

Kendra: Adaptive Internet System¹

J.A McCann, P. Howlett, J.S Crane,

**(DAAS) Distributed and Agent Based Systems Group,
Department of Computing,
School of Informatics,
City University, London EC1 0HB, UK**

jam@soi.city.ac.uk

¹ To be published in the Journal of Systems and Software, Elsevier Science, Jan 2001
Papers can only be used for review, criticism and as part of research activities. Using them for any other purpose without the permission of the publisher or author may result in breach of copyright.

1 Abstract

This paper describes an audio delivery system model called Kendra. We focus on Kendra's adaptive delivery processes and the experimentation carried out to analyse it. The system trades off quality of sound against periods of silence, therefore, adaptation changes the format of the audio data to save bandwidth when bandwidth becomes more scarce. Typically adaptive systems are composed of a monitoring component and a switching component. The Kendra monitoring component has predictive capabilities, which advise the switching component to adapt either to a better quality or to a more bandwidth friendly format when it detects lower bandwidth. The prediction element can vary the amount of data it uses in its prediction, trading-off accuracy with the system's impedance on the performance. Such systems exhibit many other tradeoffs that determine how sensitive and/or optimistic the system is to its environment, and this is explored in our experimentation. We conclude our work by discussing these network conditions and tradeoffs and how they affect on the performance of the system.

2 Introduction

The Kendra² project consists of a distributed multi-media delivery architecture that combines adaptability with distributed caching mechanisms to improve data availability and delivery performance over the Internet. The Kendra project's industrial partner is Cerbernet³ who host virtual industrial parks; typically digital marketplaces for the video, audio/radio and film industries. One of their proposed services is a distributed digital jukebox, encompassing mainly European countries and America, which delivers music to clients over the Internet. In this paper we present the Kendra adaptive delivery mechanism which uses this particular application as a proof of concept.

Essentially, Kendra delivers audio over the Internet. For example, 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. 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 music, and beyond that it is the server which 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. While music 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 more bandwidth hungry delivery method. The delivery methods are graded by how much bandwidth they save, which is offset by how

² The Kendra project is funded by the Engineering and Physical Sciences Research Council, UK, REF: GR/K91163

³ URL for Cerbernet: <http://www.cerbernet.co.uk/>

they affect the quality of the music 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 paper focuses on the Kendra system's adaptive delivery processes and the experimentation carried out to analyse it. We introduce the delivery subsystem showing how quality-of-service demands dictate the format and resolution of the audio data to adapt in response to resource changes. The Kendra model, specifying the adaptation infrastructure and salient features of the system's implementation, is described in sections 3 and 4. Section 5 looks at the experimentation environment built to test the adaptability of the system. Section 6 presents the experimental results which are then discussed. In section 7 we review related work. Finally, in sections 8 and 9 we discuss our ongoing research and then conclude.

3 The Kendra Model

The Kendra delivery subsystem comes into play after⁴ a particular audio resource has been located and requested. It comprises two distributed entities: client and service, interconnected by a network, figure 1. The function of the service component is to accept requests for resources, validate them, and create sessions to manage their delivery.

Figure 1: Delivery subsystem schematic

In the remainder of this section, we describe the basic operation of the delivery system, concentrating on component structure.

3.1 The Basic Kendra Model

The basic Kendra system presents the user with an interface as shown in figure 2. This client front-end displays the properties of the resource as it is being transferred. These are in terms of both static properties (the resolution of the data, sampling rate, etc.) and dynamic properties (the progress meter and the elapsed time). This interface also allows the user to fast-forward, pause or halt the playback altogether.

Figure 2: 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 data source at the server session component transmits a stream of data packets to the data sink at the client. The stream of data (in this case, octets) being transmitted to the data sink, is 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

⁴ This paper focuses on the delivery of the data stream once identified not the search for a particular data item. For information, we use a Kendra browser to do this, but any standard browser can be used.

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, thus avoiding the possibility of buffer overflow. 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.

3.2 Adaptation in Kendra

Adaptation in a distributed computer system involves three stages: detection of a change in resource availability, agreement on action, and the action itself. When one of the peers in the distributed system detects a change in the system resources (e.g. CPU, network bandwidth, and memory) which are available to them, the other peers are informed to initiate the process of agreeing on the action to take. Before the adaptation can take place, all peers must agree as to the adaptation to undertake and when to perform it.

In Kendra, fluctuations in available system resources during playback may require renegotiation of the quality of service parameters agreed by client and server. Broadly there are three classes of system resource, which affect the achievable quality of service: CPU, network bandwidth and memory.

Adaptation entails trading these resources off against each other. For example, compression of the audio data consumes more CPU time, but reduces the network bandwidth requirement. The most suitable choice depends on the circumstances to which the system is attempting to adapt. In all cases, the system must avoid a disaster (i.e. the client audio buffers becoming empty) which results in a period of silence. We distinguish two kinds of adverse network conditions to which the system must adapt. Firstly, **transient** loss of bandwidth manifests itself as an increase in jitter or delay variation. If it persists longer than the time taken for the buffers to drain, it then becomes a **prolonged** loss of bandwidth. At this point the application's demands for network resources have become unsustainable.

In the first instance, the client may decide to increase the server's budget (i.e. increase the buffers available at the client) using memory to provide extra buffering and accommodate short-term bandwidth loss. However, this strategy, while undemanding, is unsustainable, and persistent adverse conditions require increased use of the CPUs to transmit less data. This may be done by dropping samples or by compressing them, or a combination of the two.

The performance of an adaptation model like Kendra, is measured in terms of how well it maximises network utilisation while minimising problems. That is, the primary aim is to avoid, as much as possible, zero sound being received by the user while delivering the sound at a maximum possible quality given the network conditions. To do this we must take into account variables such as the current actual bandwidth, predicted bandwidth over a given time span, and the choice of compression algorithm. Given that in the Kendra model the decision to adapt is influenced by these factors, finding an optimal strategy is non-trivial and can be framed as a combinatorial optimisation problem (that

of allocation) which is invariably NP-hard (Papadimitriou and Steiglitz, 1982). The optimal solution could possibly be obtained using traditional techniques such as Genetic Algorithms, but due to the necessity for the system to make adaptive decisions in soft real time, this would not be a viable approach. For example, accurate measurement of network utilisation would in itself consume too many resources. Therefore a heuristic approach is taken by Kendra, using a combination of lightweight mechanisms to measure resource usage and drive adaptation. These factors influence the sensitivity of the adaptation process and how optimistic/pessimistic the system is. An adaptive system, which is sensitive to its resources, has better potential to fully utilise its resources. This comes with a cost in that environments where the resource levels fluctuate (e.g. a bursty network), an adaptive system is more likely to spend too much effort adapting and not getting on with its primary purpose. Likewise, optimistic systems have the potential to better utilise the resources available but at the cost of not being able to adapt quickly enough in response to resource problems.

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

3.2.1 Buffer Size

This is the size of the audio buffer in kilobytes, in addition to the buffers provided on the audio device. The larger the buffer the more time the system has to react to changes in bandwidth to avoid the disaster of an empty buffer. However the buffer size also has a bearing on how long startup and rebuffering takes. Both at the start, and during rebuffering (which takes place if the buffers empty), playback will not (re)commence until the buffers are full. So larger buffers entail longer periods of silence during rebuffering.

3.2.2 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. For example, if the buffer disaster threshold is configured to be 5 seconds, then the client will only request an adaptation to a lower quality of service when the predicted disaster horizon becomes less than 5 seconds in the future. So the threshold is analogous to an optimism setting for the client. That is, the lower the threshold, the more optimistic the system is that potential disasters will not happen.

