BUILDING A DISTRIBUTED VIDEO SERVER USING ADVANCED ATM NETWORK SUPPORT

Herbert Bos

University of Cambridge, Computer Laboratory, CB2 3DQ, United Kingdom

hjb1005@cl.cam.ac.uk

Abstract: We describe the network support for, as well as the implementation of, a prototype distributed video server called *BigDisk*, which enables users with small or no hard drives to combine disks in the network for video storage. In this way, storage space is limited by the total amount of disk space in the network, as opposed to the size of one specific drive. This brings the recording of feature-length films on disk within reach even of low-end users on a network with low-cost machines, by storing different segments of a video on different disks using a load-balancing policy. BigDisk is built on top of a network control architecture with resource reservation in advance which is aware of interdependencies between connections and provides a handle on the latency. Finally, replication mechanisms are discussed to provide low latency, load balancing and efficient resource utilisation for distributed video servers.

1 INTRODUCTION

Although audio and video sources and sinks are rapidly becoming ubiquitous in computer networks, the recording and playback of continous media (CM) data, is by no means a solved problem. We can identify problems both at the storage level and at the level of network control.

1.1 Addressed problems at storage level

First, most users do not have the storage space to record large high-quality video files. For example, the recording of 1 second of HDTV requires more than 17 MB of diskspace [Jardetzky et al., 1995].

Second, even when storage space is abundant, we find that most present-day disks are only able to serve a small number of simultaneous CM streams

without compromising Quality of Service (QoS) guarantees. These systems (or the network access to them) are likely to become bottlenecks. Load balancing, therefore, becomes important. Striping is often proposed as a solution, but although striping is a very useful technique, it is not without its problems (e.g. when one has to add a disk to a striping group or when different types of disks are used). Nevertheless, we recognise the usefulness of striping and do not preclude it in any way. Indeed, whenever we talk about *disks*, one may well read *striping group*[1]. Also, segmenting videos and distributing them over a number of small disks to create a virtual big disk as proposed here is really not all that different from coarse grained striping (except that the striping units may vary widely).

Third, there is a latency problem. Because CM files take up a lot of space they are generally not distributed over many nodes in the network. So each access to a video file requires access to some video server, which may be far away. This means that the latency to access a particular CM file is probably quite high, even if the user only wants to quickly browse through a number of video titles (channel hopping). Traditional techniques to alleviate the latency problem such as prefetching do not lend themselves for large CM files.

Prefetching is a technique that is proposed to speed up Web latency. Suppose a Web page contains a number of links to other pages (containing images, text, etc.). We may improve latency (perceived response) by not waiting to download a specific page until the user has clicked on the corresponding button, but instead prefetching all the data for all links the user is likely to follow. This cannot be easily extended to Web pages of video servers where the links point to potentially very large movies.

1.2 Addressed problems at network control level

Underlying the distributed applications there is network control. In the ATM world this comprises at the very least the setting up and tearing down of connections. This in turn includes routing and resource allocation. We define a control architecture as the set of protocols, policies and algorithms to control the network. Many control architectures have been proposed for ATM (e.g. Q.2931, IP switching, P-NNI, etc.), but most of these do not address the issue of resource reservation in advance and those that do (e.g. [Schill et al., 1997], [Wolf and Steinmetz, 1997]) ignore problems that are important when CM data is expected to be very common.

Such problems include the interdependency of connections. Often a specific connection is only meaningful in the context of other connections and its acceptance or rejection (possibly at some time in the future) is of little value in the absence of the acceptance or rejection of certain other connections. Reusing the resource allocation of a connection $C1$ for connection $C2$ that is known to replace a large part of $C1$ (i.e. to follow the same route with the same QoS

[1] although in that case we are probably no longer talking about low-end users

requirements) is also largely ignored. Finally, the related problem of latency in a distributed CM application reappears at the level of network control and is generally not the focus of current control architectures.

