Scalable live video streaming to cooperative clients using time shifting and video patching

Meng Guo, Mostafa H. Ammar
{mguo, ammar}@cc.gatech.edu
Networking and Telecommunication Group
College of Computing, Georgia Institute of Technology

September 16, 2003

Abstract

We consider the problem of how to enable the streaming of live video content from a single server to a large number of clients. One recently proposed approach relies on the cooperation of the video clients in forming an application layer multicast tree over which the video is propagated. Video continuity is maintained as client departures disrupt the multicast tree, using multiple description coded (MDC) streams multicast over several application layer trees. While this maintains continuity, it can cause video quality fluctuation as clients depart and trees are reconstructed around them. In this paper we develop a scheme using the transmission of a single-description coded video over an application layer multicast tree formed by cooperative clients. Video continuity is maintained in spite of tree disruption caused by departing clients using a combination of two techniques: 1) providing time-shifted streams at the server and allowing clients that suffer service disconnection to join a video channel of the time-shifted stream, and 2) using video patching to allow a client to catch up with the progress of a video program. Simulation experiments demonstrate that our design can achieve uninterrupted service, while not compromising the video quality, at moderate cost.

1 Introduction

We consider the problem of how to enable *live streaming* of video content from a single server to a large number of clients. Due to the bandwidth-intensive nature of video streams, deploying scalable streaming service with acceptable quality has always been a challenge. The straightforward solution of setting up a connection for every single request is obviously not scalable. Native network layer IP multicast could be a good solution for scalable media streaming applications, but it is not widely deployed.

One recently proposed approach relies on the cooperation of the video clients in forming an overlay network over which the video is propagated. In this approach, a client currently in the overlay network forwards the content it is receiving, and serves other client's request as a server. By distributing the transmission load evenly to the clients all over the network, the video server is no longer the bottleneck. This approach is scalable in the sense that the forwarding capability of the overlay network is growing incrementally. New clients joining the network also bring in extra bandwidth capacity to the system.

The major problem for application layer multicast is the the video discontinuity caused by the dynamics of membership. Since the clients in the group can leave at any time, other clients which are receiving video content from them have to suffer service disconnection. Although the disconnected clients can resume the service by rejoining the application layer multicast tree, this process can take time and can result in loss of video content and interruption of video reception. To make things worse, unsatisfied clients leaving the group can become a positive feedback process, causing more clients to leave, which ultimately makes the streaming media service unacceptable. In CoopNet [11], video continuity is maintained using multiple description coded (MDC) streams multicast over several application layer trees. CoopNet employs a server-based centralized control protocol, the server is responsible to make sure that multiple trees for different descriptions are uncorrelated with each other. A single client's departure can only disconnect a subset of descriptions for its children.

While CoopNet maintains video continuity, it can cause video quality fluctuation as clients depart and trees are reconstructed around them[1]. Our solution tries to provide continuous streaming service without video quality fluctuation. In this paper we develop a scheme using the transmission of a *single description coded* video over an application layer multicast tree formed by cooperative clients. Clients in the tree always receive full quality video stream. Video continuity is maintained in spite of tree disruption caused by departing clients using a combination of two techniques: (1) providing time-shifted streams at the server and allowing clients that suffer service disconnection to join a video channel of the time-shifted stream, and (2) using video patching to allow a client to catch up with the progress of the live video program. We also need a buffering scheme that allows the client to store the video packets, and to playout when needed.

In our design, the client buffers the initial part of the video content for a certain time before play out. If the client is disconnected from the multicast tree, it can play out full quality video from this buffer while reconnecting to the group. The client rejoins the group by connecting to both the original stream and a time-shifted stream. The time-shifted stream (patching stream) is used to retrieve the missed video. The original stream is used to catch up with the progress of the live video. Received video packets that are not being played out are stored at the buffer for future playout. When the missed portion of the video content is fixed, the time-shifted stream is released, and the client receives from the original stream only. The use of video patching requires that a client should have enough bandwidth for two video streams: the original stream, and the time-shifted stream.

This paper is organized as follows. We give an overview of our design in Section 2. In Section 3, we describe our design in detail. We give quantitive analysis on the requirement of the clients in section 4. Then, in Section 5 we set up the simulation environment, and show some sample results of our performance evaluation experiments. Finally, we conclude this paper in Section 5.

