A Continuous Media Transport and Orchestration Service

Andrew Campbell, Geoff Coulson, Francisco García and David Hutchison

Computing Department
Engineering Building
Lancaster University
Lancaster LA1 4YR, UK
E.mail: mpg@comp.lancs.ac.uk

ABSTRACT

The desire to transfer continuous media such as digital audio and video across packet switched networks imposes a number of new requirements on transport level communication services. This paper identifies a number of these requirements in the context of an experimental distributed multimedia infrastructure, and reports on research which addresses some of the associated issues. Particular attention is paid to two areas: (i) extended Quality of Service (QoS) provision; and (ii) support for the co-ordination of multiple related connections. We then describe an application level service, known as an orchestrator, which performs synchronisation functions over multiple related transport connections. We also outline the design and implementation of a continuous media transport service which meets the identified requirements. Finally, we outline the way in which the orchestrator and transport services are integrated into an object-based distributed multimedia application platform.

1. Introduction

The evolution of multimedia computing is being influenced both by the requirements of the new application areas and by the increasing capabilities of high-performance multiservice networks. Multiservice communications technology is evolving rapidly in high bandwidth MANs, such as DQDB and FDDI-II, and in broadband ISDN. Currently, broadband transport mechanisms such as ATM are capable of supporting a wide range of multimedia platforms with diverse QoS requirements, from low speed voice (32 Kbit/s) transmission, to very high speed HDTV (100-150 Mbit/s).

There are many application areas where the need for distributed multimedia capabilities is central. A recent survey [Williams,91] characterises such application areas as: office automation, service industry applications, retail applications, domestic applications, science and engineering, and cultural activities. The single most significant attribute of such applications is the incorporation of *continuous media*, particularly audio and video, although the transmission of other data types such as image, text and graphics is usually also involved. The significance of continuous media communications lies in the real-time, isochronous nature of the traffic which is not adequately supported by existing communications sub-systems [Blair,92].

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
COMM'92-8/92/MD,USA
© 1992 ACM 0-89791-526-7/92/0008/0099...$1.50

This paper reports on work which addresses the need for specialised transport services for continuous media traffic. We also highlight the need for co-ordination, or *orchestration*, of multiple related continuous media streams. A familiar example of this is the support of lip synchronisation of video and sound-track components of a film which are stored and transmitted as separate items.

The paper is structured as follows. We first describe, in section 2, the Lancaster experimental distributed multimedia platform. Then we present, in section 3, specific requirements which have arisen from our experiences with the platform, and also from studies of both contemporary and projected multimedia applications. In section 4, we detail our design for a continuous media transport service interface. This is followed by a description of our orchestration architecture in section 5, and a detailed presentation of the lowest level of the architecture in section 6. Finally, in section 7 we offer our conclusions.

2. An Experimental Distributed Multimedia Infrastructure

2.1 Overview of the Infrastructure

The work described in this paper was carried out in the context of an experimental distributed multimedia infrastructure which we have built over the past two years. The work has been carried out partly within the MNI project (funded under the UK SERC Specially Promoted Programme in Integrated Multiservice Communication Networks and co-sponsored by British Telecom Labs), and partly within the European Commission funded OSI 95 project. The ultimate aim of the latter project is to help develop new, OSI standard, transport protocols suited to the new environment of high-speed networks and distributed multimedia applications. In this paper we express our proposals for transport services in an OSI-compatible form. However, the OSI conventions do not imply that the results of our work are applicable only to OSI; in particular, they may also be applied in the TCP/IP world.

Our infrastructure is built around the architecture shown in figure 1. In this model, distributed multimedia applications view an object-based distributed application platform [Coulson,90] based on the Advanced Networked Systems Architecture (ANSA) [APM,89], with extensions for handling continuous media (CM) developed at Lancaster University. The platform isolates applications from the complexities of multimedia devices and CM communications. In our current configuration the platform runs transparently over two heterogeneous environments. One is a standard Ethernet/UNIX environment with Sun Sparcstations with digital audio and video boards, and the other is an experimental transputer based

environment consisting of multimedia workstations and a real-time high-speed network emulator. The Ethernet environment is used for prototyping and application development, but is unable to provide the necessary real-time performance. Once applications have been developed in this environment they are simply moved over to the transputer based system without requiring any changes.

As shown in figure 1, the transport sub-system supports the application platform. The orchestration services, which permit the co-ordination of related transport connections, are layered on top of the transport sub-system, but also extend up into the application platform. In the Ethernet environment the transport sub-system is TCP/IP, but in the transputer based environment it is an experimental CM protocol [Shepherd,91] with rate-based flow control [Cheriton,86], [Chesson,88], [Clark,88].

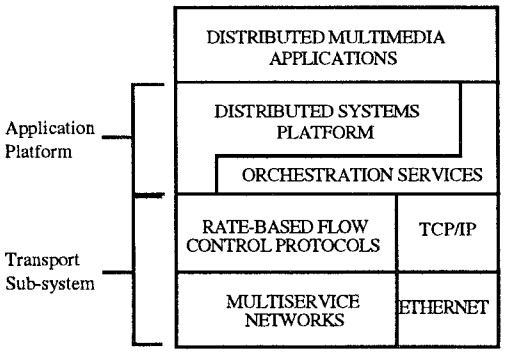

Figure 1: The Lancaster multimedia architecture.

The transputer based infrastructure consists of PCs and Sun/UNIX machines augmented with *Multimedia Network Interface* (MNI) units [Ball,90]. These units attach to conventional workstations and are responsible for interfacing the workstation to the high-speed network. In addition, they transform the host machine into a multimedia workstation by managing all CM sources and sinks at that workstation. Applications run on the host machine, and interface via RPC to the object-based platform which runs on the MNI unit.

Each MNI unit is composed of a cluster of six transputers. Two of these are respectively dedicated to audio and video A/D and D/A conversion. Of the remaining four transputers in the configuration, one runs the object-based platform, another interfaces to the network, and the remaining pair run the rate-based transport protocol. In our experimental network configuration we have two PC based multimedia workstations, a Sun 4/UNIX based multimedia workstation and a PC based storage server.

2.2 The Distributed Multimedia Platform

Distributed multimedia applications do not interact directly with the transport service interface. Instead, all communications services are viewed through two complementary abstractions:-

Invocation

At the platform level, remote interaction is modelled as the invocation of named operations in abstract data type (ADT) interfaces which are accessed in a location independent fashion. Invocation is implemented by means of an RPC protocol known as REX [APM,89] extended to provide the delay bounded communication required for the real-time control of multimedia applications [Coulson,90]. The invocation style of communication is typically used for control and event information. All CM communications are provided by the Stream abstraction described next.

