Fault Tolerant Video on Demand Services*

Tal Anker　Danny Dolev　Idit Keidar

Institute of Computer Science
The Hebrew University of Jerusalem
{anker,dolev}@cs.huji.ac.il

Laboratory for Computer Science
MIT
idish@theory.lcs.mit.edu

Abstract

This paper describes a highly available distributed video on demand (VoD) service which is inherently fault tolerant. The VoD service is provided by multiple servers that reside at different sites. New servers may be brought up "on the fly" to alleviate the load on other servers. When a server crashes it is replaced by another server in a transparent way; the clients are unaware of the change of service provider. In test runs of our VoD service prototype, such transitions are not noticeable to a human observer who uses the service.

Our VoD service uses a sophisticated flow control mechanism and supports adjustment of the video quality to client capabilities. It does not assume any proprietary network technology: It uses commodity hardware and publicly available network technologies (e.g., TCP/IP, ATM). Our service may run on any machine connected to the Internet. The service exploits a group communication system as a building block for high availability. The utilization of group communication greatly simplifies the service design.

1. Introduction

Video on demand (VoD) services are becoming popular today in hotels, luxury cruise boats, and even airplanes. As high bandwidth communication infrastructure (e.g., ATM backbone networks along with ADSL, the Internet infrastructure, etc.) is being established in many countries around the world, high bandwidth communication lines will reach millions of homes in the near future. This increasing improvement in communication technology will invite widespread utilization of VoD services in private homes, provided by telecommunication companies, cable TV providers, and via the Internet. In such an environment, scalability and fault tolerance will be key issues.

*This work was supported in part by the Ministry of Science, Basic Infrastructure Fund, Project 9762 and by Optibase Ltd.

In this paper we describe a highly available distributed VoD service. The VoD service is provided by multiple servers that may reside at different sites. The service supports smooth migration of clients from one server to another. Thus, the number of servers providing a certain service may change dynamically in order to account for changes in the load. We use a group communication system in the control plane of our service, in order to loosely coordinate the participating servers to agree upon client migration and to allow one server to take over another server's client. Our service uses a sophisticated flow control mechanism and supports adjustment of the video quality to client capabilities. We do not assume any dedicated hardware or proprietary technology: Our service uses commodity hardware and publicly available network technologies (e.g., TCP/IP, ATM). Our servers and clients may run on any machine connected to the Internet.

Current efforts in the area of VoD focus primarily on increasing the throughput of a single server by using sophisticated scheduling, caching, and file structuring. The fault tolerance issues typically being addressed concern possible disk and file failures [11, 14, 16, 18, 19, 20, 21], but do not address server failures or network partitions (with the exception of the Microsoft Tiger video server [12, 13], cf. Section 7) . Furthermore, current methods rarely address the issue of client migration and smooth provision of service while migration occurs. Thus, the concept presented in this paper complements the above techniques, in that it allows extending such VoD services to be provided by a dynamically changing number of servers.

Video transmission requires relatively high bandwidth with strict *Quality of Service (QoS)* properties (e.g., guaranteed bandwidth, bounded jitter and delays). Therefore, as any application involving video transmission, our service is best provided using QoS reservation mechanisms. However, if bandwidth is abundant and jitter rarely occurs, e.g., in a relatively not loaded LAN or small scale WAN, then some buffer space and a flow control mechanism can account for jitter periods. We have tested our VoD service on such networks with good results.

In our service architecture, each movie is replicated at a subset of the servers. When a server crashes or disconnects from its clients it is replaced by another server (holding the same movie) in a transparent way. Clients are also migrated from one server to another for load balancing purposes, e.g., when a new server is brought up. The main challenge we address is designating an alternate server and making the transition between servers smooth, so that the clients would be unaware of the change in the service provider.

This is challenging, since when the client migrates to another server, the video transmission may stop for a short period, frames may arrive twice, or may arrive out of order. We call periods at which such undesirable events occur *irregularity periods*. The duration of the irregularity period depends on the level of synchrony among the servers. Our VoD service does not assume tight coupling of the servers; in our prototype servers synchronization occurs every half a second, and the overhead for server synchronization consumes less than one thousandth of the total communication bandwidth used by the VoD service.

In order to guarantee smooth video display at such irregularity periods the client maintains a buffer of forthcoming frames. The buffer size is subject to fine tuning, depending on the expected irregularity period duration. In our experiments with a 1.4 Mbps video stream, the clients have allocated buffer space of approximately 1.7 Mbit in software in addition to 1.7 Mbit in a hardware MPEG [17] decoder.

We have designed a flow control mechanism which endeavors to keep enough frames in the buffer to account for irregularity periods and jitter, but without causing the buffers to overflow. It was challenging to tune the flow control algorithm to re-fill the client's buffers quickly (but without causing overflow) at irregularity periods. Our flow control mechanism is presented in Section 4. We tested our service both on a 10 Mbps switched Ethernet and on a small scale WAN. Our results are encouraging: The video display at times of migration (due to either server crash or load balancing) is smooth to the human observer.