2 Design Overview

In this section, we first give a general description of how our system operates. Then, we discuss the potential problem of the straightforward time shifting solution. We then apply video patching in live streaming service, and describe how the design goals are met. Finally, we give an example to illustrate the

[1]Recent work in CoopNet shows the PSNR variation with different number of descriptions [12]

system operations.

2.1 Basic Operations

Our design of the system is composed of three components: 1) a time shifting video server, 2) a level-based tree management protocol, and 3) a video stream patching scheme.

A time shifting video server S broadcasts video program in C channels. Each channel can be used to transmit one video stream. There is an application layer multicast tree associated with each channel. The server serves the clients with the original stream, and m time-shifted streams. We label these streams $s_0, s_1, \cdots, s_m$. Stream s_0 is the original stream, while s_i starts after a $i \times d$ delay. Video server is the single source of the video content, and is the root of the application layer multicast tree. It processes client requests to join, leave, and rejoin the multicast group, and is responsible for maintaining the topological structure and resource availability of the multicast tree.

When a client first joins the multicast group, it always joins a multicast tree of the original stream. If the server has free video channel available, the client connects to the server directly. Otherwise, the client joins the tree by connecting to a client already in the tree who has enough available bandwidth resources, while at the same time, has the shortest overlay path to the video server. This node join protocol guarantees that the clients in the upper level of the tree are fully loaded, before the clients in the lower level of the tree start to adopt new clients as their children. In this way, we can get a “well-shaped” wide and short multicast tree. A wide and short tree can achieve lower bandwidth consumption, and can reduce the probability of service disconnection due to ancestor node failure.

A client in the multicast tree suffers service disconnection in two cases: 1) upstream link congestion, or 2) an ancestor node’s failure. Similar to CoopNet, to detect service disconnection, the client sets a threshold value, if the packet loss rate is above the threshold, the client deems it as a service disconnection. For the case of ancestor node failure, the client detects 100% packet loss. The client manages to rejoin the group by connecting itself to another parent node. The *node rejoin delay* for a client is the time interval between the moment when the client is disconnected and the moment when the client is reconnected. We denote the *node rejoin delay* for client c as r_c. A straightforward approach for lossless video reception is: when the client rejoins the tree, it can select to join the video channel of an appropriate time-shifted video stream, so that it will not miss any video content. For example, if at time t_0, the client is disconnected, and it manages to rejoin the group at $t_0 + r_c$, the appropriate stream should be the stream with $\lceil \frac{r_c}{d} \rceil$ delay. Clients that join the same video channel form an application layer multicast tree.

2.2 Indefinite Time Shifting

A client might experience multiple service disconnections during the video reception process. Figure 1 shows an example when client c suffers multiple disconnections. In this figure, the horizontal axis is the actual time, while the vertical axis is the video playback time. The server sends out video streams with equal time shifting interval d. The client experiences three service disconnections at time t_1, t_2, and t_3. *Viewing delay* means the delay between the playback time of the video stream that the client is watching and the playback time of the original stream. *Starving period* is the time interval when the client is not receiving any video. *Freezing period* refers to the time period when the client side play out is temporarily stopped.

The client joins the original stream at time 0. During $[0, t_1]$, the viewing delay is 0. At time t_1, it is disconnected from the application layer multicast tree. It manages to reconnect to the tree at time $t_1 + \Delta_1$, the missed video is $[t_1, t_1 + \Delta_1]$. The designated time-shifted stream is s_i, where $i = \lceil \frac{\Delta 1}{d} \rceil = 1$ (assume $\Delta_1 < d$). The client begins to receive from stream s_1 at time $t_1 + d$. The viewing delay during $[t_1, t_1 + d]$ increases from 0 to d. The client is disconnected again at time t_2, and rejoins the tree at $t_2 + \Delta_2$. Note that at this time, the missed video portion is $[t_2 - \lceil \frac{\Delta 1}{d} \rceil \times d, t_2 + \Delta_2]$. The designated time-shifted stream should be s_j, where $j = \lceil \frac{\Delta 1}{d} \rceil + \lceil \frac{\Delta 2}{d} \rceil = 2$. (again, we assume $\Delta_2 < d$). The client begins to receive from stream s_2 at $t_2 + d$, and the viewing delay is increased to $2 \times d$. Similar event occurs after the third disconnection. In a general case, if a client suffers n service disconnections, it has to switch to stream s_k, where $k = \sum_{i=1}^{n} \lceil \frac{\Delta_i}{d} \rceil$, and the client viewing delay will be $k \times d$. Here, Δ_i denotes the rejoin delay after the i_{th} disconnection.