Streams

Streams are the primary extension we have made to the basic ANSA model. They represent underlying CM connections but, in keeping with the ANSA philosophy, they appear as ADT services with first class status at the programming language level. In the supporting layers, CM connections are provided by the protocol service described in this paper, but users at the platform level are isolated from the complexity of the protocol service interface. Streams contain operations to manipulate QoS in media specific terms, and also permit complex connection topologies to be established. Further details of Stream services are presented in [Coulson,91].

We have built a number of applications which run on this platform. The largest is a microscope controller developed in collaboration with a large multi-national chemical company. This provides groups of scientists with remote access to any one of a number of electron or optical microscopes located on a network. Each microscope can send its video output to a number of user workstations, and users can also create multimedia documents containing video sequences and voice annotation. In addition to the microscope application, several other test applications have been implemented including an audio/visual telephone and a video disc jockey console.

3. Requirements of Continuous Media Communications

This section summarises requirements for CM communications which were derived from the design of the Lancaster application platform. We have also gathered requirements from a wide ranging survey of both current and projected distributed multimedia applications. Further details of this survey are presented in [Williams,91]; in this paper, we concentrate exclusively on the implications of the survey for the design of CM transport services.

3.1 Simplex Connections

It is preferable in CM transport systems to provide unidirectional (simplex) virtual circuits rather than the more conventional full duplex VCs. This follows from the inherently unidirectional nature of many CM transmissions; for example, a remote camera to local video monitor connection is typical in a multimedia conferencing environment. The Stream abstraction at the platform level is also unidirectional. As resources must be explicitly reserved for CM VCs (see later), it is wasteful of network capacity to support a duplex VC if only unidirectional transfer is required, and if full duplex communication is required, it is always possible to establish a second VC. An additional disadvantage of full duplex VCs for CM transfers is that, in the general case, the quality of service required of the two directions will be different. For example it may be desired to send colour video in one direction and monochrome in the other: again the more general solution is to use two separate simplex VCs.

3.2 QoS Support

Quality of Service (QoS) parameters in current communications infrastructures allow the specification of user requirements which may or may not be supported by underlying networks. In the OSI framework, parameters dealing with connection establishment, release, and data transfer may be specified, but the required encoding necessary to allow a transport user to access them is not typically provided by transport protocols. Usually, these values are agreed between the carrier and the customer at the time the customer subscribes to a particular network service. Another function of these parameters is to form a basis for charging customers for pre-specified services.

With the emergence of digital video, audio, and other CM types, increased requirements are placed upon QoS support. For example, to support video connections high throughput is required and therefore high bandwidth guarantees will have to be made. Audio, on the other hand, will not require such a high bandwidth. Additionally, both video and audio data can tolerate some percentage loss of packets and bit errors, this of course being dependent on the encoding techniques employed for the individual media.

End-to-end delay and delay jitter (i.e. variance in delay) are new factors which must be taken into account for CM transfers. In particular, interactive CM connections impose stringent delay constraints, derived from human perceptual thresholds, which must not be violated. Delay jitter must also be kept within rigourous bounds to preserve the intelligibility of audio and voice information.

Because of the above requirements, future communication infrastructures must be enhanced to support more flexible and dynamic QoS selection so that transport users are able to precisely tailor individual transport connections to particular requirements. A set of suitable QoS parameters [Hehmann,90] which are meaningful to the transport level and the levels below, and which may be employed in characterising individual CM connections, is as follows :-

- Throughput
- End-to-end delay
- Delay jitter
- Packet Error Rates
- Bit Error Rates

At connection establishment time it should be possible to quantify and express preferred, acceptable and unacceptable tolerance levels for each of these parameters. The requested parameters should then undergo full end-to-end option negotiation. The finally agreed on tolerance levels should then be *guaranteed* for the duration of the connection (or at least an indication should be provided if the contracted values are violated: this is known as a *soft* guarantee).

3.3 Dynamic QoS Control

In distributed multimedia systems it is not sufficient to specify a QoS level, protocol profile and service class at connection time which will statically remain in force for the lifetime of the connection. A more flexible interface is needed whereby users can dynamically alter the QoS of a VC while it is active. This is required because CM information has extremely diverse QoS demands and because these demands frequently vary during the course of a single session. Such dynamicity is illustrated in the following examples:-

- if 'soft' guarantees of QoS level are selected at connection time, the QoS level may degrade, and this will result in the transport user being informed of the degradation. Rather than simply closing the VC down or accepting the degradation, the user may want to re-assess his priorities and, for example, close down another VC to save resources and then upgrade the first VC.
- the user may use the same VC successively for different purposes at different times. Examples are the transmission of full motion video interspersed with intervals of slow motion, or upgrading from monochrome to colour video, or telephone quality to CD quality audio. Another possibility is the in-service insertion of a compression module which may reduce the bandwidth requirement but increase the susceptibility to errors.

Of course, to permit QoS negotiation at the transport service interface, the necessary support must be provided either within or below the transport protocol itself. Thus mechanisms are required to alter link-level bandwidths and/ or processing and buffering resources on intermediate nodes. One possibility is the use of re-configuration at the network level in the context of network layer resource reservation protocols such as the real-time channel administration protocol [Banerjea,91] or ST-II [Topolcic,90] where resource reservations are made at intermediate nodes. Even when a connection must be torn down and re-established in order to achieve an alteration in QoS level, there are strong arguments for doing this transparently behind the transport service interface. It allows the maintenance of buffers and protocol state over the successive connections which may minimise the delay before data flow may resume; it also eases the management burden on the transport user.

3.4 Profile and Class of Service Selection

Although we advocate the use of QoS parameterisation of connections, we do *not* envisage a single fully generic transport protocol that can cater for all types of traffic equally well. Instead, due to the diverse nature of distributed multimedia applications, different protocols will be required for different types of traffic and control data. We thus envisage that communication sub-systems of the future will be composed of horizontal and vertical subdivisions in a *protocol matrix*. The user will be required to select a protocol profile that will provide a suitable service for each traffic type without duplication of functionality at the various protocol layers.

In addition to the provision of profile selection, we see the need for an extension of the more traditional notion of *class of service* selection employed in current OSI standards. In the current OSI framework, classes of transport service are selected according to the network service over which connections are to operate; that is whether the network is connectionless, connection oriented, reliable, unreliable etc.. For future multimedia applications, the notion of class of service selection could be extended to encompass more user oriented functions. For example, to provide the required flexibility in the area of error control, it should be possible for transport users to select from a set of options such as (i) error

detection and indication, (ii) error detection and correction, and (iii) error detection, correction, and indication.

3.5 Remote Connection Facility

In computer supported cooperative work (CSCW) and multimedia conferencing applications, it is natural to structure programs in terms of multiple co-operating modules which communicate via inter-process communication techniques. In this sort of environment it is often convenient for management objects (e.g. Stream services in our application platform) to request that transport services be instantiated for transport service access points (TSAPs) used by other parts of the application [Davies,91]. For example, such a management entity may request that a display device's TSAP be connected to a remote video server, camera, etc..

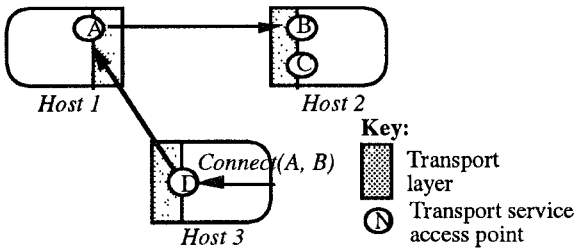

Figure 2: Remote connect scenario.

A similar *remote connect* requirement may arise where a multimedia workstation is composed of multiple loosely coupled processor/ memory pairs [Ball,90]. In such an environment it is likely that a management object running on one processor may ask for transport services to be instantiated at TSAPs on other processors. It is even possible that such managers may request transport services to be instantiated for TSAPs located on remote end-systems. In this case, initiator and source/ sink addresses will all be completely distinct. See figure 2 which illustrates a management object on host 3 making a request to connect TSAP A on host 1 to TSAP B on host 2. In all such cases, it is necessary that the transport service passes all management responses, such as connects or disconnects, to both the initiator and source addresses.

3.6 Synchronisation

There is no support in current transport service architectures for *co-ordinating* multiple related transport VCs. Transport VCs are seen as being entirely distinct from one another, and any relationship must be formed in an application dependent way in the OSI application layer. However, in distributed multimedia systems there is frequently a need to form such relationships between VCs. The prime requirement is in the area of CM synchronisation, although there are other possibilities such as linking QoS degradations on one VC to corresponding compensations on another.

To begin to address the requirements in the area of CM synchronisation, we identify two categories of synchronisation as follows:-

- *event-driven synchronisation* is the act of notifying that a event or set of events has taken place, and then causing an associated action or actions to take place. This must all be done in a *timely* manner due to the real-time nature of CM communication. For example, a user clicking on the stop button relating to a video play-out should cause the play-out to stop instantaneously.
- *continuous synchronisation* is an on-going commitment to a repetitive pattern of event driven synchronisation relationships such as the 'lip sync' relationship between the individual frames in an audio and video components of a play-out.

An example application scenario where it is required to form a continuous synchronisation relationship is a language laboratory where separate audio tracks in different languages are stored on a single server but are to be distributed to different workstations in a real-time interactive language lesson. Another example arises where it is required to associate captions from a text file with an on-going video play-out. In both of these scenarios there is a need to maintain a tight on-going linkage between real-time CM information streams.

We refer to these sorts of linkage as *orchestration*, and have developed an architecture to model and realise such relationships. We employ the term 'orchestration' rather than 'synchronisation' for two reasons. Firstly, as just stated, cross-stream relationships may encompass more than just temporal co-ordination, and secondly the term synchronisation is already overloaded and given a different emphasis in current OSI usage (the OSI concept pertains to checkpointing and state synchronisation between peer entities).

Although it is possible in some situations to support continuous synchronisation simply by multiplexing the different media onto a single VC in the correct ratios, there are strong arguments against this solution [Tennenhouse,90]:-

- the overhead and complexity of multiplexing/ demultiplexing is significant, especially when different encoding/ compression schemes are used for different media; this can lead to excessive real-time delays, especially where it would otherwise be possible to interface directly to hardware such as frame grabbers etc.
- the opportunity to process separate VCs in parallel is lost, thus reducing potential performance;
- multiplexing leads to a combined QoS which must be sufficient for the most demanding medium; this may be both expensive and unsuited to some component media types;
- multiplexing is not an option where media originate from different sources.

An analysis of the continuous synchronisation problem suggests that the following support should be provided by the infrastructure. Section 6 illustrates how our design satisfies these requirements.

- the ability to start related CM data flows *precisely* together. If the relationship is not correctly initiated, there is no possibility of maintaining a correct temporal relationship.
- the ability to start and stop the flow of CM information on sets of related connections in an atomic and near instantaneous manner. Also, if the data flow on a connection is stopped and the source is, say, fast forwarded to another point in the stored media, the play-

out should resume from the new position on re-start without old data being left in the communications buffers.

- the ability to create related VCs with the same QoS. This is so that the connections will maintain a compatible temporal transmission rate in the required ratio. For example, it may be required to associate ten sound samples with each video frame in the above mentioned scenario to maintain lip-sync.

- the ability to *monitor* the on-going temporal relationship between related VCs, and to *regulate* the VCs to perform fine grained corrections if synchronisation is being lost. It is almost inevitable that related connections will eventually drift out of synchronisation. This is due to such factors as the potentially long duration of CM connections in typical applications, the inevitable discrepancies between remote clock rates, and temporary 'glitches' occuring in individual VCs.

Finally, note that the need for continuous synchronisation support is only required when all the CM sources to be orchestrated are *stored*. Whenever *live* sources, such as the output from a camera of a microphone, are involved, it is only necessary to ensure that the latency of the VC is compatible over related VCs. This is because with live media, there is no control over when the information flow starts (it depends when the camera is switched on!), and no possibility of altering the speed of a live media flow. Even on a low grade connection, live media with constant logical rates will always play out in real-time albeit with varying degrees of delay, jitter and packet loss.

3.7 Continuous Media Data Transfer

Conventional data transfer interfaces to transport services, such as the *send()* and *recv()* systems calls found in interfaces such as TLI [Olander,86] and Berkeley sockets [Berkeley,86], or the *data.request* and *data.indication* primitives in OSI, are not optimal for CM transmissions. This is because such interfaces have no model of an *on-going commitment* to service isochronous CM information. Each buffer of data to be transferred is treated as a separate entity with no particular temporal relationship to its successors and predecessors.

The conventional interfaces are also inefficient in a CM context. Each time a system call such as *send()* or *recv()* is issued, three pieces of information are passed [Govindan,91]: *synchronisation information* (i.e. it is implicit that the data transfer should commence 'now'), *data location* (i.e. the location of the data to be transferred), and *data transfer* (perhaps involving a copy to or from system space). However, it is possible in a CM systems to avoid specifying all this information for each unit of CM information. This is because the isochronous characteristics of application/ transport data transfers are implicitly known and thus the repetitive sequence of calls becomes unnecessary.

Our experiments in this area favour the adoption of a data transfer interface based around *shared circular buffers* with access contention between separate application and protocol threads controlled by semaphores. The advantages of such a scheme are as follows:-

- data location is implicit in the value of pointers associated with the shared buffers, and no data copying is involved (especially in situations where the shared memory may be mapped directly into frame buffers etc.);

- as long as the application and the protocol are running at compatible rates no explicit producer/ consumer synchronisation need take place;

- the state of the shared buffers can be used by specialised real-time operating system schedulers [Govindan,91] to ensure that applications processing CM respect the isochronous nature of the data;

- the time spent blocking by both the application and the transport entity can be measured by monitoring the state of the synchronisation semaphores. These statistics are used by the orchestration service as will be described in section 6.