Our VoD service implementation exploits the Transis [2, 15] group communication system for synchronization among the servers, for connection establishment and for exchanging control messages, following the concepts we suggested in [6]: In [6] we described the benefits of using group communication for highly available VoD services. As a "proof of concept", we presented a preliminary VoD service prototype transmitting low bandwidth video material to clients that use software decoders. In Section 5 we describe how we exploit group communication in our current VoD service to simplify the service design. The concepts demonstrated in this work are general, and may be exploited to construct a variety of highly available servers.

2. The Environment

Our VoD service tolerates server failures and network partitions. It exploits commodity hardware and publicly available network technologies (e.g., TCP/IP, ATM); servers and clients may run on any machine connected to the Internet. As any video transmission application, our VoD service is best provided if a QoS reservation mechanism is available, e.g., when using an ATM network. However, this is not mandatory. In networks with abundant bandwidth and limited jitter, e.g., a relatively not loaded fast/switched Ethernet, or if only "soft" reservation is available (e.g., with RSVP [22]) our buffer space and flow control mechanism can account for jitter periods.

The video material is stored and transmitted in the standard MPEG [17] format. Clients use hardware MPEG decoders in order to process high bandwidth video. An MPEG encoding of a movie consists of a sequence of frames of different types: I (Intra) frames represent full images; other frame types are *incremental* and cannot be decoded without the corresponding I frames. The movie is transmitted frame by frame – a single frame is transmitted in a single message.

The communication channels used for transmitting the video material may be unreliable, in the sense that messages may be lost or arrive out of order. Our VoD service does not recover lost frames. Therefore, if the communication channel suffers message loss then a degradation occurs in the quality of the displayed movie. The VoD service uses client buffers to re-order frames that arrive out of order (i.e., insert these frames in the right place in the video stream).

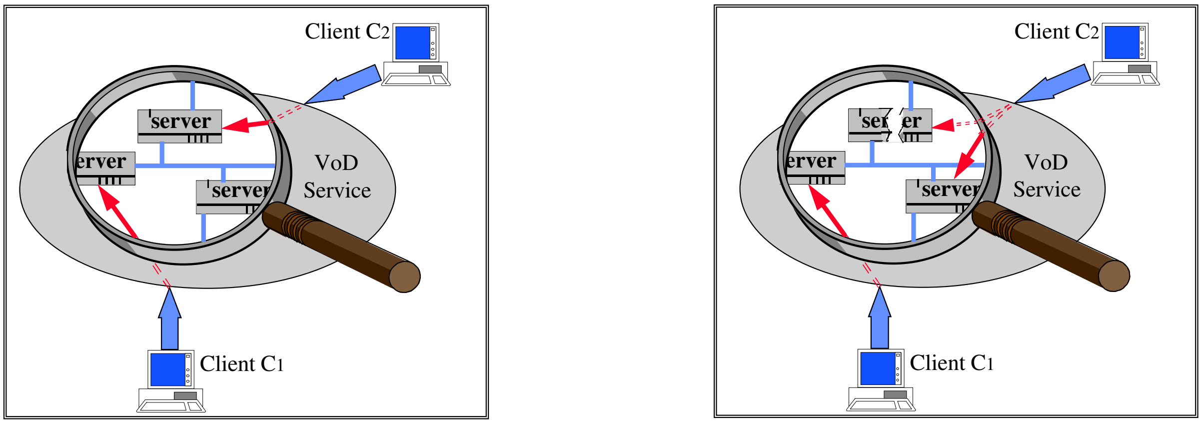
Our VoD service requires a (possibly unreliable) failure detection mechanism in order to detect server failures. It also requires a reliable multicast mechanism for low-bandwidth communication among the servers, for connection establishment and for control messages. In our prototype implementation we used the Transis [2, 15] group communication system for these tasks (cf. Section 5).

3. The Service Overview

In this section we describe the overall design of the VoD service. More details of our specific algorithms appear in the following sections.

Each movie is replicated at a subset of the servers[1]. Clients connect to the video on demand service and request a movie to watch from a list of offered movies. One of the servers that hold this movie forms a two-way connection with the client: The server transmits video material, and the client sends control messages for flow control purposes as well as for speed control and for random access within the movie. The clients have full VCR like control

[1] We assume a separate mechanism for replicating the video material.



(a) Before the server failure.

(b) Recovery from the server failure.

Figure 1. Transparent VoD services.

over the transmitted material, e.g., pause, restart, and arbitrary random access, in accordance with the ATM Forum VoD specs [10].

Each server periodically sends information about its clients to the other servers (for details, please see Section 5). When a server crashes or detaches, the remaining servers take over the clients of the crashed server, so that each client is served by exactly one server. The client is oblivious to the change, as shown in Figure 1. A similar process occurs when the servers decide to migrate clients because the load is poorly distributed, e.g., when a new server is brought up to alleviate the load. The client migration process is described in detail in Section 5.