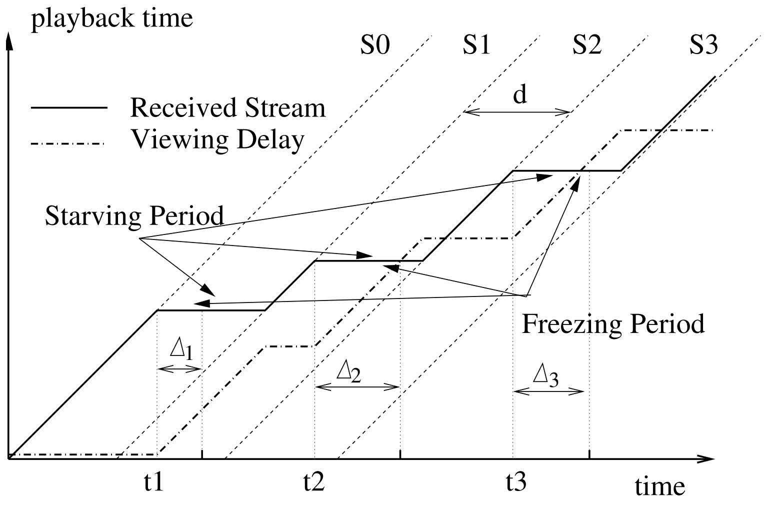


Figure 1: A sample for indefinite shifting

As shown in the Figure 1, each time the client rejoins the multicast tree, it has to join a stream with a larger time shifting value. This can result in *indefinite time shifting*, which is undesirable since the client's reception time could be much longer than the actual video program time. Also, if the time shifting value exceeds the boundary that the server can provide ($m \times d$), the client will suffer video content loss. Another problem demonstrated in this example is *video freezing* caused by *video starvation.* In the figure, the *starving period* and *freezing period* is same. Whenever the client is disconnected from the application layer multicast tree, the client side's playback is paused. In our design, we introduce *video patching* to tackle the *indefinite time shifting*; and use *initial buffering* to provide continuous video streaming.

2.3 Video Patching in Live Streaming

Video patching is a popular channel allocation technique in Video on Demand (VoD) service. It is designed to provide better server channel utilization and thus lower server load. In video patching, video channels of the VoD server are classified in two categories: regular channels and patching channels. Regular channels transmit the entire video stream, while patching channels only transmit the intial portion of a video as needed. When the server allocates a free channel to client requests, it first scans the on-going channels. If there is no regular channel distributing the requested video, or the starting time for this video is too early for the client to patch, this channel is allocated as a regular channel. Otherwise, the channel is allocated as a patching channel. Under video patching, the clients receive data from both the regular channel and the patching channel. The data from the patching channel is used to make up for the data the clients are missing from the regular channel. While receiving from the patching channel, the data being received from the regular channel is buffered by the clients for playout when needed. When the missing portion is completely received, the patching channel is released, and the clients only receive packets from the regular channel. More detailed description of video patching can be seen in [9].

In our scheme, the server sends out the original stream, as well as multiple time-shifted streams spaced by a fixed time interval d. The time-shifted stream serves as patching stream in our system. Here is how it works: when the connection to the tree is re-established after a service disconnection, a client would have missed the video from the point it is disconnected to the point it is reconnected. The reconnected client utilizes a time-shifted stream as patching stream. The patching stream is used to retrieve the missed video portion. At the same time, this client also receives the original stream, this stream is used to catch up the progress of the video program. During the patching period[2], the client is actually receiving at twice the speed as the normal stream rate. After the progress of the video is caught up, the patching stream is released, and the client receives only from the original stream.

There are several major differences between video patching in VoD service, and video patching in our scheme. 1) Different purposes: video patching in VoD service is used to reduce the access latency, while video patching in our scheme is used to provide lossless video reception. 2) Different starting time: video patching in VoD service starts at the begining of the service, in our scheme, video patching could happen at anytime during the video reception process. 3) Different releasing time: in VoD service, the patching channel is released when the video playback difference with another regular channel is fixed, while in our service, the patching channel is released when the missed video is retrieved.