At the data transfer interface we support the notion of *logical data units* for structuring CM. The boundaries of these units are preserved irrespective of their size in bytes. We apply the principle that at each time period there will always be something to transmit (i.e. one logical unit) even when CM data is variable bit rate encoded.

3.8 Group and Multicast Communications

The requirement for multicast arises from the nature of projected multimedia applications, many of which will be in the area of CSCW. In such applications a multicast facility is required for both transactional communication (RPC), and for CM connections. In transactional multicasts, it may be required that any member of the multicast group should be capable of transferring information to all other members at any time. However, in a CM based multicast session a simple 1:N topology is usually all that is required. Appropriate support for group addressing must be provided in the transport layer, but multicast support will be the responsibility of the underlying communications sub-system.

4. Design of the Continuous Media Transport Service

We now detail the design of the CM transport service used in the Lancaster experimental infrastructure. Details of the underlying transport protocol may be found in [Shepherd,91]. Because the emphasis of this paper is orchestration, we do not discuss all the issues raised in section 3: in particular multicast issues are not discussed further. We also omit discussion of some of the more traditional constituents of a complete transport system such as TSAP allocation, datagram services and priority mechanisms. We do, however, assume that such features will be available in the standard protocol matrix that we have proposed.

4.1 Connection Management Services

Connection management services deal with the establishment, release, and management of transport VCs. In the Lancaster architecture, these services are used by Stream objects and are not directly seen by applications.

4.1.1 Connection Establishment

Connection establishment is a fully confirmed service similar in nature to that employed in the OSI reference model. Full option negotiation of QoS is performed on connection, and in keeping with the requirements for CM VCs, the

negotiated QoS represents the required characteristics of the media in one direction only, from source to sink.

Three addresses are provided with each primitive to cater for remote connects as discussed in section 3.5. The initiator address represents the caller of the service and the destination and source addresses represent the two end-points which are to be connected. The addresses contain a network address to identify the end-system, and a TSAP to identify a unique endpoint within the addressed end-system. The primitives employed for connection establishment and release, together with their associated parameters, are outlined in Table 1. The QoS tolerance levels are expressed in terms of upper, lower and preferred limits of the QoS parameters described in section 3.2 and the protocol and class of service parameters are as discussed in section 3.4.

Primitives	Parameters
T-Connect.request	initiator-address, src-address, dest-address, protocol, class-of-service, QoS-tolerance-levels, vc-id
T-Connect.indication	"
T-Connect.response	"
T-Connect.confirm	"
T-Disconnect.request	initiator-address, vc-id
T-Disconnect.indication	initiator-address, vc-id, reason

Table 1: Connection establishment and release primitives and their associated parameters.

We now illustrate the implementation of the remote connect facility. Note that where it is required to form a VC in the conventional sense (i.e. where the initiator is also the sender), the caller simply sets the initiator to be the same as the source address.

When the initiating application generates a T-Connect.request primitive, this is relayed to the source entity, where on arrival a T-Connect.indication is issued to the application attached to the addressed TSAP. The source application can then accept the call with a T-Connect.response, or reject it with a T-Disconnect.request. If the request is accepted, the transport entity generates a T-Connect.request and from this point onwards the conventional connect protocol is followed. The outcome of this is eventually relayed back to the application which initiated the call. If the call is rejected then the initiating application receives a T-Disconnect.indication. A successful remote connect is illustrated in figure 3. Note that the request may be rejected by the source, the destination or the network provider.

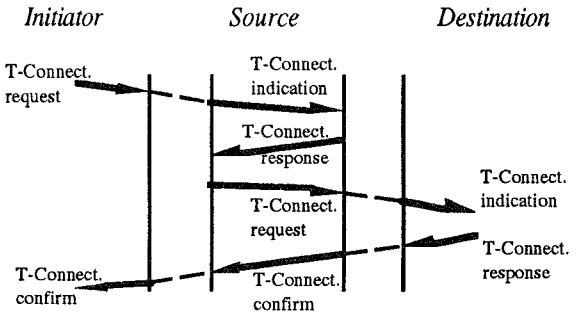

Figure 3: Successful remote connection establishment.

It is also possible, of course, for an initiator to request that a VC be remotely released. A T-Disconnect.request is sent to the source or destination where on arrival a T-Disconnect.indication is issued to the attached application. The application may then issue a T-Disconnect.request which will be relayed to the other end-point, and on arrival will generate a T-Disconnect.indication.

Primitive	Parameters
T-QoS.indication	initiator-address, src-address, dest-address, initial-QoS-tolerance-levels, sample-period, vc-id, current-QoS-tolerance-levels, error-number

Table 2: Primitive issued by transport service to notify transport user of degradation of negotiated QoS.

4.1.2 Notification of QoS Degradation

If the class of service selected from the transport protocol for a particular VC incorporates the error indication facility (see section 3.4), a primitive is required to convey to the transport user any errors or QoS degradations which may have occurred. This primitive is generated by the transport entity monitoring the VC over a suitable sample period. The primitive and its parameters are set out in Table 2. The primitive will identify the VC, provide the negotiated QoS tolerance levels, the sample period, the measured performance of the negotiated QoS tolerance levels within that sample period, and an error number to identify which of the tolerance levels have suffered degradation.

4.1.3 QoS Re-negotiation

For QoS re-negotiation, a fully confirmed service with full option negotiation similar to that used for connection establishment is employed. The primitive is detailed in Table 3. The time sequence diagram in figure 3 above is also applicable to the T-Renegotiate service.

Primitives	Parameters
T-Renegotiate.request	initiator-address, src-address, dest-address, new-QoS-tolerance-levels, vc-id
T-Renegotiate.indication	"
T-Renegotiate.response	"
T-Renegotiate.confirm	"

Table 3: Primitives employed for the re-negotiation of QoS. The primitives may be initiated by a transport protocol user, or by the transport protocol itself.

The transport protocol may need to tear down a VC and open a new one in order to supply the required QoS. Thus it should be capable of sustaining state so that it can re-synchronise once the new VC is established. If for some reason a modified service cannot be provided, the service request will be rejected with a T-Disconnect.request, and the transport user will receive a T-Disconnect.indication. However, in this latter case, the existing VC is *not* torn down; the T-Disconnect.indication simply indicates that the new service level requested can not be supported. Note that the T-Renegotiate service allows the QoS of a VC to be modified but not its protocol type or class of service.

5. Orchestration Architecture

This section presents an architecture which addresses the need for the co-ordination of multiple related CM transport streams as identified in section 3.6. We distribute the functionality of the orchestrator over three architectural regions:-

- the distributed application platform,
- an area corresponding to the OSI upper layer architecture (ULA), and
- an area closely associated with the transport layer interface (see figures 1 and 4).

It can be seen from figure 4 that orchestration is a multi-layered activity. Each layer provides *policy* to its lower neighbour and *mechanism* to its upper neighbour. This design provides both flexibility and efficiency because the lower layers are simply provided with targets, and all exceptions, error handling and re-structuring are handled in the layers above.

In essence, the operation of the three orchestration layers is as follows. The *high level orchestrator* (HLO) provides the view of orchestration seen by users of the application platform. It is a location independent ADT service interface which contains orchestration related named operations. Applications pass Stream interfaces to these operations and the HLO arranges to have the required continuous synchronisation performed by the lower layers according to a *policy* specified by the application. Policies include constraints on how 'strict' the continuous synchronisation should be and actions to take on failure.

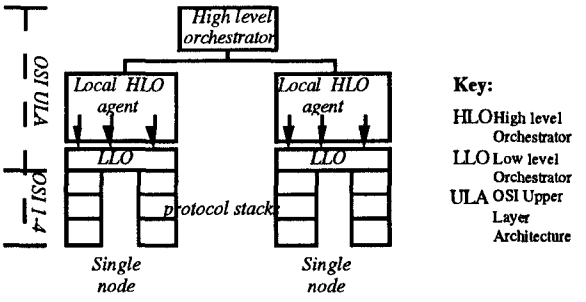

Figure 4: Three level orchestration architecture.

The HLO is responsible for finding the physical locations of the connections underlying the given Stream interfaces, and thus choosing the node from which the lower levels of orchestration will be co-ordinated. The node selected, known as the *orchestrating node*, is that common to the greatest number of VCs† (see figure 5). For example, if it was required to orchestrate separate video and audio tracks of a film stored on separate storage servers, the common sink would be designated as the orchestrating node by the HLO. Having identified the orchestrating node, the HLO creates an ADT interface onto the selected HLO agent. This is passed back to the initiating application, and enables the application to control the on-going orchestration session via invocation.

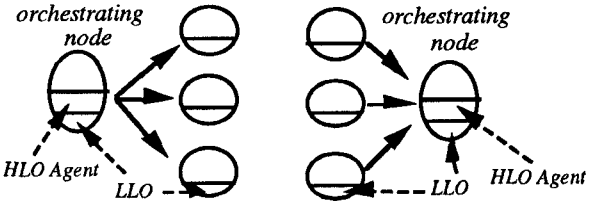

Figure 5: Orchestrating at the common node.

Below the platform level, the remaining two orchestration components are responsible for realising the behaviour and policy required by the HLO. At this level, the orchestration process is realised as *HLO agents* which monitor and regulate multiple transport VCs via a *low level orchestrator* (LLO) interface in a continuous feedback loop. For each orchestrated group of connections, a single HLO agent runs on the orchestrating node, and an LLO instance runs on all source and sink nodes of all the orchestrated VCs. The HLO only interacts with its local LLO instance (see figure 5), but the multiple LLO instances interact with each other via Orchestrator PDUs (OPDUs), on out of band connections. These connections must

† In our initial implementation we make a restriction that groups of orchestrated connections must have a common node, either at the source or the sink. With this restriction in force, we are able to use the clock at the common node as the datum for continuous synchronisation across connections, and use a simple clock synchronisation scheme. It should, however, be possible to lift this restriction without changing the basic architecture by including a general purpose clock synchronisation function (e.g. NTP [Mills,89]) within the orchestrator protocols. At the present time, however, it is not fully clear whether such a generalised facility is either required in practice or realistic in implementation.

have guaranteed bandwidth to support the necessary real-time communication of orchestration primitives. In our system, we use a special internal *control VC* associated with each transport connection [Shepherd,91].

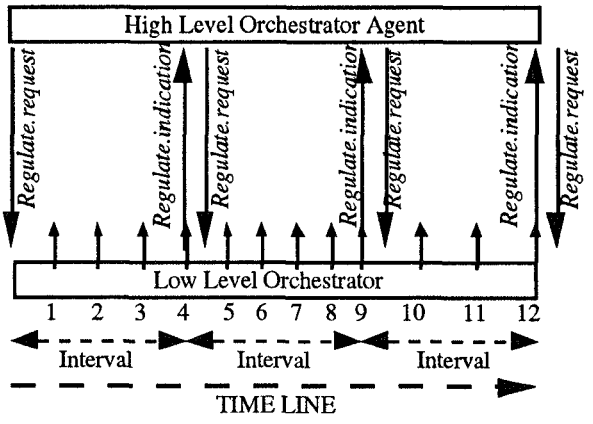

Figure 6: Interaction between HLO Agent and LLO.

Figure 6 illustrates the interaction between the HLO agent and the local LLO (note that figure 6 only illustrates the interaction required for a *single* orchestrated connection; in general multiple connections will be involved). The HLO agent supplies the LLO with *rate targets* for each orchestrated VC over specified *intervals*. These targets ensure that each orchestrated VC runs at the required rate, relative to the master reference clock maintained at the orchestration node, for the required synchronisation relationship between the orchestrated VCs to be maintained (for brevity, we do not discuss our clock synchronisation scheme in this paper). The LLO attempts to meet the required rate target over each interval for each VC, and reports back at the end of the interval on its actual success or failure. Then, on the basis of these reports, the HLO agent sets new targets for the next interval which compensate for any relative speed up or slow down among the orchestrated connections. The LLO operates on a *best effort* principle; it is the responsibility of the HLO agent to take appropriate action (e.g. set new targets or re-negotiate the connection QoS) if the LLO consistently fails to meet targets.

The small numbered arrows in figure 6 represent the delivery of quanta of CM information which are released by the sink LLO instance to the application thread at times determined by the HLO initiated targets. These quanta are known as *OSDUs*, and are the units of CM information meaningful to applications (e.g. video frame or text paragraph).

The orchestration services maintain a special *OSDU sequence number* field for each OSDU, which starts from zero from when the connection is first used; a second such field, known as an *event* field, is employed for use by the Orch.Event primitive (see later). Both these fields form part of an OPDU which is sent along with each OSDU. OSDU and OPDU boundaries are maintained by the transport service. This is possible in our system because, at connection establishment time, the application passes the maximum size of an OSDU as a QoS parameter, and this (plus the size of the OPDU) is interpreted as a lower bound on buffer size allocation.

When applications write/ read OSDUs into/ from the transport system's circular shared buffers, they write/ read from the beginning of a buffer, and may also write/ read the current OSDU size to/ from an auxiliary memory location.