3.2.3 Buffer Samples

The amount of data buffered at the client is periodically sampled. The buffer samples parameter refers to the number of these samples which are used in the linear regression calculation to predict the disaster horizon.

3.2.4 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 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), which permits the system to adapt up to a higher QoS.

3.2.5 Throughput Peak Ratio

The throughput peak ratio parameter 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 adapt⁵.

4 Implementation

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

4.1 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 3: Adaptation infrastructure -- server side

Figure 3 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. Some endpoints support bi-directional distributed communication (i.e., '*stream-source*'). Others support unidirectional distributed communication (i.e., '*control*'). Further, other endpoints communicate information

⁵ The system is currently setup so that the 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.

between the reactive communications system (which has ‘interrupt handler’ semantics) and the active component level (i.e., ‘switcher’).

4.1.1 Stream-Source

The *stream-source* provides both a sending and receiving interface. On the former it transmits blocks of stream data, while on the latter, it receives notification of the arrival of budget refreshes. Each of these behaviours has its own protocol stack. The leftmost, figure 3, is reliable because, if a budget refresh is lost, the server component will block forever, causing a failure of resource delivery. The rightmost protocol stack contains a generic ‘*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.

4.1.2 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.

4.2 Client

The client performs all of the quality-of-service monitoring⁶ -- the server merely carries out its decisions regarding the ‘what’ and ‘when’ of adaptation. The client-side adaptation infrastructure is shown in figure 4.

Figure 4: Adaptation infrastructure -- client side

4.2.1 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 calculates that the budget has

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.

expired. This is based on the amount of data which has passed since the last refresh. The *stream-sink* is parameterisable by an adapter responsible for the forwarding of data as it arrives. (A common behaviour of other interaction types is to enqueue arrived requests and notify their owner-component; these semantics are sub-optimal for this application.) Here, the sink adapter 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 adapter 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.

4.2.2 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.

4.2.3 Other Endpoints

Other endpoints, which are not shown in the diagram, are bound to the buttons in the user interface shown in figure 2. Fast-forward reconfigures the output stack connected to the audio device with a filter which discards four packets out of every five. Another endpoint is bound to the pause button. This pauses the sound by inhibiting the transmission of the server's budget refresh.

4.2.4 Best-Effort Agreement

The result of a quality-of-service management decision typically comprises not only the type of adaptation to perform but also imposes a hard deadline, before which the adaptation must be completed, based on prevailing system conditions. However, it is well known, (Versimo, 1989), that time-bounded reliable communication is impossible in the presence of general failures.⁷

Initially we employed a crude, asymmetric, optimistic two-phase protocol. Here, in the protocol's first phase, which is always initiated by the client, a request was sent specifying the sequence number at which the specified adaptation was to take place. The responder replied, confirming the changeover time. The initiator then acknowledged this confirmation. If confirmation of the original request did not arrive within a specified time-period, it would be retransmitted. If a duplicate message was received, its successor would be retransmitted. This protocol was replaced with a more robust, simpler method whereby adaptation notification is again initiated by the client, but the confirmation that the server has received the request and has begun to adapt is piggybacked on top of the first packet containing the newly formatted data.

⁷ Reliable communication in asynchronous distributed systems requires some form of timeout and retransmit implementation mechanism. Since any reasonable timeout value is finite and the number of retransmissions potentially unbounded, it is clear that total reliability is incompatible with timeliness.

4.3 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. The main characteristics of Regis are:

- structural aspects of distributed programs are orthogonal to algorithmic concerns and are expressed in a configuration language.
- programmers are not constrained to use RPC for interaction between entities. It should be possible (and easy) for them to create their own communication classes to express the style of interaction most natural for the problem at hand.
- transport protocols are orthogonal to interaction styles. Programmers can use whatever transport layer is available and configure it to achieve the desired QoS.

For more information on the Regis environment see (Magee et al., 1994). Implementing the adaptation in this middleware layer has the advantage of making our work more generic (similar approaches can be seen in (Fitzpatrick et al., 1998; Noble et al., 1997)).

4.4 Compression Algorithms

Implementations of compression algorithms can be added to this application and dynamically loaded as required into client and server components. 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 this experiment are ADPCM⁸ and μ -law⁹, which were chosen as they are both relatively lightweight and can easily be performed in real-time on an average PC. These compression algorithms are described in (Pan, 1993). In addition to the 2 compression algorithms, filters were also made available to compress from stereo to mono and from 16 bit audio to 8 bit audio.

5 Experimentation

We initially ran our implementation in a local environment over a segment of the 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 affects of the adaptive data delivery over a wider area. This section presents these experiments and discusses the results.

5.1 Parameter Defaults

The configurable parameters used in our experiments are that which were described in sections 3.3.1-3.3.5. During our experimentation we selected a single parameter (from the set discussed in section 3) and varied it to observe how it affects performance, all other parameters were left at a default value. Table 1, describes the default values given to the parameters in the absence of any different setting mentioned for an experiment.

Table 1: Experimental Default values

Concerning buffer size, in theory we could overcome the problem of larger buffers entailing longer periods of silence during rebuffering by allowing playback to restart

⁸ Adaptive differential pulse-code modulation gives a compression ratio of 4:1

⁹ μ -law compression gives a compression ratio of 2:1.

before the buffer is completely full. Our initial experimentation showed that the smaller the buffer when you restart, the greater the risk is that a disaster is going to occur straight away again. That is, we found that waiting for the buffer to completely refill usually meant that a disaster was never followed by another disaster in quick succession, however refilling sometimes resulted in a lengthy period of silence waiting for data.

5.2 Bandwidth Statistics

In order to provide repeatable network conditions in which to experiment, bandwidth statistics were collected from four sites, using an altered version of FTP to be used to build a simulation of network bandwidth. To do this we made changes to the Linux FTP source to output throughput statistics to a file. The choice of FTP is not significant, it simply provided a mechanism to gather the network bandwidth statistics. The statistics collected were then used to control the source data output rate at the server. The bandwidth statistics can be seen in Table 2. Although the network conditions to the four sites are not necessarily representative of conditions experienced when streaming audio data over the Internet, we considered the use of real network conditions to be a better compromise than using purely simulated conditions.

Table 2. Bandwidth Profiles

6 Results

6.1 Generating Results

To plot results, the bandwidth statistics and the output from the client during the experiments are combined and plotted. Each graph shows, over a 3-minute period, the adaptation (QoS) achieved by the system during playback. A point where the adaptation drops to 0 implies that a disaster (ie silence) has occurred and rebuffering is taking place for that time. An example graph can be seen in figure 5.

Figure 5. Graph showing bandwidth and adaptation

The results are then summarised into tables which are presented in this paper for succinctness. The two most important metrics are percentage bandwidth utilised¹⁰ and disaster¹¹ as typically we expect there to be a tradeoff between them, i.e. bandwidth utilisation lowers to prevent a disaster. The results of the experiments using the profiles from the four sites are now described.

6.2 Experiment 1: Uk Hensa Data Set

Table 3 shows the results of the experiment using the Hensa data set using the default values as above in table 2. Looking at figure 6, showing the behaviour of this site, we see that for the Hensa dataset there are seven occasions where the bandwidth falls close or down to zero kB/second (the longest is a period of 6 seconds).

¹⁰ Bandwidth utilisation is a measure of the average quality of the audio being sent, this can be compared with the *Fidelity* metric in (Nobel,1997) which is defined as the degree to which data presented at a client matches the reference copy at the server.

¹¹ Disaster can be seen as a measure of *Agility*, which is defined as the speed in which a system detects and responds to changes in resource availability (Nobel, 1997), similarly *Smoothness* in (Mitchell, 1998).

