future research.

Tuning factors	**Comment**	**Default Values**
Buffer Size	This is in addition to the buffers provided on the audio device. Playback will not (re)commence until the buffers are full. So larger buffers entail longer periods of silence during rebuffering.	1000 kilobytes
Buffer Disaster Threshold	This is the estimated number of seconds until disaster (i.e. an empty buffer) tolerable by the system before adapting to a lower quality of service (QoS). The predicted time until disaster, called the disaster horizon, is calculated as the amount of audio data in the buffer linearly regressed against time. If the disaster horizon is calculated to be less than the threshold then the system performs an adaptation to a lower QoS.	5 seconds
Buffer Samples	The amount of data buffered at the client is periodically sampled. The buffer samples parameter refers to the number of these samples that are used in the linear regression calculation to predict the disaster horizon.	20
Throughput Peak Ratio	Defines the ratio of the measured peak network throughput at the client to the network throughput required for a higher level of QoS, which must be achieved before the system will adapt to a higher level of QoS. For example if the peak ratio is set to 1.5, then the observed throughput must peak at 1.5 times the requirement for the higher QoS, before the system will adapt.	1.5. [4]
Throughput Samples	The average data throughput at the client is calculated by considering the arrival times and sizes of a number of audio packets. The throughput samples parameter refers to the number of audio packets that are applied in the calculation of the throughput. The peak	25 packets

[2] Adaptive differential pulse-code modulation

[3] u-law compression we use compresses 16-bit 44.1KHz samples at a ratio of 2:1.

[4] Peak must satisfy the ratio on 3 consecutive measurements before the system will adapt up. This reduces the possibility of the system over-optimistically adapting on the basis of a single peak measurement. Three measurements were seen as adequate in initial experimentation.

	throughput is then the highest throughput calculated since the last request for a peak value (i.e. the last time the system adapted to a better QoS) that permits the system to adapt up to a higher QoS.	

Table 1: Experimental Default values

Implementation

This section presents a more technical and implementation-oriented description of the structure of the delivery system. We begin by describing the server session, client components and interfaces.

Server Session
In the absence of adaptation, the behaviour that is required of the session component is very simple. That is, it merely reads blocks of audio data from the correct file and transmits them through the stream source. When the stream budget is exhausted, the session component's task is silently blocked until a refresh is received. Support for quality-of-service adaptation complicates matters somewhat; the mechanisms supporting adaptation are described below.

Figure 4 shows the adaptive communications system supporting the server side of the delivery process. Endpoints over which the session component transmits and receives data are shown respectively as empty and filled circles at the session component-communication interface[5]. Some endpoints support bi-directional distributed communication (i.e., *'stream-source'*). Others support unidirectional distributed communication (i.e., '*control'*). Further, other endpoints communicate information between the reactive communications system (which has 'interrupt handler' semantics) and the active component level (i.e., 'switcher').

Stream-Source
The *stream-source* provides both a sending and receiving interface. On the former it transmits blocks