We illustrate how the video patching scheme works, and how it eliminates *indefinite time shifting* through an example shown in Figure 2. Assume at time t_1, the client is disconnected. It rejoins the group by sending out two "rejoin" signals to the server, one is for the original stream, the other is for the patching stream. The client rejoins the original stream at time $t_1 + r_1$, the missed video portion is $[t_1, t_1 + r_1]$. It rejoins the patching stream at time $t_1 + \Delta_1$ (assume $\Delta_1 < d$), and begins to receive the patching stream s_1 at time $t_1 + d$. At time $t_1 + r_1 + d$, the missed video portion is made up, and the patching channel is released. Assume at time t_2, the client is disconnected again. At this time, the client is playing out $t_1 - d$, and its buffer stores the video portion of $[t_1 - d, t_1]$. The node rejoins the original stream at time $t_2 + r_2$, and the missed video is $[t_2, t_2 + r_2]$. It rejoins the patching stream at time $t_2 + \Delta_2$ (again, we assume $\Delta_2 < d$), and starts to receive patching stream at $t_2 + d$. During $[t_2, t_2 + d]$, the client is playing out from the buffer, the client viewing delay remains unchanged. When the client rejoins the tree, it receives the patching stream from s_1, instead of the other streams with longer time shifting values. In a general case, if a client suffers n disconnections, the client viewing delay is determined by the formula below:

[2]patching period denotes the time when a client is receiving both the original stream and the time-shifted stream

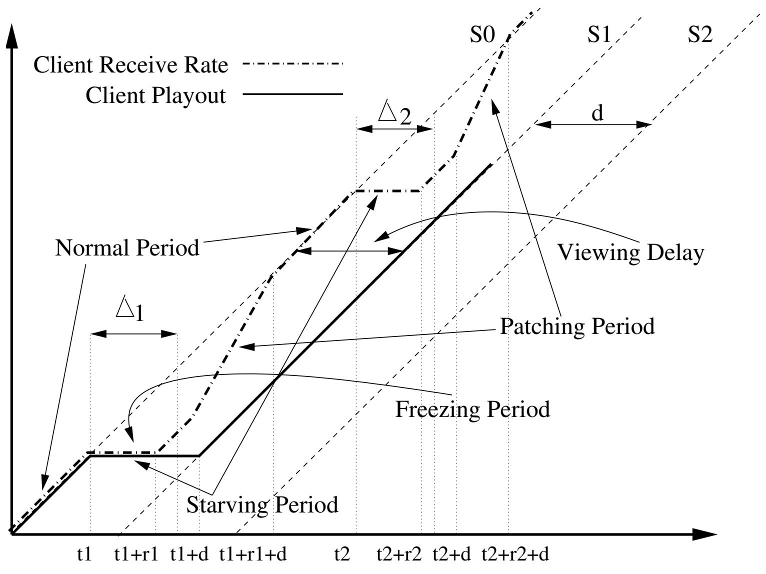


Figure 2: A sample for video patching

$$d_n = \begin{cases} \lceil \frac{\Delta_n}{d} \rceil \times d & n = 1 \\ d_{n-1} + \lceil \frac{max(\Delta_n - d_{n-1}, 0)}{d} \rceil \times d & n > 1 \end{cases}$$

Solving this formula, we get $d_n = max(\lceil \frac{\Delta_i}{d} \rceil) \times d$ ($i \in [1..n]$). Thus, the *indefinite time shifting* can be eliminated.

2.4 Continuous Video Streaming

As we stated in previous sections, the combination of a time shifting server and a video patching scheme can achieve lossless streaming, and prevent *indefinite time shifting*. But there are still some periods that the client's playback is halted, when the client is temporarily disconnected from the multicast tree.

To avoid interruption of play out during the *starving period*, buffered video accumulated during an initial playout delay can be used. In our design, when the client first joins the multicast tree, instead of immediately playing out the video stream, it can buffer the initial part of the video for a certain time. This time interval is called *initial access delay*. By waiting for an appropriate *initial access delay*, when there is a service disconnection in the future, the client can still playout the video from its buffer instead of stopping the playout.