Table 3: Hensa statistics & Figure 6

6.2.1 Analysis

Some of the experiments achieved very good bandwidth utilisation, as high as 111% in one case. This is possible due to the bursty nature of the source being overcome by the use of the buffering. Nevertheless this is offset by the long periods spent rebuffering, which are not desirable regardless of the high quality of audio during playback. A low setting for the number of buffer samples provides good performance. However, the use of a *BDT* of 5.0 or less consistently results in the need for more rebuffering, as much as 65.7 seconds in one case, which is over a third of the 3-minute period of the experiment. This network favours a large number of throughput samples in its throughput estimations and upward adaptation to be less sensitive (i.e. a *TPR* of 2).

6.3 Experiment 2: Sunsite Germany data set

This experiment uses the Sunsite Germany bandwidth statistics, the results of which are in Table 4 below. Note that this dataset simulates a higher average and peak bandwidth compared to the Hensa dataset in experiment 1, see figure 7. Further, note that the bandwidth falls close or down to zero only twice.

Table 4: Sunsite Germany statistics & Figure 7

6.3.1 Analysis

Buffering – The bandwidth utilisation is not significantly affected by the buffer size but the throughput steadily increases with buffer size. Further with a buffer size of greater than 250 kB there are no disasters or periods of silence as the buffers are large enough to sustain sound while the bandwidth falls near to zero.

Buffer Disaster Threshold - With a default buffer of 500 K we can see that there is no disaster periods. For *BDTs* less than 5 seconds the throughput and bandwidth utilisation are not affected. For a *BDT* of 7 seconds both the throughput the utilisation of bandwidth decreases by about 38%. Therefore larger *BDTs* cause the system to become overly pessimistic delivering the audio at a lower QoS.

Buffer Samples - With a sample size greater than 20kB we can see that the system is better predicting the throughput and is able to increase the bandwidth utilisation, with overall throughput increasing also. Setting the *BS* number to a value lower than 20 in this case proved to be very bad. This is due to the linear regression returning inaccurate disaster horizon predictions when using small samples sizes. An inaccurate disaster horizon prediction could cause the system to adapt to a lower QoS at a completely inappropriate time, thus wasting available bandwidth.

Throughput Peak Ratio - As the *TPR* increases with this dataset the throughput and bandwidth decrease. The lower the *TPR* the better the QoS level delivered and setting the *TPR* too high results in poor performance. That is, a *TPR* of 2.0 exhibited poor bandwidth

utilisation due to the system not adapting back up to higher levels of QoS because the measured throughput peaks never achieved the required ratio.

Throughput Samples - *TS* less than 75kB seem to have no real affect on throughput or bandwidth utilisation. A *TS* at 75kB shows a significant drop in both where the system is having problems in adapting back up.

Generally, the Sunsite-Germany bandwidth statistics provide a far more stable environment for the Kendra system. During the experiments the system rarely had to adapt below u-law compression and there was no disasters observed during experimentation when buffering was greater than 250kB. Buffering is again shown to be beneficial, however it seems that there is a point after which adding more buffering does not provide additional performance. For example using 1000kB of buffering only manages an extra 1% bandwidth utilisation over using 750kB of buffer space. Buffer Disaster Thresholds even lower than 5 seconds are shown to give even better bandwidth utilisation. However it remains to be seen whether such low *BDTs* would be appropriate under less stable bandwidth conditions.

6.4 Experiment 3: Sunsite in Norway.

Table 5 presents the results of the same set of experiments again using different bandwidth statistics provided by the Sunsite-Norway FTP site (figure 8). This will show if the parameters shown to be significant in the previous experiments remain so over a more bursty link.

Table 5: Sunsite Norway Statistics & Figure 8

6.4.1 Analysis

Buffering – The larger the buffer space the better the throughput and bandwidth utilisation, but note how low the quality of sound is (less than 50% bandwidth utilisation) even up to a 500kB buffer. With a 250kB buffer there are 9 episodes of silence compared with only 1 with the 750kB buffer. Further, with a buffer of 1000kB we observe relatively high disaster times due to rebuffering. This is because less time is being spent in rebuffering in the smaller 750kB buffer than the 1000kB buffer.

Buffer Disaster Threshold – As *BDT* increases the throughput decreases, as does the bandwidth utilisation. On the otherhand the level of disaster is lessened considerably. Therefore, in this type of network environment a small *BDT* of 1-3 seconds is not long enough allow the system to adapt down to a lower QoS and keep the system more stable.

Buffer Samples –The system is more stable at a low QoS level with lower BS, but when the amount of data sampled increases throughput and more interestingly periods of silence increase also (we discuss this later).

Throughput Peak Ratio – *TPR* does not significantly affect throughput, bandwidth utilisation or time taken to refill the buffer, for this data set. Looking at the graphs, we

can see that the larger the *TPR* the higher QoS is delivered, but there are 3 periods of silence as opposed to two in *TPRs* that are less than 2.

Throughput Samples – This also does not really affect the bandwidth utilisation but when the number of samples become greater than 50, the system is more able to match the delivery to the throughput and consequently there are no periods of disaster.

Generally, the Sunsite-Norway bandwidth statistics provide nothing like the stability of the previous German bandwidth. The link to Norway was very bursty which suggest that buffering would be crucial over such a link. In this case, buffering of 250kB is shown to be inadequate, and results in poor bandwidth utilisation of 31% and 22 seconds spent rebuffering. However, using a large buffer of 1000kB achieves a bandwidth utilisation of 102%. It can also be seen that the use of such a large buffer has also resulted in the system having the problem with filling its buffers. That is, 14.7 seconds were spent rebuffering after disasters compared to just 3.9 seconds for the 750kB buffer. Lower Buffer Disaster Thresholds did perform well over a bursty connection, but as before, the lower the *BDT*, despite better bandwidth utilisation, more disasters occur and hence more rebuffering is required. The number of Buffer Samples again shows the inverse relationship between bandwidth utilisation and rebuffering.

6.5 Experiment 4: Sunsite UK

We ran the same set of experiments again using the Sunsite-UK bandwidth statistics, which can be seen in figure 8.

Table 6: Sunsite UK Statistics & Figure 8

6.5.1 Analysis

The results for the *Buffering* and *Buffer Disaster Threshold* parameters show similar behaviour to the Norway result. That is, the larger the buffer the better the performance, but due to the time taken to fill up a larger buffer the more silence was observed. A higher *BDT* means slightly lower performance but with better rebuffering times.

Buffer Samples, *Throughput Peak Ratio* and *Throughput Samples* do not really affect the performance or number of disasters recorded.

As expected, buffering is a crucial component over such a bursty connection. However, even with 1000kB of buffer, over 20 seconds is spent rebuffering. Again, reducing the *BDT* to 1-second (an optimistic setting) increases the bandwidth utilisation to a more respectable 37%, but at the expense of more rebuffering time. In this case it seems beneficial to make the system act pessimistically by setting the *BDT* to 7.0 seconds or higher, which maintains a similar bandwidth utilisation and reduces the requirement for rebuffering. As before, it is shown that the default number of buffer samples of 20 is perhaps too low to give an accurate regression and disaster horizon, and using 30 or 40 buffer samples improves performance.

6.6 Results Summary

There are some notable trends in the results of these experiments which are summarised below:

- For networks that are showing good bandwidth and are not too bursty we see that the ideal buffer size is just greater than the measured peak bandwidth and a lower disaster horizon is preferred. 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 it is still feasible to have a buffer size just greater than the measured peak bandwidth. This environment prefers pessimistic settings with high disaster horizon thresholds and more samples taken to estimate systems resources (both buffer and throughput).
- In low bandwidth environments a buffer size just greater than the measured peak bandwidth is workable, but to increase the quality of sound a larger buffer is preferable. High disaster horizon thresholds are also preferable but such environments work better with low numbers of buffer samples with high numbers of throughput samples. Which 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 downward and sensitive and yet optimistic about adapting upwards.
- Although promising results were achieved by manually configuring the parameters such as disaster threshold, it is clear that better results would be attainable if the parameters themselves were dynamically adapted during a session. Dynamically adapting parameters would act like a dampening mechanism on the system's tendency to over-steer (i.e. adapt too many times leading to disaster). This would be possible by instrumenting the client to monitor the results of its own adaptation requests. Thus if a change of compressor resulted in a disaster, the client could adapt its parameters in order for adaptations to be more cautious. Dynamic adaptation of the parameters is an area for future research.