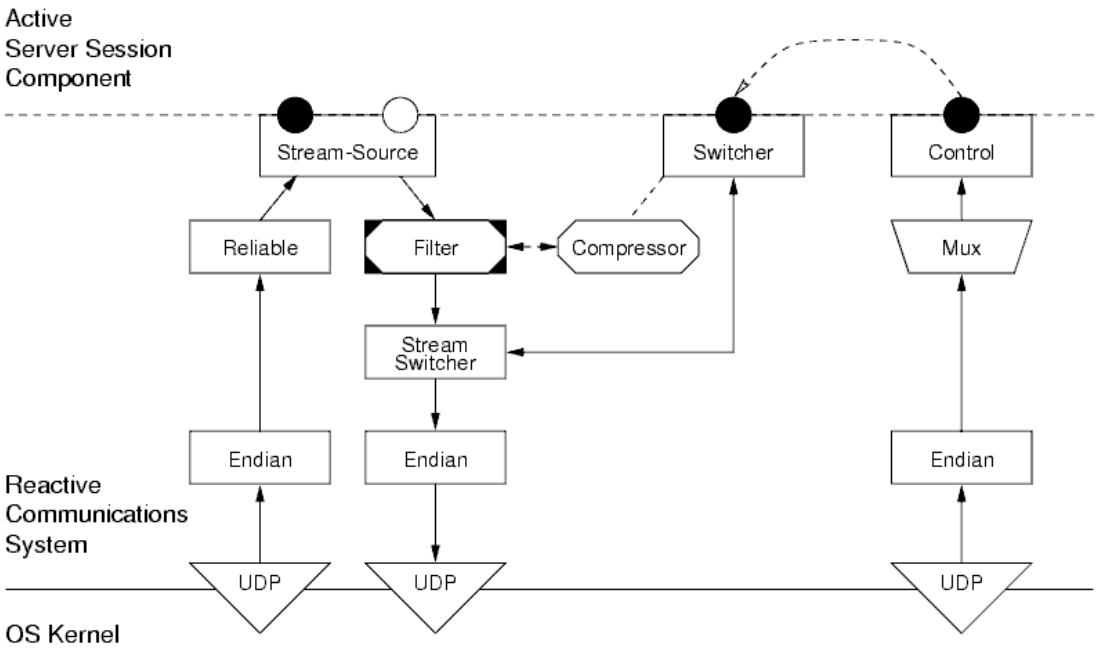

Figure 4: Adaptation infrastructure -- server side

of stream data, while on the latter, it receives notification of the arrival of budget refreshes. Each of these behaviours has an individual protocol stack. The rightmost protocol stack contains a generic

[5] This is Darwin notation (Magee94).

'*filter*' element permitting controlled modification of the stack's behaviour. Since this is used to transform the application data, and the transformation can be lossy, it is located immediately under the stream-source endpoint. Underneath the *filter*, the *stream switcher* component adds an identifier for the current filter configuration to each outgoing packet. The *endian* component simply adds a byte-ordering tag, before the data is sent to the UDP interface.

Control and Switcher
The session component receives client requests to change compressor at the *control* endpoint. On receipt of a request to change the compressor configuration, the *switcher* endpoint performs the reconfiguration and informs the *stream switcher* of the change. The ID of the compressor (the transcoding algorithm currently configured into the filter) is piggybacked on to every single packet by the *stream switcher* component. This means the server can switch as soon as it gets a request to change compressor due to less or better bandwidth detected. Therefore the client will always be using the correct compressor as it can see which compressor was used on each packet. In other words, there is no reply from the server in response to a switchover request just a change in the compressor ID, which is read by the client on getting a packet. This carries the additional benefit of handling packet loss as every packet contains an identifier (ID) of the compressor that the server applied.

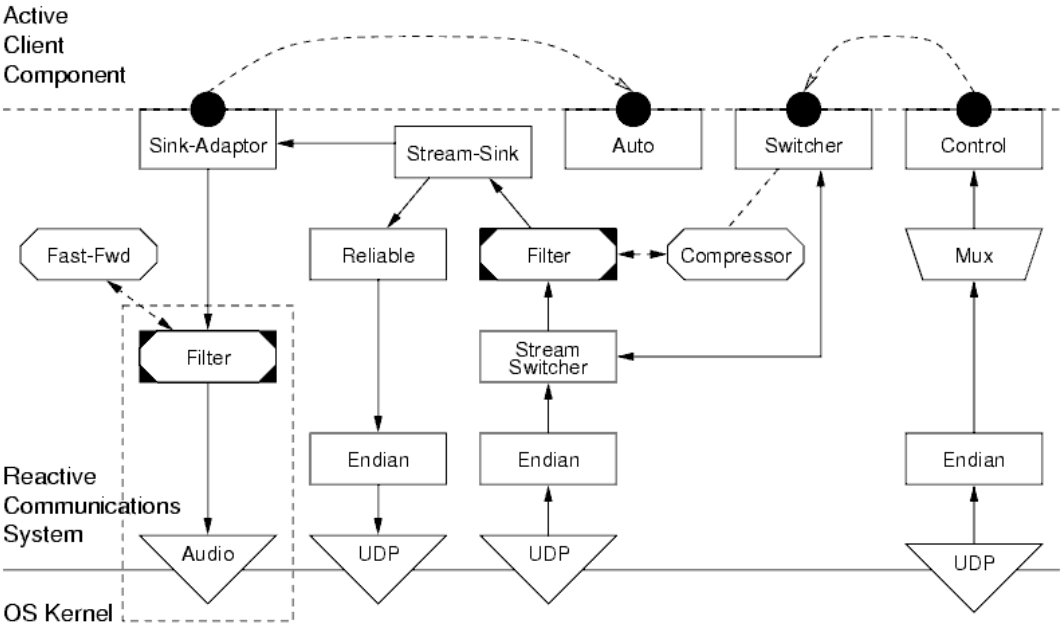

Figure 5: Adaptation infrastructure -- client side

Client

The client performs all of the quality-of-service monitoring[6]. The server merely carries out its decisions regarding what compression should be used and when is the best time to do this. The client-side adaptation infrastructure is shown in figure 5.