The client maintains two buffers of forthcoming frames: one in the hardware decoder and one in software (cf. Section 4.2 for a discussion of buffer sizes). Received video frames are first stored in the software buffer and then streamed into the hardware decoder. In case of buffer overflow, frames need to be discarded: If a frame arrives when the buffer is full, we discard one of the frames in the buffer to make space for the new frame. When possible we discard an incremental frame and not an I (full image) frame.

The buffers allow smooth video display at migration times. The software buffer is also used for re-ordering of video frames that arrive out of order. Our-of-order frames can be inserted to the right place in the video stream only if they arrive before they should be streamed into the hardware decoder. We discard frames that arrive after the hardware decoder consumed frames that follow them. The flow control mechanism's task is to keep enough frames in the buffer to account for irregularity periods and for re-ordering, while avoiding buffer overflow. The flow control mechanism is described in Section 4.

4. Flow Control

Our VoD service uses a loosely-coupled feedback-based flow control mechanism: The client sends flow control messages to the server in order to dynamically adjust the transmission rate. The server maintains the current rate per client, and adjusts it according to the client's flow control requests. Clients may request to either increase or decrease the transmission rate by a certain Δ. In our prototype implementation this Δ is one frame per second. If, for example, the server transmits 25 frames per second to a certain client, and an increase request arrives from this client, then the server changes the rate to 26 frames per second.

The client's flow control module endeavors to keep enough frames in the buffers to account for irregularity periods, and also to allow for re-ordering of frames that arrive out of order. Albeit, it must be careful not to increase the transmission rate too much and not to cause buffer overflow.

The client does not try to deduce at which rate the server is transmitting the video, it only keeps track of the buffers' occupancy (i.e., the number of frames in the buffers). The flow control mechanism attempts to keep the number of frames in the buffer between the *low water mark* and the *high water mark* thresholds. If the number of frames falls below the low water mark, then the transmission rate is increased, and if the buffer is full above the *high water mark*, the transmission rate is decreased.

Due to network delay, the transmission rate does not change instantaneously. Therefore, after increasing the transmission rate sufficiently to surpass the low water mark, the client must start requesting to slow the transmission rate down before the occupancy surpasses the high water mark. Likewise, the client must request to increase the transmission rate before the occupancy falls below the low water mark. Thus, when the number of frames in the buffer is

Value of Buffer Occupancy			Check Frequency	Request to send
from	to	**and**		
0	critical threshold-1		f_urgent	emergency
critical threshold	low water mark -1		f_urgent	increase
low water mark	high water mark-1	$<$ previous occupancy	f_normal	increase
low water mark	high water mark-1	$>$ previous occupancy	f_normal	decrease
high water mark	full		f_urgent	decrease

Figure 2. The Client's Flow Control Policy

between the low and high water marks, the client adjusts the transmission rate according to the change in the buffers' occupancy: If the buffers contain more frames than they had contained when the previous flow control request was sent, the client requests to decrease the transmission rate, and vice versa. If the buffer occupancy is the same, no request is emitted.

When the buffer occupancy is between the high and low water marks, flow control messages are sent at a relatively small frequency, *f_normal*. When the buffer occupancy is not between the high and low water marks, the flow control messages are more urgent, and are therefore sent at a higher frequency, *f_urgent*. In our prototype, when the occupancy is between the low and high water marks flow control messages are sent every 8 received frames, and otherwise the frequency is doubled. In addition, when the client's buffer occupancy falls below a certain critical threshold, the client sends an *emergency* request. The handling of emergency requests (by the server) is described in Section 4.1. The client's flow control policy is summarized in Figure 2.

4.1. Handling Emergency Situations

When the buffer occupancy falls below a critical threshold, the emergency mechanism kicks in. Such a situation typically occurs when the client migrates to another server (due to a server failure or load balancing) and also at startup time and when the client requests random access to a different part of the movie.

In such cases, the client sends an emergency request to the server. The server responds by temporarily increasing the transmission rate in order to re-fill the clients' buffers very quickly. In order to avoid overflowing the clients' buffers, the server does not persist with the high transmission rate for a long period. Instead, the additional transmitted bandwidth decays with time.

The number of frames per second transmitted to a client is the sum of the latest known transmission rate[2] plus an *emergency quantity*. The emergency quantity decays by a certain percentage every second. While the emergency quantity is greater than zero, the server ignores all flow control requests from the client.

[2] A default transmission rate is used at startup.

The base emergency quantity q and the decay factor $f \in (0, 1)$ are chosen so that the total number of additional frames desired is the sum of the decaying sequence (of values truncated to integers): $\sum_i q \cdot f^i$. There is a tradeoff involved in the selection of these parameters: When starting with a high base quantity q, the buffers fill up faster to allow coping with message re-ordering and additional emergencies smoothly. However, the risk of overflow is greater and for a few seconds additional transmission bandwidth consumption is very high. If QoS reservation mechanisms are used, this can be costly.