1.3 Contribution

In this paper, we first describe a prototype distributed video server (DVS) called *BigDisk* that addresses all three of the problems of section 1.1. It places few demands on the machines in the network (indeed, these could even be low-end PCs with small disks). The load on the disks is automatically balanced and a solution is proposed that both improves latency considerably and enables prefetching for CM.

BigDisk serves as an example application to demonstrate an advanced control architecture that allows reservation of resources in the network in advance and provides support for new types of connection, complete with QoS guarantees. BigDisk is an innovative distributed application storing and retrieving CM data for low-end and high-end users alike.

We will show that the DVS itself can be very simple since it is supported by the control architecture for ATM which gives the DVS guarantees as to the future availibility of resources as well as to the cell loss ratio (CLR) and latency. The control architecture is not intended to replace existing ones. Instead it is expected to run alongside control architectures such as Q.2931, IP-Switching, etc., where needed.

1.4 Overview

In section 2, we give a high level overview of the operations of the DVS, in section 3 we introduce the control architecture that makes it all possible. Given the functionality of the control architecture, section 4 sketches how the recording and playback of the DVS was implemented. In section 5, we discuss the problem of latency as well as mechanisms to reduce it. Finally, we summarise and draw conclusions in section 6.

2 RECORDING AND PLAYBACK WITH THE DVS

In this paper, we are not concerned with the problem of guaranteed rate storage servers[2]. Interested readers should refer to [Anderson et al., 1992], [Lo, 1994], [Bosch and Mullender, 1996], [Jardetzky et al., 1995]. From our point of view, the issue of whether the lower-level storage server is capable of delevering guaranteed rate is an endpoint problem. We assume that the request that is sent to the DVS (and through the DVS to the network) is based on the endpoint QoS constraints that are only known at the application level[3]. In cheap systems, users will simply use the cheap disks in their PCs for storing and re-

[2]Nevertheless, we have built a simple CM storage server for experimentation purposes.
[3]Of course, this *should* be taken into account to calculate end-to-end QoS guarantees

trieving. High-end users may prefer to use guaranteed-rate storage servers for high-quality video. Both types of users are supported.

For the time being, we assume that the principal aim of the DVS is to provide users with limited diskspace with a virtual 'BigDisk', which is comprised of all disks on the network on which the user has write permission. In the DVS each designated disk stores a segment (e.g. 10 minutes) of a large video. Figure 1 illustrates how a large video file has been segmented and distributed over multiple disks in the network and in specific what the playback consists of.

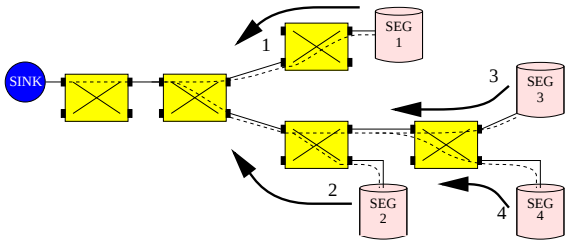

Figure 1 Video segments distributed over multiple disks

BigDisk should scale to a large number of disks. The front-end of the DVS is a graphical user interface, while the backend essentially consists of calls into the control architecture. The front-end enables users to inspect the various disks on the network, add new disks to the system and select the mode for this DVS session. The BigDisk DVS has two modes: a recording mode and a playback mode.