We illustrate our solution in Figure 3. The client connects to the server at time 0, instead of playing out immediately, it buffers the initial D time unit video. At time D, it begins to play out the video from its buffer, while it keeps receiving from the original stream. As to how large this D should be, there is a tradeoff between access latency and video continuity. Obviously, longer access latency can result in smoother video reception. In the case of time shifting without video patching, the initial delay should be $D \geq \sum_{i=1}^{n} \lceil \frac{\Delta_i}{d} \rceil \times d$. If video patching is introduced, the initial delay can be reduced to $D \geq max(\lceil \frac{\Delta_i}{d} \rceil) \times d$ under non overlapping failures.

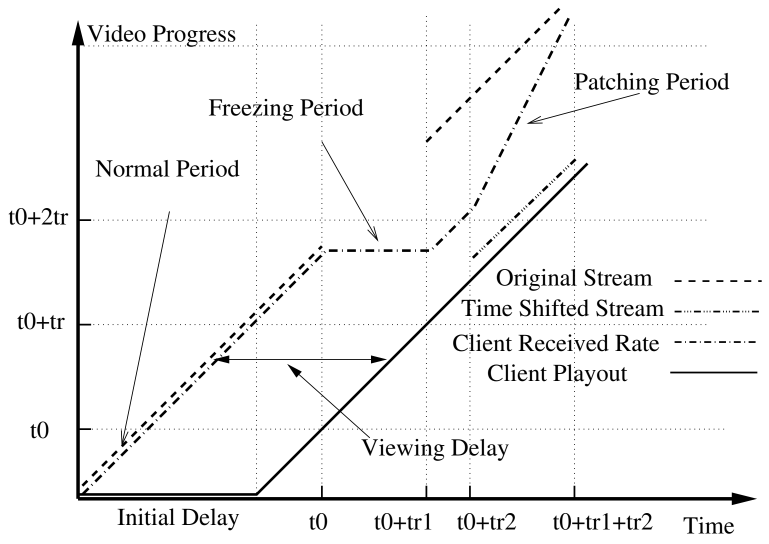


Figure 3: Continuous video streaming with initial delay

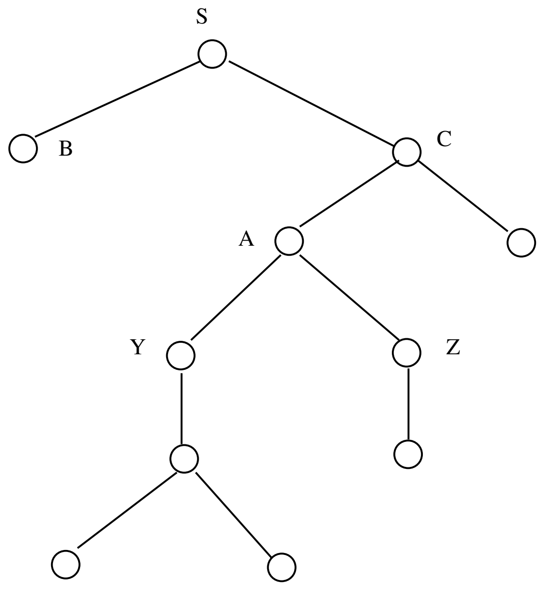
In Figure 3, at time t_0, the client is disconnected. Since the client buffer stores video $[t_0 - D, t_0]$, it can still play from the buffer while reconnecting to the server. If the initial delay D is larger than the *node rejoin delay*, the client buffer will not be drained out before it is reconnected to the server. Once the client is reconnected, it receives the time-shifted (patching) stream as well as the original stream, until the missed video is made up. During this process, the client does not experience any pause of playout, neither does it suffer any extra delay. If D is smaller than the *node rejoin delay*, then the client side video playback has to be paused.