Stream-Sink
Resource data arrives at the rightmost UDP layer and progresses through the protocol layers above. If the client's *stream switcher* component detects a change in the compressor configuration, (i.e. the piggybacked compressor ID has changed), then the *switcher* endpoint is informed so that reconfiguration of the client's *filter* can be performed. Once the filter has transformed the data, it reaches the *stream-sink*. This layer is responsible for refreshing its peer source's budget when it

[6] Currently, the monitoring carried out by the client only takes into account network activity. The CPU requirement is not considered when selecting a compressor. This is an area for future investigation.

calculates that the budget has expired. This is based on the amount of data, which has passed since the last refresh. The *stream-sink* is parameterisable by an adaptor responsible for the forwarding of data as it arrives. The sink adaptor forwards data directly to the audio device if the latter has sufficient space in its internal buffers. Only if the audio device is full, is data queued, and a timer set to remind the adaptor to drain the queue when room becomes available. The queue is configured with a limited size, but queue overflow is not a problem as the server only sends data, which is within the budget requested by the client. Therefore reducing the number of packets lost and consequently less packet retransmissions is necessary.

Switcher
The client switcher's function is similar to the session switcher -- it switches protocols on cue; the cue being the compressor ID which is read by the client on getting a packet.

Implementation Environment

The Kendra system was implemented in Regis on a set of PC's running Linux. Regis is a middleware environment for constructing distributed programs and systems. For more information on the Regis environment see (Magee et al., 1994). The system was engineered as components as we found this was the most appropriate way to build a dynamic system that calls upon components on demand. Further implementing the adaptation in this middleware layer has the advantage of making our work more generic

Summary of Experiment Results

We initially ran our implementation in a local environment over a segment of the City University's Ethernet. This showed the adaptive delivery to be excellent in that there were no periods of silence due to bandwidth fluctuations or due to the switchover mechanism itself. We then decided to observe the effects of the adaptive data delivery over a wider area. A full account of the experiments and results can be obtained from (McCann et. al. 2000).

Using profiles of wide area bandwidth measurements we were able to experiment further. Traces of actual Internet bandwidth were gathered and these represent those profiles, this is described in much more detail in (McCann). We observed notable trends in the results of these experiments, which are summarized below:

- For networks that show good bandwidth and are not too bursty we observed that a lower disaster horizon is preferred. We also noted that the results highlighted that the number of samples (both buffer and throughput) affects the performance in that, when increased, they allow the system to deliver data at a higher quality without compromising disaster time.
- As expected, in very bursty environments we observed pessimistic settings with high disaster horizon thresholds and more samples taken to estimate systems resources (both buffer and throughput) are preferred. Therefore in bursty conditions it was better to reduce the quality of the audio so that playback could be continuous.
- In low bandwidth environments, a larger buffer size is preferable. We also found that high disaster horizon thresholds are also preferable but such environments work better with low numbers of buffer samples and with high numbers of throughput samples. This means that in low bandwidth environments it seems to be better to have settings at a non-sensitive level and yet be pessimistic about adapting downwards and a more sensitive setting and yet optimistic about adapting upwards.

Intelligent Caching in Kendra

As introduced before, caching decreases latency, network- and server-load by storing a copy of the resource closer to (or even at) the client. To do this, essentially the cache mechanism must ensure it contains the data needed by a majority of the clients or will produce the best performance saving. This requires it to adapt to the queries that are being asked of the server and the type of data that the server provides. Caches are storage areas of a finite size therefore when data is no longer useful it must be withdrawn from the cache; the cache replacement policy determines which objects to remove and is essentially the 'intelligence'. The most popular cache replacement algorithm is based on discarding the object that is the Least Recently Used (LRU) (Chankhunthod et. al. 96). An alternative mechanism employs a Least Frequently Used (LFU) policy to discard objects that are not being accessed that frequently (Chankhunthod et. al. 96). The latter policy suffers from allowing stale objects that have been referenced frequently in the past to remain unnecessarily in the cache, therefore not taking current

trends into account. LRU suffers in that it can fill the cache with object that has been accessed only once, but an object that has a higher probability of being reaccessed will be discarded due to it not being accessed of late. We found that both these weaknesses are particularly prominent when we examine the behaviour of audio and photographic website accesses. That is, our studies on media data such as music and photographs show that access behaviour can change with external trends as well as data access is very close to a Zipfian like distribution (Zipf 49). It is the exploitation of these two observations that leads us to our alternative cache replacement policy developed specifically for web proxy caches. The difference is in that it adapts to different classes of users based on their access trends. As the amount of adaptivity in this caching mechanism is low we only describe our results for brevity and completeness.