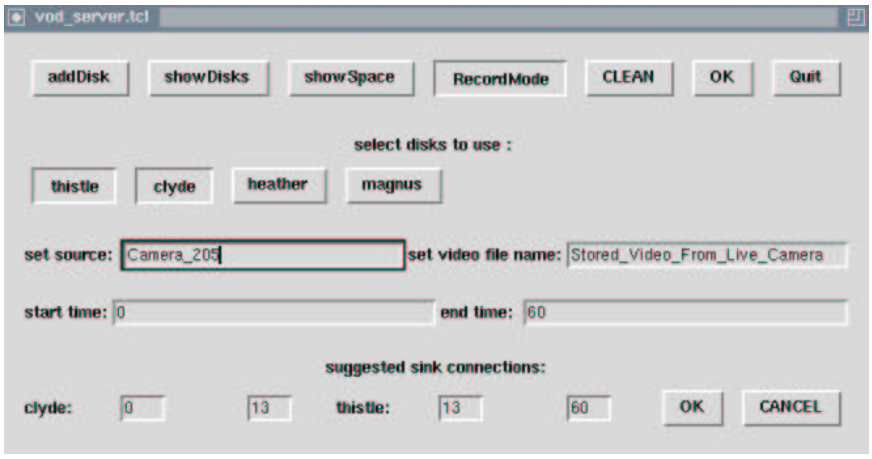

Figure 2 Recording mode

In recording mode (see figure 2), users select the disks on which to record the CM data. This means that even if a video fits entirely on the local drive, users may still choose to distribute it over a number of disks to balance the

load. After users have selected the disks that act as sinks, and the video stream that acts as source, they specify the name for this video (under which it will be stored on all disks) and the start time and end time for the stream (e.g. in case of a recorded television program, these can be the start time and end time of the program).

Finally, the user presses OK at which point the DVS suggests a particular segmentation and distribution corresponding to the user's choices. This suggestion is based on a *policy* such as load balancing. A simple load-balancing scheme that we have experimented with is one in which a user has been allocated varying amounts of diskspace on different disks and the DVS tries to guarantee the same relative disk usage on all disks involved. The suggestion can be overruled if the user so desires and new policies can be plugged in on the fly.

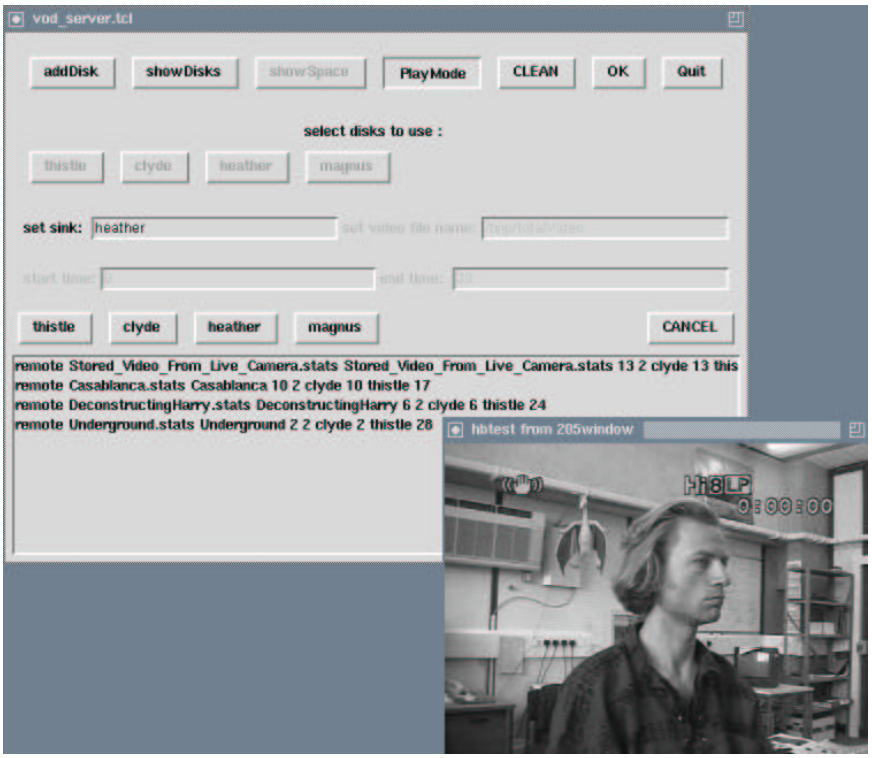

Figure 3 Playback mode

In playback mode (see figure 3), users select a video to play, on any of the disks in their system. When the video is selected, the DVS queries a small video database to find out which segments make up this video and where they are stored. It will start playing them in the right order and at the appropriate times.