7 Related Work

There has been a recent increase in interest in configurable or extensible operating systems (Engler et al., 1995; Berhad et al., 1998), database systems (Batory et al., 1988; Boncz and Kersten, 1995) and communication subsystems (Van Renesse et al. 1996). Here the focus is the customisation of the system services to a particular application and not necessarily the adaptation of that service under a given runtime condition. Our research has concentrated on the latter.

Work on best effort delivery of video conferencing across packet-switched networks was reported in (Jaffay et al., 1994). They developed a best effort protocol that attempts to improve jitter, congestion and packet loss, and low latency. The architecture is in essence similar to ours in that it looks at buffer management to help with jitter problems. It differs in that it adapts to network congestion by changing the audio/video transport queue length and by using an algorithm to combine the audio and video frames into

network packets based on their respective deadlines. This algorithm requires a certain amount of redundancy and incurs an extra 10% bandwidth for a conference. Further, this work is very closely tied to video conferencing and the delays that are tolerable within that application domain.

The SWEB++ project's aim is to improve response time of www applications (Andresen and Yang, 1998). This system carries out its adaptation by mapping a scheduling scheme to a task graph. Using client resource information they partition and schedule the computation and communication tasks over multi-processors and their client systems in an optimum way. This system has the disadvantage that the programmer must describe the task graph in advance. Similarly (Hiltunen, 1998) used the idea of micro-protocols to allow adaptive configuration of a real-time dependable channel. This allows the customisation of communication services for different types of applications. They describe a methodology based on the understanding of relationships between software modules which describe combinations of micro-protocols which will execute correctly (in terms of performance, security and reliability).

Work on the Odyssey system is also very relevant (Nobel et al., 1997). Here they describe a framework which supports adaptation for mobile computing. This system, like ours, has the advantage that the adaptation decision is the responsibility of the application (in our case the client) which makes the framework more general. Further their system takes multi-tasking into account to balance performance for more than one application using the resources. On the otherhand, for continuously streamed media they require that the different levels of media quality is stored as tracks where the system can switch between them – unlike Kendra where we change the data's format through filters on demand, yet our performance is positively comparable to theirs. Nevertheless, their performance studies are less realistic (they do not really measure burstiness and use idealised waveforms to represent network conditions, where we have used actual network measurements). Both Odyssey and Kendra use passive resource monitoring which is more bandwidth friendly.

The closest commercial product to our system is Real Audio¹² which supports adaptive streaming. Similar to Odyssey, adaptation is achieved by storing each resource in a number of different compression formats and maintaining a time-index into each of the formats. This has the advantage for non-live resources of allowing pre-compression to be performed off-line. This also allows a much greater range of compressors to be used, especially those, such as MPEG layer-3, which cannot be performed in real time in software on today's hardware. On the otherhand, this method requires a large amount of replication redundancy.

8 Future Plans

We believe we can improve the adaptation process by building a monitoring system which takes machine load and multi-tasking into consideration. Improvements to the adaptation strategy will also allow the server to initiate adaptation (not merely for

¹² <http://www.real.com>

symmetry). This is because the client currently will find it hard to distinguish which part of the network problem is actually due to server load. Consequently, server monitoring of its own load could allow either end to calculate the load on the network. We are also looking at ranking the different adaptations where subsequent adaptation is selected based on a distance metric.

Further we would also like to experiment further by looking at the affects many Kendra sessions have on the network to see how they compete for resources and how well it adapts in that environment. This then can be compared with reservation schemes such as RSVP, ATM etc.

The audio application is itself merely an instance of a more general class of application in which data must be transferred across a network to an output device with certain timing guarantees. For example we are looking at Audio/Video streaming to see if lessons learned here can be applied there. Another example, in which disasters are more serious, is that of a CD-writer. In this case, the data stream cannot be interrupted -- if it is, the partially written CD is ruined. Current CD-writing programs recommend that, while writing, the data is stored locally and the machine is used for nothing else. We feel that the adaptive buffering mechanism could be used to accommodate transient machine load caused by concurrently executing applications. (For safe network transfer, some form of bandwidth reservation is probably necessary.)

9 Conclusion

In this paper we have described a distributed architecture which has been designed to deliver audio streams over the Internet focusing on performance improvement. This architecture ensures better use of bandwidth by adapting its data delivery method to suit network conditions. The adaptation method chosen depends on the level of performance deterioration or improvement observed in the network. This aims to avoid the situation where silence is observed by the user when listening to music while keeping the quality of sound as high as possible.

In this paper we have discussed our implementation in its controlled experimental environment. We have discussed the elements that determine how sensitive and/or optimistic the system is to its environmental resources, and how this affects the system's overall performance. We believe that this work can contribute to other fields of architecture research wishing to use dynamically bound system services and environmental adaptation for performance and reliability purposes¹³.

10 References

Andresen, D., Yang, T., SWEB++: Partitioning and Scheduling for Adaptive Client-Server Computing on WWW', Proc. of 1998 *SIGMETRICS Workshop on Internet Server*

¹³ The authors would like to thank the reviewers for their useful comments regarding this paper.

Performance, on-line proceedings <http://www.cs.wisc.edu/~cao/WISP98-program.html>
June 1998.

Batory, D., Barnett, J., Garza, J., Smith, K., Tsukuda, K., Twichell, B., Wise, T., Genesis: An Extensible Database Management System, *IEEE Transactions on Software Engineering*, 1711-1730, November 1988.

Bershad, B.N., Savage, S., Pardyak, P., Sirer, E.G., Fiuczynski, M.E., Bercker, D., Chambers, C., Eggers, S., Extensibility, Safety and Performance in the SPIN Operating System, Proceedings of the *SIGOPS'95*, Colorado, USA, 46-62, December 1995.

Boncz, P., Kersten, M.L., Monet: an impressionist sketch of an advanced database system, *BIWIT'95, Basque Int. Workshop on Information Technology*, Spain, 240-251 July 1995.

Engler, D.R., Kaashoek, M.F., O'Toole, J.W., Exokernel: An Operating System Architecture for Application-Level Resource Management, Proceedings of the *15th ACM SOSP*, Colorado, USA, 251-266, December 1995.

Fitzpatrick, T, Blair, G, Coulson G, Davies, N and Robin, P, Software Architecture for Adaptive Distributed Multimedia Applications, *IEE Proceedings – Software*, 145 (5), 163-171, October 1998.

Hiltunen, M.A., Configuration Management for Highly-Customizable Services, *The 4th International Conference on Configurable Distributed Systems*, 197-205, May 1998.

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

Jeffay, K., Stone, D.L., Smith, F.D., Transport and Display Mechanisms for Multimedia Conferencing Across Packet-Switched Networks, *Computer Networked and ISDN Systems*, 26 (10), 1281-1304, July 1994.

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

Mitchell, S., Naguib, H., Coulouris, G., Kindberg, T., Dynamically Reconfiguring Multimedia Components: A Model-Based Approach', *Proceeding of SIGOPS EW98*, Sintra, Portugal, online proceedings <http://www.acm.org/sigops/EW98/>, September, 1998

Noble, B.D., Satyanarayanan, M., Narayanan, D., Tilton, H.E., Flinn, J., Walker K.R., Agile Application-Aware Adaptation for Mobility. *Proceedings of the 16th ACM Symposium on Operating System Principles*, France, 210-222, October 1997.

