15

DISTRIBUTED MULTIMEDIA SYSTEMS

This chapter draws substantially on a review paper by R.G. Herrtwich [Herrtwich 1995] to whom we are grateful for permission to use his material.

15.1 Introduction

Modern computers can handle streams of continuous, time-based data such as digital audio and video. This capability has led to the development of distributed multimedia applications such as networked video libraries, Internet telephony and video-conferencing. Such applications are viable with current general-purpose networks and systems although the quality of the resulting audio and video is often less than satisfactory. More demanding applications such as large-scale video conferencing, digital TV production, interactive TV and video surveillance systems are beyond the capabilities of current networking and distributed system technologies.

Multimedia applications demand the timely delivery of streams of multimedia data to end-users. Audio and video streams are generated and consumed in real time and the timely delivery of the individual elements (audio samples, video frames) is essential to the integrity of the application. In short, multimedia systems are real-time systems: they must perform tasks and deliver results according to a schedule that is externally determined. The degree to which this is achieved by the underlying system is known as the *quality of service* (QoS) enjoyed by an application.

Although the problems of real-time system design had been studied before the advent of multimedia systems and many successful real-time systems were developed [Kopetz and Verissimo 1993], they have not generally been integrated with more general-purpose operating systems and networks. The nature of the tasks performed by these existing real-time systems, such as avionics, air traffic control, manufacturing process control and telephone switching, differs from those performed in multimedia applications. The former generally deal with relatively small quantities of data and with relatively infrequent *hard deadlines* and failure to meet any deadline can have serious or even disasterous consequences. In such cases, the solution adopted has been to over-specify the computing resources and to allocate them on a fixed schedule that ensures that worst-case requirements are always met.

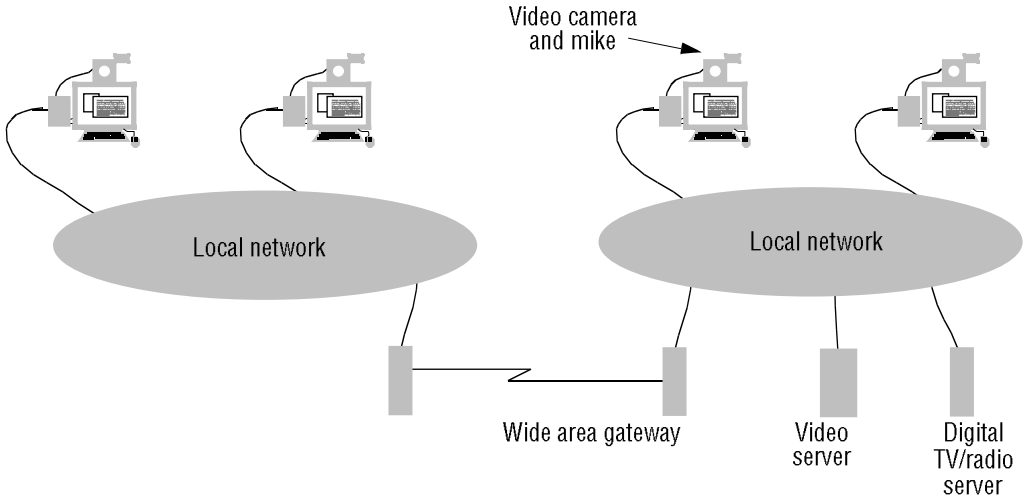
The planned allocation and scheduling of resources to meet the needs of multimedia and other applications is referred to as *quality of service management.* Most current operating systems and networks do not include the QoS management facilities needed to support multimedia applications.

Multimedia applications require the continuous processing and transmission of bulky streams of data at high bandwidths and with frequent deadlines (e.g. there is a deadline for the delivery of each video frame to its destination) but the consequences of failure are less serious – a small proportion of missed deadlines can often be tolerated.

The consequences of failure to meet deadlines in multimedia applications can be serious, especially in commercial environments such as video-on-demand services, business conferencing applications and remote medicine, but the requirements differ significantly from those of other real-time applications:

- Multimedia applications are often highly distributed and operate within general-purpose distributed computing environments. They therefore compete with other distributed applications for network bandwidth and for computing resources at users' workstations and servers.

Figure 15.1 A distributed multimedia system.



- The resource requirements of multimedia applications are dynamic. A video conference will require more or less network bandwidth as the number of participants grows or shrinks. Its use of computing resources at each user's workstation will also vary, since, for example, the number of video streams that have to be displayed varies. Multimedia applications may involve other variable or intermittent loads. For example, a multimedia lecture might include a processor-intensive simulation activity.
- Users often wish to balance the resource costs of a multimedia applications with other activities. Thus they may be willing to reduce their demands for video bandwidth in a conferencing application in order to allow a separate voice conversation to proceed, or they may wish a program development or a word-processing activity to proceed while they are participating in a conference.

QoS management systems are intended to meet all of these needs, managing the available resources dynamically and varying the allocations in response to changing demands and user priorities. A QoS management system must manage all of the computing and communication resources needed to acquire, process and transmit multimedia data streams, especially where the resources are shared between applications.

Figure 15.1 illustrates a typical distributed multimedia system capable of supporting a variety of applications such as desktop conferencing and providing access to stored video sequences, broadcast digital TV and radio. The resources for which QoS management is required include network bandwidth, processor cycles and memory capacity. Disk bandwidth at the video server may also be included. We shall adopt the generic term *resource bandwidth* to refer to the capacity of any hardware resource (network, central processor, disk subsystem) to transmit or process multimedia data.

In an open distributed system, multimedia applications can be started up and used without prior arrangement. Several applications may co-exist in the same network and even on the same workstation. The need for QoS management therefore arises

regardless of the *total quantity* of resource bandwidth or memory capacity in the system. QoS management is needed in order to *guarantee* that applications will be able to obtain the necessary quantity of resources at the required times, even when other applications are competing for the resources.

Some multimedia applications have been deployed even in today's QoS-less computing and network environments. These include:

Web-based multimedia: These are applications that provide best-effort access to streams of audio and video data published via the Web. They have been successful when there is little or no need for the synchronisation of the data streams at different locations. Their performance is constrained by the limited bandwidth and variable latencies found in current networks and by the inability of current operating systems to support real-time resource scheduling. For audio and low-quality video sequences, the use of extensive buffering at the destination to smooth out the variations in bandwidth and latency results in continuous and smooth display of video sequences but with a source-to-destination delay that may reach several seconds.

Network phone and audio conferencing: This type of application has relatively low bandwidth requirements, especially when efficient compression techniques are used. But its interactive nature demands low round-trip delays and these cannot always be achieved.

Video on demand services: These supply video information in digital form; retrieving the data from large online storage systems and delivering them to the end-user's display. These are successful where sufficient dedicated network bandwidth is available and where the video server and the receiving stations are dedicated. They also employ considerable buffering at the destination.

Highly interactive applications pose much greater problems. Many multimedia applications are cooperative (involving several users) and synchronous (requiring the users' activities to be closely coordinated). They span a wide spectrum of application contexts and scenarios. For example:

- A simple video conference involving two or more users, each using a workstation equipped with a digital video camera, microphone, sound output and video display capability. Application software to support simple teleconferencing is widely available [CU-SeeMe, Netmeeting, Viz], but its performance is severely limited by today's computing and network environments.

- A music rehearsal and performance facility enabling musicians at different locations to perform in an ensemble [Konstantas et al. 1997]. This is a particularly demanding multimedia application because the synchronization constraints are so tight.

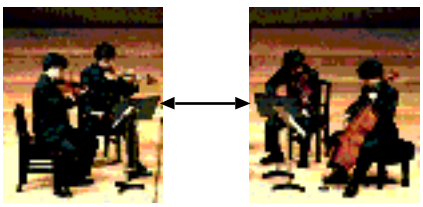
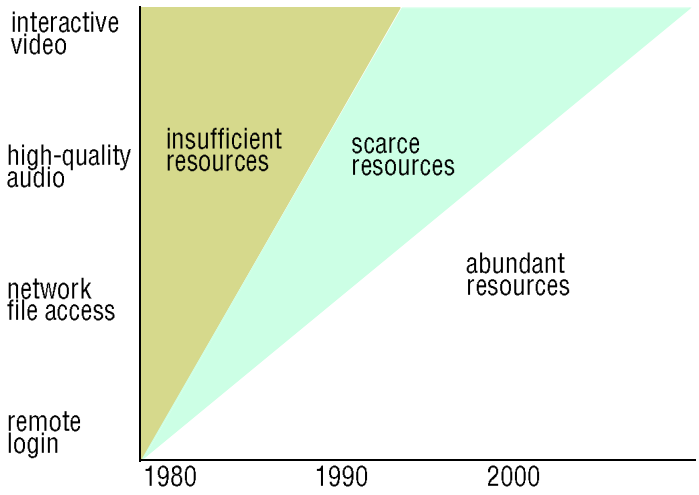


Figure 15.2 The "Window of Scarcity" for computing and communcation resources.