3 THE CONTROL ARCHITECTURE

Note that the high level operations described in the previous section, introduce the following problem (see also figure 1). If we agree to record (play out) a particular video, we have to guarantee that all segments of the video can be stored (played). For example, we must avoid the situation where we unexpectedly find out when we get to the last segment of the video, that the recording (playback) of this segment is rejected after all, because the connection setup request is rejected (e.g. because of growing load in that segment of the network). This is the problem of *temporal connection interdependency* [Bos, 1998], i.e. the acceptance of one call, sometime in the future, depends on the acceptance and time of establishment of some other call to be meaningful.

Note also that in the previous section no mention whatsoever was made of the network. During normal operation, the network is completely transparent to the BigDisk users. They only need to specify which set of disks they would like to use for the video, or which video they would like to play out on which machine and the DVS backend ensures that if the user request is accepted then all connections are established at the appropriate times, and with guarantees as to the availability of the resources.

This is only possible because of the underlying control architecture. It is clear that to give any guarantees about the future availibility of resources, it must be possible to reserve resources in advance. However, it is shown in [Bos, 1998] that these guarantees go beyond a simple end-to-end advance reservation scheme.

3.1 Problems with traditional end-to-end reservation

Consider the playback scenario in which two source disks, connected to the same switch contain two consecutive segments of a video[4]. This is illustrated in figure 1 by file servers 3 and 4. In other words, the connections from both sources to the sole sink, are almost identical and follow one another seamlessly with the same QoS. End-to-end advance reservation of connections, however, precludes the reuse of part of an existing connection. In all probability, this results in a very large handoff overhead, because a new connection has to be set up from endpoint to endpoint which may involve many switches. In the worst case, it means that all resources upto and including the sink are bothered with the setup of a connection that is exactly the same as its predecessor.

The only way to give the required guarantees is to make n separate end-to-end reservations, where n is the total number of connections needed to play an entire video file. This not only introduces a large, unnecessary overhead, maintaining n separate connections ($n \gg 1$) also presents a formidable management and administration task. It would be much easier if we introduced a new type of connection which can have multiple sources (in sequence) or multiple sinks or both. For example, in the latter case, one tear-down request suffices to remove

[4]Similar arguments can be made for the recording mode.

all of the state corresponding to the (potentially complex) set up connection from the network.

3.2 Advance reservations without source modelling

The control architecture described in this paper has a simple interface that allows for advance reservation of connections of arbitrarily complex nature. The advance reservation mechanism decouples the time of request for resources from the time of allocation of resources. This enables applications to say to the control architecture: at 8pm tonight I want to combine the following resources to produce this service. A simple example of such a reservation is a reservation of connections from one t.v. receiver to a number of low-end disks. If the control architecture accepts the request, it essentially enters into a contractual agreement with the application to provide the requested resources and connections at the requested times.

In the current implementation, the description of what resources are required for a connection is very simple and based on peak rate. This is because we believe that accurate source characterisation (modelling) is prohibitively difficult. However, to alleviate potential over-conservativeness, an effective bandwidth estimation based on measurements of cell arrivals is used for all *active* connections (as opposed to *future* connections) as described in [Bos, 1998].

Connection sequences and connection patterns. To enable the DVS to give the necessary guarantees to deal with temporal connection interdependency, we now introduce the following 2 new connection types:

1. *Connection sequence*: this type has an arbitrary number of sources that follow each other seamlessly and a single sink that may not even be aware of the number or location of the upstream sources or hand-offs. Identical portions of the connections of subsequent sources are automatically reused. The operation to reserve and create this type of connection takes as arguments a reference for a sink (e.g. a display program on a particular machine), a start time, and a list of source records each of which specifies a source, a peak rate and an end time[5].

2. *Connection patterns*: this type has the same source sequence, combined with an arbitrary list of (possibly overlapping) sinks. The operation is largely the same as a connection sequence, but the hand-off process is more complicated. The operation takes the same arguments as the connection sequence except that there is now a random list of sink records each of which contains a sink reference as well as a period $[t_{start}, t_{end}]$ during which this sink is active. Overlapping sink intervals effectively indicate multicasts during the overlapping periods.

[5]The end time is the time until which this source i is active and serves as the *start time* for source $i + 1$.

Both types of connections can be reserved in advance, modified and made resilient to failures. A more detailed description of these and other operations of the control architecture can be found in [Bos, 1998].

Network programmability. The operations discussed in the previous section probably suffice for most applications. Our claim in the first paragraph of section 3.2, however, was that advance reservation of connections of an arbitrarily complex nature were supported. The way we achieved this goal was by making the control architecture *extensible* (sometimes also called *elastic*). Applications are able to load their own code into the control architecture, effectively allowing them to implement any policy they want to. A very low-level and same address-space interaction with the control architecture allows this code to be highly efficient. Here, as elsewhere, the issues to do with resource reservation (e.g. reserving bandwidth on a particular switch port for some time interval in the future) are decoupled from the issues to do with resource allocation (e.g. setting up a connection across a switch from a reserved input [port, vpi, vci] to a reserved output [port, vpi, vci]). Both resource reservation and resource allocation are programmable by applications. It is beyond the scope of this paper to discuss the mechanisms (or the corresponding security issues) in detail.

3.3 Failures

What happens if certain resources (e.g. switches or hosts) are unavailable? Again without going into detail, we state that the control architecture has a number of hard-state failure recovery procedures (i.e. state has to be explicitly removed from the network) that enable it to recover gracefully from most failures. In case of failure, the connection state is in principle removed and all corresponding resources released. It is possible, however, to declare connections (and sequences and patterns) *persistent* which means that even if the connection manager itself goes down and comes up again, the connections will remain in, or be restored to, their correct states at that time. A more detailed description is given in [Bos, 1998].

3.4 One of many control architectures

As mentioned before, the control architecture which supports the DVS is not meant to replace any existing control architectures. It is meant to run alongside them wherever and whenever it is needed. This is part of our belief that there is no one-size-fits-all solution for network control, which has led to the development of switchlets here in the Cambridge Computer Laboratory as described in [van der Merwe and Leslie, 1998]. Switchlets allow one to partition resources on switches, where each partition can be controlled by its own control architecture.

The control architecture described here is distributed in nature and uses a CORBA-based distributed processing environment. It falls under what is

called *Open ATM Network control* and runs on general purpose workstations. It consists roughly of the following components (see also figure 4). Firstly, there is a federated trading mechanism to find and obtain offers for services, for example video producing services on the source side or display services on the sink side. Secondly, for each host that wants to communicate, there is a local host manager (HM), which provides the interface to the control architecture and controls some local resources. Thirdly and most importantly, there is the connection manager (CM) where all the actual network control is exercised[6]. It is the control manager that talks to the switch(let) and sets up the connection. This is also where such things as admission control take place. Finally, there are clients and servers that export service offers to the traders. A more detailed description of the control architecture can be found in [Bos, 1998].

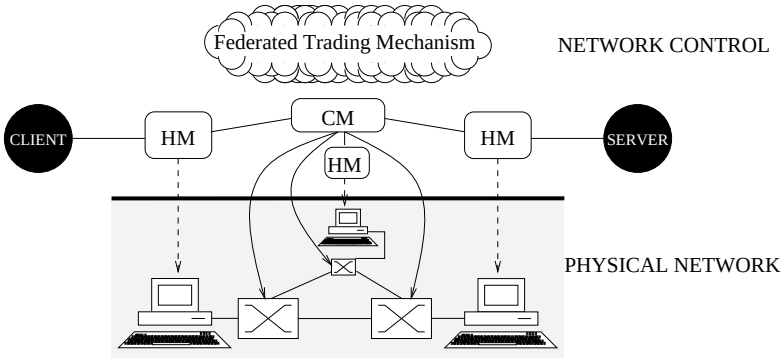

Figure 4 Components of the architecture

4 IMPLEMENTATION OF THE DVS

Acting as a client to the control architecture and using the *connection sequence* and *connection pattern* types described in (3.2), the DVS can easily provide the necessary guarantees. The operation of the BigDisk backend (i.e. the control architecture) on a request from the DVS is now roughly as follows:

1. *Playback:*
   (a) make an advance reservation for a *connection sequence* involving the appropriate disks for the appropriate (segment-)times and the sink for the entire duration of the video;
   (b) if successful, return the reservation identifier to the DVS (which enables the DVS to change the reservation) and sleep until t_{start}, the requested time to start the playback[7];

[6]figure 4 suggests that there is only 1 (central) CM. In reality there may be multiple CMs active simultaneously
[7]in case this is not 'now'

(c) at t_{start}, set up the connection (allocating the appropriate amount of bandwidth) and call back the sink and the first source disk server in the sequence (essentially telling them to start receiving and sending, respectively);

(d) at the end of each segment, flip the connection to the next source and call its callback function (also: tear down the dangling connection to the previous source and notify it of the fact) and so on until all sources have finished;

(e) at that point, free all resources, notify the currently active sink and the currently active source and remove all state from the system

2. *Recording:*

(a) reserve a *connection pattern* with one source (e.g. the t.v. program you want to record) and n sinks (the disks on which you want to spread this video);

(b) if succesful, return the reservation identifier and wait until the program starts;

(c) set up the connection and notify the source and the first sink;

(d) at the end of the pre-reserved time of segment 1, flip the connection to the next sink, clean up dangling connections, and repeat this process until all segments are stored;

(e) free all resources, remove all state

All the BigDisk front-end has to do is call the operations (1a) and (2a) and execute the call back functions (that it has registered) when they are called by the backend. The rest of the DVS consists of a small distributed database to manage the video segments and a storage server that is CORBA aware and can be prompted to play or record video.

5 LATENCY

The latency resulting from using the control architecture of section 3 is potentially much lower than in the case of end-to-end connection setups that come with other advance reservation schemes (or with more traditional control architectures such as Q.2931 or P-NNI). Even so, when setting up a number of sources in sequence, latency becomes a very important issue if we want to ensure smooth playback.

We define the *initial latency* as the time it takes before the first video data starts arriving at the sink, measured from the time we issued the connection request. We define the *follow-up playback latency* as the time between the arrival at the sink of the last cell of one source and the first cell of the next on a source handoff in playback mode. Finally, we define the *recording gap* as the number of cells that get lost due to the time it takes to flip the switch connection in record mode. The recording gap can be 0, if on the cut-off switch

we set up a connection to the new sink first before removing the old connection (effectively a short-lived multi-cast). In that case, we only have a few cells that are duplicated on the old and new sink[8].

Initial latency also, is generally not a problem in video playback, because all it means is that a video only starts at the client side a few milliseconds after the client issued the request. At any rate, the reservation in advance gives you a way to control the lower bound of the latency, in the sense that can specify you don't want the playout to start until a specific time t_{start}. At t_{start}, when the connection is set up, we see that the latency is lower in the control architecture described here (compared to traditional control architectures), because at that time we have already communicated the request to the control architecture, exercised call admission control, allocated resources, etc. This time will be saved on the initial latency.

5.1 Follow-up playback latency

This leaves us with the follow-up playback latency. This is a hard and important problem. The follow-up latency means that if we simply were to play the data immediately when it arrives at the sink, there would be *glitches* in the playout at the time of handoffs. These gliches have a length of the follow-up playback latency and are undesirable. To avoid them, we stick a small buffer at the sink, aimed at absorbing the latency (as well as possible jitter). The question is: how small can this buffer be?

Note that in the case of BigDisk, it is probably adequate to use a buffer that is simply 'big enough'. In other words, if we buffer a few seconds of video, this will be plenty to aborb all follow-up latency and all jitter at the expense of the initial latency (which increases by a few seconds). Although adequate, this is not a very satisfactory solution and, in fact, we are able to do to better. The control architecture described in this paper is capable of giving probabilistic upper (and lower) bounds on the follow-up playback latency, across a single switch and even end-to-end. Although it is beyond the scope of this paper to discuss the mechanism in detail, we briefly discuss the user-level view.

Essentially, DVS clients are allowed to specify a latency probability bound (lpb) which they submit to the control architecture. The control architecture is then able to calculate the corresponding delay D_{lpb}. For example, an lpb of 99 % means that the client wants to know the delay bound that 99 % of the traffic stays under. In other words: $P(delay > D_{0.99}) \leq 0.99$. This delay bound can be used to determine the size of the buffer required to guarantee smooth playout.

[8]which may increase the follow-up playback latency slightly, when the video is played back

5.2 Segment replication

What if the network cannot meet the desired guarantees of a client's request? There are two options. The first option is to reject the request. This is what's most commonly done. The other option is to try and enable the system to accommodate your request. This is what's advocated in [Bos, 1997] and also in this paper. Note that the reasons for not accepting a request can vary greatly. For example, it may be that the advance reservation fails because the network's capacity is already fully reserved for the desired interval. This is an example of a request that would be rejected because of other traffic in the network. It may also be that a request would be rejected because the video server is simply too many hops away to meet the delay/latency demands, *regardless* of the presence or absence of other traffic. This request could *never* be satisfied.

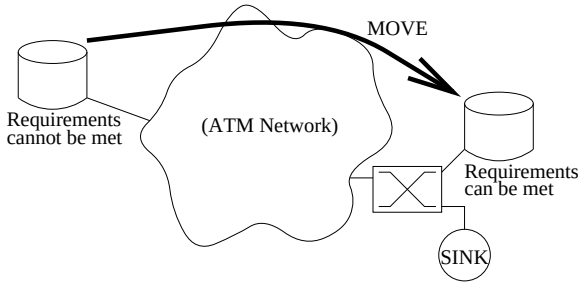

Figure 5 Moved segment is able to meet QoS requirements

Segment replication. Both problems would be addressed if we moved the segment stored on that problematic file server to a place in the network from where the required guarantees can be satisfied, e.g. to a disk on the same switch as the sink (see figure 5).

For example, assume that segment i in a connection sequence is the segment that is causing the problems (i.e. the call to this disk would not be accepted because it can't meet the required guarantees). We define t_i as the time that the playout of segment i should start. Now, observing that we have from $t = now$ until $t = t_i$ to move things around in the network in such a way that at t_i we will be able to play back segment i, we propose the following admision control policy:

1. If the request can be admitted by the network *as is*, i.e. without shuffling segments around, admit the request.

2. If not, see if segment i can be replicated on another disk from where the guarantees can be met (before t_i), if so admit the request.

3. If not, see if we can move segments for already accepted requests to other disks from where their own guarantees can still be met and which makes

room for the new request's connection to segment i on some disk from where its required guarantees can be met.

4. If not, reject the request.

Currently, only the first 2 steps in the admission process have been implemented for the DVS. So, segments get automatically replicated to a lightly loaded or better-located disk F_{light}, if this enables the control architecture to meet the required guarantees. The playout will then take place from F_{light}. All this is transparent to the client. It is shown in simulation that the implemention of this policy will increase the number of accepted requests significantly [Bos, 1997].

What should happen to the replicated segment on F after the playout is the topic of future research. It could be removed immediately, so as not to waste resources or it could be kept (at least for some time) to offload a highly loaded disk (or the part of the network en route to this disk, etc). It would be trivial to implement the strategy advocated in [Dan et al., 1995].

5.3 First segment replication

The segment replication policy of the previous section enables us to provide lower latency than would be possible if we were to use the original placement of segments. This can be exploited in a different way as well. Consider a system with a large number of video files. Although replication is often proposed to improve availability and latency for these essentially write-only files, it makes no sense to replicate even a subset of these videos due to their enormous sizes.