We experimented with different such sequences. In our prototype, we chose to increase the bandwidth consumption at emergency periods by no more than 40% of the mean bandwidth. Thus, for transmitting a 30 frames per second movie, we set the base emergency quantity q to 12. We use a decay factor f of 0.8, so the resulting sequence sum is 43 frames. Note that if the service were to use QoS reservation, e.g., over an ATM network, then it would need to reserve an additional *variable bit rate (VBR)* channel for emergency periods, varying to at most 40% of the *constant bit rate (CBR)* channel reserved for normal periods.

We further elaborated the emergency recovery mechanism to transmit a smaller emergency quantity at less serious emergency situations. We set two critical thresholds: If the client's buffer occupancy falls below 15%, the base emergency quantity is 12 frames, as explained above. If the buffer occupancy falls below 30% but not below 15%, the base quantity is set to 6 frames, and the resulting sequence sums up to 15 additional frames.

4.2. Choosing Buffer Sizes and Thresholds

The buffer size is chosen to account for irregularity periods occurring at emergency situations (migration due to server failure or load balancing). The flow control mechanism endeavors to keep the buffer occupancy always above the low water mark. Therefore, the low water mark should reflect the number of frames needed to account for irregularity periods. The duration of the irregularity period is at most the sum of the server synchronization skew and the take over time. In our prototype implementation, the server synchronization skew is half a second in the worst case. The take over time is affected by the failure detection time-out and by the time required for information exchange among the servers. In our tests on a local area network, the take over time was half a second on the average. Additional delay may be introduced by process scheduling since we do not use a real-time operating system.

We have chosen the buffer sizes to contain approximately 2.4 seconds of video, the low water mark to be 73% of the total buffer space and the high water mark to be 88% of the buffer space. Thus, when the buffers are full up to the low water mark, they account for an irregularity period of approximately 1.7 seconds. We tuned the gap between the low and high water marks to be large enough to allow the flow control algorithm to keep the buffer occupancy in this range, yet not larger than needed in order not to consume excess buffer space. Likewise, the margin between the high water mark and the top of the buffer is essential in order to avoid buffer overflow. Using a sophisticated mechanism for handling emergency requests allowed us to make this margin very small. All of these values are subject to fine tuning according to the specific run-time environment.

Note that our buffer sizes account for a single emergency situation. A second emergency situation can be handled smoothly only after the buffers are re-filled to contain sufficient frames. In order to guarantee smoothly coping with additional emergency situations occurring before the buffers start to re-fill, the buffer size should be enlarged. If there is not enough video material in the buffers to account for the duration of the irregularity period, the situation cannot be handled smoothly, i.e., some video material is delayed or skipped and a human observer can notice the jitter (usually during no more than a second).

4.3. Adjusting the Quality of the Video Material

Some clients' communication or computation capabilities may not allow for processing of high quality video, e.g., if they use a slow modem to communicate or if they do not have hardware video decoders. In such cases, the client may request lower quality video consisting of less frames per second. When such a request arrives, the server starts skipping frames, and transmits only the number of frames per second which suits the client's capabilities. This is done by transmitting all the I (full image) frames, and some of the other frames, as the capabilities allow.

5. Exploiting Group Communication

Our VoD service exploits a *group communication system (GCS)* [1]. The use of group communication simplifies achieving fault tolerance and dynamic load balancing and provides a convenient framework for the overall service design.

Group communication introduces the notion of group abstraction which allows processes to be easily arranged into multicast groups. Thus, a set of processes is handled as a single logical connection identified by a logical name. Within each group, the GCS provides reliable multicast and membership services. The reliable multicast services deliver messages to all the current members of the group. The *membership* of a group is the set of currently live and connected processes in the group. The task of the membership service is to maintain the membership of each group and to deliver the membership to the group members whenever it changes.

5.1. The Service Group Layout

Our service creates the following three kinds of multicast groups, as shown in Figure 3.

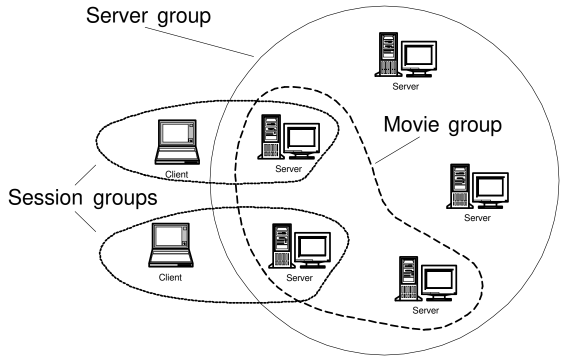


Figure 3. The group layout of the VoD service.