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

Van Renesse, R., Birman, K., Maffeis, S., Horus, A flexible group communication system, *Communications of the ACM*, 39(4), 76-83, April 1996.

Verssimo, P., Real-Time Communication, in *Distriubted Systems* (Mullender S., ed.), Addison-Wesley, 447-490, (1989).

Pan, D. Y., Digital Audio Compression, *Digital Technical Journal*, 5(2): 1-14, 1993.

Papadimitriou, Ch.H., Steiglitz, K., *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, HJ. 1982.

11 Figures

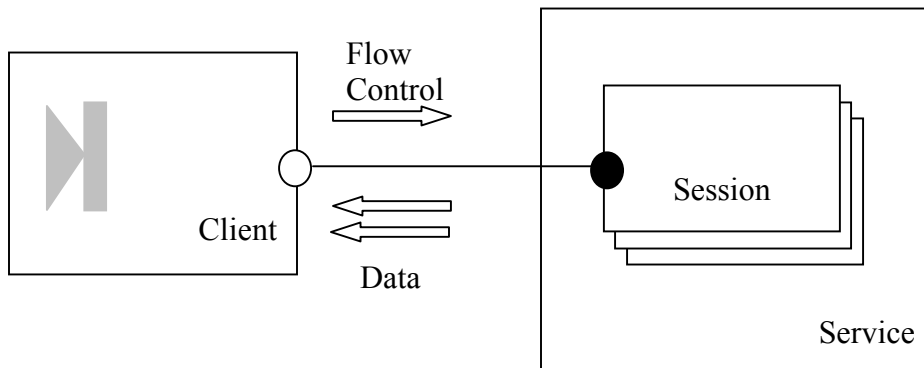


Figure 1: Delivery subsystem schematic

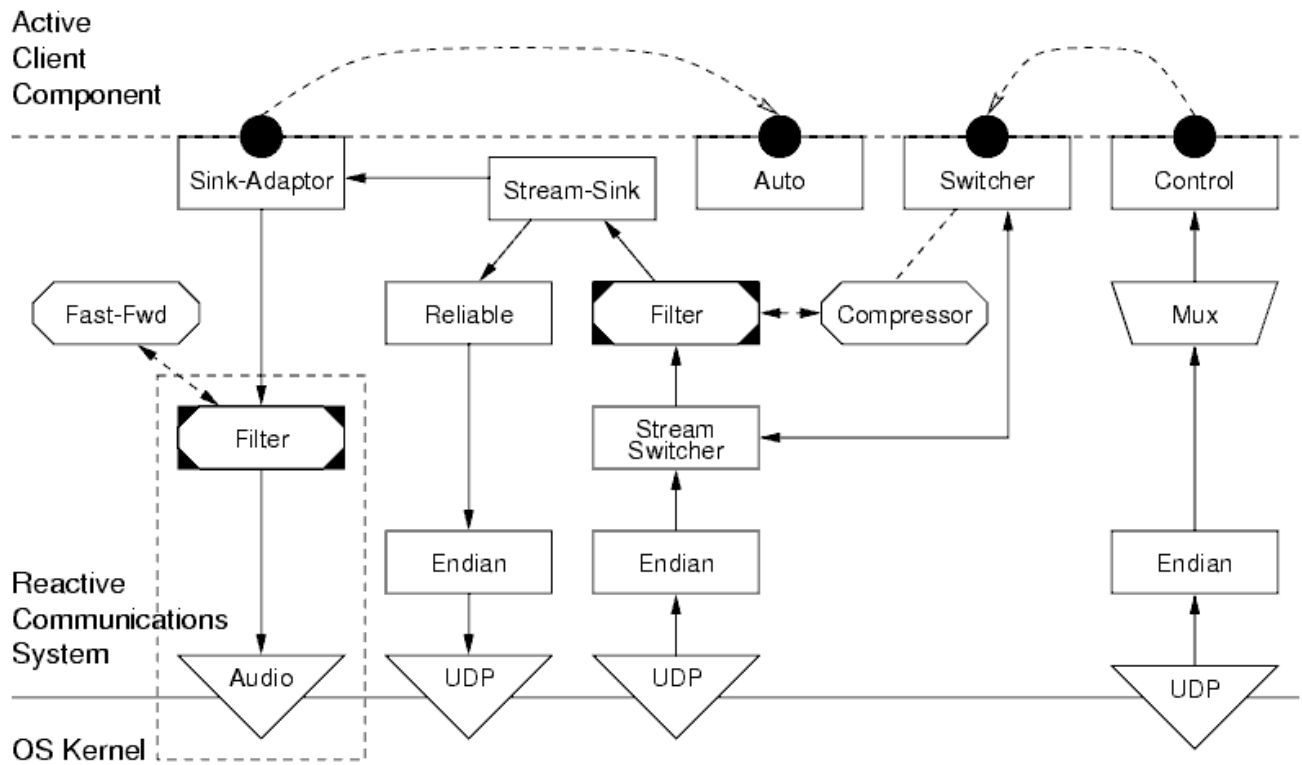


Figure 4: Adaptation infrastructure -- client side

Buffering	1000 kilobytes
Buffer Disaster Threshold	5 seconds
Buffer Samples	20
Throughput Peak Ratio	1.5
Throughput Samples	25 packets

Table 1: Experimental Default values

FTP Site	Total Transferred (kB)	Total Time (secs)	Average Rate		Highest Rate (kB/sec)	Lowest Rate (kB/sec)
			(kB/min)	(kB/sec)		
www.hensa.ac.UK	12860.6	196.3	3931.9	65.5	276	0
sunsite.doc.ic.ac.UK	12796.3	96.6	7950.6	132.5	925	0
sunsite.informatik.rwth-aachen.de	12563.0	77.8	9693.2	161.6	420	2
sunsite.uio.no	12538.7	157.5	4775.4	79.6	360	0