Applications such as these require:

Low latency communication: Round trip delays < 100 ms, so that interaction between users appears to be synchronous.

Synchronous distributed state: If one user stops a video on a given frame, the other users should see it stopped at the same frame.

Media synchronisation : All participants in a music performance should hear the performance at approximately the same time (Konstantas *et al*. [1997] identified a requirement for synchronisation within 50 ms). Separate soundtrack and video streams should maintain 'lip sync', e.g. for a user commenting live on a video playback, or a distributed Karaoke session.

External synchronisation: In conferencing and other cooperative applications, there may be active data in other formats, such as computer-generated animations, CAD data, electronic whiteboards, shared documents. Updates to these must be distributed and acted upon in manner that appears at least approximately synchronized with the time-based multimedia streams.

Such applications will run successfully only in systems that include rigorous QoS management schemes.

The window of scarcity ◊ Many of today's computer systems provide some capacity to handle multimedia data, but the necessary resources are very limited. Especially when dealing with large audio and video streams, many systems are constrained in the quantity and quality of streams they can support. This situation has been depicted as the *window of scarcity* [Anderson et al. 1990b]. While a certain class of applications lies within this window, a system needs to allocate and schedule its resources carefully in order to provide the desired service (see Figure 15.2). Before the window of scarcity is reached, a system has insufficient resources to execute the relevant applications. This

Figure 15.3 Characteristics of typical multimedia streams.

	Data rate (approximate)	*Sample or frame size*	*frequency*
Telephone speech	64 Kbits/sec	8 bits	8,000/sec
CD-quality sound	1,400 Kbits/sec	16 bits	44,000/sec
Standard TV video (uncompressed)	120 Mbits/sec	up to 640 x 480 pixels x 16 bits	24/sec
Standard TV video (MPEG-1 compressed)	1.5 Mbits/sec	variable	24/sec
HDTV video (uncompressed)	1,000–3,000 Mbits/sec	up to 1920 x 1080 pixels x 24 bits	24–60/sec
HDTV (MPEG-2 compressed)	10–30 Mbits/sec	variable	24–60/sec

was the situation for multimedia applications before the mid-eighties. Once an application class has left the window of scarcity, system performance will be sufficient to provide the service even under adverse circumstances and without customized mechanisms.

It is likely that multimedia applications will remain in the window of scarcity for the foreseeable future. Advances in system performance are likely to be used to improve the quality of multimedia data, to include higher frame rates and greater resolution for video streams or to support many media streams concurrently, for example in a video-conferencing system. More demanding applications, including virtual reality and real-time stream manipulation ("special effects") can extend the window of scarcity almost indefinitely.

In Section 15.2 we review the characteristics of multimedia data. Section 15.3 describes approaches to the allocation of scarce resources in order to achieve QoS and Section 15.4 discusses methods for scheduling them. Section 15.5 discusses methods for optimizing the flow of data in multimedia systems. Section 15.6 describes the Tiger Video Server, a low-cost scalable system for the delivery of stored video streams to large numbers of clients concurrently.

15.2 Characteristics of multimedia data

We have referred to video and audio data as continuous and time-based. How can we define these characteristics more precisely? The term 'continuous' refers to the user's view of the data. Internally, continuous media are represented as sequences of discrete values which replace each other over time. For example, the value of an image array is replaced 25 times per second to give the impression of a TV-quality view of a moving

scene; a sound amplitude value is replaced 8000 times per second to convey telephone quality speech.

Multimedia streams are said to be *time-based* (or *isochronous*) because timed data elements in audio and video streams define the semantics or 'content' of the stream. The times at which the values are played or recorded affect the validity of the data. Hence systems that support multimedia applications need to preserve the timing when they handle continuous data.

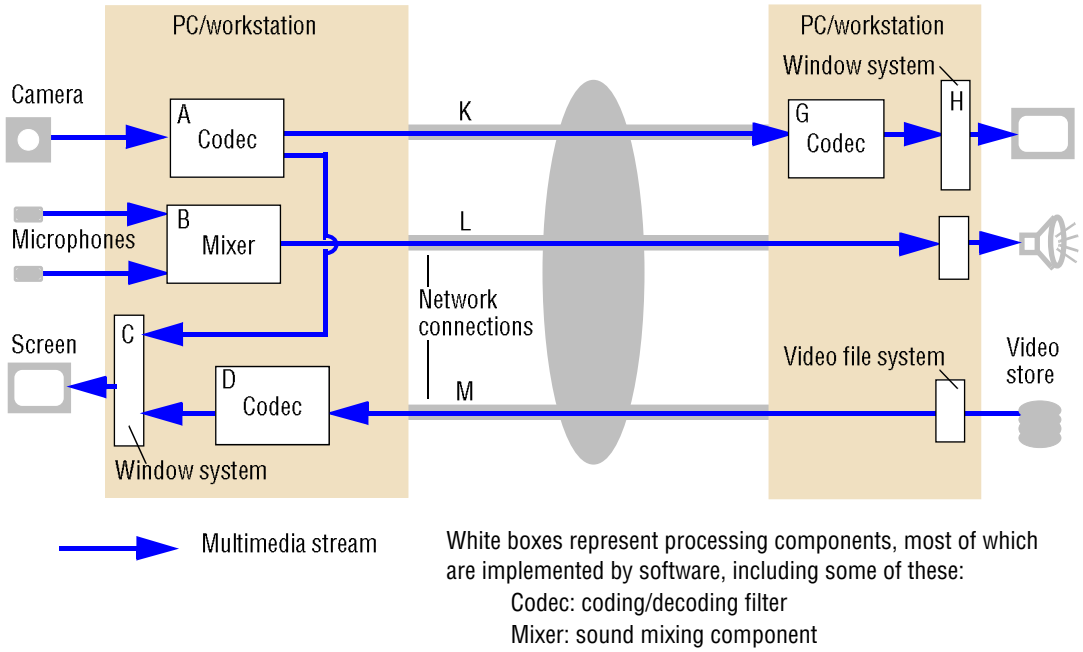
Multimedia streams are often bulky. Hence systems that support multimedia applications need to move data with greater throughput than conventional systems. Figure 15.3 shows some typical data rates and frame/sample frequencies. We note that the resource bandwidth requirements for some are very large. This is especially so for video of reasonable quality. For example a standard TV video stream requires more than 120 Mbits/sec., which exceeds the capacity of a 100 Mbit/s Ethernet network. CPU capacities are also stretched; a program that copies or applies a simple data transformation to each frame of a standard TV video stream requires at least 10% of the CPU capacity of a 400 MHz PC. The figures for high-definition television streams are even higher, and in many applications, such as video conferencing, there is a need to handle multiple video and audio streams concurrently. The use of compressed representations is therefore essential, although transformations such as video mixing are difficult to accomplish with compressed streams.

Compression can reduce bandwidth requirements by factors between 10 and 100, but the timing requirements of continuous data are unaffected. There is intensive research and standardisation activity aimed at producing efficient, general-purpose representations and compressions methods for multimedia data streams. This work has resulted in various compressed data formats such as GIF, TIFF and JPEG for still images and MPEG-1, MPEG-2 and MPEG-4 for video sequences. We do not detail these here; several other sources, e.g. [Buford 1994] and [Gibbs and Tsichritzis 1994] provide reviews of media types, representations and standards and the Web page of [Szentivanyi 1999] is an additional source of references to documentation on current multimedia standards.

Although the use of compressed video and audio data reduces bandwidth requirements in communication networks it imposes substantial additional loads on processing resources at the source and destination. This processing has often been supplied through the use of special-purpose hardware to process and despatch video and audio information – the video and audio coders/decoders (Codecs) found on video cards manufactured for personal computers. But increasing central processor power of personal computers and multiprocessor architectures are likely to enable them to perform much of this work in software using software coding and decoding filters. This approach offers greater flexibility with better support for application-specific data formats, special-purpose application logic and the simultaneous handling of multiple media streams.

The compression method used for the MPEG video formats is asymmetric, with a complex compression algorithm and simpler decompression. This tends to help its use in desktop conferencing, where compression is often performed by a hardware Codec but decompression of the several streams arriving at each user's computer is performed in software, enabling the number of conference participants to vary without regard to the number of Codecs in each user's computer.

Figure 15.4 Typical infrastructure components for multimedia applications



15.3 Quality of service management

When multimedia applications run in networks of personal computers they compete for resources at the workstations running the applications (processor cycles, bus cycles, buffer capacity) and in the networks (physical transmission links, switches, gateways). Workstations and networks may have to support several multimedia and conventional applications. There is competition between the multimedia and conventional applications, between different multimedia applications and even between the media streams within individual applications.

The concurrent use of physical resources for a variety of tasks has long been possible with multi-tasking operating systems and shared networks. In multi-tasking operating systems the central processor is allocated to individual tasks (or processes) in a round-robin or other scheduling scheme that shares the processing resources on a *best-efforts* basis amongst all of the tasks currently competing for the central processor.

Networks are designed to enable messages from different sources to be interleaved allowing many virtual communication channels to exist on the same physical channels. The predominant local-area network technology, Ethernet, manages a shared transmission medium in a *best-efforts* manner. Any node may use the medium when it is quiet. But packet collisions can occur and when they do sending nodes wait for random backoff periods in order to prevent repeated collisions. Collisions are likely to occur when the network is heavily loaded and this scheme cannot provide any guarantees regarding the bandwidth or latency in such situations.

Figure 15.5 QoS specs for components of the application shown in Figure 15.4.

Component	*Bandwidth*	*Latency*	*Loss rate*	*Resources required*
Camera	Out: 10 frames/sec, raw video 640x480x16 bits	–	Zero	–
A Codec	In: 10 frames/sec, raw video Out: MPEG-1 stream	Interactive	Low	10 ms CPU each 100 ms; 10 Mbytes RAM
B Mixer	In: 2 x 44K bits/sec audio Out: 1 x 44K bits/sec audio	Interactive	Very low	1 ms CPU each 100 ms; 1 Mbytes RAM
H Window system	In: various Out: 50 frames/sec framebuffer	Interactive	Low	5 ms CPU each 100 ms; 5 Mbytes RAM
K Network conn.	In/Out: MPEG-1 stream, approx. 1.5 Mbits/sec	Interactive	Low	1.5 Mbits/sec, low-loss stream protocol
L Network conn.	In/Out: Audio 44K bits/sec	Interactive	Very low	44 K bits/sec, very low-loss stream protocol

The key feature of these resource allocation schemes is that they handle increases in demand by spreading the available resources more thinly between the competing tasks. Round-robin and other best-efforts methods for sharing processor cycles and network bandwidth cannot meet the needs of multimedia applications. As we have seen, the timely processing and transmission of multimedia streams is crucial for them. Late delivery is valueless. In order to achieve timely delivery, applications need guarantees that the necessary resources will be allocated and scheduled at the required times.

The management and allocation of resources to provide such guarantees is referred to as *quality of service management*. Figure 15.4 shows the infrastructure components for a simple multimedia conferencing application running on two personal computers, using software data compression and format conversion. The white boxes represent software components whose resource requirements may affect the quality of service of the application.

The figure shows the most commonly-used abstract architecture for multimedia software, in which continuously flowing *streams* of media data elements (video frames, audio samples) are processed by a collection of processes and transferred between the processes by inter-process connections. The processes produce, transform and consume continuous streams of multimedia data. The connections link the processes in a sequence from a *source* of media elements to a *target* at which it is rendered or consumed. The connections between the processes may be implemented by networked connections or by in-memory transfers when processes reside on the same machine. For the elements of multimedia data to arrive at their target on time, each process must be allocated adequate CPU time, memory capacity and network bandwidth to perform its designated task and must be scheduled to use the resources sufficiently frequently to enable it to deliver the data elements in its stream to the next process on time.

In Figure 15.5 we set out resource requirements for the main software components and network connections in Figure 15.4 (note the corresponding letters against components in these two figures). Clearly, the required resources can only be guaranteed

if there is a system component responsible for the allocation and scheduling of those resources. We shall refer to that component as the *Quality of Service manager*.

Figure 15.6 shows the QoS manager's responsibilities in the form of a flowchart. In the next two sub-sections we describe the QoS manager's two main sub-tasks:

- *Quality of service negotiation*. The application indicates its resource requirements to the QoS manager. The QoS manager evaluates the feasibility of meeting the requirements against a database of the available resources and current resource commitments and gives a positive or negative response. If it is negative, the application may be reconfigured to use reduced resources and the process is repeated.
- *Admission control*. If the result of the resource evaluation is positive, the requested resources are reserved and the application is given a Resource Contract, stating the resources that have been reserved. The contract includes a time limit. The application is then free to run. If it changes its resource requirements it must notify the QoS Manager. If the requirements decrease, the resources released are returned to the database as available resources. If they increase, a new round of negotiation and admission control is initiated.

In the remainder of this section we describe techniques for performing these subtasks in further detail. Of course, while an application is running, there is a need for fine-grained scheduling of resources such as processor time and network bandwidth to ensure that real-time processes receive their allocated resources on time. Techniques for this are discussed in Section 15.4.

15.3.1 Quality of service negotiation

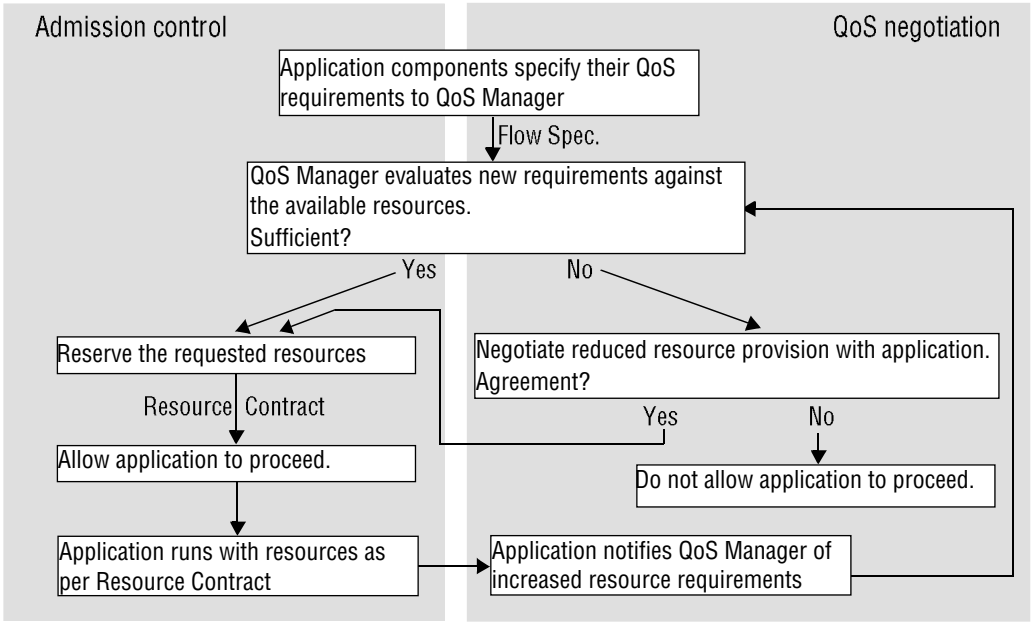
To negotiate QoS between an application and its underlying system, an application must specify its QoS requirements to the QoS Manager. This is done by the transmission of a set of parameters. Three parameters are of primary interest when it comes to processing and transporting multimedia streams: *bandwidth*, *latency*, and *loss rate*.

Bandwidth: The bandwidth of a multimedia stream or component is the rate at which data flows through it.

Latency: Latency is the time required for an individual data element to move through a stream from the source to the destination. Of course this may vary depending on the volume of other data in the system and other charactersitics of the system load. This variation is termed *jitter* – formally, jitter is the first derivative of the latency.

Loss rate: Since the late delivery of multimedia data is of no value, data elements will be dropped when it is impossible to deliver them before their scheduled delivery time. In a perfectly-managed QoS environment, this should never happen, but as yet, few such environments exist for reasons outlined earlier. Furthermore, the resource cost of guaranteeing on-time delivery for every media element is often unacceptable – it is likely to involve the reservation of resources far exceeding the average requirement in order to deal with occasional peaks. The alternative that is adopted is to accept a certain rate of data loss – dropped video frames or audio samples. The acceptable ratios are

Figure 15.6 The QoS Manager's task.



usually kept low – seldom more than 1% and much lower for quality-critical applications.

The three parameters can be used:

1. To describe the characteristics of a multimedia stream in a particular environment. For example, a video stream may require an average bandwidth of 1.5 Mbits/sec and because it is used in a conferencing application it needs to be transferred with at most 150 ms delay to avoid conversation gaps. The decompression algorithm used at the target may still yield acceptable pictures with a loss rate of 1 frame out of 100.
2. To describe the capabilities of resources to transport a stream. For example, a network may provide connections of 64 Kbits/sec bandwidth, its queuing algorithms guarantee delays less than 10 ms and the transmission system may guarantee a loss rate smaller than 1 in 10^6.

The parameters are interdependent. For example:

- Loss rate in modern systems rarely depends on actual bit errors due to noise or malfunction; it results from buffer overflow and from time-dependent data arriving too late. Hence, the larger bandwidth and delay can be, the more likely is a low loss rate.
- The smaller the overall bandwidth of a resource is compared to its load, the more messages will accumulate in front of it and the larger the buffers for this accumulation need to be to avoid loss. The larger the buffers become, the more

likely it is that messages need to wait for other messages in front of them to be serviced – that is, the larger the delay will become.

Specifying the QoS parameters for streams ◊ The values of QoS parameters can be stated explicitly (e.g. for the camera output stream in Figure 15.4 we might require *bandwidth*: 50 Mbits/sec, *delay*: 150 ms, *loss*: < 1 frame in 10^3) or implicitly (e.g. the bandwidth of the input stream to the network connection K is the result of applying MPEG-1 compression to the camera output).

But the more usual case is that we need to specify a value and a range of permissible variation. Here we consider this requirement for each of the parameters:

Bandwidth: Most video compression techniques produce a stream of frames of differing sizes depending on the original content of the raw video. For MPEG, the average compression ratio is between 1:50 and 1:100, but this will vary dynamically depending on content; for example, the required bandwidth will be highest when the content is changing most rapidly. Because of this, it is often useful to quote QoS parameters as maximum, minimum or average values, depending on the type of QoS management regime that will be used.

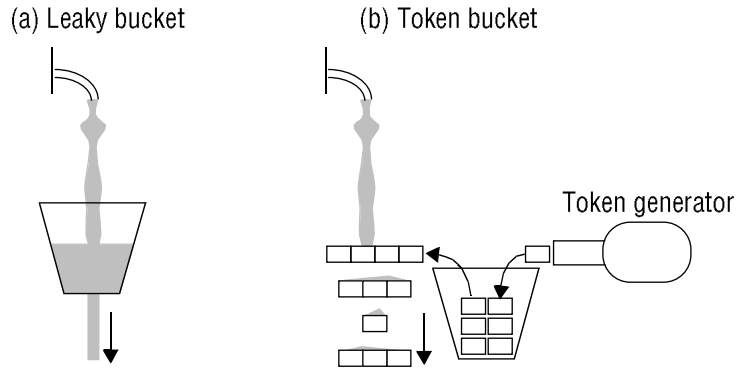
Another problem that arises in the specification of the bandwidth is the characterization of *burstiness*. Consider three streams of 1 Mbits/s. One stream transfers a single frame of 1 Mbit every second, the second is an asynchronous stream of computer-generated animation elements with an average bandwidth of 1 Mbit/s the third sends a 100 bit sound sample every microsecond. Whereas all three streams require the same bandwidth, their traffic patterns are very different.

One way to take care of irregularities is to define a burst parameter in addition to rate and frame size. The burst parameter specifies the maximum number of media elements that may arrive early – that is, before they should arrive according to the regular arrival rate. The model of *linear-bounded arrival processes* (LBAP) used in [Anderson 1993] defines the maximum number of messages in a stream during any time interval t as $Rt + B$ where R is the rate and B is the maximum size of burst. The advantage of using this model is that it nicely reflects the characteristics of multimedia sources: multimedia data read from disks is usually delivered in large blocks and data received from networks often arrives in the form of sequences of smaller packets. In this case the burst parameter defines the amount of buffer space required to avoid loss.

Latency: Some timing requirements in multimedia result from the stream itself: if a frame of a stream does not get processed with the same rate at which frames arrive, backlog builds up and buffer capacity may be exceeded. If this is to be avoided, a frame must on average not remain in a buffer for longer than $1/R$, where R is the frame rate of a stream, or a backlog will occur. If backlogs do occur, the number and size of the backlogs will affect the maximum end-to-end delay of a stream, in addition to the processing and propagation times.

Other latency requirements arise from the application environment. In conferencing applications, the need for apparently instantaneous interaction amongst the participants makes it necessary to achieve absolute end-to-end delays of no more than 150 ms to avoid problems in the human perception of the conversation. Whereas for the replay of stored video, to ensure a proper system response to commands such as *Pause* and *Stop*, the maximum latency should be in the order of 500 ms.

Figure 15.7 Leaky bucket algorithm and token bucket algorithms



A third consideration for the delivery time of multimedia messages is jitter – variation in the period between the delivery of two adjacent frames. Whereas most multimedia devices make sure that they present data at its regular rate without variation, software presentations (for example, in a software decoder for video frames) need to take extra care to avoid jitter. Jitter is essentially solved by buffering, but the scope for jitter removal is limited, because total end-to-end delay is constrained by the consideration mentioned above, so the playback of media sequences also requires media elements to arrive before fixed deadlines.

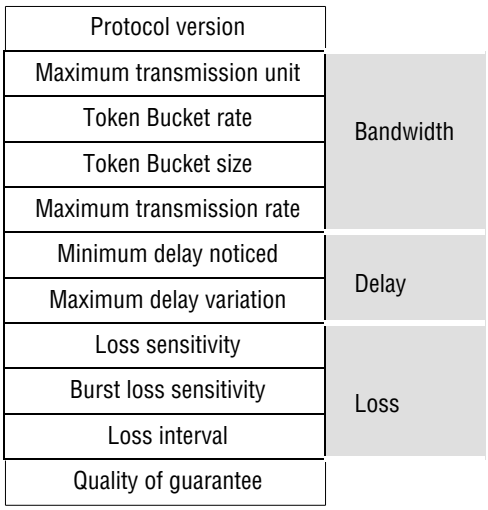
Loss rate: Loss rate is the most difficult QoS parameter to specify. Typical loss rate values result from probability calculations about overflowing buffers and delayed messages. These calculations are either based on worst-case assumptions or on standard distributions. Neither of these is necessarily a good match for practical situations. However, loss rate specifications are necessary to qualify the bandwidth and latency parameters: two applications may have the same bandwidth and latency characteristics; they will look dramatically different when one application loses every fifth media element and the other loses only one in a million.

As with bandwidth specifications, where not just the volume of data sent in a time interval but its distribution over the time interval is important, a loss rate specification needs to determine the time interval during which to expect a certain loss. In particular, loss rates given for infinite time spans are not useful as any loss over a short time may exceed the long-term rate significantly.

Traffic shaping ◊ Traffic shaping is the term used to describe the use of output buffering to smooth the flow of data elements. The bandwidth parameter of a multimedia stream typically provides an idealistic approximation of the actual traffic pattern that will occur when the stream is transmitted. The closer the actual traffic pattern matches the description, the better a system will be able to handle the traffic, in particular when it uses scheduling methods that are designed for periodic requests.

The LBAP model of bandwidth variations calls for regulation of the burstiness of multimedia streams. Any stream can be regulated by inserting a buffer at the source and by defining a method by which data elements leave the buffer. A good illustration of this method is the image of a leaky bucket (Figure 15.7a): the bucket can be filled

Figure 15.8 The RFC 1363 Flow Spec.



arbitrarily with water until it is full; through a leak at the bottom of the bucket water will continuously flow. The leaky-bucket algorithm ensures that a stream will never flow with a rate higher than *R*. The size of the buffer *B* defines the maximum burst a stream can incur without losing elements. *B* also bounds the time for which an element will remain in the bucket.

The leaky bucket algorithm completely eliminates bursts. Such elimination is not always necessary as long as bandwidth is bounded over any time interval. The token bucket algorithm achieves this while allowing larger bursts to occur when a stream has been idle for a while (Figure 15.7b). It is a variation of the leaky bucket algorithm in which tokens to send data are generated at a fixed rate *R*. They are collected in a bucket of size *B*. Data of size *S* can only be sent if at least *S* tokens are in the bucket. The send process then removes these *S* tokens. The token bucket algorithm ensures that over any interval *t* the amount of data sent is not larger than $Rt + B$. It is, hence, an implementation of the LBAP model.

High peaks of size *B* can only occur in a token-bucket system when a stream has been idle for a while. To avoid these bursts, a simple leaky bucket can be placed behind the token bucket. The flow rate *F* of this bucket needs to be significantly larger than *R* for this scheme to make sense. Its only purpose is to break up really large bursts.

Flow specifications ◊ A collection of QoS parameters is typically known as a flow specification, or *flow spec* for short. Several examples of flow specs exist and are all similar. In Internet RFC 1363 [Partridge 1992], a flow spec is defined as eleven 16-bit numeric values (Figure 15.8) that reflect the QoS parameters discussed above in the following way:

- The maximum transmission unit and maximum transmission rate determine the maximum bandwidth required by the stream.
- The token bucket size and rate determine the burstiness of the stream.

- The delay characteristics are specified by the minimum delay that an application can notice (since we wish to avoid over-optimization for short delays) and maximum jitter it can accept.
- The loss characteristics are defined by the total acceptable number of losses over a certain interval and the maximum number of consecutive losses.

There are many alternatives for expressing each parameter group. In SRP [Anderson et al. 1990a] the burstiness of a stream is given by a maximum workahead parameter that defines the number of messages a stream may be ahead of its regular arrival rate at any point in time. In [Ferrari and Verma 1990] a worst-case delay bound is given: if the system cannot guarantee to transport data within this time span, the data transport will be useless for the application. In RFC 1190, the specification of the ST-II protocol [Topolcic 1990], loss is represented as the probability for each packet to be dropped.