Server group consists of all the VoD servers. The client uses this group at startup in order to connect to the VoD service. The client communicates with the abstract server group and is therefore completely unaware of particular VoD server identities.

Movie group (per movie) consists of those VoD servers that have a copy of a particular movie. This group is used by the servers to consistently share information about clients that are currently watching this movie, for fault tolerance purposes (cf. Section 5.2 below).

Session group (per client) consists of the client watching a movie and the server that is currently communicating that movie to the client. The client uses this group to send control information to the VoD server.

5.2. Fault Tolerance and Dynamic Load Balancing

Let us consider what happens within a single movie group $G(M)$ corresponding to a movie M. Each member of $G(M)$ uses the reliable multicast service to periodically multicast to the other members of $G(M)$ information about its clients who are watching M. This information includes the offsets of its clients in the movie M and their current transmission rates: a total of a few dozens of bytes.

In our prototype implementation the servers multicast this information every half a second. Thus, the servers are kept synchronized within half a second with respect to the clients' positions in the movie, while the storage space and bandwidth required for this information is negligible w.r.t. the buffer space and bandwidth required for the video transmission.

Whenever the membership of $G(M)$ changes (e.g., as a result of a server crash or join), the members of $G(M)$ receive a notification of the new membership. Upon receiving this notification, the servers evenly re-distribute the clients among them. If the notification reflects a server failure, each remaining server in $G(M)$ uses its knowledge about all the clients in order to deterministically decide which clients it now has to serve. When new servers join, the servers first exchange information about clients, and then use it to deduce which clients each of them will serve.

In order to take over a client, a server simply joins the client's session group and resumes the video transmission starting from the offset and transmission rate that were last heard from the previous server.

5.3. The Benefits of Using Group Communication

The use of group communication greatly simplifies the service design. In particular, it provides the following advantages:

1. The **group abstraction** simplifies connection establishment and allows for transparent migration of clients while maintaining a simple client design. The clients are oblivious to the number and identities of the servers providing the service.

2. The **membership service** detects conditions for client migration, both for re-distributing the load, and for achieving fault tolerance.

3. The **reliable group multicast semantics** facilitates information sharing among the servers, in order to allow them to consistently agree about client migration.

4. Using **reliable multicast**, we guarantee that client control messages will reach the servers.

Our VoD service prototype was implemented using the Transis group communication system. The server was implemented in C++, using only around 2500 lines of code. The client was implemented in C, using only around 4000 lines of code (excluding the GUI and the video display module). Without the Transis services, such an application would have been far more complicated, and the code size would have turned out significantly larger.

6. Performance Measurements

We implemented the VoD service using UDP/IP for video transmission. We used the Transis [2, 15] group communication system (running over UDP/IP) for membership and reliable messages. The servers run on PCs running BSDI UNIX. The video is stored and transmitted in MPEG [17] format. The clients run on Windows 95/NT; the video is decoded by the clients using Optibase hardware decoders. The performance measurements shown below were obtained with the following parameters: Approximately 1.4 Mbps, 30 frames per second MPEG movie; allocated software buffers for 37 frames; 204 KB hardware buffers (approximately 1.2 seconds of video); the servers synchronize their states every 1/2 second.

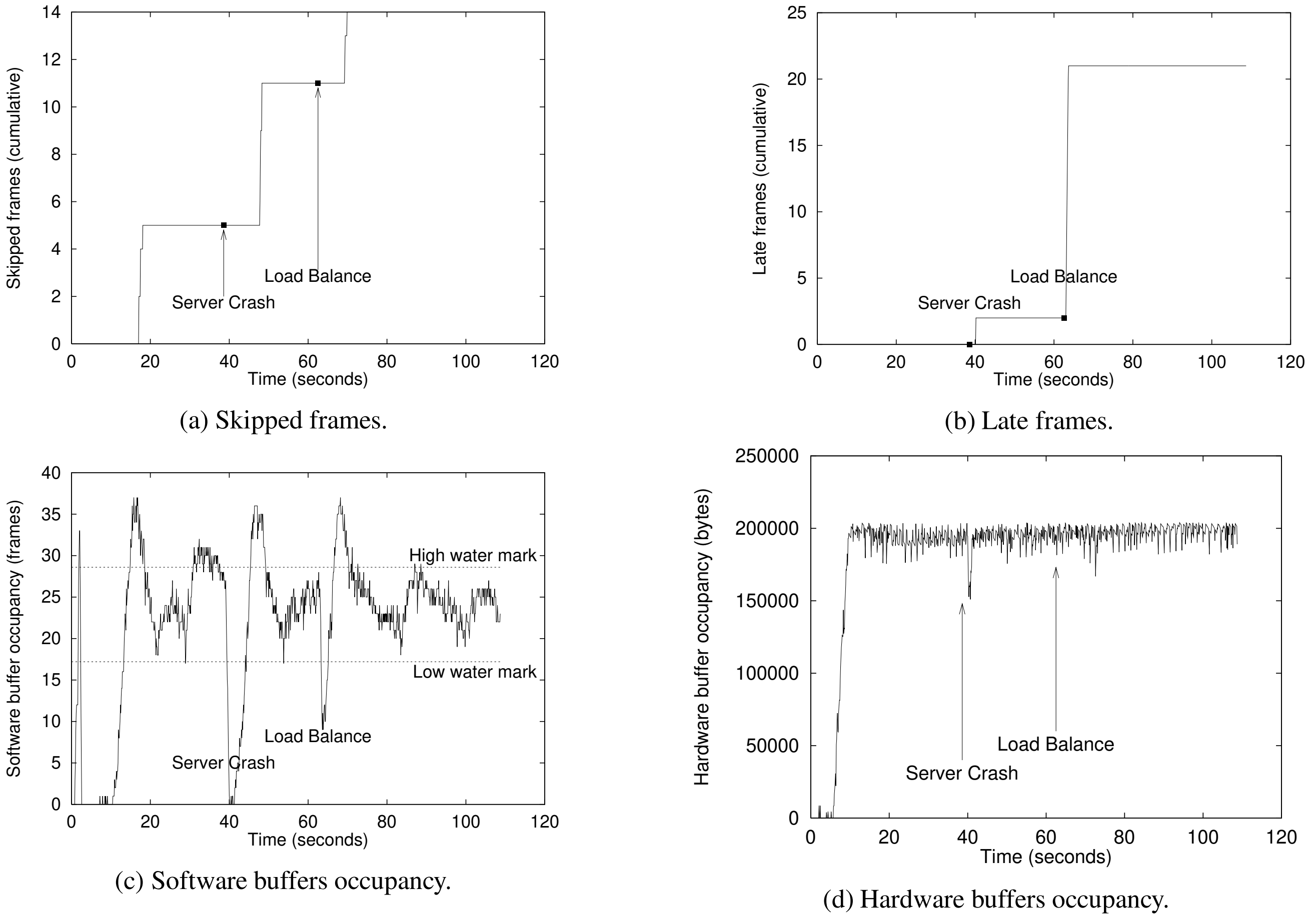
6.1. Performance Measurements in a LAN

Below, we present typical performance measurements obtained while testing the VoD service on a 10 Mbps switched Ethernet. The measurements were collected by a VoD client watching a movie in the following scenario: Approximately 38 seconds after the movie began, the server transmitting this movie was terminated and the client was migrated to another server. Approximately 24 seconds later, a new server was brought up and the client was migrated to it for load balancing purposes.

6.1.1. Overcoming the Irregularity of Video Transmission

Figure 4(a) depicts the cumulative number of frames that were skipped (i.e., not displayed to the user) as a function of time. Running on a LAN, we did not encounter message loss, and frames were discarded only due to buffer overflow occurring during recovery from emergency situations[3]. Figure 4(a) shows that no more than six frames were skipped following each emergency period (at startup, server failure, and migration due to load balancing). Due to our policy not to discard I (full image) frames in cases of buffer overflow, none of the skipped frames was an I frame. The frame loss caused a slight transient degradation of the video image that

[3] Note the correlation between skipped frames and the peak software buffer occupancy (depicted in Figure 4(c)).



(a) Skipped frames.

(b) Late frames.

(c) Software buffers occupancy.

(d) Hardware buffers occupancy.

Figure 4. Overcoming the irregularity of video transmission in a LAN.

lasted less than a second; this degradation was not noticeable to a human observer.

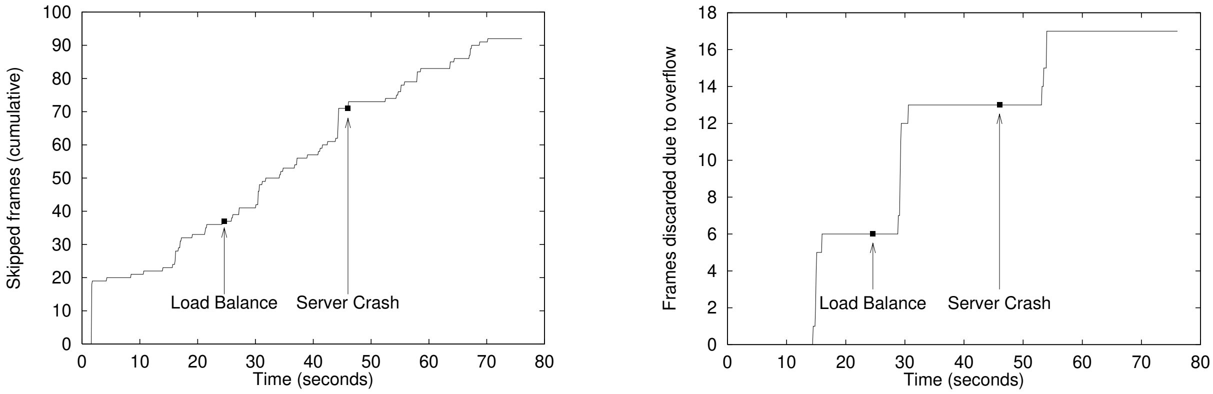
Figure 4(b) shows the cumulative number of late frames (i.e., frames that were discarded because they arrived after they should have been displayed). Running on a LAN, messages do not arrive out of order, and the only late arriving frames are those that arrive twice[4] at migration times. Since the servers are not perfectly synchronized, when a client migrates from one server to another certain frames may be transmitted by both servers. This occurs since we take a conservative (pessimistic) approach, preferring duplicate transmission of frames over missed frames.