The two metrics used to quantify the relative performance of the cache replacement policies are hit-rate (HR) and byte hit-rate (BHR). Hit-rate measures the number of requests for data items which were found in the cache. A replacement strategy that achieves a high hit-rate will reflect an improvement in retrieval latency from the user's point of view --- they perceive that the object has been delivered more quickly. A replacement strategy that improves the byte hit-rate (i.e. hit-rate over size of object) reflects bandwidth savings as it lowers the volume of data being sent over the network.

The full results of our simulations are presented in (McCann 2000). The experiments took the form of simulations using access log data to generate client requests. The cache at the service proxy was implemented to simulate conventional LRU and FIFO caching policies as well as our Kendra caching model. We varied the caching policy and simulations were also run using two different cache sizes. The results in terms of response time saving or retrieval latency are very encouraging. We observed that the Kendra caching policy provides a hit-rate improvement over traditional caching methods of between 10% and 40%, with many results achieving a greater than 90% hit-rate. Our byte hit-rates were not as successful and for many experiments there was no obvious improvement in byte hit-rate, meaning that in many cases there is no network bandwidth to be saved by the Kendra policy. However two experiments did demonstrate significant byte hit-rate savings; both these sites contain significantly larger files than those of the other sites, which indicate that the Kendra policy saves bandwidth when the data is coarse grained like that found in sites which contain video/audio and photographic content.

The caching experiments show that our caching policy is suited toward sites that provide quite large data items, like graphics or music. As mentioned above we also found that for some experiments the byte-hit-rate was unexpectantly low. After analysis we believe this to be due to the long tail in the Zipfian distribution having a more significant effect on the performance of the system than first thought. The web caching community has not studied this issue; therefore we strongly believe that this needs further exploration.

Discussion and Future Work

The adaptive delivery system described above can be termed as both a source-based and receiver-based rate control system using Wu's terminology (Wu et. al. 2001). This is because it is the sender that is responsible for adapting the stream yet it does this on request from the receiver. Examples of other source-based adaptive systems can be found in Wu's approaches and directions paper (Wu et. al. 2001). These systems can be described as being either buffer-based or loss based adaptation schemes (Wang et. al. 99). Loss based schemes adjust the rate based on the packet loss experience by the receiver whereas buffer based schemes use the occupancy of a buffer on the transmission path as a measure of congestion. Kendra similarly uses a buffer mechanism but unlike traditional buffer schemes the buffer is already part of the client playback system and not an extra buffer added to the sender [e.g. see Jacobs et al in Wang]. Furthermore placing the buffer at the client end measures the rate where it is felt the most therefore potentially giving more accurate feedback. Pure receiver-driven methods typically have the sender send multiple layers, each layer being a different encoding of the stream at differing QoS levels. When the receiver detects congestion it tunes the received transmission. These methods require multicast groupings. The Kendra system, though using multicast, in theory does not require that multicast to exist. This means we do not foresee a problem with the sender serving multiple client requests at the same time as it is the receiver (client) that is dictating the stream's format and the server can just supply that format. Furthermore it does not require all streams of data to be sent over the network only, the requested stream, thus better utilising network bandwidth. On the other hand, the current version of the Kendra system is encoding the stream on demand, which was fine for our demonstration system but would not be for a production system, as it would require real-time encoding and the large computing resource to do that. Furthermore, in a non-multicast version, multiple clients

would be requesting differing streams at differing QoS levels depending on the bandwidth each client perceives, and this would put a great load on the system. Therefore pre-encoded streams would improve performance (while taking up more disk storage space). Finally, unlike lots of the research in this field we did do experiments on the sensitivity and varied the many parameters using realistic network loads. However we never studied the effects of multiple Kendra systems and how this affects the network bandwidth as a whole. This and the other aspects mentioned above are the focus of our future work.

We discussed earlier that reservation based techniques to provide QoS can waste bandwidth and do not provide for graceful degradation of application performance. Further, as we saw in our experimental results, sometimes the adaptive system will fail when there is just not enough resource. As a result, a newer promising body of research is looking at combining both adaptivity and reservation techniques. It is expected that feedback from a monitoring component will allow the application to modify resource reservations dynamically (Foster et. al. 2000). This has the potential of better sharing the network while providing a guaranteed lower level QoS to the application or user.