All the above examples provide a continuous spectrum of QoS values. If the set of applications and streams to be supported is limited, it may be sufficient to define a discrete set of QoS classes, for example, telephone-quality and high-fidelity audio, live and playback video, etc. The requirements of all classes must be implicitly known by all system components; the system may even be configured for a certain traffic mix.

Negotiation procedures ◊ For distributed multimedia applications, the components of a stream are likely to be located in several nodes. There will be a QoS manager at each node. A straightforward approach to QoS negotiation is to follow the flow of data along each stream from the source to the target. A source component initiates the negotiation by sending out a flow spec to its local QoS manager. The manager can check against its database of available resources whether the requested QoS can be provided. If other systems are involved in the application, the flow spec is forwarded to the next node where resources are required. The flow spec traverses all the nodes until the final target is reached. Then the information on whether the desired QoS can be provided by the system is passed back to the source. This simple approach to negotiation is satisfactory for many purposes, but it does not consider the possibilities for conflict between concurrent QoS negotiations starting at different nodes. A distributed transactional QoS negotiation procedure would be required for a full solution to this problem.

Applications rarely have fixed QoS requirements. Instead of returning a Boolean value on whether a certain QoS can be provided or not, it is more appropriate for the system to determine what kind of QoS it can provide and let the application decide on whether it is acceptable. In order to avoid over-optimized QoS or to abort the negotiation when it becomes clear that the desired quality is not achievable, it is common to specify a desired and worsts value for each QoS parameter. An application may specify that it desires a bandwidth of 1.5 Mbits/s, but would also be able to handle 1 Mbits/s or that delay should be 200 ms, but 300 ms would be the worst-case that is still acceptable. As only one parameter can be optimized at the same time, HeiRAT [Vogt et al. 1993] expects the user to define values for only two parameters and leaves it to the system to optimize the third.

If a stream has multiple sinks the negotiation path forks according to the data flow. As a straightforward extension to the above scheme, intermediate nodes can aggregate QoS feedback messages from the targets to produce worst-case values for the QoS parameters. The available bandwidth then becomes the smallest available bandwidth of

all targets, the delay becomes the longest of all targets, and the loss rate becomes the largest of all targets. This is the procedure practice by sender-initiated negotiation protocols such as SRP, ST-II or RCAP [Banerjea and Mah 1991].

In situations with heterogeneous targets, it is usually inappropriate to assign a common worst-case QoS to all targets. Instead, each target should receive the best possible QoS. This calls for a receiver-initiated negotiation process rather then a sender-oriented one. RSVP [Zhang et al. 1993] is an alternative QoS negotiation protocol in which targets connect to streams. Sources communicate the existence of streams and their inherent characteristics to all targets. Targets can then connect to the closest node through which the stream passes and derive data from there. In order for them to obtain data with the appropriate QoS, they may need to use techniques such as filtering (discussed in Section 15.5).

15.3.2 Admission control

Admission control regulates access to resources to avoid resource overload and to protect resources from requests that they cannot fulfil. It involves turning down service requests should the resource requirements of a new multimedia stream would violate existing QoS guarantees.

An admission control scheme is based on some knowledge of both the overall system capacity and the load generated by each application. The bandwidth requirement specification for an application may reflect the maximum amount of bandwidth that an application will ever require, the minimum bandwidth it will need to function, or some average value in between. Correspondingly, an admission control scheme may base its resource allocation on any of these values.

For resources that have a single allocator, admission control is straightforward. Resources that have distributed access points, such as many local area networks, require either a centralized admission control entity or some distributed admission control algorithm that avoids conflicting concurrent admissions. Bus arbitration within workstations falls into this category – however, even multimedia systems that perform bandwidth allocation extensively do not control bus admission as bus bandwidth is not considered to be in the window of scarcity.

Bandwidth reservation ◊ A common way to ensure a certain QoS level for a multimedia stream is to reserve some portion of resource bandwidth for its exclusive use. In order to fulfil the requirements of a stream at all times, a reservation needs to be made for its maximum bandwidth. This is the only possible way to provide guaranteed QoS to an application – at least as long as no catastrophic system failures occur. It is used for applications that cannot adapt to different QoS levels or become useless when quality drops occur. Examples include some medical applications (a symptom may appear in an x-ray video just at the time when video frames are dropped) and video recording (where dropped frames will result in a flaw in the recording that is visible every time the video is played).

Reservation based on maximum requirements can be straightforward: when controlling access to a network of a certain bandwidth B, multimedia streams s of a bandwidth b_s can be admitted as long as $\Sigma b_s <= B$. Thus a token ring of 16 Mb/s may support up to 10 digital video streams of 1.5 Mb/s each.

Unfortunately, capacity calculations are not always as simple as in the network case. To allocate CPU bandwidth in the same way requires the execution profile of each application process to be known. Execution times, however, depend on the processor used and often cannot be determined precisely. While several proposals for automatic execution time calculation exist [Mok 1985], [Kopetz et al. 1989], none of them has achieved widespread use. Execution times are usually determined through measurements which often have wide error margins and limited portability.

For typical media encodings such as MPEG, the actual bandwidth consumed by an application may be significantly lower than its maximum bandwidth. Reservations based on maximum requirements may then lead to wasted resource bandwidth: requests for new admissions are turned down although they could be satisfied with the bandwidth that is reserved, but not actually used by existing applications,

Statistical multiplexing ◊ Because of the potential under-utilization that can occur, it is common to overbook resources. The resulting guarantees, often called statistical or soft guarantees to distinguish them from the deterministic or hard guarantees introduced before, are only valid with some (usually very high) probability. Statistical guarantees tend to provide better resource utilization as they do not consider the worst case. But just as when resource allocation is based on minimum or average requirements, simultaneous peak loads can cause drops in service quality; applications have to be able to handle these drops.

Statistical multiplexing is based on the hypothesis that for a large number of streams the aggregate bandwidth required remains nearly constant regardless of the bandwidth of individual streams. It assumes that when one stream sends a large quantity of data, there will also be another stream that sends a small quantity and overall the requirements will balance out. This, however, is only the case for uncorrelated streams.

As experiments show [Leland et al. 1993], multimedia traffic in typical environments does not obey this hypothesis. Given a larger number of bursty streams, the aggregate traffic still remains bursty. The term *self-similar* has been applied to this phenomenon, meaning that the aggregate traffic shows similarity to the individual streams of which it is composed.

15.4 Resource management

To provide a certain QoS level to an application, not only does a system need to have sufficient resources (performance), it also needs to make these resources available to an application when they are needed (scheduling).

15.4.1 Resource scheduling

Processes need to have resources assigned to them according to their priority. A resource scheduler determines the priority of processes based on certain criteria. Traditional CPU schedulers in time-sharing systems often base their priority assignments on responsiveness and fairness: I/O intensive tasks get high priority to guarantee fast response to user requests, CPU-bound tasks get lower priorities and overall, processes in the same class are treated equally.

Both criteria remain valid for multimedia systems, but the existence of deadlines for the delivery of individual multimedia data elements changes the nature of the scheduling problem. Real-time scheduling algorithms can be applied to this problem as discussed below. As multimedia systems have to handle both discrete and continuous media, it becomes a challenge to provide sufficient service to time-dependent streams without causing starvation of discrete-media access and other interactive applications.

Scheduling methods need to be applied to (and coordinated for) all resources that affect the performance of a multimedia application. In a typical scenario, a multimedia stream would be retrieved from disk and then sent through a network to a target station where it is synchronized with a stream coming from another source and finally displayed. The resources required in this example include disk, network, and CPUs as well as memory and bus bandwidth on all systems involved.

Fair scheduling ◊ If several streams compete for the same resource it becomes necessary to consider fairness and to prevent ill-behaved streams from taking too much bandwidth. A straightforward approach to ensure fairness is to apply round-robin scheduling to all streams in the same class. Whereas in [Nagle 1987] such a method was introduced on a packet-by-packet basis, in [Demers et al. 1989] the method is used on a bit-by-bit basis which provides more fairness with respect to varying packet sizes and packet arrival times. These methods are known as fair queuing.

Packets cannot actually be sent on a bit-by-bit basis, but given a certain frame rate it is possible to calculate for each packet when it should have been sent completely. If packet transmissions are ordered based on this calculation, one achieves almost the same behaviour as with actual bit-by-bit round robin, except that when a large packet is sent, it may block the transmission of a smaller packet that would have been preferred under the bit-by-bit scheme. However, no packet is delayed longer than for the maximum packet transmission time.

All basic round-robin schemes assign the same bandwidth to each stream. To take the individual bandwidth of streams into account, the bit-by-bit scheme can be extended so that for certain streams a larger number of bits can be transmitted per cycle. This method is called weighted fair queuing.

Real-time scheduling ◊ Several real-time scheduling algorithms have been developed to meet the CPU scheduling needs of applications such as avionics industrial process control. Assuming that the CPU resources have not been over-allocated (which is the task of the QoS Manager), they assign CPU timeslots to a set of processes in a manner that ensure that they complete their tasks on time.

Traditional real-time scheduling methods suit the model of regular continuous multimedia streams very well. Earliest-deadline-first (EDF) scheduling has almost become a synonym for these methods. An EDF scheduler uses a deadline that is associated with each of its work items to determine the next item to be processed: the item with the earliest deadline goes first. In multimedia applications, we identify each media element arriving at a process as a work item. EDF scheduling is proven to be optimal for allocating a single resource based on timing criteria: if there is a schedule that fulfils all timing requirements, EDF scheduling will find it [Dertouzos 1974].

EDF scheduling requires one scheduling decision per message (i.e. per multimedia element). It would be more efficient to base scheduling on elements that exist for a longer time. Rate-monotonic (RM) scheduling is a prominent technique for

real-time scheduling of periodic processes that achieves just this. Streams are assigned priorities according to their rate: the higher the rate of work items on a stream, the higher the priority of a stream. RM scheduling has been shown to be optimal for situations that only utilize a given bandwidth by less than 69% [Liu and Layland 1973]. Using such an allocation scheme, the remaining bandwidth could be given to non-real-time applications.

To cope with bursty real-time traffic, the basic real-time scheduling methods should be adjusted to distinguish between time-critical and non-critical continuous-media work items. In [Govindan and Anderson 1991] deadline/workahead scheduling is introduced. It allows messages in a continuous stream to arrive ahead of time in bursts, but applies EDF scheduling to a message only at its regular arrival time.

15.5 Stream adaptation

Whenever a certain QoS cannot be guaranteed or can only be guaranteed with a certain probability, an application needs to adapt to changing QoS levels, adjusting its performance accordingly. For continuous-media streams, the adjustment translates into different levels of media presentation quality.

The simplest form of adjustment is to drop pieces of information. This is easily done in audio streams where samples are independent from each other, but it can immediately be noticed by the listener. Drop-outs in a video stream encoded in Motion JPEG, where each frame stands for itself are more tolerable. Encoding mechanisms such as MPEG, where the interpretation of a frame depends on the values of several adjacent frames, are less robust against omissions: it takes a longer time to recover from errors and the encoding mechanism may, in fact, amplify errors.

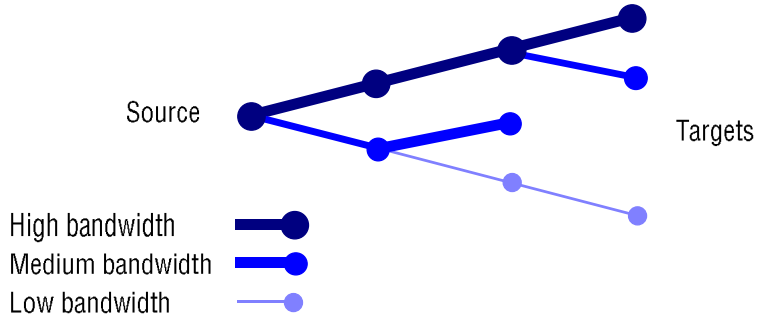
If there is insufficient bandwidth and data is not dropped, the delay of a stream will increase over time. For non-interactive applications this may be acceptable, although it can eventually lead to buffer overflows as data is accumulated between the source and sink. For conferencing and other interactive applications, increasing delays are not acceptable, or must exist only for a short period. If a stream is behind its assigned playout time, its playout rate should be increased until it gets back on schedule: while a stream is delayed, frames should be output as soon as they are available.

15.5.1 Scaling

If adaptation is performed at the target of a stream, the load on any bottleneck in the system is not decreased and the overload situation persists. It is useful to adapt a stream to the bandwidth available in the system before it enters a bottleneck resource in order to resolve contention. This is known as scaling.

Scaling is best applied when live streams are sampled. For stored streams, it depends on the encoding method how easy it is to generate a downgraded stream. Scaling may be too cumbersome if the entire stream has to be decompressed and encoded again just for scaling purposes. Scaling algorithms are media-dependent although the overall scaling approach is the same: to subsample a given signal. For audio information, such subsampling can be achieved by reducing the rate of audio sampling.

Figure 15.9 Filtering.



It can also be achieved by dropping a channel in a stereo transmission. As this example shows, different scaling methods can work at different granularities.

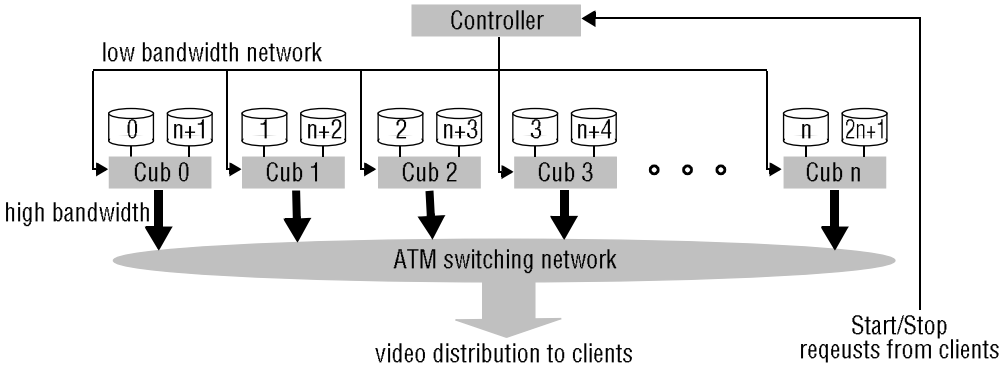
For video, the following scaling methods are appropriate:

- *Temporal scaling* reduces the resolution of the video stream in the time domain by decreasing the number of video frames transmitted within an interval. Temporal scaling is best suited for video streams in which individual frames are self-contained and can be accessed independently. Delta compression techniques are more difficult to handle as not all frames can be easily dropped. Hence, temporal scaling is more suitable for Motion JPEG than for MPEG streams.
- *Spatial scaling* reduces the number of pixels of each image in a video stream. For spatial scaling, hierarchical arrangement is ideal because the compressed video is immediately available in various resolutions. Therefore, the video can be transferred over the network using different resolutions without recoding each picture before finally transmitting it. JPEG and MPEG-2 support different spatial resolutions of images and are well-suited for this kind of scaling.
- *Frequency scaling* modifies the compression algorithm applied to an image. This results in some loss of quality, but in a typical picture, compression can be increased significantly before a reduction of image quality becomes visible.
- *Amplitudinal scaling* reduces the colour depths for each image pixel. This scaling method is, in fact, used in H.261 encodings to arrive at a constant throughput although image content varies.
- *Colour Space* scaling reduces the number of entries in the colour space. One way to realize colour space scaling is to switch from colour to greyscale presentation.

Obviously, combinations of these scaling methods are possible.

A system to perform scaling consists of a monitor process at the target side and a scaler process at the source. The monitor keeps track of the arrival times of messages. When messages get delayed, it is an indication of some bottleneck in the system. The monitor then sends a *Scale-Down* message to the source and it reduces the bandwidth of the stream. After some period of time, the source scales the stream up again. Should the bottleneck still exist, the monitor will again detect a delay and scale the stream down

Figure 15.10 Tiger Video Server hardware configuration



[Delgrossi et al. 1993]. The fundamental problem of the scaling approach is to find good heuristics to avoid unnecessary *Scale-Up* operations and to prevent the system from oscillating.

15.5.2 Filtering

As scaling modifies a stream at the source, it is not always suitable for applications that involve several receivers: when a bottleneck occurs on the route to one target, this target sends a *Scale-Down* message to the source and all targets receive the degraded quality although some would have no problem in handling the original stream.

Filtering is a method that provides the best possible QoS to each target by applying scaling at each relevant node on the path from the source to the target (Figure 15.9). RSVP [Zhang et al. 1993] is an example of a QoS negotiation protocol that supports filtering. Filtering requires that a stream can be partitioned into a set of hierarchical sub-streams, each adding a higher level of quality. The capacity of nodes on a path determines the number of sub-streams a target receives. All other sub-streams are filtered out as close to the source as possible (perhaps even at the source) to avoid transfer of data that is later thrown away. A sub-stream is not filtered at an intermediate node if somewhere downstream a path exists that can carry the entire sub-stream.

15.6 Case Study: The Tiger Video Server

A video storage system that supplies multiple real time video streams simultaneously is seen as an important system component to support consumer-oriented multimedia applications. Several research systems of this type have been developed and some have evolved into products (see [Chung 1998]). One of the most advanced of these is the Tiger Video File Server developed at the Microsoft Research Labs [Bolosky *et al.* 1996].