In this paper we are mainly concerned with the lowest level orchestration functionality associated with the transport layer, as this level provides the fundamental orchestration mechanism. Therefore we do not present the orchestration functionality at the application platform or OSI application layer in any more detail, but concentrate instead on the LLO interface and functionality.

6. The Low Level Orchestrator

The LLO orchestration interface consists of two sets of OSI-like primitives as follows:-

- *primitives for priming, starting and stopping orchestrated groups of connections, and*
- *primitives for controlling and monitoring individual orchestrated connections*

The first set operates over a grouping of transport connections. The primitives provide the ability to atomically *prime*, *start* and *stop* the flow of data in these connections both atomically and instantaneously as required by the analysis in section 3.6. By 'priming' we mean that the receiver's buffers are initially filled with the required data for a particular connection, in preparation for a subsequent *start* command. This allows related media flows to be started simultaneously, which is a fundamental requirement for subsequent continuous synchronisation.

The second set of primitives operate on single transport connections in an orchestrated grouping. They enable the controlling HLO agent to set and monitor the above mentioned flow rate targets. As stated above, connections will attempt to meet these targets on a best effort basis. Primitives are also provided to report back to the HLO agent on the actual performance achieved at the end of each interval.

6.1 LLO Instantiation

We assume that before an HLO agent attempts to instantiate an LLO orchestrating service, the VCs to be orchestrated have already been established. It is also assumed that the initiator of the LLO service (i.e. an HLO agent) resides on the common orchestration node.

Table 4 summarizes the primitives involved and their parameters. The initiator issues a Orch.request which causes an OPDU to be passed to the LLO instance at each source and sink of all VCs in the orchestrated group. This OPDU contains an orch-session-id supplied by the HLO. Each source and sink determines whether or not it is able to support the requested orchestration request, and replies to the initiating LLO. If accepted by all the remote LLO services, the HLO will receive a Orch.confirm, or if rejected a Orch.Release.indication giving a reason why orchestration was rejected. Rejection may occur because some LLO instance has no table space available, or because one or more of the specified VCs do not exist etc..

Primitives	Parameters
Orch.request	orch-session-id, list-of-vc-ids
Orch.confirm	"
Orch.Release.request	orch-session-id
Orch.Release.indication	orch-session-id, reason

Table 4: Orchestration request and release primitives together with their associated parameters.

Orchestration is released by issuing a Orch.Release.request. Orchestration will also be released implicitly if all the VCs in an orchestrated session are closed.

6.2 Group 1: Primitives for Priming, Starting and Stopping

The primitives to prime connections, and atomically start and stop the data flow on groups of orchestrated VCs are illustrated in Table 5.

6.2.1 Orch.Prime

The Orch.Prime primitive is used to ensure that multiple streams of remotely stored CM data can be started together in a synchronised way. It is also useful in ensuring that time critical data can be pre-fetched and made available when required. A third application of Orch.Prime arises when it is required to clean out the buffers in an end to end connection. This need arises when a user stops a media play-out and then wishes to seek to another part of the media before resuming. If the buffers were not flushed in this situation, a short burst of media buffered from the previous play would be discernible.

The time sequence operation of the Orch.Prime primitive is shown in figure 7. Orch.Prime.indication primitives are passed to the application threads associated with each end of each orchestrated VC. On receipt of the Orch.prime.indication, each application thread is expected to respectively start generating data for transmission or preparing to accept data. If any application thread is not in a position to do this it can reply with a Orch.Deny rather than a Orch.Prime.response. In either case, the result is passed back to the LLO at the orchestration node.

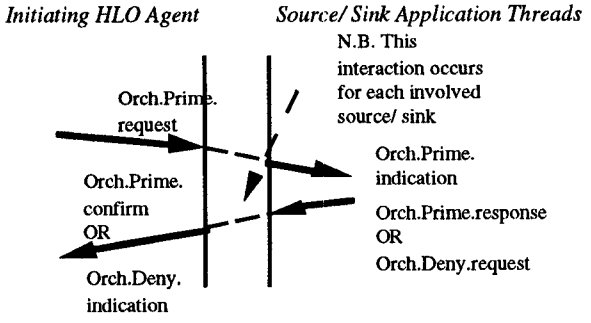

Figure 7: Orch.Prime.

As data begins to arrive at the sinks, the sink LLOs allow the receiver's communications buffers to fill, but prevent the data from being delivered to the receiving application threads. Note that this implies a close implementation relationship between the LLO and the transport service. When the receive buffers are eventually full, each sink LLO notifies the LLO, which eventually relays either a Orch.Prime.confirm or a Orch.Deny.indication to the originating HLO. At this point, the source will also be blocked by the protocol's flow control mechanism, but the pipeline is filled and ready to go.

Primitives	Parameters
Orch.Prime.request	orch-session-id
Orch.Prime.indication	"
Orch.Prime.response	"
Orch.Prime.confirm	"
Orch.Start.request	orch-session-id
Orch.Start.indication	"
Orch.Start.response	"
Orch.Start.confirm	"
Orch.Stop.request	orch-session-id
Orch.Stop.indication	"
Orch.Stop.response	"
Orch.Stop.confirm	"
Orch.Add.request	orch-session-id, vc-id
Orch.Add.indication	"
Orch.Add.response	"
Orch.Add.confirm	"
Orch.Remove.request	orch-session-id, vc-id
Orch.Remove.indication	"
Orch.Remove.response	"
Orch.Remove.confirm	"
Orch.Deny.request	orch-session-id, reason
Orch.Deny.indication	"

Table 5: Orchestration primitives for priming, starting and stopping.

6.2.2 Orch.Start

Having primed an orchestrated set of connections, the HLO will eventually issue a Orch.Start.request. This re-starts the transport protocol and unblocks the previously filled receive buffers so that they may be emptied by the sink application thread. As data is waiting at all the sinks, all the receiving application threads in the orchestrated group will start to receive data at (almost) the same instant. A Orch.Start.indication will be sent to each source and sink application thread as a result of the Orch.Start.request in an analogous manner to that illustrated in figure 7. However, if the system is in a primed state, these threads will not need to take any special action as they are already set up to produce/ consume data, but are blocked by the protocol.

6.2.3 Orch.Stop

Orch.Stop instantaneously freezes the flow of data in the specified VCs. Note, however, that the flow of data can not actually be stopped until the underlying protocol's flow control mechanism can take effect. As with the Orch.Prime primitive, the receive buffers are made unavailable to the application sink thread before they are drained so that data is available for a subsequent primed start. Also, the underlying protocol's flow control mechanism will eventually block the source application threads in the same way as Orch.Prime.

6.2.4 Orch.Add and Orch.Remove

The Orch.Add and the Orch.Remove primitives are employed to either add or remove a particular VC or VCs from an orchestrated group. The time sequence operation is again similar to figure 7. Note that when VCs are removed from an orchestrated group they are not disconnected and thus data may still be flowing.