2.5 An Example of System Operations

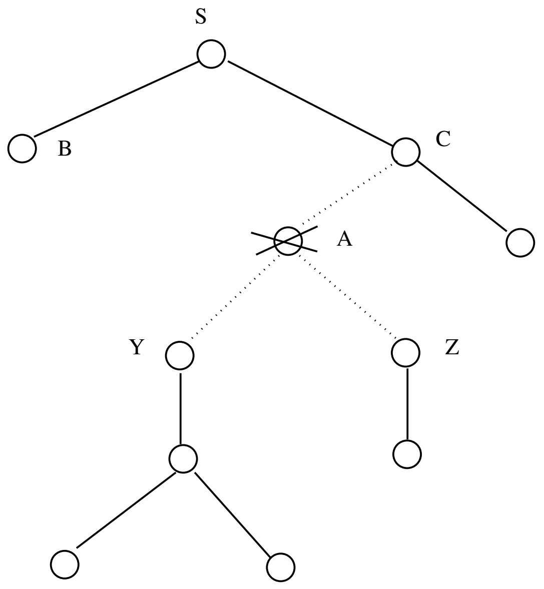
Figure 4 shows several snapshots of the application layer multicast tree during the video streaming process. Figure 4(a) shows the multicast tree where all the clients in the tree receive the original stream. At time t_0, node A leaves the group either due to congestion or node failure. This causes the subtrees rooted at nodes Y, and Z to suffer service disconnection. This is shown as Figure 4(b).

Nodes Y and Z send "rejoin" message to server S respectively. We describe the rejoin process for node Y, node Z has similar rejoin process. The "rejoin" message includes the rejoin request for both the original stream and the time-shifted stream. To reconnect the client to the original stream, server S selects a client which is currently in the multicast tree as the parent node of Y. In this case, client B is selected. Node Y connects with client B at time t_{Y1}, and begins to receive the original stream from it. At the same time, the server allocates a free video channel to send out the patching stream to Y. Assume the server starts to send out the patching stream at t_{Y2}, and stream s_i is the designated patching stream, where $i = \lceil \frac{t_{Y2}-t_0}{d} \rceil$. Figure 4(c) shows the multicast tree structure at time t_1, when both nodes Y and Z are in patching period. Figure 4(d) shows the tree structure when both patching channels for nodes Y and Z are released.

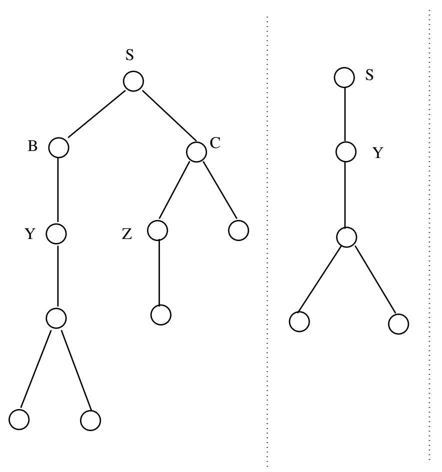
We analyze the influence of this service disconnection to node Y and its children. Node Y begins to receive the original stream at time t_{Y1}, it misses video $[t_0, t_{Y1}]$ due to service disconnection. The patching stream starts at t_{Y2}. Assume at time t_0, the client buffer has b_0 time unit of video content. When the



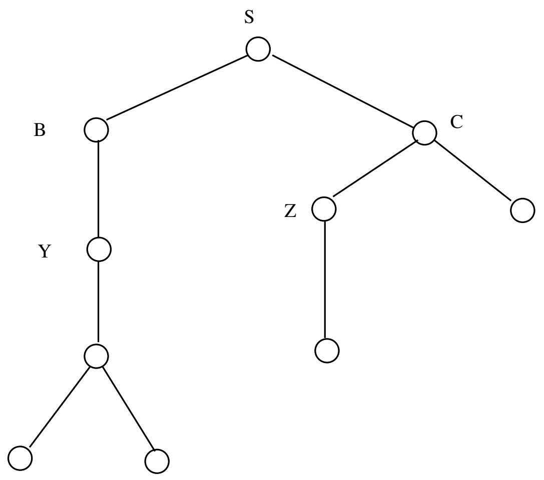
(a) the original tree



(b) node A disconnected



(c) patching structure



(d) new tree

Figure 4: The example of system operations

client is disconnected, node Y playout the video content from its buffer. If $b_0 \geq t_{Y2} - t_0$, then the client's viewing process is not interrupted before the patching stream begins to feed the client. Otherwise, the client may suffer a pause of $t_{Y2} - t_0 - b_0$. At time $t_0 + t_{Y2} + t_{Y1}$, the patching period finishes and the patching stream is released.

We analyze the influence of this service disconnection to node Y and its children. