Multimedia Tools and Applications, 16, 99–136, 2002

Distributed Video Production: Tasks, Architecture and QoS Provisioning*

RAOUF BOUTABA rboutaba@bbcr.uwaterloo.ca
Department of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, Ontario N2L 3G1, Canada

NED NING REN, YASSER RASHEED AND ALBERTO LEON-GARCIA
Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario M5S 3G4, Canada

Abstract. In this paper, we apply a top-down approach to describe the architecture components of Distributed video production (DVP) systems. We first introduce the concept of a Distributed video production environment and describe typical DVP tasks involved. We then propose a generic layered functional model that captures the characteristics of the DVP architecture. Subsequently, we identify system and network performance parameters and discuss possible network protocol stack realizations for the transport of DVP traffic. Finally, we propose a quality of service-sensitive model for resource allocation and optimization.

Keywords: Distributed video production, real-time video, MPEG2, quality of service, ATM, internet protocol and network adaptation

1. Overview

Advances in networking and computing technologies have made modern multimedia systems a reality. Applications such as videoconferencing and video-on-demand will become a part of our everyday life in the near future. The standardization of high definition television (HDTV) is the result of developments in multimedia technology, and signals the beginning of a new era in the broadcasting industry. High quality programs with interactive user interfaces will be delivered to homes over a large number of channels, resulting in radical changes to viewer experiences.

The booming multimedia technology has also placed great demands on today's video production industry. Not only higher program quality and more complex special effects are required to satisfy new viewer-expectations, but a larger quantity of program contributions are also required in a short period of time. The new requirements must be satisfied by revolutionary changes to production facilities.

Production equipment used in today's video production facilities is usually very expensive. Most equipment are specialized to a set of dedicated tasks and interconnected by cables, which results in expensive equipment centralized to a single physical location—the studio. However, the task of video production is distributed in nature. Production resources

*This project is partially funded by the Canadian Institute for Telecommunication Research (CITR).

and live video contributions usually come from more than one physical location, e.g. daily news, sports events or even weather forecasting. The actual shootings of footages do not always happen inside of the studio, however editing and final touch-ups of the programs need to be done inside the studio—where equipment belongs. Moreover, this production process is not a single person's task, but rather requires cooperative working of personnel from each stage of the production, such as the communication between scriptwriters and the director, cooperation between cameramen and editors, ... etc. Therefore, the operational nature of video production requires a distributed cooperative environment. Furthermore, equipment inside each studio is dedicated to the workloads that are local to the studio, and therefore may not be fully utilized. Every studio needs to purchase its own expensive switcher, editor, recorder and other complex equipment. In case the studio does not have the right equipment to perform a certain task, the work would then have to be contracted-out to other production houses where it could be properly performed. This contracting-out process is expensive and has a long turn-around time. It is also unreliable to some extent since certain production decisions have to be made by the contracting studio while the program is being processed there.

The development of new high-speed network technologies is promising to provide quality of service (QoS) guarantees to high bandwidth applications. On the other hand, the development of MPEG2 standards enables flexible digital representation of high quality video. The combination of these two technologies is expected to provide a solution to the problem existing in the production industry today: the lack of support for distributed cooperative working environment and the inefficient utilization of expensive studio equipment.

In this paper, we discuss a distributed video production (DVP) approach, which replaces cable connections in traditional studios with high-speed network connections. Since the network is not restricted to connecting local studio equipment, it can also interconnect equipment from remote studios, therefore enabling users at different physical locations to work on a single task, jointly over the network. This creates a distributed working environment for the production process. Production resources such as video processors, devices for storage, compression, mixing and special effect may then be distributed around the network in the form of various servers as shown in figure 1. These resources can all be shared remotely among different users, thus ensuring effective utilization of the resources and removing the need for each studio to own its dedicated equipment. The DVP environment will therefore enable broadcasters to improve economic efficiency and time-to-market of broadcast program productions.

1.1. Objectives

The objectives of this paper are threefold:

- Identify different distributed video production tasks and their related quality of service requirements
- Propose a layered architecture for distributed video production systems and identify typical system and network requirements
- Propose a QoS-sensitive model for resource allocation and optimization

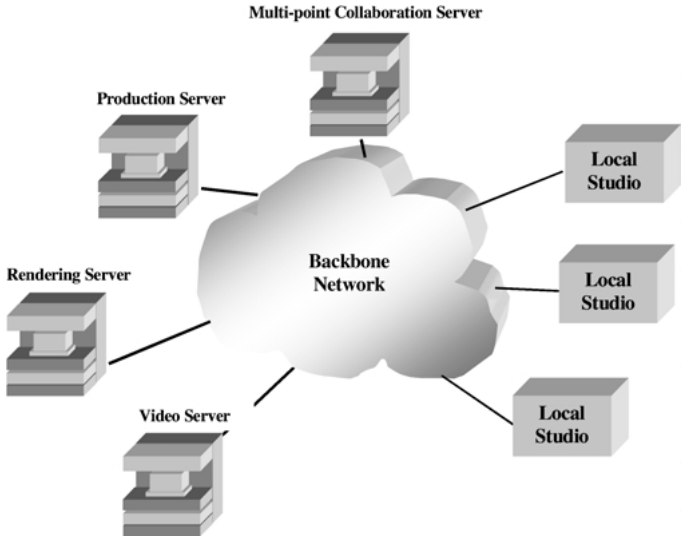

Figure 1. A typical distributed video production system.

1.2. Outline

The paper is organized as follows. In Section 2, we introduce the distributed video production environment and identify the different tasks performed in DVP applications. In Section 3, we present a general architecture for distributed video production environments. Subsequently in Section 4, we discuss QoS requirements for DVP applications including bandwidth, delay and error requirements, present possible protocol stack realizations for the transport of DVP traffic over high-speed networks and discuss the suitability of different network adaptation protocols for DVP applications. Finally in Section 5, we propose a QoS-sensitive model for resource allocation and optimization in a DVP environment.

2. Distributed video production tasks

In this Section, we describe several typical tasks performed in a traditional production studio. We then discuss how these tasks may be carried out in a distributed video production environment and present a possible setup for the DVP version of each task.

2.1. Editing

Editing is one of the most common tasks performed in a studio. Its main purpose is to arrange footages together and form the final presentation program. At the time of writing this paper, computer-based non-linear editing is becoming the standard. Analog video is digitized, stored on computer disks and processed digitally. The random access nature of

computers enables editors to arrange footages in a dynamic fashion, making the non-linear editing process more efficient and flexible than traditional linear editing.

The editing process usually consists of two phases: the off-line phase and the on-line phase. During the off-line phase, producers and editors experiment with various footage arrangements, and create a draft model of the final program in the form of an edit decision list (EDL). The EDL contains information about how footages are connected together and what switcher effects should be used to connect them. These effects include cutting, wiping, dissolving, fading and keying, etc. During the on-line phase, the final program is created; effects are performed according to the EDL. In addition, complex digital effects (computer graphics and animations), that may have not been rendered for the off-line version, are also performed.

We discuss two possible situations for editing tasks in a DVP setting. In the first scenario, which we call "*local editing*" (see figure 2), the local studio possesses its own high quality editing system, and can therefore perform editing (especially on-line editing) locally. However, the local studio might not have all the footages required for the edit. In this case, missing footages stored in a digital format on a remote video server may then be retrieved, as they are needed in the edit. Typically, the edit would be in the form of cuttings between programs stored on the server and programs generated locally. A good example is a news program showing a reporter announcing news events inside the studio, and subsequently the sequence changes to previously recorded footages about the event. The previously recorded footages might be physically stored on the remote DVP video server. The local studio issues command signals to the DVP video server and controls the program stream that is sent back, as shown in figure 2. The command signals include instructions such as play, pause and search. The program stream sent back to the local studio includes video, audio and data streams. Possible content of the data stream includes time codes, synchronization signals and meta-data. The latter includes description about each portion of the program such as the date and time the scene was created, location of the scene and notes made by the director at the time of shooting etc.

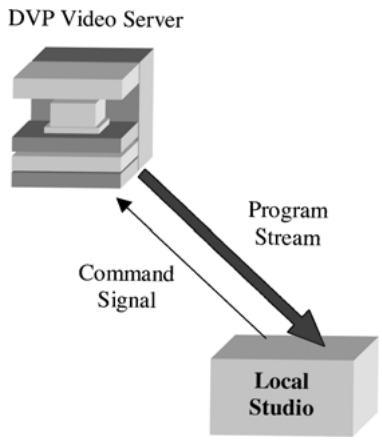

Figure 2. DVP local editing.

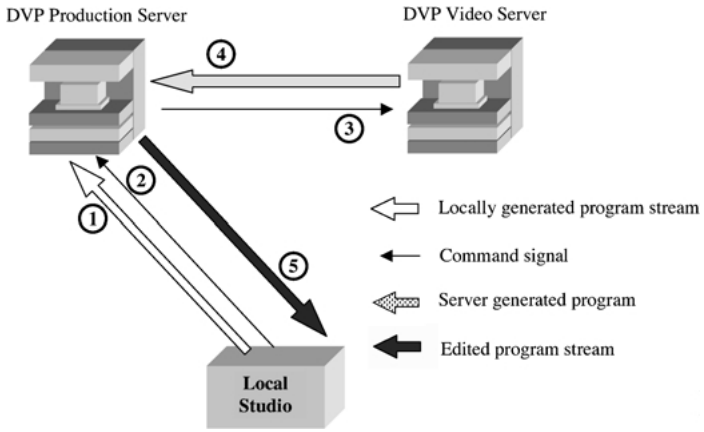

Figure 3. DVP remote editing.

The second scenario of editing in a DVP environment is remote editing. In this case, the local studio does not possess local high quality editing equipment. Therefore, the actual editing will have to be done remotely. In a conventional studio setup, this would result in a time consuming contracting process as mentioned earlier. A solution is to use a production server in a DVP environment as shown in figure 3, where locally generated program streams are delivered to a DVP production server, along with command signals to control the editing process. The production server may request additional program streams required by the edit from a DVP video server. The editing process is then carried out on the production server under the control of the local studio, and the final edited program is sent back to the studio.

2.2. Keying

Keying is the process of displaying different sources of videos at various positions of the screen at the same time. A key is an "electronic hole cut into a picture, and the area cut away is filled with a different video source" [6]. One example of a key is the digital effect box over a newscaster's shoulder. While the person is describing a news event, graphics or video footages related to the event can be displayed in the box. In this case, the keying equipment cuts a "*hole*" in the studio camera's video signal and fills it with the graphics or video footages from another source.

Studios that do not have high quality keying equipment available locally may use remote keying techniques in the DVP environment. Video streams that have to be keyed together can be transported to the production server through the DVP transport network. The keying operation is carried out there on the production server and the resultant video is delivered back to the studio.

In some cases, local studios may only have the foreground video clips available but not the background clips. An example could be a news program where a footage of a reporter in the foreground is shot in the studio while the background is composed of news event

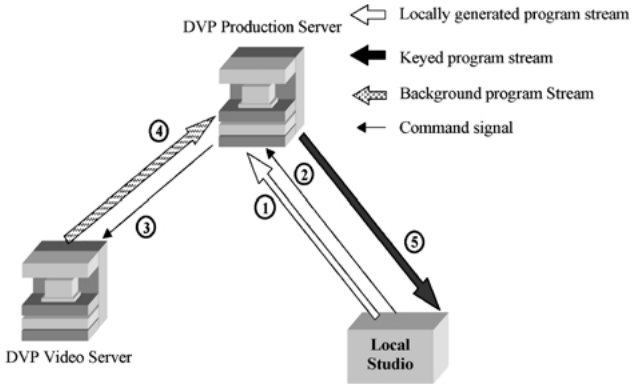

Figure 4. DVP remote keying.

clips located on a remote DVP video server. In this case, the production server has to obtain the background clips from the video server, mix them into the foreground stream using information carried by the key signal and return the result back to the local studio (see figure 4).

2.3. Virtual studio production

Virtual studio production allows mixing of live video with synthetic or natural imagery. An example is a footage of live actors shot in the studio, mixed with computer generated graphics or outdoor scenarios that create the illusion that the performing is actually carried out on the artificially generated set. The combination of the live video and the artificial background is also called a "*virtual set*."

In virtual studio production, various techniques are used to synchronize the foreground camera with the background camera (or virtual camera if the background is computer rendered). These techniques involve feeding the virtual camera with movement information and such that it matches changes in the foreground. The differences lie in how this tracking information is derived. Some approaches involve installing sensors on the foreground camera to record movement, zooming and focusing information of the camera. The tracking data is sent to the background-rendering computer and the virtual camera attempts to match the movements of the foreground camera according to changes in these parameters. This technique enables synchronization between live and artificial video objects. Other approaches utilize optical tracking techniques that place special markers in the foreground set such as on the blue wall. Markers must be blue as well, however they may have a different shade than the blue color of the wall. Image processing tools use the position of these markers as references and calculate the camera tracking information. The background set is then re-rendered according to the tracking information, while the blue markers are keyed out from the final video by keying devices.

In the case where local studios do not have virtual studio production equipment, this task can be executed in a remote fashion within the DVP environment. A possible setup is

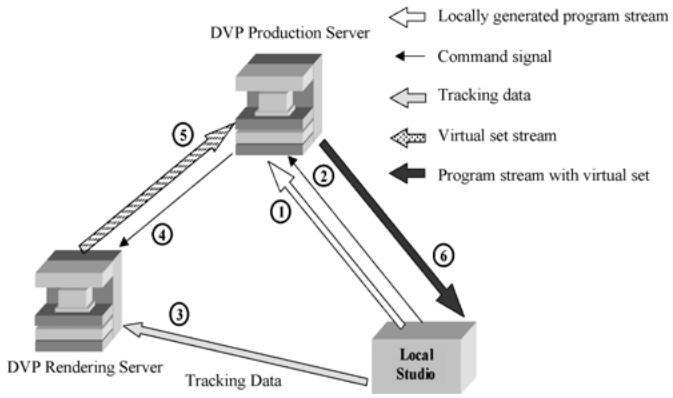

Figure 5. DVP virtual studio production.

shown in figure 5. The local studio sends the foreground video stream to the DVP production server; the studio also specifies parameters of the background set in the command signal. The production server chooses a rendering server according to the type of background set that has to be rendered, and identifies the rendering server to the local studio. Upon reception of information about the rendering server, the local studio transmits tracking data and rendering parameters to the rendering server. The rendering server processes the request and delivers rendered set to the production server where they are mixed with the live foreground video and transported back to the local studio.

2.4. Distributed joint production

Another possible production task that can be performed in the DVP environment is the distributed joint production. In this scenario, several geographically dispersed studios can jointly work on a program production. An example is a remote news interview program where people involved in the program are located in different studios. The DVP environment allows a distributed interview to be carried out among them. The distributed production process may utilize DVP resources such as various DVP servers available on the DVP transport network, therefore allowing more advanced remote productions to be carried out in the DVP environment. More details of the distributed joint productions are discussed in Section 3.2.4.

In this section, we presented a brief description for typical tasks performed in distributed video production. Based on these production tasks, we present next a preliminary model for a layered architecture for the DVP environment.

3. Architecture of distributed video production systems

The objective of this section is to discuss the architecture components of DVP systems and propose a preliminary functional model that can capture the system characteristics and

requirements. In our model, we take into consideration the different DVP tasks (explained in Section 2), their system requirements, in addition to the computational, storage and network transport resources needed to support them. In our work, we adopt a layered model in order to highlight the separation of functions in each layer of the architecture.

Distributed video production systems consist of local studios and a set of remote servers interconnected through a high-speed backbone network, as was shown in figure 1. Local studios connect to the backbone network through a DVP interface system. Different types of DVP servers exist, each with different functionality. Possible servers include media servers that provide storage for media clips, rendering servers that provide graphics and animation rendering, production servers that are responsible for production tasks and multi-point production servers for coordination of remote joint productions. This DVP architecture is potentially scalable by adding new servers to the network to satisfy increasing usage demand or the emergence of new task requirements. We first discuss local studios in Section 3.1, then explain our proposed model for the architecture of DVP servers in the following section.

3.1. Local studio and DVP interfacing system

Most of conventional studios are currently migrating their analog infrastructure to digital, in an attempt to leverage the advantages of digital compression, storage and transmission technologies. In a digital environment, traditional analog routing and switching devices are replaced with a high speed Local Area Network (LAN) [3], thus enabling local studios to communicate locally and potentially utilize remote production resources.

A DVP interfacing system connects the local studio LAN to the backbone network, as shown in figure 6, and provides local studios with the ability to query and request services from remote DVP servers. DVP interfaces resemble subscriber terminal units (STU) in a video on demand (VoD) system in hiding the actual physical location of the remote server; control workstations on the local studio LAN use DVP remote resources without realizing these resources are actually remote to them. The DVP interface may also adapt to heterogeneous technologies used in different studios by resolving compatibility issues between local equipment and signal formats used on the DVP backbone network. DVP

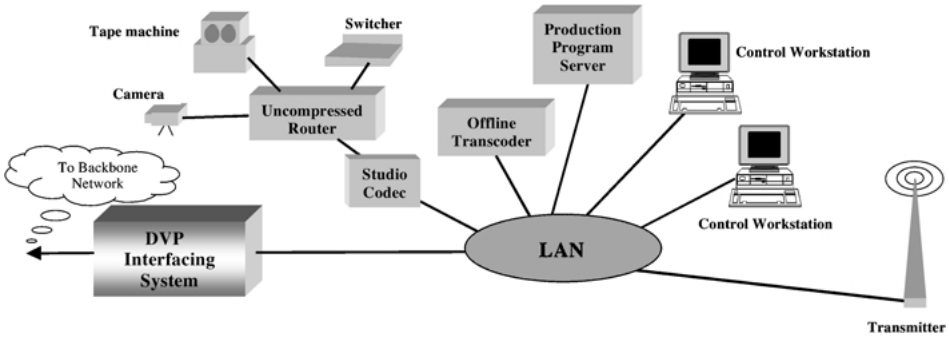

Figure 6. Local studio and DVP interfacing system.

interfaces therefore hide the details of the DVP system from the local studio technology. Finally, the local studio DVP interface is responsible for negotiating and matching QoS guarantees from both environments. QoS requested by local workstations are translated and communicated to remote systems on the DVP backbone network.

3.2. DVP servers

In our architectural model, various types of DVP servers share a common layered architecture from a functional point of view. We divide our generic layered server functional model into three levels, namely, the application, system and transport levels. The application level consists of application software that provides clients with required services, while the system level represents hardware components that support the application software, and finally, the transport level includes functions such as network and storage device interfaces. Each server has a management backplane that performs system management functions including operation control and resource partitioning.

For every DVP server type, the relative levels may perform different functions. In the following sections, we first describe our layered functional for a generic DVP server, then explain the role of each type of servers in detail.

3.2.1. Server architecture. We divide the generic DVP server into four entities: the three levels mentioned above in addition to the management back plane, as shown in figure 7. The application level represents the service layer, which contains software that provides each DVP server with various service capabilities, such as editing, rendering, keying, network access interfaces, . . . etc. Some service layer functions supporting basic server functionality may be common to all DVP server types, such as network access and management interfaces. Meanwhile, other functions of the service layer may be unique to each individual server and specifically designed according to services the server has to provide. Examples are editing, rendering, keying and production software suites.

The system level consists of functional systems and coding/synchronization layer. The functional system is composed of the necessary hardware components that support software

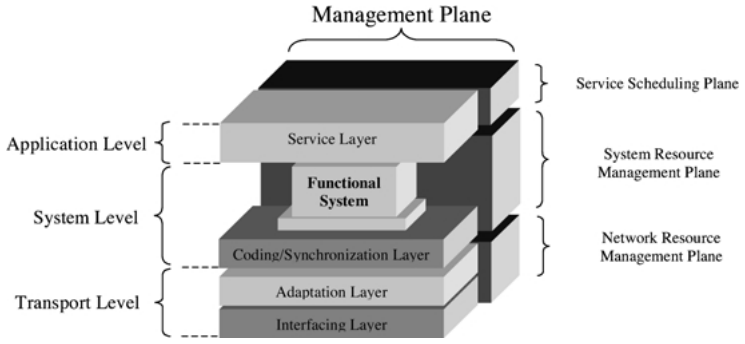

Figure 7. DVP server architecture.

applications. Functional systems are in general distinct for each server, depending on the tasks the server has to perform. Examples are dedicated hardware equipment capable of providing storage, rendering and multi-point control capabilities. The coding/synchronization layer is responsible for proper coding and decoding of streams as they are passed between the system level and the transport level. This layer is also responsible for inter- and intra-media synchronization. Delay jitter introduced by the transport network, storage system and other server systems are compensated at this layer using appropriate buffering techniques. Examples of coding/synchronization layers include the MPEG coding/systems layer and real-time transport protocol (RTP). More details on the coding and synchronization are presented in Section 4.

The transport level is divided into an adaptation layer and an interfacing layer. The adaptation layer performs the necessary mapping between the application and the network packet formats, in addition to adapting the QoS delivered by the network to the level of QoS requested by the application. Examples of adaptation layers include TCP and UDP for IP networks, and AAL1, AAL2 or AAL5 for ATM networks. The interfacing layer consists of the network transport system, which includes software interfaces (e.g., sockets) in addition to hardware network interface cards (e.g., Ethernet or ATM). More detail on the adaptation and interfacing layer is presented in Section 4.

3.2.1.1. Resource management plane. Each server has a management back plane that can handle system and network resource management functionality. The management system is divided into three planes according to the three system architecture levels we described above (application, system and transport levels). The three management planes are service scheduling plane, system resource management plane and network resource management plane as shown in figure 7.

The service-scheduling plane receives resource utilization information for system and network resources. Based on the resources available throughout the system and from the network, it provides a list of currently available services to local studios. This service list is communicated to the local DVP interface. Service requests originating from local studio sites are granted, modified or denied based on availability of resources. We describe the service scheduling in more detail in Section 5.

The system resource management plane manages DVP's resources on a system-wide basis. It is different from conventional system management modules due to the fact that DVP deals mainly with multimedia data. The plane takes the unique high capacity, real-time and continuous characteristic of multimedia data into account while managing the system. The system management plane contains a multimedia operating system which deals with issues such as real-time CPU scheduling, memory management, I/O scheduling and multimedia file system management. The system resource information is passed to the service-scheduling plane, and is used to make admission decisions for service requests.

3.2.2. Server functionality. We have defined four types of servers in the last section, which share similar system components with distinct functional systems as shown in figure 8. Following is a discussion of the role of each of these servers.

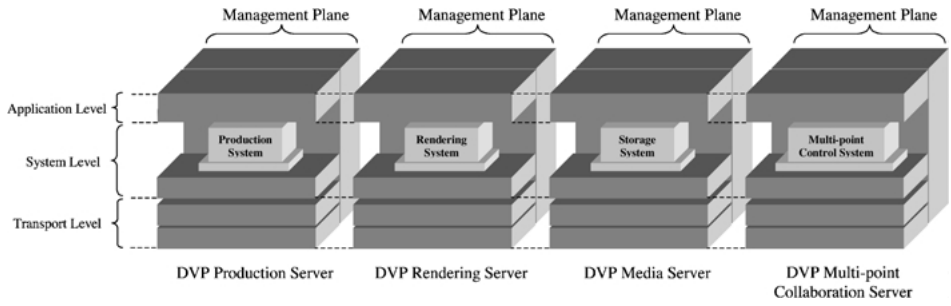

Figure 8. DVP servers layered functional model.

3.2.2.1. Production server. The production server supports day-to-day production tasks required in a typical studio. These tasks include various types of wiping, dissolving, keying and non-linear editing as was discussed in Section 2. Depending on the capacity of the server, the production server may have hundreds of input and output ports in order to accommodate the needs of connected users such that all users can be served simultaneously. Each user issues production commands to the system as if it was local to them. These commands are then transmitted by the local DVP interface to the production server, along with video footages originated from the user site. Video streams and command streams enter the production system through a reserved input port for the user and are processed or mixed with video footages from other sources, such as a video server. The processed streams are finally sent back to the user through an output port.

The input and output ports, server and network resources must all be reserved prior to the beginning of a production operation. User specified QoS requirements are translated by the DVP interface at the local studio and communicated to the server. Management mechanisms in the server keep track of available server resources and decide whether user requested QoS could be accommodated.

3.2.2.2. Rendering server. The rendering server is responsible for providing computer-generated graphics, as requested by users. Rendering needs of DVP are usually satisfied by powerful processors inside the rendering server. The rendering tasks include generating text characters, studio logos, digital effects, real-time animations and virtual sets. Some of these tasks are very complex in nature and especially if performed in production quality, they would require enormous amounts of computational power, let aside the multi-user and multitasking environment the rendering server has to support. Graphics-specialized processors are likely to be customized to support the heavy processing needs.

3.2.2.3. Media server. Media servers store large collections of footages, serving as a media database for daily production needs of the DVP facility. Storage systems usually have large capacities in order to support high quality media. In addition, in order to satisfy the real-time characteristics of production tasks, the storage system must be able to access and serve data in real-time. The requirements for media storage and real-time data access are similar to VoD servers. This will likely result in a similar design for storage systems in DVP media servers.

In order to reduce the amount of storage space required, media streams may be stored in compressed formats. Compression algorithm such as MPEG-2 attempt to reduce file sizes while effectively preserving media quality. Layered coding may be used in order to accommodate the heterogeneous quality requirements existing in the DVP environment. Unlike media quality required in VoD environments where users are commonly satisfied with broadcast quality images, the quality requirements encountered in DVP are heterogeneous, depending on different operations each users performs. A user constructing an EDL in an off-line editing process may not need high image quality, therefore only a few base layers of the video program may suffice to reconstruct basic quality images. On the other hand, a user performing a final edit or live-broadcast may require production or broadcast quality, consequently more layers of the program would be needed for reconstruction into full production quality images. The layered coding approach can potentially accommodate heterogeneous quality requirements without trans-coding, thus avoiding possible quality degradation at the expense of increased coding overhead associated with the layered format. Data placement algorithms should consider the layered media file formats while storing them.

3.2.2.4. Multi-point collaboration server. The multi-point collaboration server (MCS) enables multi-site joint productions. Its role is similar to MCU (Multi-point control unit) in a videoconferencing. In order to support interconnections among multiple sites, each geographical region has a MCS assigned, to which local studios in the region initiate multi-site production requests. The MCS contacts the rest of the participants to set up the session. Participants include other MCSs and servers. The MCSs decide how the multi-point connections are established and how media streams are interchanged between sites. MCSs are also responsible for managing distributed sessions by monitoring on-going sessions, performing multi-point information distribution and group cooperation control.

We will use a remote interview example in order to illustrate the role of MCSs and some of their similarities to MCUs in videoconferencing. In this example, three local studios wish to participate in an interview, a production server processes the interview footages and adds effects such as cutting between views of participants. For each of the three studios, the DVP system also forwards streams from the other two sites to them. This allows the participants to communicate with each other (figure 9). In addition, in order to enhance the viewing experience of the program further, a computer generated background or a virtual set may be added to the scene. This would create the illusion that participants are side-by-side co-existing in the virtual set. In this case, keying information has to be transmitted from the studios to the production server, which indicates which part of the shot belong to the background and can be keyed out (key signals are not shown in the figure for simplicity). A rendering server generates the required virtual set, and the final produced footages is stored on a video server.

Local studios send their full quality media streams to the MCSs, which are then forwarded to the production server. Each person in the interview needs to see and hear participants of other sites, but it is not necessary for them to receive full quality streams since a low-resolution image and telephone quality voice stream are adequate for the purpose. For this reason, MCS1 can extract a low-resolution version of the stream originating from studio1

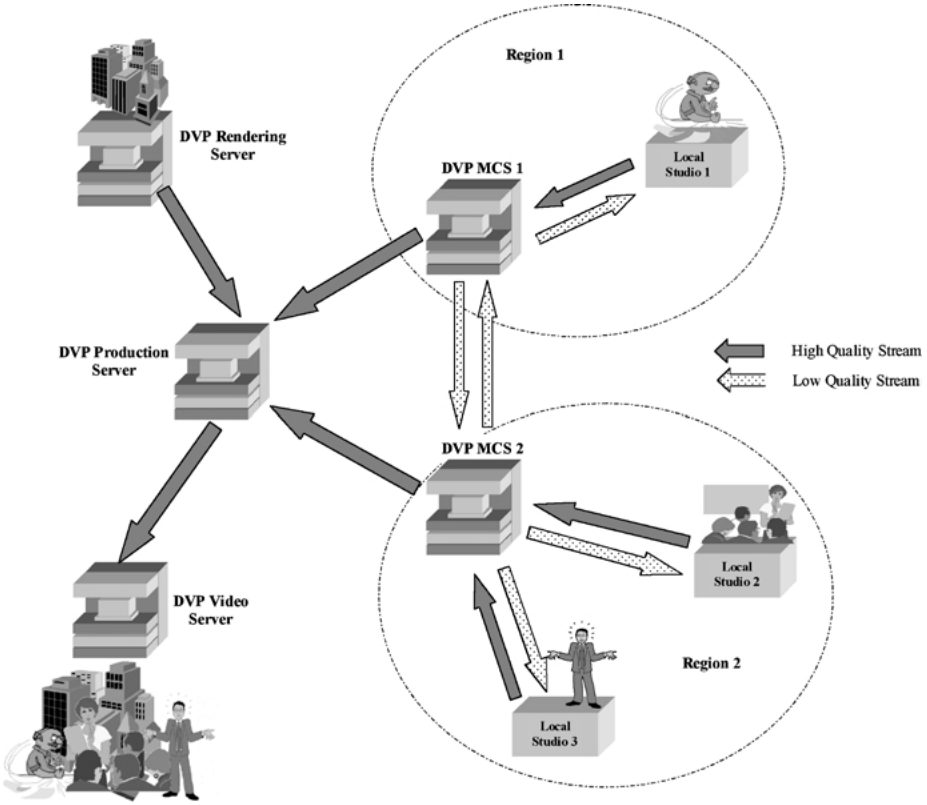

Figure 9. Multi-point production with MCS.

and send it to MCS2 where it is mixed with low-resolution version streams from studio 2 and 3. The mixed low-resolution version is then forwarded to each studio by the MCSs. This media mixing process is similar to audio and video mixing performed by MCUs in videoconferencing.

MCSs are also coordinators for joint production. In figure 9, a MCS can grant production control to one of the local studios to make it the host site for the production. For example, when the control is granted to local studio2, it will be able to feed the production server with camera control signals through MCS2. The camera control signal will then be used to control all camera movements at the three local studios and also within the virtual set created by the rendering server. Once studio2 is granted the right to drive all cameras, it will be the host of the production process and production resources provided by other sites; rendering server, production server, video server and local studios will all seem to be local resources to studio2. Operators inside each local studio may communicate through intercom or videoconferencing tools supported by communication channels established between MCSs in order to make joint production decisions, while studio2 can carry out the final production. Alternatively, the control right of the production may also be permanently assigned to a host studio and passed among local studios dynamically. Each studio may

place a request for the control right, and the host studio decides which studio should be granted the control right temporarily. When the studio finishes its task, the right is returned to the host studio. The transferring and policing of control right is also coordinated by MCSs.

We discussed in this section the architecture of DVP systems and proposed a layered functional model capable of capturing the DVP system characteristics. Our functional model depends on the DVP task requirements and in particular, the system and network performance requirements. In the next section, we discuss the network requirements in more detail while elaborating on the transport network architecture.

4. Transport network and system requirements for DVP applications

The DVP system can be divided into three components: system hardware, system software and network transport. In order to support the real-time and high bandwidth characteristics of DVP traffic, there are certain requirements for each of system component. In Sub-Section 4.1, we discuss the hardware and software requirements. Communication support issues are discussed in Section 4.2. These discussions serve as a baseline for DVP system implementation requirements. Other requirements may be needed in the future in order to obtain a complete set of network and system requirements.

4.1. DVP system requirements

4.1.1. Processing unit. Due to the high volume and real-time nature of media data, processors in DVP systems must be able to handle large amounts of data and thus require relatively high processing capability. Currently two types of processors can be found in a multimedia system [9]: dedicated multimedia processors and general-purpose processors with multimedia support. Dedicated multimedia processors are typically customized to perform specific multimedia operations, and are designed to provide large capacity and high efficiency. Complex and computationally intensive multimedia functions such as video/audio compression, 3D processing and virtual reality requires support from these dedicated processors. Advanced general-purpose processors provide support for multimedia by extending their instruction set to include multimedia instructions. These extended instructions can be executed in parallel, and provide support for generic operations in multimedia processing. An example is Intel's MMX Pentium processor, which achieves 65% to 370% performance improvements with extended multimedia support [22]. The emergence of these multimedia-extended processors allows many multimedia functions to be implemented in software, and significantly reduces cost and increases flexibility.

In order to support the high capacity demand of DVP applications, it is possible that customized dedicated processors will have to be implemented in DVP servers. This will enable servers to sustain large aggregated processing loads. Furthermore, these processors can be customized to have a programmable architecture, which will enable DVP servers to be more flexible in order to accommodate future application requirements. For end-systems inside local studios, a mixture of general-purpose processors and dedicated processors can be used. The general-purpose processors will be handling less demanding tasks, with dedicated

processors to back them up and support a few computationally intensive tasks. In order to lower costs and achieve higher efficiencies, dedicated processors used in local studios may not have to be programmable; flexibility can be achieved with software supported by general-purpose processors. Since computational requirements for local studios may not be as high as for DVP servers, the software and general-purpose processor combination should be adequate for most tasks with supplement of a few dedicated processors.

4.1.2. System BUS. Inside a system, the bus links all components together, and is responsible for transferring data between these components. A block of data usually has to traverse the bus more than twice in order to be processed by different devices. Therefore, the bus is one of the traffic concentration points in a system. In addition, bus speed is usually lower compared to processor speed, this is because the physical length of a bus has to be long enough in order to connect all devices together, however the top speed of bus operations is limited by the maximum signal propagation delay on the wires. Also, as more devices are attached to the bus, the total capacitance of the bus increases and this in turn slows down signal propagation further. It can take more than 5 ns for a pulse to traverse a 300 mm bus with a dozen boards attached to it [4], and this would limit the bus speed to within 100 MHz. Since the speed of modern processors increases, while bus speed is not equally catching up, bus speed is becoming a bottleneck for system performance.

For DVP applications that deal with continuous streams, large amounts of data have to be exchanged between processor, memory and I/O devices. The amount of traffic involved is likely to overwhelm the system bus. A possible solution to avoid congestion on the system bus is to introduce local buses between high traffic devices, to serve as "expressways" between high traffic devices and take a portion of the traffic off the main bus. However, a local bus can only transfer data between devices attached to it. In case additional data processing has to be performed by devices that are not on local buses, media data will still have to go thorough the main bus, thus reducing flexibility in the application design.

Another possible solution is to replace the system bus with a switch architecture, also known as crossbar or cross-point. Each system device is connected to other devices through the switch upon request. Unlike in the conventional bus approach where a large number of devices share a single communication path and only one of them can transmit data at a time, the switch approach allows data transfers to occur between multiple pairs of devices at the same time. This improves the overall system throughput. Furthermore, since a device will be connected to only one other device (a switch port), the length of the wire needed is shorter, and the total capacitive load on the wires is much smaller than a typical bus with dozen of devices attached. These factors result in faster signal propagation and further increases the communication speed of attached devices.

However, the switch architecture has its disadvantages. The complexity of the switch is a square function of the number of devices attached, and thus the switch can not be used to connect large numbers of devices. In such a situation, conventional bus architecture may be a more economical choice despite its lower performance. More details about the switch architecture mechanisms can be found in [10] and [23]. It is also possible that the switch architecture would be used to support high-speed devices in DVP servers, while slower devices may be connected by conventional bus architecture to reduce system cost.

4.1.3. Storage system. Storage devices in the DVP system must have large capacity and high throughput in order to provide access to a large amount of media data in a short time. Moreover, data blocks must be read or written in an uninterrupted fashion in order to support the continuous nature of DVP media data. Special data placement, disk scheduling and admission control techniques should be employed to allow continuous data access, while disk arraying techniques should be applied in order to support high capacity and throughput.

Magnetic disks are the most suitable medium for storing media data due to their short access time, high transfer rate and random access capability. However, a single disk is not sufficient to store large multimedia files, especially in a server environment, where many files have to be stored and many users have to be served simultaneously. The solution is to group a number of disk drives together and share the load. These drives operate in parallel and thus the aggregated capacity and throughput of the disk group is proportional to the number of disks in the group. This technique is used in the redundant array of inexpensive disks (RAID) technology, where inexpensive drives are combined to increase both storage throughput and capacity. The RAID technology also uses extra disks to store redundant data information, this overcomes the unreliability associated with inexpensive drives. Currently, there are eight level of RAID functionality defined, each level has its unique characteristics to serve different aspects of system requirements. A summery of the RAID technologies can be found in [24].

Data placement techniques are used in multimedia storage management in order to support continuous data accesses. As mentioned earlier, the real-time nature of media files requires data to be accessed in an uninterrupted fashion. If data segments of a file are placed on the disk in a scattered manner, as in the case of conventional file systems such as disk operating system (DOS), then during file access, extra disk seeks may have to be performed to locate the next data segment. These intra-file-seek operations introduce jitter in the media file access and thus increase the initial access delay and consequently buffer requirements of the storage system. Therefore, in order to support real-time accessing, the structure of data layout on the disk must be optimized. One possible approach is contiguous placement, which places successive segments of a media stream consecutively on the disk. To retrieve a particular stream, only one seek is required to locate the beginning of the file. However, contiguous placement is likely to cause fragmentation. Constrained placement [25, 40] is a variation of the contiguous placement method, which avoids fragmentation by leaving constrained gaps between consecutive data blocks in a media stream. These gaps are used to store other media streams, whose continuity can be ensured by adjusting their own block size.

In addition, since data transfer rate of a disk is usually higher than the playback rate of a single stream, a number of streams can be served simultaneously in a multiplexed fashion. In order to ensure efficient resource utilization, as many streams as possible should be served concurrently. However, admission control of services must be in place to prevent overload, this is especially true for DVP servers where available resources are consumed rapidly with increasing user requests. Moreover, multiplexed access and admission control alone may not be sufficient to guarantee optimal resource utilization, therefore parallel file access must also be scheduled in an orderly fashion in order to maximize the overall

throughput while minimizing low seek time and latency. A number of disk scheduling algorithms exist, however traditional algorithms which do not consider the real-time nature of media data are not suitable for managing storage access in DVP systems. A multimedia disk scheduling algorithm not only has to optimize disk throughput, minimize seek latency, maintaining fairness among tasks and avoiding starvation, but also has to consider the start time and deadline of real-time tasks. Details of real-time disk scheduling algorithms such as scan-earliest-deadline-first and group-sweeping-scheduling can be found in [37] and [29].

4.1.4. System software. In order to achieve good utilization and obtain optimal performance in the system, hardware resources must be properly managed by an operating system (OS). System resources to be managed include processing power, memory space, I/O bandwidth and storage space. An OS hides the physical characteristics of these devices and provides simpler interfaces to user applications.

Traditional operating systems such as Unix are largely concerned with throughput, utilization and fairness aspects of system management; resources are allocated mostly according to fairness among applications. This makes them inappropriate for managing real-time multimedia tasks, which have distinct priorities, based on each task's urgency. A multimedia OS [21, 37] should be able to manage a mixture of real-time and non-real-time tasks. For real-time tasks, QoS guarantees are provided and are supported by mechanisms that perform QoS specification, admission control, resource management, scheduling and policing. QoS specifications are needed in order to indicate to the OS how many resources a new task would require. Based on these QoS parameters, schedulability tests can check whether there are enough resources to accommodate the new task. If the request can be accepted, the resource manager allocates the required capacity for the task and the scheduling mechanism arranges for the execution of the task. A policing mechanism is required to prevent tasks from over-using system resources. A multimedia OS should be able to deal with both real-time and non-real-time tasks since they are both important to DVP applications. While priorities are given to real-time tasks and their QoS requirements are being satisfied, non-real-time tasks would have lower priority but they can not be neglected. Tasks should be scheduled in a proper way such that it prevents starvation of non-critical tasks while avoiding priority inversion of real-time tasks.

System resources managed by a multimedia OS can be partitioned into four categories: process, memory, I/O and file management.

Process management is responsible for scheduling the execution of each task (or process). Continuous-media processing has a periodic nature, this makes the specification and scheduling of processing requirements easier. These periodic tasks perform repeated and similar operations each with their own deadlines, the goal of the process manager is to schedule as many tasks as possible while their deadlines are all met. The problem is how to schedule the tasks effectively while maintaining a low scheduling overhead caused by operations such as schedulability test. The two most important scheduling algorithms are Earliest-deadline-first (EDF) and rate-monotonic. Both algorithms are preemptive and assign priority to processes according to their deadline and request rate respectively. EDF is a dynamic algorithm and it frequently recalculates process priorities, thus its scheduling

overhead is higher. The advantage of EDF is that its processor utilization can reach 100% without failing task deadlines. The rate-monotonic algorithm is static since it fixes priority of a task at the beginning, and this causes more context switching. However, since rate-monotonic algorithms have less scheduling overhead, they are more suitable for a purely periodic task environment. More details on process scheduling can be found in [37] and [35].

Memory management is responsible for assigning (and releasing) memory to processes. Typically, virtual memory is also used to store less frequently used data, which may be swapped between physical memory and storage by paging. However, media data in multimedia systems are usually very large, a page fault would incur long swapping delay and possibly lead to missing task deadlines. Therefore, memory swapping for real-time tasks should be avoided. One possible approach is to lock the data in the memory and prevent it from being swapped to the disk, however this technique should be used with care since media data are large in size and memory space may be limited. Other approaches take advantage of the periodic nature of media tasks and pre-fetches data into the memory.

I/O management is responsible for ensuring efficient data transfers between I/O subsystems and main memory. Performance of I/O is limited by many factors such as bus bandwidth and OS overhead. Traditionally, I/O devices are controlled by device drivers that exist in the kernel space of the memory. When user applications place high-level requests to an I/O device, such as reading a line, the request is translated into a system call into the OS. The OS kernel breaks up the system call and places several low-level requests to the device driver, and the I/O device is then reached through the device driver. This hierarchy of operation requests result in overhead. Furthermore, when requested data are delivered back from the I/O device, several copying operations have to be performed in order to transfer the data thorough every level of the hierarchy. This results in inefficient operation and degrades real-time performance of the system. Each data copying between kernel space and user space would require a context switch between user and kernel processes, and these context switches incur even more operations to swap processor register to memory and vice versa.

It is clear that direct data transfers between I/O devices must be supported in a multimedia system. Data exchange could occur at the device driver level in the kernel space without going through the application space [19]. Of course, this is assuming that no application level modification has to be performed on acquired data. It is also possible to make devices "autonomous" [40], these devices would be smart enough to manage flow control and also simple operations such as checksum, then data can be exchanged between devices without going through the main memory at all. Moreover, Raghavan and Tripathi suggested granting applications direct access to I/O devices without going thorough the kernel. This would save overheads associated with system calls and context/domain switches. However, issues related to access control and run-time management must be carefully considered in such an implementation.

File management is responsible for providing users with file access and making efficient use of storage resources. The main difference of a multimedia OS file manager from conventional file manager is related to the continuous, real-time nature of media data. These issues have already been discussed in the storage management section.

4.2. Network transport requirements

This sub-section presents a preliminary quantification of network performance requirements for typical multimedia applications. Transport network performance is usually measured by three sets of parameters, namely bandwidth, delay and error/loss requirements. It is clear from our discussion of DVP tasks in Section 2 that these tasks will place stringent demands on the underlying transport networks due to their high quality requirements and interactive nature. However, it is noteworthy that, similar to most multimedia applications, the actual QoS parameters depend largely on specific implementations and media contents. After consulting a number of references, we have collected a set of representative QoS parameters for typical multimedia and video distribution services. These values reflect general requirements likely to be encountered in distributed video production environments.

4.2.1. Bandwidth requirements. Bandwidth requirements in multimedia applications are usually very demanding compared to most data applications. Multimedia, and in particular video information, requires relatively large amounts of bandwidth in order to achieve proper QoS levels. The requirements are even more exigent in a video production environment, where uncompressed production quality video requires transport bandwidth of 166–270 Mbps. The upcoming High definition Television standard further pushes these values up to 1.5 Gbps. However, with the advancements in compression algorithms such as MPEG2, standard definition TV production may be accommodated with lower bandwidth requirements. In Table 1, we attempt to summarize our findings in terms of bandwidth requirements for typical multimedia applications likely to be encountered in DVP applications.

4.2.2. Delay, delay variation and inter-media skew requirements. The second set of performance measures is concerned with delay performance. This includes constant delay, variable delay and inter-stream (skew) performance parameters. It is a well-known fact that real-time applications impose relatively more stringent requirements on network delay performance, when compared with data applications. In particular, for interactive multimedia applications, very low transport delays are sought in order to achieve proper end-to-end quality of service.

Table 1. Bandwidth requirements for multimedia [24, 32, 39].

Application	Characteristics	Bit rate
Videoconference	H.261	64 Kbps to 2 Mbps
VCR quality	MPEG-1	1.2–1.5 Mbps
TV quality	MPEG-2 MP@ML	2–10 Mbps
Postproduction	MPEG-2 4:2:2@ML	10–50 Mbps
	Uncompressed	166–270 Mbps
HDTV	Compressed (distribution)	19.2–60 Mbps
	Compressed (production)	270–300 Mbps
	Uncompressed	1.5 Gbps

DVP applications scan a large spectrum of delay requirements. As an example, typical videoconferencing applications may withstand 200–500 ms end-to-end, while an interactive distributed joint production application may not tolerate more than 20–100 ms end-to-end [32, 38]. This may in turn impose a bound on the maximum physical distance between servers involved in such an application.

Delay performance is also measured based on delay variation or "jitter." Jitter is a measure of the difference in delay experienced by different packets in the network due to variation in buffer occupancy in intermediate switching nodes. Another form of jitter is inter-stream jitter or "skew", which measures the difference in delay as seen by separate streams pertaining to the same application (such as audio and video). In order to ensure proper intra-stream synchronization, low delay variation is often required.

Jitter may be compensated using buffering techniques at the receiving end. However larger the jitter, longer the buffering delay experienced, which in turn increases the overall end-to-end delay. Therefore, low delay variation must be guaranteed for interactive applications, which require small end-to-end delay by its nature.

Accurate delay variation requirements are not well defined and depend largely on specific media types and system implementations (e.g., decoder design). A general guideline is that for television quality video, jitter should be kept below 10 ms [11]. Additionally, the MPEG2 standard recommends decoders with ± 4 ms of jitter tolerance. Many comprehensive experiments have been carried out in [36] to measure the human perception of inter-media skew, and provide a general guideline of ± 80 ms for inter-stream synchronization for playback application.

4.2.3. Transport error requirements. The third set of network performance measures is concerned with bit errors and packet losses introduced in the network. Error requirements for multimedia applications vary according to the application. However in general, occasional transport errors may be more tolerable when compared with data applications where error requirements are much tighter. This is due to the fact that multimedia information is presented to the user at a relatively rapid rate, therefore single errors may not be perceivable by the human eye. Nevertheless, we should note that this mostly applies to playback type of applications. DVP tasks such as remote keying may require tight error bounds for the camera tracking signals. Finally, compression algorithms add a multiplication factor to single bit errors, depending on the location of the error. This translates into a trade-off between bandwidth and error requirements.

Table 2 presents a summary of error requirements for typical multimedia applications in terms of expected duration of operation with no errors. These values are in turn translated to bit error rates at the physical transmission level.

4.3. The need for network adaptation techniques

Transport networks used in DVP environments may also be used for the transport of traffic types other than real-time video, such as day-to-day file and electronic mail exchange. These traffic types are rather of a non-real-time nature and may generally have different requirements. Moreover, it is obvious from our previous discussion that there exists a variety of

Table 2. Error rate requirements for multimedia applications [32].

Service	Error requirements	BER
Videoconferencing	30 min error free	BER < 1e-6
MPEG-1 core	20 min error free	BER < 4e-10
MPEG-2 MP@ML "TV quality"	30 min error free	BER < 6e-11
MPEG-2 MP@ML Postproduction	1 hr error free	BER < 2e-11

DVP applications, each with different bandwidth and QoS requirements, depending on the tasks and components involved. In order to accommodate such a heterogeneous traffic mix, while having a common network transport framework, application class-specific adaptation functions need to be implemented. These adaptation functions would then perform the necessary mapping between the application and the network QoS and performance parameters. Network transport architectures and adaptation techniques are discussed next in detail.

Based on our previous discussion, it is clear that transport networks used for the real-time traffic need to have special capabilities in order to deliver the required QoS for DVP applications. In this sub-section, we present possible protocol stack realizations for DVP based on the state-of-the-art and examine the different alternatives for network adaptation techniques.

4.3.1. Possible protocol stack realizations for DVP. In order to understand the different alternatives for a transport network to support DVP applications, we start by classifying the different types of video information sources, then work our way down in the protocol stack, identifying at each level the different protocol combinations and functionalities. Figure 10 shows examples of protocol stack realizations for DVP applications.

At the media encoder level, the encoder type may range from a constant bit rate (CBR) encoder to a variable bit rate (VBR) encoder. The encoder may also be of a layered-type,

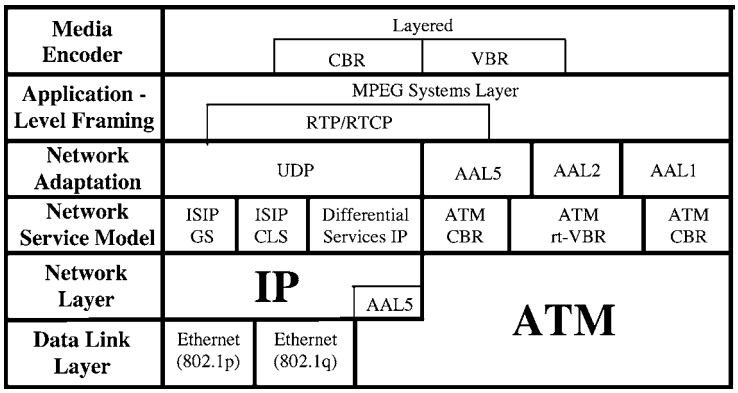

Figure 10. Protocol stack realizations for DVP networks.

generating several media layers, each of which falling into one of the previous categories. Note that for a layered-type encoder, different layers may be passed to the level below as individual streams, or alternatively, layers may be multiplexed to form a combined stream.

The next protocol level in the stack is the application-level framing layer. This is the first layer to prepare the video information for transmission over the network. Typical functions at this level include adding time stamps, sequence numbers, multiplexing identifiers and any other information needed for proper recovery at the receiver. Examples of application-level framing layers are the MPEG Systems layer [15] and the real-time transport protocol (RTP) [33]. While essentially RTP and the MPEG Systems have many features in common, there is still a possibility to use RTP to transport MPEG systems packets in order to provide full-compatibility with systems using RTP as their default application level-framing protocol. It should be noted that the encoder level and the application-level framing layer correspond to the coding/synchronization layer in the DVP system architecture (figure 7).

Application packets are then passed to the network adaptation layer. We use the term "*network adaptation layer*" in the figure to include ATM Adaptation Layers (AAL) and IP transport protocols. This layer corresponds to the adaptation layer of figure 9. Network adaptation protocols are intended to provide a level of adaptation of the QoS delivered by the underlying network to the QoS requested by the application. The adaptation functions are explained in detail in Section 5.2. In the case where RTP is used for application framing, a network adaptation protocol is needed for encapsulation of RTP packets. This would typically be either user datagram protocol (UDP) for IP or AAL type 5 for ATM networks. For the MPEG Systems layer case without RTP, any underlying network adaptation protocol may be used, although it would typically be an ATM network in the transport, hence an ATM adaptation layer for the adaptation.

There are several alternatives for the transport network service class. We show the network service in the figure for the purpose of relating the different adaptation layers and network layer protocols. For IP networks, Integrated Services IP models such as the guaranteed service (GS) [34] or the controlled-load service (CLS) [41] classes may be used to provide the necessary QoS requirements, in which case the reSerVation protocol (RSVP) [5] would serve as a signaling protocol. Alternatively, Differential-Services IP classes may be used to provide higher priority for the video traffic over other less-demanding traffic types. For ISIP and Differential-services IP models, the combination of RTP/RTCP over UDP is expected to form the basis for real-time network adaptation. On the other hand, for ATM networks, it is more likely that either a constant bit rate or a real-time variable bit rate connection would be appropriate, as defined in the user-network interface version 4.0 [1].

The bottom two layers of figure 10 explore the network protocol layer and the data link technology alternatives. When ATM is used as a network layer, it provides its own data link layer mechanism. For IP networks, either the Ethernet 802.1p or 802.1q standard may provide the data link layer for IP. The 802.1p standard provides priority scheduling features for the switched Ethernet standards, while the 802.1q standard provides QoS support through virtual LAN (VLAN) tagging. Alternatively, ATM may provide a data link layer for IP, in which case AAL5 would be used to map IP packets onto ATM cells [20]. Various other alternatives are proposed for the transport of IP over ATM, however it is out of the scope of this paper to discuss these alternatives [14].

4.3.2. Network adaptation functions. We now focus our attention on the network adaptation functions at various levels of the protocol stack. We use the term "*network adaptation*" here to include the second and third level from the top in figure 10. In other words, this includes application-level framing protocols such as RTP/RTCP and MPEG Systems, in addition to IP transport and ATM adaptation layer protocols. In general, network adaptation functions (at both application and network levels) are needed for two complementary purposes: 1) mapping application traffic and QoS parameters to network-level performance parameters, and 2) adapting the QoS delivered by the network to the QoS level requested by the application. The adaptation functions are usually classified under four categories:

4.3.2.1. Timing recovery and synchronization. Timing recovery and synchronization functions attempt to restore the temporal relationship between the packets constituting a real-time information stream. This is achieved by means of time stamps or adaptive clock recovery techniques at the receiver. Additionally, a playout buffer is used in order to absorb the jitter introduced in the network. Note that the delay introduced in the playout buffer affects the end-to-end delay of the connection and should not cause the overall delay to exceed the tolerable delay bound. Another level of synchronization is inter-stream synchronization. This function usually deals with synchronization between several streams pertaining to a single application, such as in the case of audio and video or in the case of multi-layered video streams.

4.3.2.2. Error/loss control. Packet errors and losses are mainly caused by bit errors and buffer overflows in the network. Error control techniques are usually based on automatic repeat reQuest (ARQ), forward error correction (FEC) techniques, or a combination of both. For real-time applications with tight delay constraints, FEC techniques are usually favored over their ARQ counterparts. Moreover, FEC schemes are usually coupled with byte or packet interleaving in order to add robustness to the scheme, provided that the interleaving/de-interleaving delay is kept within the tolerable amounts, as specified by the application.

4.3.2.3. Stream multiplexing. Multiplexing is usually used in order to avoid inter-stream synchronization problems by aggregating the different streams belonging to one or more application and transmitting them over the network as one unit. This results in a reduction in the inter-stream jitter to a relatively small and controllable amount. Multiplexing is also used to achieve better network utilization by means of statistical multiplexing.

4.3.2.4. Shaping and flow control. Shaping is used to reduce the burstiness of the traffic, and hence decrease the probability of packet loss due to buffer overflows inside the network. However, shaping usually incurs delays, depending on the shaping interval technique and parameters, which in return impose constraints on the amount of shaping that may be applied.

Flow control provides means to adapt the application's rate to the network congestion level as it occurs. This feature relies on the applications being adaptive in order to function properly. Shaping and flow control functions are not widely implemented in practical real-time adaptation protocols.

4.3.2.5. Network adaptation protocols. This section provides a survey of the state-of-the-art in ATM and IP network adaptation and transport protocols. This includes a survey on AAL1, AAL2, AAL5 and RTP/RTCP. For details on MPEG2 systems, the reader is referred to [15] and [16].

AAL type 1 was originally developed for CBR real-time connections. Timing recovery and synchronization may be achieved by one of two techniques: synchronous residual time stamps and adaptive clock recovery. Cell loss detection is achieved using a 3-bit sequence number field. For cell error and loss correction, two FEC schemes with byte interleaving have been defined [17]: a long interleaving scheme and a short interleaving scheme for loss- and delay-sensitive applications, respectively. Both schemes use Reed-Solomon block codes with erasure correction capabilities. The performance of AAL1 for the transport of CBR MPEG2 has been reported in [26] and [28].

AAL1 might also be successfully used for VBR traffic with a piece-wise nature [27]. This is achieved by sending special synchronization cells, identified by setting the user-to-user bit in the cell header, to convey rate change indication to the AAL1 receiver. The performance of this scheme is limited by the overhead introduced by these special cells in the case of frequent rate changes.

AAL2 has recently been developed, primarily designed for VBR voice with a strict end-to-end delay constraint [8], with special emphasis on voice for mobile applications. Nevertheless, it may well be suited for VBR video and multimedia applications. AAL2 is divided into a common part sublayer and a service specific sublayer. A CPS packet header contains an 8-bit channel identifier for multiplexing, a 6-bit length indicator and a 5-bit header error correction field for header error protection. AAL2 multiplexing capability is considered a strong feature of the protocol, and is used for dynamic channel assignment for mobile applications. In order to use AAL2 for the transport of MPEG2 video, a video-specific SSCS sublayer should be defined to include necessary timing and error control functions. At the time of writing this paper, this is still an open research issue.

AAL5 development was originally driven by the computer industry, in search for an efficient and reliable replacement for AAL3/4 for non-real-time VBR traffic. Due to its wide availability in ATM products, AAL5 is now used in the transport of CBR MPEG2 over ATM [2] and [18]. Each AAL5 packet is segmented into one or more ATM cells, the last of which is marked by setting its corresponding ATM user-to-user bit in the cell header. The trailer cell contains a 32-bit CRC field for error detection, however, no error correction mechanism is defined for AAL5; one bit error may cause the entire AAL5 packet to be discarded. AAL5 does not provide timing recovery and relies on the application to react to any jitter introduced in the network. Another disadvantage of AAL5 is that cell multiplexing is not possible. For the transport of CBR video, several packing schemes have been proposed, the most popular of which is to pack every two MPEG2 Transport Stream packets into one AAL5 PDU [2]. Although a null SSCS is used in this proposal, a video-specific SSCS may be defined to provide necessary timing and error control. However, since cell multiplexing is not possible, FEC schemes with interleaving might not be feasible. The performance of AAL5 for the transport of CBR MPEG2 video is reported in [12] and [39].

The real-time transport protocol (RTP) was primarily designed for multiparty multimedia conferencing on the Internet [33]. RTP is an application-level framing protocol that provides a framework for adding real-time support for multimedia applications over the existing best-effort service model.

RTP and RTCP provide applications with time stamps, sequence numbers, payload type identification, session participants information and QoS monitoring. RTP/RTCP do not provide length indicators and therefore rely on the underlying transport or AAL protocol (often UDP or AAL5) to provide the necessary packet encapsulation. QoS monitoring is achieved by having receivers transmit reports with their observed connection QoS. Error control was not included in early implementations of RTP. This is currently being investigated in [30].

The implementation of RTP is often integrated into the application. RTP requires additional payload and application-specific documentation (e.g., [31] and [13]) for the transport of MPEG video.

4.3.3. Comparison of network adaptation techniques. Table 3 summarizes the features of the application and network-level adaptation techniques. No clear distinction can be made between the different protocols. Furthermore, there exist redundant functions implemented in more than one of them. This makes a choice even harder to achieve. However, it should be noted that other factors might affect the choice for network adaptations such as compatibility with existing equipment or availability of products. Moreover, in some situations, redundancy is favored for adding robustness to the overall system.

5. DVP service provision: Service scheduling with QoS

5.1. Service access model

Access to DVP servers will be mainly done according to a client/server model where the end user at the local studio executes DVP tasks such as retrieving multimedia objects from a multimedia server. In such a distributed environment, a number of end users from several local studios may need to access a number of DVP servers through communication networks.

Table 3. Summary of network adaptation protocol capabilities.

	MPEG systems	AAL1	AAL2	AAL5	RTP	UDP
Time stamps	Yes	Yes	No	No	Yes	No
CRC error detection	No	SAR header	CPS header	Entire packet	No	Optional
Sequence numbers	4-bit	3-bit	1-bit	No	16-bits	No
Forward error correction (FEC)	Optional	Yes	No	No	Possible	No
Multiplexing	Yes	No	Yes	No	Yes	Yes
Encapsulation	Yes	Yes	Yes	Yes	No	Yes
Overhead	$\geq$2.12%	2%	$\leq$4%	$\geq$8 bytes	$\geq$12 bytes	8 bytes

This scenario reveals the requirement for an access model, which allows servicing users according to their QoS requirements while optimizing DVP server resource utilization. For that purpose, we have defined a DVP service access platform, which involves two basic components. The first one, at the user host, acts as the user access interface. It controls user access to the requested DVP server, and checks if the required QoS parameters are satisfied. The second component, at the server end, interfaces with the DVP server. It manages access requests to the server functionality and keeps track of the server load information. Hereafter, the first component will be referred to as the *user agent* and the second one as the *DVP server agent*. Furthermore, depending on the networked system topology, the access model can involve a number of additional management components acting as intermediates between user agents and DVP server agents. These intermediate managers allow the directing of user requests to the most appropriate servers in a transparent and dynamic manner. QoS requirements and DVP servers' availability are taken into account in the process of directing users' requests.

Two types of managers are defined, the *virtual studio access manager* (VSAM), and the *domain manager* (DM). The latter is responsible for a domain, grouping a set of DVP servers according to different criteria such as geographical constraints, DVP server functionality, or others. VSAM is responsible for a set of domains grouped according to organizational policies. It acts as a smart directory service gathering both functional and management information on the DVP domains and makes this information available to domain managers and local studios. Functional information concerns mainly the DVP tasks and multimedia content supported within the DVP domains. Management information is primarily used for managing user access to DVP servers by directing user requests to the appropriate domains based on: load information; QoS requirements; and cost constraints. The domain manager monitors the DVP servers within its domain through the server agents associated with these DVP servers. It collects state as well as load information, and performs the appropriate allocation of user requests. It also detects multimedia server failures whenever they occur. As a result, VSAM, and domain managers support the appropriate exchange of messages between user agents and server agents. In addition, they maintain state information that allows the determination of the most suitable DVP server in response to an end user request.

The introduced agents and managers can be configured (their number, location, dependencies) to suit a given physical topology or a given DVP service provision policy. They intervene between local studios, i.e., user hosts, and existing (or future) DVP servers, i.e., server machines, to increase service availability as well as to optimize resource utilization.

Figure 11, shows an example configuration of Virtual Studio Access architecture, illustrating the distributed feature of the access model. The example configuration consists of a two-level hierarchy where the virtual studio access manager is the root, domain managers constitute the first level, and DVP Server Agents the leaves. However, the virtual studio architecture may contain as many domain levels as necessary according to the size and feature of the overall system, e.g., the number of DVP servers, their distribution, the network topology, etc. The performance of the overall system is also an important design choice to determine the number of domain levels in the access platform. Indeed, involving a large number of intermediate domain managers in the decision making process will affect the

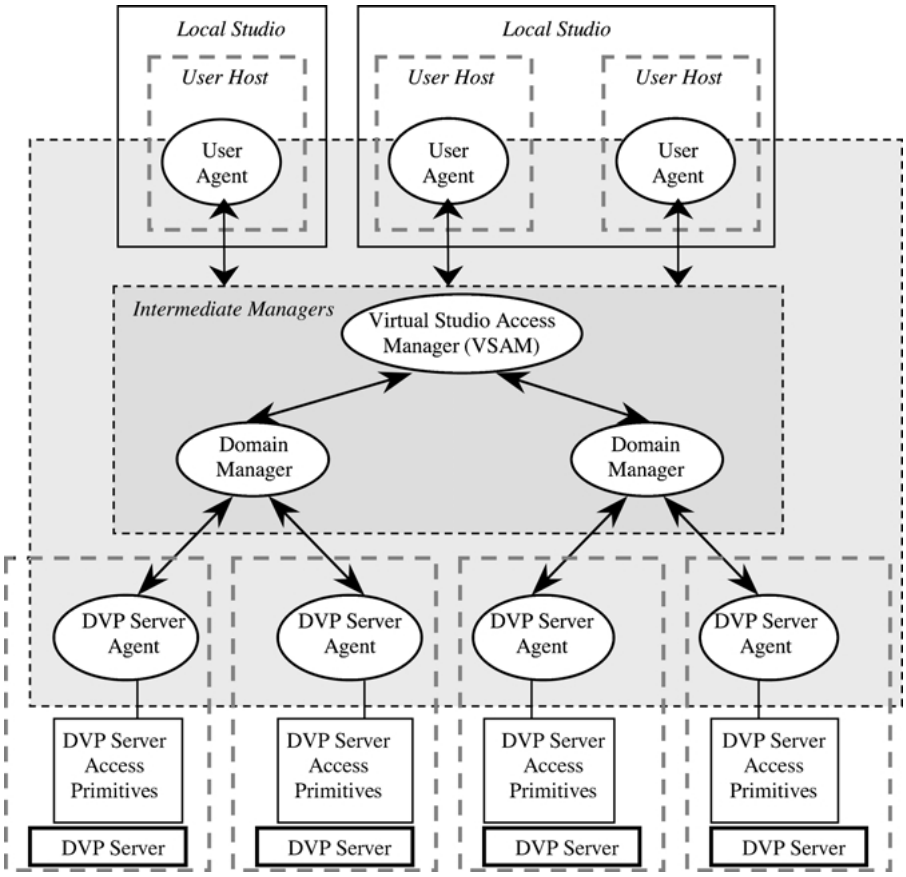

Figure 11. Virtual studio access architecture.

performance in terms of response time. The following subsections present in detail, the functions provided by the virtual studio architectural components.

5.1.1. The user agent. The user agent consists of three components: *user interface*, *QoS manager*, and a *client controller* (as it is shown in figure 12). The user interface consists of two main parts. The first one provides the various DVP tasks such as a way to search and select a video object from a video server. The second one allows the specification of the desired QoS such as the quality for the presentation. It also allows to set the access cost constraints and to renegotiate the QoS parameters during the DVP service provision, e.g., during a video object display.

Whenever, a user selects a DVP task with specific QoS requirements, the user interface component invokes the QoS manager, which starts by checking whether the client machine characteristics, such as the screen size and color, support the requested QoS. If not, the QoS manager sends to the user a reject message, possibly with an alternative offer, through

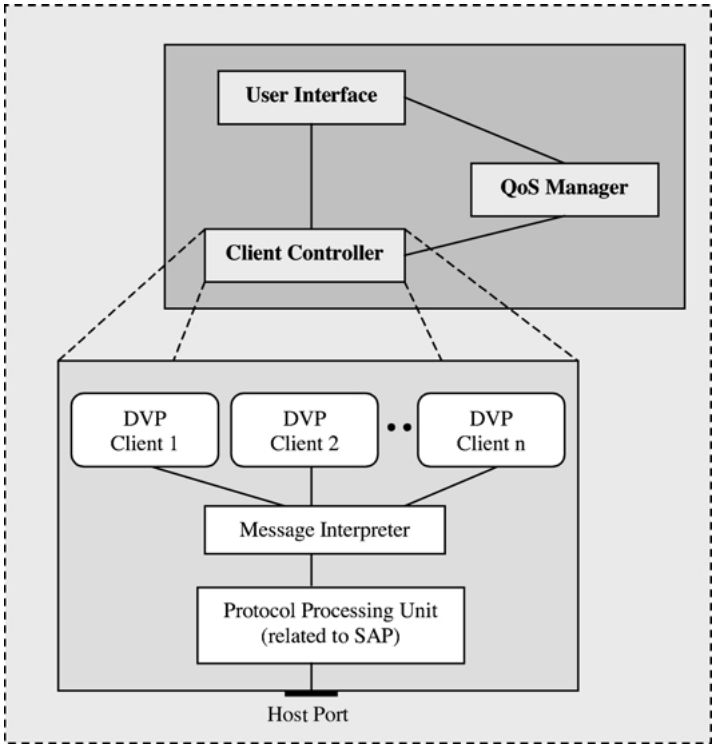

Figure 12. User agent architecture.

the user interface. In this case, the user might abandon the request, accept the alternative offer if any, or initiate a re-negotiation. If the specified QoS parameters conform to the capabilities of the user's host, the QoS manager invokes the client controller together with its user requirements, i.e., QoS, and user workstation capabilities like the available decoder or operating system. The client controller builds a request message before sending it to the domain manager. It then waits, at a specific host port, for a response while initiating a timer. If the timer expires, or the user request is rejected by the DVP system, a reject message is displayed at the user interface. Otherwise, the client controller invokes the appropriate DVP client to display the multimedia object. This is done via a message interpreter responsible for mapping virtual studio access protocol data units to a specific primitive that starts the corresponding DVP client. The latter can be any of the available research and commercial DVP tools such as audio/video players or other media processing software.