Design goals ◊ The main design goals for the system were the following:

Video on demand for a large number of users: The typical application is a service that supplies movies to paying clients. The movies are selected from a large stored digital movie library. Clients should receive the first frames of their selected movies within a few seconds of issuing a request and they should be able to perform pause, rewind and fast forward operations at will. Although the library of available movies is large, a few movies may be very popular and they will be the subject of multiple unsynchronized requests, resulting in several concurrent but time-shifted playings of them.

Quality of service: Video streams must be supplied at a constant rate with a maximum jitter that is determined by the (assumed small) amount of buffering available at the clients and a very low loss rate.

Scalable and distributed: The aim was to design a system with an architecture that is extensible (by the addition of computers) to support up to 10,000 clients simultanously.

Low cost hardware: The system was to be built using low-cost hardware ('commodity' PCs with standard disk drives).

Fault tolerant: The system should continue to operate without noticable degradation after the failure of any single server computer or disk drive.

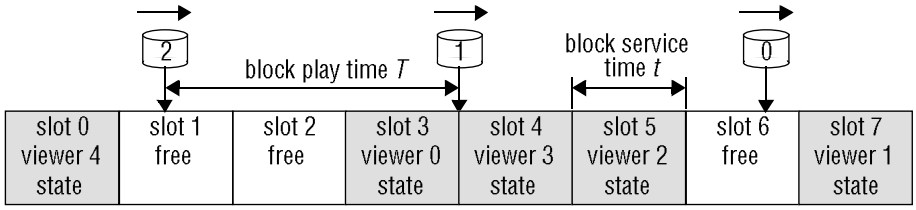
Taken together, these requirements demand a radical approach to the storage and retrieval of video data and an effective scheduling algorithm that balances the workload across a large number of similar servers. The primary task is the transfer of high-bandwidth streams of video data from disk storage to a network and it is this load that has to be shared between the servers.

Architecture ◊ The Tiger hardware architecture is shown in Figure 15.10. All of the components are off-the-shelf products. The Cub computers shown in the figure are identical PCs with the same number of standard hard disk drives (typically between 2 and 4) attached to each. They are also equipped with Ethernet and ATM network cards. The Controller is another PC. It is not involved in the handling of multimedia data and is responsible only for the handling of client requests and the management of the work schedules of the Cubs.

Storage organization ◊ The key design issue is the distribution of the video data amongst the disks attached to Cubs in order to enable them to share the load. Since the load may involve the supply of multiple streams from the same movie as well as the supply of streams from many different movies, any solution based on the use of a single disk to store each movie is unlikely to achieve the goal. Instead, movies are stored in a striped representation across all disks as described below. This leads to a failure model in which the loss of a disk or a Cub results in a gap in the sequence of every movie. This is dealt with by a storage mirroring scheme that replicates the data and a fault tolerance mechanism as described below.

Striping: A movie is divided into *blocks* (chunks of video of equal play time, typically around 1 second, occupying about 0.5 Mbytes) and the sequence of blocks that make up a movie (typically about 7000 of them for a two-hour movie) are stored on disks attached to different Cubs in a sequence indicated by the disk numbers shown in Figure 15.10. A movie can start on any disk. Whenever the highest-numbered disk is reached, the movie is 'wrapped around' so that the next block is stored on disk 0 and the process continues.

Figure 15.11 Tiger schedule



Mirroring: The mirroring scheme divides each block into several portions called *secondaries*. This ensures that when a Cub fails, the extra workload of supplying data for blocks on the failed Cub falls on several of the remaining Cubs and not just one of them. The number of secondaries per block is determined by a *decluster factor*, *d* with typical values in the range 4 to 8. The secondaries for a block stored on disk *i* are stored on disks *i+1* to *i+d*. Note that, provided that there are more than *d* Cubs, none of these disks are attached to the same Cub as disk *i*. With a decluster factor of 8, approximately 7/8ths of the processing capacity and disk bandwidth of Cubs can be allocated to fault-free tasks. The remaining 1/8th of its resources should be enough to to serve secondaries when needed.

Distributed schedule ◊ The heart of Tiger's design in the scheduling of the workload for the Cubs. The schedule is organized as a list of *slots*, where each slot represents the work that must be done to play one block of a movie – that is, to read it from the relevant disk and transfer it to the ATM network. There is exactly one slot for each potential client receiving a movie (called a *viewer*) and each occupied slot represents one viewer receiving a real-time video data stream. The viewer state is represented in the schedule by the address of the client computer, the identity of the file being played, the viewer's position in the file (the next block to be delivered in the stream) the viewer's play sequence number (from which a delivery time for the next block can be calculated) and some bookkeeping information.

The schedule is illustrated in Figure 15.11. The *block play time T* is the time that will be required for a viewer to display a block on the client computer, typically about 1 second and assumed to be the same for all the stored movies. Tiger must therefore maintain a time interval *T* between the delivery times of the blocks in each stream, with a small allowable jitter that is determined by the available buffering at the client computers.

Each Cub maintains a pointer into the schedule for each disk that it controls. During each block play time it must process all of the slots with block numbers that fall on the disks it controls and delivery times that fall within the current block play time. The Cub steps through the schedule in real time processing slots as follows:

1. Read the next block into buffer storage at the Cub.
2. Packetize the block and deliver it to the Cub's ATM network controller with the address of the client computer.
3. Update viewer state in the schedule to show the new next block and play sequence number and pass the updated slot to the next Cub.

These actions are assumed to occupy a maximum time *t* which is known as the *block service time*. As can be seen in Figure 15.11, *t* is substantially less than the block play time. The value of *t* is determined by the disk bandwidth or the network bandwidth, whichever is smaller. (The processing resources in a Cub are adequate to perform the scheduled work for all the attached disks). When a Cub has completed the schedule tasks for the current block play time it is available for unscheduled tasks until the start ofthe next play time. In practice, disks do not provide blocks with a fixed delay and to accommodate their uneven delivery the disk read is initiated at least one block service time before the block is needed for packetizing and delivery.

A disk can handle the work to service *T/t* streams and the values of *T* and *t* typically result in a value > 4 for this ratio. This and the number of disks in the entire system determines the number of viewers that a Tiger system can service. For example, a Tiger system with 5 Cubs with 3 disks attached to each can deliver approximately 70 video streams simultaneously.

Fault tolerance ◊ Because of the striping of all the movie files across all of the disks in a Tiger system, failure of any component (a disk drive or a Cub) would result in a disruption of service to all clients. The Tiger design remedies this by retrieving data from the mirrored secondary copies when a primary block is unavailable because of the failure of a Cub or a disk drive. Recall that secondary blocks are smaller than primary blocks in the ratio of the decluster factor *d* and that the secondaries are distributed so that they fall on several disks that are attached to different Cubs.

When a Cub or a disk fails, the schedule is modified by an adjacent cub to show several *mirror viewer states*, representing workload for the *d* disks that hold the secondaries for those movies. A mirror viewer state is similar to a normal viewer state but with different block numbers and timing requirements. Because this extra workload is shared amongst *d* disks and *d* Cubs, it can be accommodated without disrupting the tasks in other slots, provided that there is a small amount of spare capacity in the schedule. The failure of a Cub is equivalent to the failure of all of the disks attached to it and is handled in a similar manner.

Network support ◊ The blocks of each movie are simply passed to the ATM network by the Cubs that hold them, together with the address of the relevant client. The QoS guarantees of ATM network protocols (see Chapter <Networks>, Section <ATM>) are relied upon to deliver blocks to client computers in sequence and in time. The client needs sufficient buffer storage to hold two primary blocks, the one that is currently playing on the client's screen and one that is arriving from the network. When primary blocks are being served the client need only check the sequence number of each arriving block and pass it to the display handler. When secondaries are being served, the *d* Cubs responsible for a declustered block deliver their secondaries to the network in sequence and it is the client's responsiblity to collect and assemble them in its buffer storage.

Other functions ◊ We have described the time-critical activities of a Tiger server. The design requirements called for the provision of fast-forward and rewind functions. These functions call for the delivery of some fraction of the blocks in the movie to the client in order to give the visual feedback typically provided by video recorders. This is done on a best-effort basis by the Cubs in unscheduled time.

The remaining tasks include the management and distribution of the schedule and and the management of the database of movies, deleting old and writing new movies onto the disks, maintaining an index of movies.

In the initial Tiger implementation schedule management and distribution was handled by the Controller computer. Because this constitutes a single point of failure and a potential performance bottleneck, schedule management was subsequently redesigned as a distributed algorithm [Bolosky *et al.* 1997]. Management of the movie database is performed by Cubs in unscheduled time, in response to commands from the Controller.