6.3 Group 2: Primitives for Regulating and Monitoring

Primitives	Parameters
Orch.Regulate.request	orch-session-id, vc-id, target-OSDU#, max-drop#, interval-length, interval-id
Orch.Regulate.indication	orch-session-id, vc-id, interval-id, OSDU#, dropped#, proto-block-times, app-block-times
Orch.Delayed.request	orch-session-id, vc-id, source-or-sink, interval-length, OSDUs-behind
Orch.Delayed.indication	"
Orch.Delayed.response	"
Orch.Delayed.confirm	"
Orch.Event.request	orch-session-id, vc-id, event-pattern
Orch.Event.indication	"

Table 6: Orchestration Primitives for Regulation and Monitoring

The primitives used to implement the regulation and monitoring functions on orchestrated VCs are illustrated in Table 6 and described in the following sub-sections. For brevity, we do not discuss the time sequence operation of these primitives.

6.3.1 Orch.Regulate

6.3.1.1 Orch.Regulate.request

This primitive is issued by the HLO agent to set a flow rate target for the forthcoming interval for individual VCs in an orchestrated group as explained in section 5. Parameters include the orchestration session ID, the ID of the VC to be controlled, the length of the forthcoming interval, and an interval-id to identify the corresponding Orch.Regulate.indication. The target-OSDU# parameter denotes the OSDU sequence number which should ideally be delivered to the sink application thread at precisely the end of the interval. This is the way in which the required flow rate target is specified: the required rate is calculated as ((*target-OSDU# - current-OSDU#*) / *interval-length*).

When the LLO is attempting to meet the requested flow rate target for a connection, there are three possible cases: it may be on target, behind target or ahead of target. If the connection is on target, no special action need be taken. If, however, either of the other cases is true, the following compensatory strategies are available to the LLO:-

- if a connection is behind, its sole compensatory strategy is to drop OSDUs. The final parameter, max drop#, states the maximum number of OSDUs which the VC may discard in order to achieve its flow rate target. All such discards are performed at the source by incrementing the source shared buffer pointer. This permits the source application thread to immediately insert another OSDU and thus overwrite the previous one before it is sent. This strategy may help a delayed VC to catch up and meet the link delivery target in the case that it is lack of transport bandwidth which is causing the delay.
- if, on the other hand, a connection is *ahead* of schedule, the compensatory action is simply to block. Note that, as with the Orch.Stop primitive, the flow control mechanism of the underlying transport protocol must be capable of rapid adaptation for this to be feasible. The rate based mechanism used in our protocol [Shepherd,91], has proved to be adequate for this purpose. Note also that, for both these compensatory actions, the LLO must take responsibility for attempting to spread compensatory actions over the length of the target interval to avoid unnecessary jitter.

If these compensatory actions are not available to the LLO (e.g. a max-drop# of zero will often be chosen where a no-loss medium such as voice is involved), then the necessary corrections must be taken by the HLO agent on the basis of the information gathered via the Orch.Regulate.indication primitive.

6.3.1.2 Orch.Regulate.indication

This primitive is used to report back to the HLO agent on the performance actually achieved by each orchestrated connection in comparison to the targets set by the HLO for the interval just completed. The interval-id parameter is used to match the indication to a prior request. The LLO will continuously generate Orch.Regulate.indications at the end of each interval length as established by the last Orch.Regulate.request. The statistics reported include the OSDU# *actually* delivered at the end of the interval, the number of OSDUs *actually* dropped, and the times spent blocking by both the application and protocol threads at both the source and sink ends of the VC. This blocking time information is gathered by associating timers with the shared circular buffer semaphores described in section 3.7.

The blocking time information is used by the HLO agent to determine which part of the system was responsible for any failure to meet the flow rate target. Based on this information, the HLO can take compensatory action if required. For example, if the application threads spent an excessive amount of time blocked, the protocol throughput was presumably too low and the HLO may re-negotiate the QoS of the VC. Alternatively if the protocol threads were blocked, the application threads were presumably slow in producing/ consuming data. In this latter case the HLO will probably issue a Orch.Delayed primitive.

6.3.3 Orch.Delayed

This primitive is issued by the HLO in response to the situation described above. The effect is to cause an indication to be delivered to the application thread(s) causing the delay. The intended interpretation of a Orch.Delayed.indication is that the thread is not running sufficiently fast to produce/ consume data at a rate required by the client of the location independent orchestration service. Applications so informed may take any appropriate action such as requesting more processor resources, or they may merely give up by replying with a Orch.Deny.request.

6.3.4 Orch.Event

This primitive is used to register an interest in a particular application defined event associated with some OSDU; it thus provides support for event-driven synchronisation.

To register an interest in some application defined event, a Orch.Event.request is issued to an LLO instance at the sink end of an orchestrated VC, together with a value representing the event. Subsequently, the sink LLO instance will match this value, which is not interpreted in any way by the LLO, against the values in the event fields of the OPDUs associated with incoming OSDUs (see section 5) on the set of orchestrated VCs. If the event in the OSDU matches the registered bit pattern, a Orch.Event.indication is raised. To actually cause an event to be initiated, the event fields of OSDUs may optionally be set by the source application thread when writing an OSDU.

An example of use of the event mechanism is when a change of encoding is being signalled in the data stream such as the introduction of a particular compression scheme. It would obviously be possible to implement such a scheme in an ad-hoc manner in the application layer, but this would require that application threads examine each incoming OSDU. The present scheme avoids complicating application code, permits system dependent optimisations to be made, and also permits OSDUs to be dumped directly into, say, a video frame buffer.

7. Conclusions

We have outlined the design of a CM transport service and an associated orchestration service which permits real-time co-ordination between distinct transport connections. Our transport service has simplex VCs with flexible QoS configuration, including re-negotiation, and a facility for establishing connections remotely. Details of related work in the field of CM transport protocols may be found in [Wolfinger,91] and [Hehmann,91]. Our work is notable for the close integration of protocol concerns with those of the distributed application platform that we consider an essential part of future systems building.

We also describe an orchestration architecture consisting of three components: a high level orchestrator which makes HLO services available from our object-based application platform, HLO agents which control and monitor orchestrated connections, and low level orchestrators which sample and regulate the flow of CM information over intervals as directed by the HLO agent. The orchestration system is architecturally separate from the transport sub-system although the two components must be intimately related in implementation.