During the DVP service provision, the user may experience a QoS degradation manifested for example as an unacceptable presentation quality. In such an occurrence, the user agent notifies the appropriate domain manager asking for an alternative DVP server capable of delivering the service with the contracted QoS. In the current implementation, the state of the DVP task, e.g., a video object display is registered at the time the task is interrupted, e.g., the position of the last video scene displayed. This allows the alternative DVP server

to continue the delivery of a service started by another server. The new DVP server restarts the DVP task based on the state parameters registered earlier.

The object-based design of the user agent allows for higher flexibility and easier maintenance. As an example to support a new DVP application, one can integrate the appropriate user interface without changing the existing objects of the user agent.

5.1.2. The DVP server agent. A DVP server agent is delegated to represent a DVP server within the virtual studio environment. The DVP server agent continuously waits at a specific service port for requests coming from its domain manager. When a request is received, the server agent checks first the capacity of the DVP server to deliver the requested service with the desired QoS. This leads to either starting the delivery by the server, or sending a reject message to the user. The server agent is composed of two main parts, or functions. The first part deals with the communication with other components of the virtual studio architecture like the domain manager, and the user agent. The second part is related to the DVP server represented by the corresponding server agent in our access model. Its implementation is specific to the encapsulated DVP server. The advantage of this design is the ability to change from one DVP server to another by changing only the corresponding part in the server agent. Similarly, introducing a new DVP server in the virtual studio environment necessitates the provision of the corresponding interoperability functions at the server agent.

In the context of the virtual studio service requests management, a DVP server agent exchanges messages with the domain manager concerning its operational state, and its availability. This allows the SAP to detect faulty DVP servers. For this purpose, the DVP server agent periodically notifies the domain manager of its availability to handle service requests. Notification frequencies need not to be equal for all DVP server agents in the system. The notification frequency depends on a number of factors, such as the reliability of the server. In addition, it takes into account how much importance we place on these factors. Such factors can be computed according to collected statistics on the past behavior of the server.

The load information of a DVP server referred to as the DVP server availability is computed as the number of service requests the server can handle with a given QoS at a given time. This highly depends on the internal implementation of the DVP server such as the processing architecture, storage capacity, access performance, etc. The DVP server agent computes the load of the DVP server it encapsulates based on the load model described in Section 5.2.

5.1.3. The virtual studio access manager. The role of the virtual studio access manager is to redirect user requests to the appropriate DVP domain. That is the domain containing the most appropriate DVP server to handle the request. For this purpose, VSAM maintains two state information tables, namely `VSAM_DVPtask_Table` and `VSAM_Load_Table`. `VSAM_DVPtask_Table` contains two attribute types: `DVPtask_Id` and `List_of_Domains`. It gives, for each DVP task identified by `DVPtask_Id`, the list of domains containing at least one DVP server that supports this task. Each domain in the `List_of_Domains` is identified by its `Domain_Id`. `VSAM_Load_Table` gives for each domain, identified by a `Domain_Id`, the load of this domain.

Upon receipt of a request from a domain manager or a local studio, VSAM determines the domains with at least one DVP server that can handle the request. It uses `VSAM_DVPtask_Table` for this purpose. The load information contained in `VSAM_Load_Table` is sorted from the lightly loaded domain to the highly loaded one. According to these tables, domain managers are successively requested to deliver the multimedia object. A reject message is send back if none of the domains can satisfy the request.

The load information for each domain maintained by VSAM is received from the corresponding domain manager. The latter computes the domain load as the weighted average sum of the load levels of the various DVP servers within the domain.

5.1.4. The domain manager. Domains are logical structures used as flexible means for clustering DVP servers in order to control resources utilization. Each domain contains a domain manager, which maintains the global view of the encapsulated DVP servers. Based on global and timely knowledge of servers state, a domain manager can offer end users better quality and fault tolerant access to the DVP services supported by these servers. The state information on a DVP server concerns mainly its operational state, e.g., out of service, and its availability, e.g., load, response-time, etc.

As described previously, domains can be defined according to geographical, organizational, service type, service performance, and other criteria. The various domains are organized into a hierarchy of domains to provide a system-wide access to DVP services. The domain structuring allows reducing the management complexity of the overall virtual studio environment. Indeed, a domain manager performs its management task autonomously, but may cooperate with other domain managers in the context of the global access management and service scheduling tasks. The number and size of the domains can be determined based on several factors such as investment versus revenue for the DVP service provider, and/or quality versus price of the offered services to local studios.

Similar to VSAM, but within a single domain, the domain manager redirects user requests to the appropriate DVP server. It maintains mainly two state information tables, namely `D_DVPtask_Table` and `D_Load_Table`. Emphasis is given on the load distribution capability. Therefore the state information is limited to DVP servers' load information. `D_DVPtask_Table` contains two attribute types: `DVPtask_Id` and `List_of_DVPserver_DVP-task`. It gives, for each DVP task identified by `DVPtask_Id`, the list of servers supporting the requested DVP task together with the QoS characteristics. Each item in the `List_of_DVPserver_DVPtask` is composed of the DVP`server_Id` supporting `DVPtask_Id` and a list of QoS charachteristics. `D_Load_Table` gives for each DVP server, identified by a `SVPserver_Id`, the load of this server.

Upon receipt of a request from a user agent or from VSAM, the domain manager determines, from `D_DVPtask_Table`, the DVP servers supporting the requested DVP task with the QoS characteristics required by the user and supported by the user workstation. The domain manager compiles the load information contained in `D_Load_List` to determine the most appropriate DVP server with respect to the user request. For that purpose, `D_Load_List` is sorted from the lightly loaded server to the highly loaded one. In practice, The domain manager submits the user request to the DVP Server Agent responsible for the least loaded server supporting the requested DVP task. This process is repeated until a

server succeeds to deliver the requested service, or all the servers fail. In the latter case, a notification is sent to VSAM.

The load information for each DVP server that is maintained by the domain manager is received from the corresponding DVP Server Agent. The latter computes the load of the DVP server it is responsible for, based on the QoS-sensitive load model described in Section 5.2.

5.2. Multidimensional load model

5.2.1. QoS-sensitive load model of DVP servers. One of the main criteria used to find the most appropriate DVP server to which to redirect users service requests, is the load of DVP servers. Usually, a DVP server's load is computed as the number of processes currently executed divided by the server's speed, or equivalently the number of service requests currently handled divided by the server's speed (a function of the CPU and disk access speeds).

In a multi-service environment supporting different QoS, the load information, as defined above, does not reflect the real load, and may lead to wrong placement decisions. Indeed, the "least loaded server", handling the smallest number of service requests, may not correspond to the server which has the smallest load volume. For example, a server performing a task that requires a large amount of resources, e.g., decoding and displaying a high resolution video, is effectively more loaded than a server performing tens of light tasks, e.g., text processing. Furthermore, there are no guarantees that the selected least loaded server can support the user QoS requirements.

To alleviate these limitations we propose a scheme, which provides an accurate information about the capacity of a server to support a given request by performing extensive measurements prior to service operation. This scheme allows to determine the server which is lightly loaded; and is able to support the user request with the desired QoS.

In the proposed scheme, QoS requirements are grouped into QoS classes. Each class represents a range of QoS parameters' values. A service request is associated with a QoS class and, thus, requires a certain amount of network and system resources from the DVP server.

At a given time, the state of a given DVP server can be defined as the set of service instances currently supported by the server. Based on this classification, we can determine, through experiments, the set of *non-blocking, semi-blocking* and *blocking* states for a given DVP server. A DVP server is in a non-blocking (respectively blocking) state if it can (not) support new service request(s) without affecting the service instances currently supported. When the DVP server is in a semi-blocking state, certain new service requests can be supported while others cannot. This QoS classification and server states determination are formalized as follows:

Let us define CL as the set of QoS classes a DVP server can support, and n the cardinality of CL.

Definition 1. St is defined as the set of service instances currently supported by server S at time t. It reflects the state of a given DVP server at a given time.

We note St=(V(cl^1), V(cl^2), .., V(cl^n)), where V(cl^i) as the number of currently provided service instances belonging to QoS class cl^i; 1 <= i <= n and n = Cardinal(CL).

Definition 2. A finite state machine (FSM) that represents a given DVP server S is a tuple (S, S^0, SR, TSR, T), where:

- S is the set of states in which the DVP server can still accept new requests (non-blocking states),
- S^0 is the initial state of the DVP server,
- SR represents the set of service requests submitted to the DVP server,
- TSR is a set of received service termination requests generated by local studios, and
- T: S {S^0} * SR $\rightarrow$ S is a transition function.

The transition function T operates as follows:

When the DVP server S receives a service request, for example sr, belonging to QoS class cl^k at time $t + \delta$, two options are possible: Either the DVP server is in a blocking state and rejects the sr request, or it is in a non-blocking state and processes this request. In the latter case the DVP server transits from state St to St'=(V(cl^1),.., V(cl^k)+1,.., V(cl^n)).

When the DVP server receives a service termination request at time $t + \delta$, say tsr, corresponding to a service request of QoS class cl^k, it transits from state St to $St'' =$ (V(cl^1),.., V(cl^k)-1,2 .., V(cl^n)).

5.2.2. Domain load. As stated previously, our Virtual studio access model introduces the domain concept as a means for grouping multimedia servers to obtain aggregated views of the DVP network and system resources, and hence a more easily manageable distribution of the overall load in the virtual studio. Typically, domains are used to redirect users' requests to the appropriate DVP servers based on the global state information maintained at these domains. In addition to the load of each DVP server in the domain, we may need to compute the load of the domain as a whole. This is particularly needed, if a hierarchical domain structuring is adopted for the system where domains can be members of upper level domains. In this case the load of the domains is used to decide the directing of service requests to the most appropriate domain.

The load information of a domain is a function of the load of the DVP servers contained in this domain. In the next subsection, a simple approach is used to compute a domain load as the weighted average sum of the load levels of the DVP servers in this domain. The selection of the weights associated to the various DVP servers depends on several factors, such as the reliability and availability of the DVP servers. These factors can be statistically determined based on the past behaviors of the DVP servers. In practice, the designer of the virtual studio assigns weights to the DVP servers involved in the Virtual Studio. Weights are first statically assigned based on either the designer servers' exploitation policy or the characteristics of the servers or both. The characteristics of the DVP servers are obtained from the server's vendor specifications such as the performance of the server in terms of processing power, storage capacity, access speed and so on. The weights are then

dynamically adjusted according to statistics on servers operation and behavior. The domain manager keeps track of DVP servers past behaviors by maintaining, into a log, several information attributes concerning the operational state, the administrative state, the health and availability of each DVP server. Statistics are computed using the information in the log and used to adjust the weights assigned to the DVP servers, for example on a day of the week or on a time of the day basis. The weights can also be adjusted to reflect a new policy of the Virtual Studio service provider. The load information for each DVP server maintained at the domain is obtained from this DVP server according to the multidimensional load model described previously.

5.2.3. Information exchange policy. There are a number of approaches to exchange state information between the DVP servers involved in the load distribution process. These mainly implement a polling procedure, which may be initiated by the client host or by the server. In the context of our domain-based structuring of the virtual studio environment, load information exchange is based on a server-initiated polling mechanism implemented for each individual domain. Several variants of server-initiated polling can be envisaged for load information exchange within a domain. Among them there are:

– Each DVP server periodically sends notifications about its current load.
– A DVP server notifies about its current load whenever it changes.
– The DVP server sends notifications only when significant load levels are reached.

In order to minimize the number of messages exchanged in the system, the last policy is adopted for the exchange of load information. According to the defined load model, a DVP server doesn't need to send a load update notification each time it transits to a new state except if this one is a blocking or a semi-blocking state. Therefore, only the DVP servers capable of supporting user requests with the desired QoS will be considered during task placement decisions.