Table 2. Bandwidth profiles

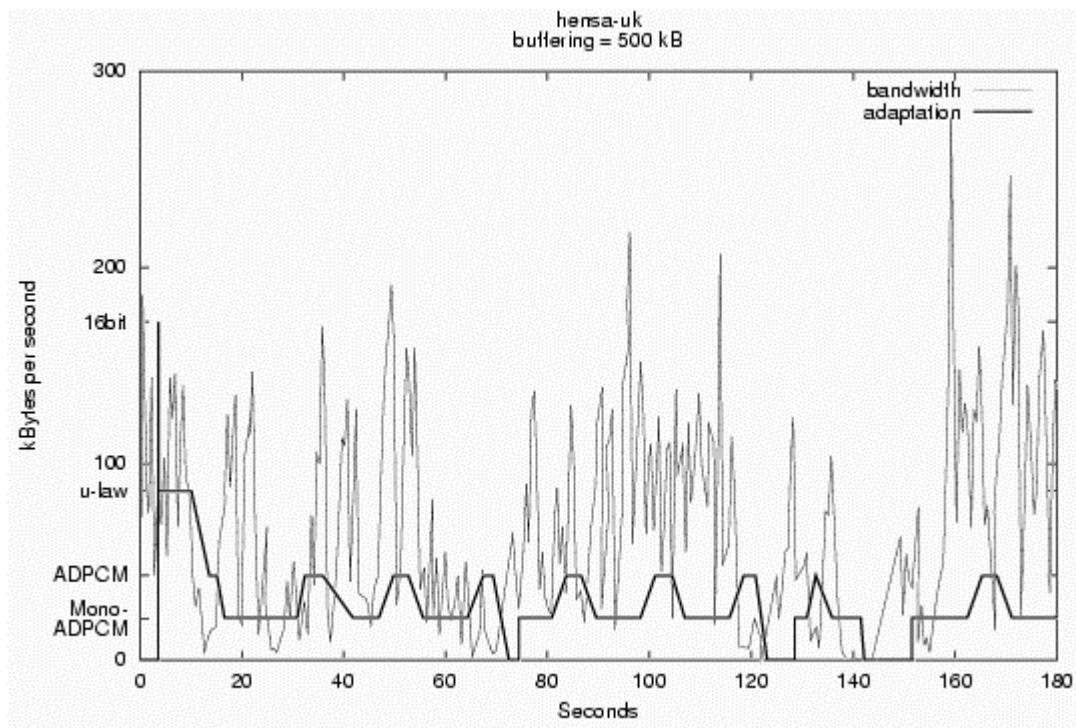


Figure 5. Graph showing bandwidth and adaptation

Experiment	Total Transferred over 3 minutes (kB)	Average Rate (kB/min) (kB/sec)		Bandwidth utilised(%)	Disaster due to Rebuffering (secs)
Variable: Buffering					
Hensa II Buffering 250 kB	5425.4	1808.5	30.1	46	23.3
Hensa II Buffering 500 kB	8725.0	2908.3	48.5	74	36.4
Hensa II Buffering 750 kB	7853.1	2617.7	43.6	67	25.0
Hensa II Buffering 1000 kB	9875.1	3291.7	54.9	84	35.6
Variable: Buffer Disaster Threshold					
Hensa II Buffer Disaster Horizon 1.0 secs	12705.4	4235.1	70.6	108	65.7
Hensa II Buffer Disaster Horizon 3.0 secs	10724.6	3574.9	59.6	91	62.8
Hensa II Buffer Disaster Horizon 5.0 secs	9875.1	3291.7	54.9	84	35.6
Hensa II Buffer Disaster Horizon 7.0 secs	6293.5	2097.8	35.0	53	19.3
Variable: Buffer Samples					
Hensa II Buffer Samples 10	13091.0	4363.7	72.7	111	52.6
Hensa II Buffer Samples 20	9875.1	3291.7	54.9	84	35.6
Hensa II Buffer Samples 30	11414.4	3804.8	63.4	97	47.1
Hensa II Buffer Samples 40	8412.9	2804.3	46.7	71	52.3
Variable: Throughput Peak Ratio					
Hensa II Throughput Peak Ratio 0.5	12820.8	4273.6	71.2	109	48.5
Hensa II Throughput Peak Ratio 1.0	11747.7	3915.9	65.3	100	43.5
Hensa II Throughput Peak Ratio 1.5	9875.1	3291.7	54.9	84	35.6
Hensa II Throughput Peak Ratio 2.0	8537.4	2845.8	47.4	72	32.2
Variable: Throughput Samples					
Hensa II Throughput Samples 25	9875.1	3291.7	54.9	84	35.6
Hensa II Throughput Samples 40	10796.0	3598.7	60.0	92	49.5
Hensa II Throughput Samples 50	9699.3	3233.1	53.9	82	35.4
Hensa II Throughput Samples 75	10331.7	3443.9	57.4	88	29.3

Table 3: Hensa statistics

Experiment	Total Transferred over 3 minutes (kB)	Average Rate (kB/min) (kB/sec)		Bandwidth utilised(%)	Disaster due to Rebuffering (secs)
Variable: Buffering					
Sunsite – Germany Buffering 250 kB	23728.9	7909.6	131.8	82	7.5
Sunsite – Germany Buffering 500 kB	24383.3	8127.8	135.5	84	0.0
Sunsite – Germany Buffering 750 kB	26189.3	8729.8	145.5	90	0.0
Sunsite – Germany Buffering 1000 kB	26607.7	8869.2	147.8	91	0.0
Variable: Buffer Disaster Threshold					
Sunsite - Germany Buffer Disaster Horizon 1.0 secs	26309.9	8770.0	146.2	90	0.0
Sunsite - Germany Buffer Disaster Horizon 3.0 secs	26318.4	8772.8	146.2	91	0.0
Sunsite - Germany Buffer Disaster Horizon 5.0 secs	24383.3	8127.8	135.5	84	0.0
Sunsite - Germany Buffer Disaster Horizon 7.0 secs	9886.3	3295.4	54.9	34	0.0
Variable: Buffer Samples					
Sunsite – Germany Buffer Samples 10	5402.1	1800.7	30.0	19	0.0
Sunsite - Germany Buffer Samples 20	24383.3	8127.8	135.5	84	0.0
Sunsite - Germany Buffer Samples 30	26322.7	8774.2	146.2	91	0.0
Sunsite - Germany Buffer Samples 40	26311.5	8770.5	146.2	90	0.0
Variable: Throughput Peak Ratio					
Sunsite - Germany Throughput Peak Ratio 0.5	26386.6	8795.5	146.6	91	0.0
Sunsite - Germany Throughput Peak Ratio 1.0	26301.9	8767.3	146.1	90	0.0
Sunsite - Germany Throughput Peak Ratio 1.5	24383.3	8127.8	135.5	84	0.0
Sunsite - Germany Throughput Peak Ratio 2.0	13712.4	4570.8	76.18	47	0.0
Variable: Throughput Samples					
Sunsite - Germany Throughput Samples 25	24383.3	8127.8	135.5	84	0.0
Sunsite - Germany Throughput Samples 40	26316.2	8772.1	146.2	90	0.0
Sunsite - Germany Throughput Samples 50	26315.3	8771.8	146.2	90	0.0
Sunsite - Germany Throughput Samples 75	13612.9	4537.6	75.6	47	0.0

Table 4: Sunsite Germany statistics

Experiment	Total Transferred over 3 minutes (kB)	Average Rate (kB/min) (kB/sec)		Bandwidth utilised(%)	Disaster due to Rebuffering (secs)
Variable: Buffering					
Sunsite – Norway Buffering 250 kB	4455.2	1485.1	24.8	31	22.0
Sunsite – Norway Buffering 500 kB	6220.0	2073.3	34.6	43	6.0
Sunsite – Norway Buffering 750 kB	10445.2	3481.7	58.0	73	3.9
Sunsite – Norway Buffering 1000 kB	14659.9	4886.6	81.4	102	14.7
Variable: Buffer Disaster Threshold					
Sunsite - Norway Buffer Disaster Horizon 1.0 secs	9944.2	3314.7	55.2	69	13.8
Sunsite - Norway Buffer Disaster Horizon 3.0 secs	8766.8	2922.3	48.7	61	9.2
Sunsite - Norway Buffer Disaster Horizon 5.0 secs	6220.0	2073.3	34.6	43	6.0
Sunsite - Norway Buffer Disaster Horizon 7.0 secs	5847.5	1949.2	32.5	41	3.2
Variable: Buffer Samples					
Sunsite – Norway Buffer Samples 10	5164.7	1721.6	28.7	36	1.4
Sunsite - Norway Buffer Samples 20	6220.0	2073.3	34.6	43	6.0
Sunsite - Norway Buffer Samples 30	8848.3	2949.4	49.2	62	8.7
Sunsite - Norway Buffer Samples 40	10043.6	3347.9	55.8	70	19.4
Variable: Throughput Peak Ratio					
Sunsite - Norway Throughput Peak Ratio 0.5	8241.7	2747.2	45.8	58	5.6
Sunsite - Norway Throughput Peak Ratio 1.0	6281.1	2093.7	34.9	44	6.2
Sunsite - Norway Throughput Peak Ratio 1.5	6220.0	2073.3	34.6	43	6.0
Sunsite - Norway Throughput Peak Ratio 2.0	6680.9	2227.0	37.1	47	8.3
Variable: Throughput Samples					
Sunsite - Norway Throughput Samples 25	6220.0	2073.3	34.6	43	6.0
Sunsite - Norway Throughput Samples 40	5515.4	1838.5	30.6	38	4.0
Sunsite - Norway Throughput Samples 50	5816.9	1939.0	32.3	40	0.0
Sunsite - Norway Throughput Samples 75	6097.9	2032.6	33.9	43	0.0