A solution for this problem is to replicate only the first (small) segment of each video over many network nodes, while keeping a single replica of the other segments (preferably *not* on a central server). This is shown in figure 6. The initial latency to the video file will then be very small (there is always a first segment on a disk nearby) and the time to play this first segment can be used to move the remaining segments close enough to satisfy the QoS requirements.

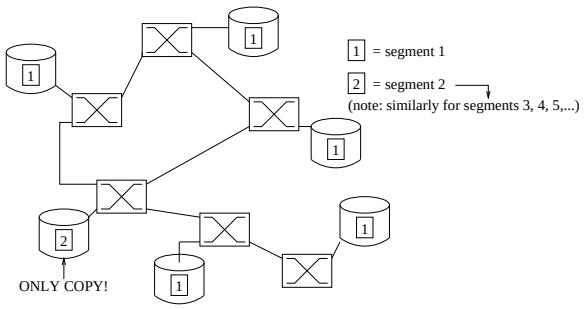

Figure 6 The first segment of a video is heavily replicated

For this purpose, the user interface shown in figure 2 has an extra option which enables users to replicate specific segments on other disks. Observe that

the replication of the first segment over many nodes is essentially a form of caching.

Video prefetching. We might be able to use the above to deal with the prefetching problem described in the introduction. Suppose we have a Web page with a large number of links to very large video files. As stated before, it makes no sense to prefetch all the data corresponding to all the links that the user might follow. On the other hand, we could prefetch the first segment of each these files and even if these would not fit completely on our local host, they could be stored temporarily on BigDisk, i.e. on some local disks in our network. This would allow users to quickly browse through a large number of videos without incurring large latencies each time a new video button is clicked.

6 CONCLUSIONS

We have described how a novel network control architecture can support an innovative distributed video server. The video server (and applications like it) would be difficult if not impossible to implement with traditional control architectures due to the lack of support for temporal connection interdependency and latency bounds. A prototype implementation of both the control architecture and the video server has been achieved. The control architecture allows reservations in advance and is able to give probabilistic latency guarantees. Replication mechanisms were described to improve latency and decrease the rejection ratio in a network by load balancing. The segmentation of videos, which started out as necessity because of limited storage space and the need for load balancing, is exploited in ways to make possible such things as video caching, video prefetching and sharing of a large number of local disks.

References

Anderson, D. P., Osawa, Y., and Govindan, R. (1992). A file system for continuous media. *ACM Transactions on Computer Systems*, 10(4):311–337.

Bos, H. (1997). An active distributed file server for continuous media. In *Proceedings of the 2nd European Research Seminar on Advances in Distributed Systems*, pages 93–98.

Bos, H. (1998). Efficient reservations in open atm network control using online measurements. *International Journal of Communication Systems*, 11(4):247–258.

Bosch, P. and Mullender, S. J. (1996). Cut-and-Paste File Systems: Integrating Simulators and File Systems. In *Proceedings USENIX Conference*, San Diego, California.

Dan, A., Kienzle, M., and Sitaram, D. (1995). A dynamic policy of segment replication for load-balancing in video-on-demand servers. *Multimedia Systems*, 3(3):93–103.

Jardetzky, P. W., Sreenan, C. J., and Needham, R. M. (1995). Storage and synchronisation for distributed continuous media. *Multimedia Systems*, 3(4):151–161.

Lo, S. (1994). *A Modular and Extensible Network Storage Architecture*. PhD thesis, University of Cambridge Computer Laboratory, Computer Laboratory, Pembroke Street, Cambridge CB2 3QG, U.K. Also published as Technical Report No. 326.

Schill, A., Kuehn, S., and Breiter, F. (1997). Resource reservation in advance in heterogeneous networks with partial atm infrastructures. In *Proceedings of INFOCOM'97*.

van der Merwe, K. and Leslie, I. (1998). Service specific control architecture for atm. *IEEE Journal on Selected Areas In Communications*, 16(13):424–436.

Wolf, L. and Steinmetz, R. (1997). Concepts for resource reservation in advance. *Multimedia Tools and Applications*.