Conclusion

In this chapter we described why adaptive systems are becoming popular as an alternative to guarantee methods of providing QoS support. That is we showed that reservation techniques are limited by scale as well as their uptake. We introduced our adaptive audio delivery system, namely Kendra, which showed how adaptivity can improve performance. We summarised the experimental results we obtained from testing Kendra in realistic environments showing that the understanding of what is actually happening in the system is difficult given the large number of parameters that can be tuned. There is obviously more work to be carried out in the field of adaptive Internet applications. However, how well an adaptive web system adapts to its environment depends on how well we, and ultimately it, understands its environment. Currently, there is a large body of work that is looking at network characterisation and topology discovery. This research has highlighted the vast complexity involved, showing that understanding the Internet and its growth is not an easy task.

Acknowledgements

The author would like to thank Dr. Stephen Crane and Paul Howlett for their work on the Kendra project, which was carried out wile we were in City University, London[7]. The Engineering and Physical Sciences Research Council[8] in the UK funded this project and Cerbernet Ltd[9] was our Industrial Partner.

References

Bestavros A. & Cunha C. (1996). Server-initiated Document Dissemination for the WWW. In IEEE Data Engineering Bulletin, 19(3), 3-11. http://www.cs.bu.edu/faculty/best/res/papers/debull96.ps

Breslau L., Cao P.: Fan L., Phillips G. & Shenker S. (1998). On the Implications of Zipf's Law for Web Caching, Computer Sciences Department, University of Wisconsin-Madison, Technical Report no.1371, April, 1998, http://www.cs.wisc.edu/~cao/papers/zipf-implication.ps.Z

Chankhunthod P., Danzig P., Neerdaels C., Schwartz M. & Worrell K. (1996) A hierarchical Internet object cache. In Proceedings of USENIX.

Foster I., Sander V., Roy, A.(2000). A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In Proceedings of the 8th International Workshop on Quality of Service (IWQOS, 2000). 181-188.
http://www.computer.org/conferen/meta96/- kerherve/kerherve.html

[7] http://www.soi.city.ac.uk/
[8] http://www.epsrc.ac.uk/
[9] Atlantic Telecom now owns Cerbernet Ltd.

ISO/IEC JTC1/SC21/WG7 Secretariat, Standards Association of Australia, (1995). Reference Model of Open Distributed Processing, part 1: Overview. Document ITU-T X.903 | ISO/IEC 10746-1. PO Box 1055, Strathfield, NSW, Australia 2135.

Kerhervé B., A. Pons, G. von Bochmanun & A. Hafid. (1996) Metadata Modeling for Quality of Service Management in Distributed Multimedia Systems. In Proceedings of the First IEEE Metadata Conference. IEEE Press.

Magee J., Dulay N., & Kramer J. (1994) A Constructive Development Environment for Parallel and Distributed Programs. In IEE/IOP/BCS Distributed Systems Engineering, 1(5) 304-312.

McCann J.A. (2000) The Kendra Cache Replacement Policy and its Distribution. In World Wide Web An International Journal 3(4). Baltzer Science Publishers, ISSN 1386-145X. 231-240.

McCann J.A., Howlett P., & Crane J.S., (2000) Kendra: Adaptive Internet System. In the Journal of Systems and Software. 55(1), Elsevier Science, 3-17.

Rizzo L. & Vicisano L. (2000) Replacement Policies for a Proxy Cache. In IEEE/ACM Transactions on Networking. 8(2) 158-170. http://www.iet.unipi.it/~luigi/caching.ps.gz

Turner J.S. (1986) New directions in communications (or 'which way to the information age?'). In IEEE Communications. 24(10) 8-15.

Wang X. & Schulzrinne H. (1999) Comparison of Adaptive Internet Multimedia Applications. Institute of Electronics, Information and Communication Engineers Transactions, E82-B, 806-818.

Weibel S., Kunze J., Lagoze C., and Wolf M. (1998) Dublin Core Metadata for Resource Discovery. 2413 in IETF. The Internet Society, 27

Wessels D. (1996) The Squid Internet Object Cache. In Proceedings of Web Caching on Internet Conference, Warsaw, Poland.

Williams S., Abrams M.: Standridge C., Abdulla G. & Fox E.(1996). Removal Policies in Network Caches for World-Wide Web Documents, ACM SIGCOMM '96

Wu D., Hou T., Zhu W., Zhang Y-Q., Peha J. (2001). Streaming Video over the Internet: Approaches and Directions. In IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Streaming Video, 11(3), 282-300.

Zipf G. K. (1949) Human Behavior and the Principle of Least-Effort, Addison-Wesley, Cambridge, MA.