Performance and scalability ◊ The initial prototype was developed in 1994 and used five 133MHz Pentium PCs each equipped 48 Mbytes of RAM and three 2Gbyte SCSI disk drives, an ATM netwrok controller and running Windows NT. This configuration was measured under a simulated client load. When serving movies to 68 clients with no faults in the Tiger system the delivery of the data was perfect – no blocks were lost or delivered to clients late. With one Cub failed (and hence three disks) the service was maintained with a data loss rate of only 0.02%, well within the design goal.

Another measurement that was taken was the startup latency to deliver the first block of a movie after receipt of a client request. This will be highly dependent on the number and the position of free slots in the schedule. The algorithm used for this initially would place a client request in the nearest free slot to the disk holding block 0 of the requested movie. This resulted in measured values for the startup latency in the range 2 to 12 seconds. Recent work has resulted in a slot allocation algorithm that reduces clustering of occupied slots in the schedule leaving free slots distributed more evenly in the schedule and improving average startup latency [Douceur and Bolosky 1999].

Concluding comments ◊ Although the initial experiments were made with a small configuration, later measurements were made with a 14 Cub, 56 disk configuration and the distributed scheduling scheme described by Bolosky *et al.* [1997]. The load that could be serviced by this system scaled successfully to deliver 602 simultaneous 2 MBit/second data streams with a loss rate of less than 1 block in 180,000 when all Cubs are functioning. With one Cub failed, less than 1 in 40,000 blocks was lost. These results are impressive and appear to bear out the claim that a Tiger system could be configured with up to 1000 Cubs servicing up to 30,000-40,000 simultaneous viewers. The Tiger system has evolved into a software product entitled Microsoft NetShow Theater Server [Microsoft 2000].

References

[Anderson 1993] Anderson, D.P. (1993), Meta-Scheduling for Distributed Continuous Media. *ACM Transactions on Computer Systems*, Vol. 11, No. 3.

[Anderson *et al.* 1990a] Anderson, D.P., Herrtwich, R.G. and Schaefer, C. (1990), *SRP – A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet*. Technical Report 90-006, International Computer Science institute, Berkeley.

[Anderson *et al.* 1990b] Anderson, D.P., Tzou, S., Wahbe, R., Govindan, R. and Andrews, M. (1990), Support for Continuous Media in the DASH System. *Tenth International Conference on Distributed Computing Systems*, Paris.

[Banerjea and Mah 1991] Banerjea, A. and Mah, B.A. (1991), The Real-Time Channel Administration Protocol. *Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Heidelberg.

[Bolosky *et al.* 1996] Bolosky, W., Barrera, J., Draves, R., Fitzgerald, R., Gibson, G., Jones, M., Levi, S., Myhrvold, N. and Rashid, R. (1996), The Tiger video fileserver, *6th NOSSDAV Conference*, Zushi, Japan, April. http://www.research.microsoft.com/~bolosky/papers/

[Bolosky et al. 1997] Bolosky, W., Fitzgerald, R. and Douceur, J. (1997), Distributed schedule management in the Tiger video fileserver, *16th ACM Symposium on Operating System Principles*, pp. 212-223, St. Malo, France, October. http://www.research.microsoft.com/~bolosky/papers/

[Buford 1994] Buford, J.K. (1994), *Multimedia Systems*, Addison–Wesley Publishers.

[Cheng 1998] Cheng, C. K. (1998) A Survey of Media Servers, Hong Kong University CSIS, November, http://www.csis.hku.hk/~ckcheng/papers/video.ps

[CU-SeeMe, Netmeeting, Viz] <References to be added>.

[Cruz 1991] Cruz, R. (1991), A Calculus for Network Delay. *IEEE Transactions on Information Theory*, Vol. 37, No. 1.

[Delgrossi *et al.* 1993] Delgrossi, L., Halstrick, C., Hehmann, D., Herrtwich, R.G., Krone, O., Sandvoss, J. and Vogt, C. (1993), Media Scaling for Audiovisual Communication with the Heidelberg Transport System. *ACM Multimedia '93*, Anaheim.

[Demers *et al.* 1989] Demers, A., Keshav, S., Shenker, S. (1989), Analysis and Simulation of a Fair Queueing Algorithm. *ACM SIGCOMM '89*.

[Dertouzos 1974] Dertouzos, M.L. (1974), Control Robotics – The Procedural Control of Physical Processes. *IFIP Congress*.

[Douceur and Bolosky 1999] Douceur, J.R. and Bolosky, W. (1999), Improving Responsiveness of a stripe-scheduled media server. *SPIE Proceedings* Vol. 3654. Multimedia Computing and Networking. pp.192-203. http://www.research.microsoft.com/~bolosky/papers/thrifty/mmcn99.ps

[Ferrari and Verma 1990] Ferrari, D. amd Verma, D. (1990), A Scheme for Real-Time Channel Establishment in Wide-Area Networks. *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 4.

[Gibbs and Tsichritzis 1994] Gibbs, S.J. and Tsichritzis, D.C. (1994), *Multimedia Programming*, Addison-Wesley Publishers.

[Govindan and Anderson 1991] Govindan, R. and Anderson, D.P. (1991), Scheduling and IPC Mechanisms for Continuous Media. *ACM Operating Systems Review*, Vol. 25, No. 5, pp.68-80.

[Hyman *et al.* 1991] Hyman, J., Lazar, A.A., Pacifici, G. (1991), MARS – The MAGNET-II Real-Time Scheduling Algorithm. *ACM SlGCOM '91*, Zurich.

[Herrtwich 1995] Herrtwich, R.G. (1995), Achieving Quality of Service for Multimedia Applications, *ERSADS '95, European Research Seminar on Advanced Distributed Systems*, l'Alpe d'Huez, France, April.

[Konstantas *et al.* 1997] Konstantas, D., Orlarey, Y., Gibbs, S. and Carbonel, O. (1997), Distributed Music Rehearsal, *Proc. International Computer Music Conf. 97*. Also available at: http://cuiwww.unige.ch/OSG/publications/TR97/TR97Contents.html

[Kopetz *et al.* 1989] Kopetz, H. *et al.* (1989), Distributed Fault-Tolerant Real-Time Systems – The MARS Approach. *IEEE Micro*, Vol. 9, No. 1.

[Kopetz and Verissimo 1993] Kopetz, H. and Verissimo, P. (1993), Real Time and Dependability Concepts, in Mullender (Ed.), *Distributed Systems*, Edition 2, Addison-Wesley 1993.

[Liu and Layland 1973] Liu, C.L. and Layland, J.W. (1973), Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, Vol. 20, No. 1.

[Leland *et al.* 1993] Leland, W E., Taqqu, M.S., Willinger, W., Wilson, D.V. (1993), On the Self-Similar Nature of Ethernet Traffic. *ACM SIGCOMM '93*, San Francisco.

[Microsoft 2000] Microsoft Corporation (2000), *NetShow Theater Server Web Page*. http://www.microsoft.com/Theater/

[Mok 1985] Mok, A.K. (1985), SARTOR – A Design Environment for Real-Time Systems. *Ninth IEEE COMP-SAC*.

[Laursen *et al*. 1994] Laursen, A., Olkin, J. and Porter, M. (1994), Oracle media server: Providing consumer based interactive access to multimedia data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):470-477, June.

[Nagarajan and Kurose 1992] Nagarajan, R. and Kurose, J. (1992) On Defining, Computing, and Guaranteeing Quality-of-Service in High-Speed Networks. *INFOCOM '92*.

[Nagle 1987] Nagle, J. (1987), On Packet Switches with Infinite Storage. *IEEE Transactions on Communications*, Vol. 35, No 4.

[Partridge 1992] Partridge, C. (1992), A Proposed Flow Specification. *Internet RFC 1363*, Network Information Center.

[Steinmetz and Engler 1993] Steinmetz, R. and Engler, C. (1993), *Human Perception of Media Synchronization*. Technical Report 43.9310, IBM European Networking Center, Heidelberg.

[Szentivanyi 1999] Szentivanyi, S. (1999), *Index to Multimedia Information Sources*. http://dutiem.twi.tudelft.nl/projects/MultimediaInfo/

[Topolcic 1990] Topolcic, C. (Ed.) (1990), Experimental Internet Stream Protocol, Version 2. *Internet RFC 1190*, Network Information Center.

[Turner 1986] Turner, J.S. (1986), New Directions in Communications (or Which Way to the Information Age). *IEEE Communications*, Vol. 24, No. 10

[Vogt *et al.* 1993] Vogt, C., Herrtwich and R.G, Nagarajan, R. (1993), HeiRAT – The Heidelberg Resource Administration Technique: Design Philosophy and Goals. *Kommunikation in verteilten Systemen*, München, Informatik aktuell, Springer.

[Zhang *et al.* 1993] Zhang, L, Deering, S.E., Estrin, D., Shenker, S. Zappala, D. (1993), RSVP – A New Resource Reservation Protocol. *IEEE Network Magazine*, Vol. 9, No. 5.