The exchanged load information, referred to hereafter as Load_Level (LL), is expressed as:

$$LL = \Sigma w_i * b_i,$$

where b_i is a boolean that represents the server state with respect to service requests belonging to cl^i.$b_i = 0$ if the DVP server cannot support any new service request of class cl^i; otherwise $b_i = 1$; initially $t_i = 1$ for $i = \{1, \ldots, n\}$.

w_i indicates the weight associated with cl^i ($w_i < w_j$ means that $cl^i < cl^j$ which in turn means that service requests from cl^j require more server resources than those from cl^i). Hence, the values of w_i, for $1 <= i <= n$, depend on the classification of cl^i, $1 <= i <= n$, in terms of grade of service.

5.2.4. Service access policy. The most frequently invoked tasks are supported by a large number of DVP servers. Similarly, the most frequently accessed video objects are replicated on a large number of video servers. In this perspective, a replication scheme such as the one proposed in [7] can be used. The frequency of accesses to video objects (respectively, task invocations) is equally assigned to all objects (respectively, DVP tasks) statically at

system start up. Access frequency is then incremented each time the object or the task is invoked. The Virtual Studio access architecture easily determines the access frequency for each object as all requests go through it. This is done with a reasonable overhead as the access model is designed to manage the invocation of a relatively limited number of tasks.

Given a user request to perform a DVP task with given QoS requirements, the role of the access architecture is to locate the appropriate DVP server to handle this request. A service access policy is defined to guide the access architecture in the process of dynamically locating the DVP server capable of delivering the requested DVP service while satisfying user QoS requirements. Several factors are taken into account by the service access policy. These are:

1. load of the DVP servers;
2. QoS characteristics of the DVP task; and
3. user QoS requirements and cost constraints.

The ultimate goal of our access model is to minimize the blocking probability of user requests. Therefore, we define the service access policy in such a way to avoid rejecting a local studio request while there are DVP servers in the virtual studio, which might satisfy this request. The following steps summarize the implemented service access policy:

1. Identify the DVP servers capable to perform the requested DVP task; this gives a list of candidate DVP servers.
2. From the list in (1), select the DVP server which can perform the requested DVP task while satisfying the most user QoS requirements, and which has the lightest break load.
3. Start the DVP service delivery (e.g., displaying a video object or executing a DVP task); go to step (2) in case the selected DVP server experiences rapid load fluctuation leading to a violation of the service agreement.

6. Summary and conclusion

Distributed video production systems are seen as a solution for most of the economical and technical challenges that a current studio faces. In this paper, we presented a comprehensive framework for DVP architectures. We described in detail typical DVP tasks and their related system and network performance requirements. We proposed a layered architecture model that captures the system and network characteristics of DVP environments. We finally proposed a QoS-sensitive model for resource allocation and optimization for distributed video production systems. Our architecture addresses several key issues in the design and implementation of DVP systems. We believe that with the advancements in computational and networking technologies, DVP systems will soon become a reality.

Acknowledgment

We would like to thank Mr. Leigh Chang and Mr. Richard Kupniki of Leitch Technology for providing valuable feedback during this project.

References

1. ATM Forum Technical Committee, "Traffic management specification version 4.0," ATM Forum af-tm-0056.000, April 1996.
2. ATM Forum Technical Committee, "Audiovisual multimedia services: Video on demand specification 1.1," ATM Forum af-saa-0049.001, March 1997.
3. B. Bhatt, D. Birks, and D. Hermreck, "Digital television: Making it work," IEEE Spectrum, Oct. 1997.
4. A. Boxer, "Where Buses Can Not Go," IEEE Spectrum, Feb. 1995.
5. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reSerVation protocol (RSVP)—Version 1 functional specification," IETF RFC 2205, Sept. 1997.
6. S.E. Browne, "Video editing: A postproduction primer," 3rd edition, Focal Press: Boston, 1996.
7. A. Dan and D. Sitaram, "An online video placement policy based on bandwidth to space ratio (BSR)," in Proceedings of ACM SIGMOD'95, San Jose, 1995.
8. Draft new ITU-T Recommendation I.363.2, "B-ISDN ATM adaptation layer type 2 specification," Madrid, Nov. 1996.
9. B. Furht, Processor Architectures for Multimedia. Kluwer Academic Publishers: Norwell, Massachusetts, 1998.
10. M.D. Hayter and D.R. Mcanley, "The Desk Area Network," ACM Operating Systems Review, Vol. 25, No. 4, 1991.
11. D. Hehmann, M. Salmony, and H. Stuttgen, "Transport services for multimedia applications on broadband networks," Computer Communications, Vol. 13, No. 4, May 1990.
12. P. Hodgins, "Timing recovery for MPEG video over AAL5," M.A.Sc. Thesis, University of Toronto, Toronto ON, Sept. 1995.
13. D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar, "RTP paload format for MPEGI/MPEG2 video," draft-ietf-avt-mpeg-new-01.txt, June 1997.
14. IETF, "Internetworking over NBMA working group home page," <http://www. com21.com/pages/ietf.html>
15. ITU-T Recommendation H.222.0|ISO/IEC 13818-1, "Generic coding of moving pictures and associated audio information: Systems," April 1995.
16. ITU-T Recommendation H.222.1, "Multimedia multiplex and synchronization for audiovisual communication in ATM environments," 1996.
17. ITU-T Recommendation I.363.1, "B-ISDN ATM adaptation layer (AAL) specification, Type 1 and 2," 1996.
18. ITU-T Recommendation J.82, "Transport of MPEG-2 constant bit rate television signals in B-ISDN," July 1996.
19. D. Kandlur, D. Saha, and M. Willebeek-LeMair, "Protocol architecture for multimedia applications over ATM networks," IEEE Journal on Selected Areas in Communications, Vol. 14, No. 7, Sept. 1996.
20. M. Laubach, "Classical IP and ARP over ATM," IETF RFC 1577, Jan. 1994.
21. G. Lu, Communication and Computing for Distributed Multimedia Systems, Artech House: Norwood, MA, 1996.
22. A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for Multimedia PCs," Communications of the ACM, Vol. 40, No. 1, 1997.
23. T. Pfeifer, "Micronet machines—New architecture approaches for multimedia end-systems," in Proceedings of 4th International Workshop of Network and Operating System Support for Digital Audio and Video, Lancaster House, Lancaster, U.K., Nov. 1993, pp. 29–40.
24. S.V. Raghavan and S.K. Tripath, Networked Multimedia Systems: Concepts, Architecture, and Design, Prentice-Hall, NJ, 1998.
25. P.V. Rangan, H.M. Vin, and S. Ramagathan, "Designing an on-demand multimedia service," IEEE Communications Magazine, Vol. 30, July 1992.
26. Y. Rasheed and A. Leon-Garcia, "Implementation model and performance verification for AAL1 carrying CBR MPEG2 traffic," IEEE ATM Workshop '95, Washington DC, Oct. 1995.
27. Y. Rasheed and A. Leon-Garcia, "ATM adaptation layers for VBR MPEG2 video," Poster Presentation, CITR AGM '95, Vancouver BC, Oct. 1995.

28. Y. Rasheed and A. Leon-Garcia, "AAL1 with FEC for the transport of CBR MPEG2 video over ATM networks," IEEE INFOCOM '96, San Francisco CA, March 1996.
29. A.L.N. Reddy and J.C. Wyllie, "I/O issues in a multimedia system," IEEE Computer, Vol. 27, No. 3, March 1994.
30. J. Rosenberg and H. Schulzrinne, "An RTP payload format for generic forward error correction," draft-ietf-avt-fec-01.txt, Nov. 1997.
31. H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," RFC 1890, Jan. 1996.
32. M. Schwartz and D. Beaumont, "Quality of service requirements for audio-visual multimedia services," ATM Forum, July 1994. ATM94-0640.
33. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC 1889, Jan. 1996.
34. S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," IETF RFC 2212, Sept. 1997.
35. R. Steinmetz, "Analyzing the multimedia operating system," IEEE Multimedia, Spring 1995.
36. R. Steinmetz, "Human perception of jitter and media synchronization," IEEE Journal on Selected Areas in Communications, Vol. 14, No. 1, Jan. 1996.
37. R. Steinmetz and K. Nahrstedt, Multimedia: Computing, Communications, and Applications, Prentice-Hall, NJ, 1995.
38. Task Force for Harmonized Standards for the Exchange of Program Material as Bit Streams, "First Report: User Requirements," European Broadcasting Union and Society of Motion Picture and Television Engineers. April 1997.
39. C. Tryfonas, "MPEG-2 transport over ATM networks," M.A.Sc. Thesis, University of California Santa Cruz, Sept. 1996.
40. H.M. Vin and P.V. Rangan, "Designing a multi-user HDTV storage server," IEEE JSAC, Vol. 11, Jan. 1993.
41. J. Wroclawski, "Specification of the controlled-load network element service," IETF RFC 2211, Sept. 1997.

Raouf Boutaba is an Assistant Professor in the Department of Computer Science at the University of Waterloo since 1999. Before he was with the Electrical and Computer Engineering Department of the University of Toronto. Before that and for three years he was the Director of the Telecommunications and Distributed Systems Division in the Computer Science Research Institute of Montreal. He has been an adjunct Professor at the University of Montreal since 1995. Dr. Boutaba conducts research in integrated network and systems management, wired and wireless multimedia networks, and quality of service control in the Internet. He founded and chaired the IFIP/IEEE International Conference on the Management of Multimedia Networks and Services in 1997. He is the chairman of the IFIP working group on Networks and distributed systems management and a member of the advisory editorial board of the International Journal of Network and Systems Management Dr. Boutaba is the recipient of the Premier's Research Excellence Award in 2000.

Ned Ning Ren was born in Shanghai, P.R.China in 1972. He received a B.A.Sc and M.A.Sc degree in Electrical and Computer Engineering from University of Toronto in 1996 and 1998. Ned has done research on distributed multimedia application, system and resource management. He is currently working as a senior software engineer on advanced VPN management platform in California.

Yasser Rasheed was born in Cairo, Egypt, in 1969. He received his B.Sc. in Electrical Engineering from Cairo University, Egypt, in 1991 and received his M.A.Sc. and Ph.D. degrees in Electrical and Computer Engineering from the University of Toronto, Canada, in 1995 and 2000 respectively. Yasser is currently holds the position of a Staff Network Software Engineer at Intel Architecture Labs, Intel Corporation. His interests include high-speed networking, media distribution in the homes, and Quality of Service.

Alberto Leon-Garcia received the B.S., M.S., and Ph.D. degrees in electrical engineering from the university of Southern California, in 1973, 1974, and 1976 respectively. He is a Full Professor in the Department of Electrical and Computer Engineering of the University of Toronto and he currently holds the Nortel Institute Chair in Network Architecture and Services. In 1999 he became an IEEE fellow for "*For contributions to multiplexing and switching of integrated services traffic*". He teaches undergraduate and graduate courses in communication networks, and conducts research in resources management of broadband networks and service end systems, switch and router design, Internet performance, and wireless packet access networks. He is currently leading a team that is developing

a programmable network node that can be used for the rapid prototyping of packet network protocols. He is also Director of the Master of Engineering in Telecommunications program. Dr. Leon-Garcia was Editor for Voice/Data Networks for the IEEE TRANSACTIONS ON COMMUNICATIONS from 1983 to 1988 and Editor for the IEEE INFORMATION THEORY NEWSLETTER from 1982 to 1984. He was Guest Editor of the September 1986 Special Issue on Performance Evaluation of Communications Networks of the IEEE SELECTED AREAS ON COMMUNICATIONS. He is also author of the textbooks *Probability and Random Processes for Electrical Engineering* (Reading, MA: Addison-Wesley), and *Communication Networks: Fundamental Concepts and Key Architectures*, co-authored with Dr. Indra Widjaja and published by McGraw-Hill.