Note that a different behavior occurs in case of a server failure than in case of migration due to load balancing. When a server fails, there is a longer intermission in the transmission since failure detection takes time. Therefore, at such times, buffer occupancy drops lower (please see below). At load balance time, on the other hand, the new server starts transmitting video material approximately at the same time that the old server stops transmitting. Due to discrepancy between the servers, some frames are transmitted twice, as observed in the late frames graph (Figure 4(b)).

[4] A duplicate frame is considered late.

6.1.2. Buffer Occupancy

The occupancy of the client's buffers as a function of time is displayed in Figures 4(c) and 4(d). Figure 4(c) shows that the software buffers reach their mean occupancy (around 23 frames) after approximately 14 seconds. While no emergency events occur, the buffer occupancy oscillates between the low and high water marks. The software buffer occupancy drops to zero when the client is migrated due to a server failure, and drops to approximately $1/4$ of its capacity when the client is migrated for load balancing purposes. The buffers are re-filled quickly, and therefore buffer overflow occurs following recovery from emergency periods. Figure 4(d) shows that the hardware buffers fill up approximately 10 seconds after the first frame of the movie arrives the client. The hardware buffer occupancy drops to approximately $3/4$ of its capacity following server crash.



(a) Total number of skipped frames.

(b) Frames discarded due to buffer overflow.

Figure 5. Skipped frames in a WAN.

6.2. Measurements in a Small Scale WAN

We have tested our VoD service between the Hebrew and Tel Aviv Universities, which are seven hops apart on the Internet. We used UDP/IP without any QoS reservation mechanisms. The measurements below were collected by a VoD client watching a movie. Approximately 25 seconds after the movie began, a new server was brought up and the client was migrated to it for load balancing purposes. Approximately 22 seconds later, the server transmitting this movie was terminated and the client was migrated to another server.

Figure 5(a) depicts the cumulative number of frames that were skipped (i.e., not displayed to the user) as a function of time. As one can observe, when running on the Internet without reservation mechanisms, a certain percentage of the messages are lost. Therefore, the quality of displayed video is inferior to the quality observed on a LAN. At irregularity periods additional frames are skipped due to buffer overflow. This is demonstrated in Figure 5(b), which depicts the cumulative number of frames that were discarded due to buffer overflow.

The client's buffer occupancy and number of late frames observed on a WAN exhibit similar behavior to that observed on a LAN. Due to lack of space, we do not include the graphs here.

7. Related Work

Current research in the area of VoD often focuses either on improving the performance of a single server [11, 14, 19, 20, 21], or on parallel servers with dedicated hardware [16, 18]. The improved performance of a single server is achieved by techniques such as sophisticated file organization [11, 19, 20], novel QoS aware disk scheduling algorithms [14, 20, 21], data fault tolerance [11, 19, 20] and admission control and resource (e.g., buffers) reservations [14, 19, 20] ([14] deals also with network QoS).

Current research rarely addresses the issue of smooth provision of service in the presence of server and communication failures. The only exception that we are aware of is the Microsoft Tiger [12, 13] video file service which is highly scalable. Tiger uses striping of movies across several servers.

The Tiger architecture differs from ours in that it assumes that the set of servers is tightly coupled and connected via a fast communication network. In their architecture, multiple servers serve the same clients. A sophisticated scheduler is utilized to synchronize the servers. In our architecture, each client is served by one server at a given time and the servers can be geographically apart.

Using Tiger, a special reconfiguration process needs to be executed when a new server or a new movie is added, in order to re-stripe the movies. With our service, a new server can be brought up without any special preparations, and new movies can be added "on the fly" by storing them on machines where servers are running.

The Tiger system smoothly tolerates the failure of one server, but not necessarily two failures even if the failures are not concurrent, and even if the total number of servers is very large. In contrast, our VoD service does not set a hard limit on the number of server failures tolerated. If a movie is replicated k times, then up to $k - 1$ failures are tolerated.

8. Conclusions and Future Work

We have presented a fault tolerant video on demand service which is provided by multiple servers. When a server crashes it is replaced by another server in a transparent way. The clients are unaware of the change in the service provider. New servers may be brought up "on the fly" to alleviate the load on other servers. In test runs of our implementation, such transitions are not noticeable to a human observer who uses the service. The concepts demonstrated in this work are general, and may be exploited to construct a variety of highly available servers.