Table 5: Sunsite Norway Statistics

Experiment	Total Transferred over 3 minutes (kB)	Average Rate (kB/min) kB/sec)		Bandwidth utilised(%)	Disaster due to Rebuffering (secs)
Variable: Buffering					
Sunsite – UK Buffering 250 kB	399.2	1133.1	18.9	14	66.4
Sunsite – UK Buffering 500 kB	4516.9	1505.6	25.1	19	41.3
Sunsite – UK Buffering 750 kB	6169.9	2056.6	34.3	26	19.3
Sunsite – UK Buffering 1000 kB	6430.3	2143.4	35.7	27	21.5
Variable: Buffer Disaster Threshold					
Sunsite - UK Buffer Disaster Horizon 1.0 secs	8711.1	2903.7	48.4	37	25.9
Sunsite - UK Buffer Disaster Horizon 3.0 secs	8484.5	2828.2	47.1	36	28.8
Sunsite - UK Buffer Disaster Horizon 5.0 secs	6430.3	2143.4	35.7	27	21.5
Sunsite - UK Buffer Disaster Horizon 7.0 secs	6059.9	2020.0	33.7	25	10.9
Variable: Buffer Samples					
Sunsite – UK Buffer Samples 10	5715.5	1905.2	31.8	24	14.2
Sunsite - UK Buffer Samples 20	6430.3	2143.4	35.7	27	21.5
Sunsite - UK Buffer Samples 30	7579.1	2526.4	42.1	32	22.2
Sunsite - UK Buffer Samples 40	7718.8	2572.9	42.9	32	17.4
Variable: Throughput Peak Ratio					
Sunsite - UK Throughput Peak Ratio 0.5	6582.9	2194.3	36.6	28	17.9
Sunsite - UK Throughput Peak Ratio 1.0	6601.7	2200.6	36.7	28	17.9
Sunsite - UK Throughput Peak Ratio 1.5	6430.3	2143.4	35.7	27	21.5
Sunsite - UK Throughput Peak Ratio 2.0	6530.3	2176.8	36.3	27	15.2
Variable: Throughput Samples					
Sunsite - UK Throughput Samples 25	6430.3	2143.4	35.7	27	21.5
Sunsite - UK Throughput Samples 40	6372.0	2124.0	35.4	27	19.8
Sunsite - UK Throughput Samples 50	6913.4	2304.5	38.4	29	26.9
Sunsite - UK Throughput Samples 75	6508.9	2169.6	36.2	27	19.7