We make an assumption in the design of our transport service and orchestrator that the underlying transport protocol employs *rate-based* flow control [Cheriton,86], [Chesson,88], [Clark,88] as opposed to a traditional window based technique [Postel,81], [Stallings,87]. This assumption has emerged from our experimentation with real-time CM, where we have found rate-based flow control to be admirably suited for transporting CM. Attractive characteristics include the de-coupling of flow control from the error control mechanism, and the natural correspondence between the notions of continuous data flow and rate controlled transmission. In the future, we aim to experiment with other types of buffer management and flow control schemes to determine pragmatically to what extent they are suitable for the support of CM transmissions and orchestration. A second assumption is that when the protocol is operating in an internet environment, a network level resource reservation protocol such as ST-II [Topolcic,90] or SRP [Anderson,91] will need to be used to guarantee resources in intermediate nodes.

The context of our work is that it forms part of the OSI 95 project in which the Lancaster role is to develop new transport and higher level communications protocols suited to the new environment of high-speed multiservice networks and distributed multimedia applications. OSI 95 is participating in an initiative to introduce a New Work Item into ISO/OSI on High-Speed Transport Service and Protocol. The OSI 95 project is not the only international attempt to introduce a new high-speed transport protocol: ANSI X3S3.3 has recently issued draft descriptions of a new high speed transport service (HSTS) and protocol (HSTP), which are for further study within OSI 95. A further issue currently being studied is the generalisation of our transport and orchestration model to multicast applications.

Finally, a number of practical issues remain to be more fully investigated. These include the orchestration of VCs with no common node, the coupling between the orchestration layers and the transport system, the efficient handling of multicast orchestration, and the use of other transport protocols in our architecture. Also, further experimentation with applications in the field needs to be conducted to provide proof by experience of the value of the work described here.

Acknowledgements

Part of this work was carried out within the MNI project (funded under the UK SERC Specially Promoted Programme in Integrated Multiservice Communication Networks (grant number GR/F 03097) and co-sponsored by British Telecom Labs), and part within the OSI 95 project (ESPRIT project 5341, funded by the European Commission).

References

[Anderson,91] Anderson, D.P., R.G. Herrtwich, and C. Schaefer. "SRP: A Resource Reservation Protocol for Guaranteed Performance Communication in the Internet", *Internal Report* University of California at Berkeley, 1991.

[APM,89] APM Ltd. "The ANSA Reference Manual Release 01.01", Architecture Projects Management Ltd., Poseidon House, Castle Park, Cambridge, UK, July 1989.

[Ball,90] Ball, F., D. Hutchison, A.C. Scott, and W.D. Shepherd. "A Multimedia Network Interface." *3rd IEEE COMSOC International Multimedia Workshop (Multimedia '90),* Bordeaux, France, Nov 1990.

[Banerjea,91] Banerjea, A., and B.A. Mah. "The Real-Time Channel Administration Protocol." *Second International Workshop on Network and Operating System Support for Digital Audio and Video,* IBM ENC, Heidelberg, Germany, Springer Verlag, 1991.

[Berkeley,86] Berkeley. "UNIX Programmer's Reference Manual, 4.3 Berkeley Software Distribution", Computer Systems Research Group, Computer Science Division, Univ. of California at Berkeley, CA, USA. April 1986.

[Blair,92] Blair, G.S., F. Garcia, D. Hutchison and W.D. Shepherd. "Towards New Transport Services to Support Distributed Multimedia Applications", *Presented at Multimedia '92: 4th IEEE COMSOC International Workshop*, Monterey, USA, April 1-4, 1992.

[Cheriton,86] Cheriton, D.R. "VMTP: A Transport Protocol for the Next Generation of Communication Systems." *ACM SIGCOMM '86* Aug 1986,

[Chesson,88] Chesson, G. "XTP/PE Overview." *Proc. 13th Conference on Local Computer Networks,* Minneapolis, Minnesota, Oct 1988.

[Clark,88] Clark, D., et al. "NETBLT: A High Throughput Transport Protocol." *ACM SIGCOMM,* Pages 353-359, 1988.

[Coulson,90] Coulson, G., G.S. Blair, N. Davies, and A. Macartney. "Extensions to ANSA for Multimedia Computing", *To appear in Computer Networks and ISDN Systems* MPG-90-11, Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK. Oct 25 1990.

[Davies,91] Davies, N., G. Coulson, N. Williams, and G.S. Blair. "Experiences of Handling Multimedia in Distributed Open Systems", *Presented at SEDMS '92, Newport Beach CA, April 1992*; also available from Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK. November 1991.

[Govindan,91] Govindan, R., and D.P. Anderson. "Scheduling and IPC Mechanisms for Continuous Media." *Thirteenth ACM Symposium on Operating Systems Principles,* Asilomar Conference Center, Pacific Grove, California, USA, SIGOPS, Vol 25, Pages 68-80.

[Hehmann,90] Hehmann, D.B., M.G. Salmony, and H.J. Stuttgen. "Transport services for multimedia applications on broadband networks." *Computer Communications* Vol 13 No. 4, 1990, Pages 197-203.

[Hehmann,91] Hehmann, D.B., R.G. Herrtwich, W. Schulz, T. Schuett, and R. Steinmetz. "Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High Speed Transport System" *Second International Workshop on Network and Operating System Support for Digital Audio and Video,* IBM ENC, Heidelberg, Germany, 1991.

[Mills,89] D.L. Mills, "Internet Time Synchronisation: the Network Time Protocol", *Internet Request for Comments No. 1129* RFC-1129, October 1989.

[Olander,86] Olander, D.J., G.J. McGrath, and R.K. Israel. "A Framework for Networking in System V." *Proceedings of the 1986 Summer USENIX Conference,* Atlanta, Georgia, USA, Pages 38-45.

[Postel,81] J. Postel. "Transmission Control Protocol", *Internet Request for Comments No. 793* RFC-793, September 1981.

[Shepherd,91] Shepherd, W.D., D. Hutchison, F. Garcia and G. Coulson. "Protocol Support for Distributed Multimedia Applications." *Second International Workshop on Network and Operating System Support for Digital Audio and Video,* IBM ENC, Heidelberg, Germany, Nov 18-19 1991.

[Stallings,87] W. Stallings. "Handbook of Computer Communications, Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards.", Macmillan, New York, 1987.

[Tennenhouse,90] Tennenhouse, D.L., "Layered Multiplexing Considered Harmful", *Protocols for High-Speed Networks*, Elsevier Science Publishers B.V. (North-Holland), 1990.

[Topolcic,90] Topolcic, C. "Experimental Internet Stream Protocol, Version 2 (ST-II)", *Internet Request for Comments No. 1190* RFC-1190, October 1990.

[Williams,91] Williams, N., G.S. Blair. "Distributed Multimedia Application Survey", *Internal Report N. MPG-91-11.* Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK. April 1991.

[Wolfinger,91] Wolfinger, B., and M. Moran. "A Continuous Media Data Transport Service and Protocol for Real-time Communication in High Speed Networks." *Second International Workshop on Network and Operating System Support for Digital Audio and Video,* IBM ENC, Heidelberg, Germany, 1991.