In our current implementation, the video material is transmitted using UDP/IP. Connection establishment, control and sharing of state among the servers are performed using the services of the Transis group communication system, which also runs over UDP/IP. The use of group communication greatly simplifies the service design.

We intend to port and test the VoD service over ATM networks: The video material will be transmitted via native ATM UNI 3.1 [8] or UNI 4.0 [9] connections. We intend to continue using group communication for connection establishment, control, and sharing of state. We will use a GCS geared to WAN, based on the ideas in [3, 4, 5]. This GCS will either run over classical UDP/IP with LAN emulation over ATM (LANE) [7], or directly over native ATM.

Acknowledgments

We thank Gregory Chockler for his contribution to the preliminary version of the VoD service, and for his helpful comments and suggestions regarding the presentation style of this paper.

References

[1] ACM. *Commun. ACM 39(4), special issue on Group Communications Systems*, April 1996.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. In *22nd IEEE Fault-Tolerant Computing Symposium (FTCS)*, July 1992.

[3] T. Anker, D. Breitgand, D. Dolev, and Z. Levy. CONGRESS: CONnection-oriented Group-address RESolution Service. Tech. Report CS96-23, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, December 1996. Available from: http://www.cs.huji.ac.il/~transis.

[4] T. Anker, D. Breitgand, D. Dolev, and Z. Levy. CONGRESS: Connection-oriented group-address resolution service. In *Proceedings of SPIE on Broadband Networking Technologies*, November 2-3 1997.

[5] T. Anker, G. Chockler, D. Dolev, and I. Keidar. Scalable group membership services for novel applications. In M. Mavronicolas, M. Merritt, and N. Shavit, editors, *Networks in Distributed Computing (DIMACS workshop)*, volume 45 of *DIMACS*, pages 23–42. American Mathematical Society, 1998.

[6] T. Anker, G. Chockler, I. Keidar, M. Rozman, and J. Wexler. Exploiting group communication for highly available video-on-demand services. In *Proceedings of the IEEE 13th International Conference on Advanced Science and Technology (ICAST 97) and the 2nd International Conference on Multimedia Information Systems (ICMIS 97)*, pages 265–270, April 1997.

[7] The ATM Forum. *LAN Emulation Over ATM Specification - Version 1.0*, February 1995.

[8] The ATM Forum Technical Committee. *ATM User Network Interface (UNI) Specification Version 3.1*, June 1995. ISBN 0-13-393828-X.

[9] The ATM Forum Technical Committee. *ATM User-Network Interface (UNI) Signalling Specification Version 4.0, af-sig-0061.000*, July 1996.

[10] The ATM Forum Technical Committee. *Audiovisual Multimedia Services: Video on Demand Specification 1.0, af-saa-0049.000*, January 1996.

[11] S. Berson, L. Golubchik, and R. R. Muntz. Fault tolerant design of multimedia servers. In *ACM SIGMOD International Symposium on Management of Data*, pages 364–375, May 1995.

[12] W. J. Bolosky, J. S. Barrera, R. P. Draves, R. P. Fitzgerald, G. A. Gibson, M. B. Jones, S. P. Levi, N. P. Myhrvold, and R. F. Rashid. The Tiger video fileserver. In *Proceedings of the Sixth Inernational Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, April 1996.

[13] W. J. Bolosky, R. P. Fitzgerald, and J. R. Douceur. Distributed schedule management in the Tiger video fileserver. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 212–223, October 1997.

[14] T. cker Chiueh, C. Venkatramani, and M. Vernick. Design and implementation of the stony brook video server. In *Software Practice and Experience*. To Appear.

[15] D. Dolev and D. Malkhi. The Transis approach to high availability cluster communication. *Commun. ACM*, 39(4), April 1996.

[16] R. Haskin and F. Schmuck. The Tiger Shark file system. In *Proceedings of IEEE Spring COMPCON*, Feb. 1996.

[17] ISO/IEC 13818 and ISO/IEC 11172. *The MPEG Specification*. http://www.mpeg2.de/.

[18] J. Y. B. Lee. Parallel video servers: A tutorial. *IEEE MultiMedia special issue on Video Based Application*, 5(2), April – June 1998.

[19] C. Martin, P. S. Narayan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini Multimedia Storage and Management. Kluwer Academic. To appear.

[20] P. Shenoy, P. Goyal, S. Rao, and H. Vin. Design and implementation of symphony: An integrated multimedia file system. In *ACM/SPIE Multimedia Computing and Networking (MMCN'98)*, pages 124–138, January 1998.

[21] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-a disk array management system for video files. In *ACM Multimedia*, pages 393–400, August 2–6 1993.

[22] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. In *IEEE Network*, September 1993. The RSVP Project home page: http://www.isi.edu/div7/rsvp/rsvp.html.