Table 6: Sunsight UK Statistics

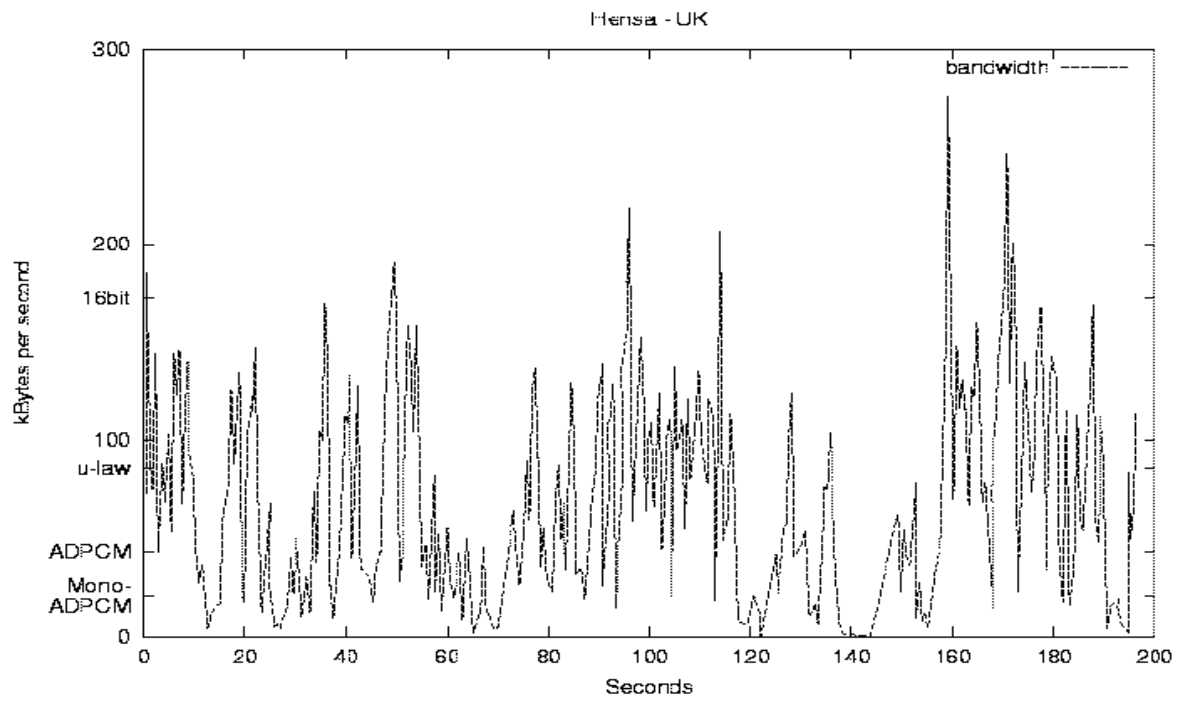


Figure 6. Hensa UK

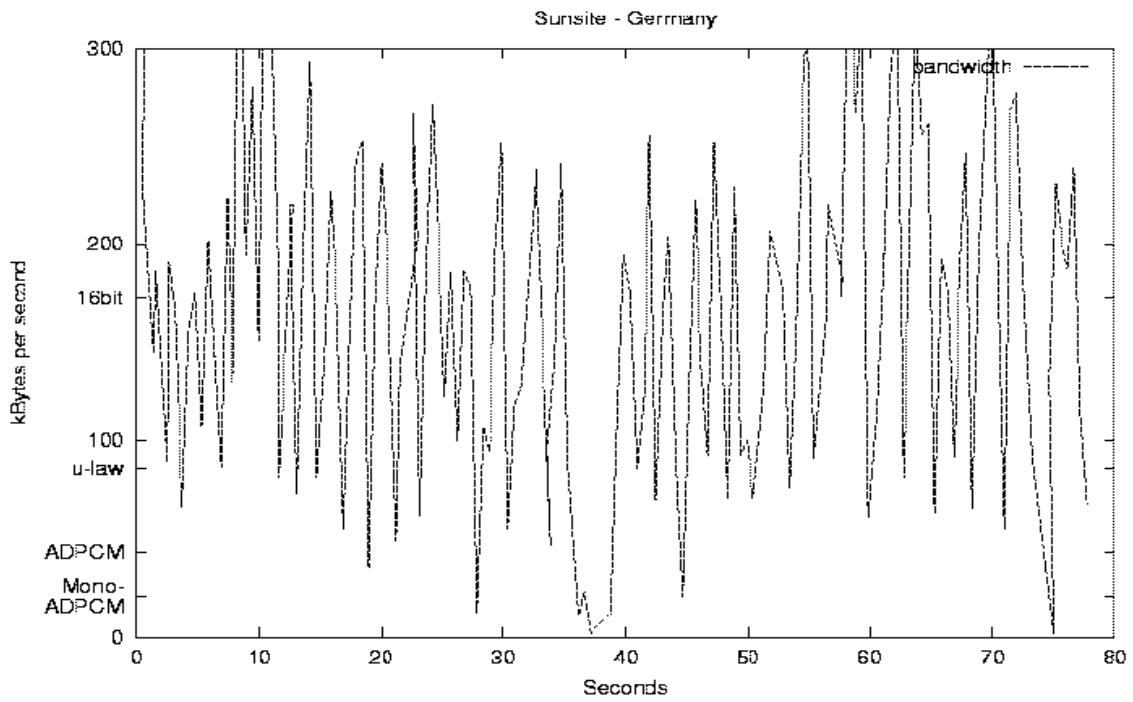


Figure 7. Sunsite Germany

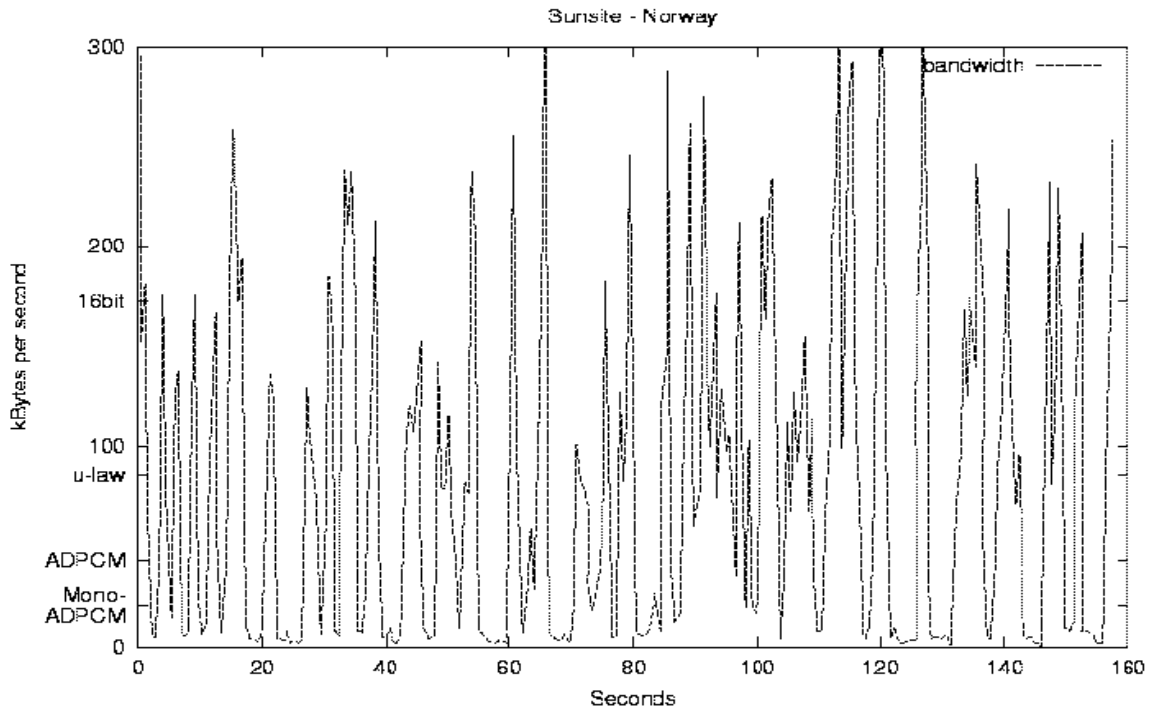


Figure 8. Sunsite Norway

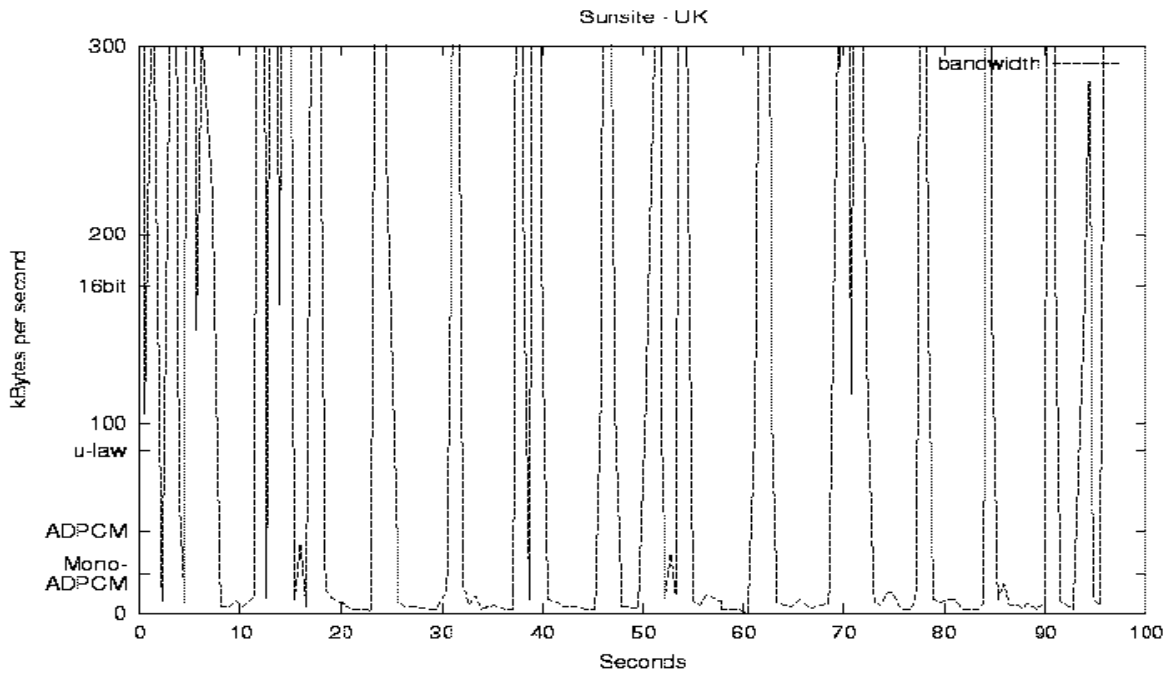


Figure 9. Sunsite UK

12 Biographies

Julie McCann received her DPhil doctorate in 1992 from the University of Ulster. This research looked at measurement and prediction tools for parallel database systems. From the same university she obtained BSc (Hons) degree in Computing Science and Information Systems. From 1986-87 she worked in the Bank of Ireland, Computing Centre as a Technical Performance Analyst specialising in IMS database systems. Previous to this she has also worked as a programmer in the Health Sector.

In 1992 Julie joined City University, Computer Science Dept. where she became the leader of the Systems Architecture Research Centre, managing many Operating System, Database and Distributed systems research projects. She currently lectures in Database Systems and Interoperable Information Systems. Her research focuses on high performance extensible operating and database systems. Julie is a Chartered Engineer and a Member of the British Computer Society.

Paul Howlett obtained an MSc in Advanced Computer Science from the University of Manchester in 1997. His MSc included research into the simulation of cloth for use in computer animation. He has a BSc(Hons) degree in Computer Science from the same university. Since completing his MSc, Paul has worked as a research assistant in the Department of Computing at City University, where his work focuses on distributed multimedia systems.

Stephen Crane received his PhD doctorate in 1997 from Imperial College, London. His research focused on Dynamic Binding for Distributed Systems. Previously, in 1990, he received his Masters degree in Computer Science in Trinity College Dublin and in the same university he obtained a BAI degree in Computer Engineering, 1987.

Steven joined City University in 1997 on the Kendra project. He has extensive research experience in the areas of reconfigurable, extensible parallel and distributed systems, and dynamic reconfiguration management.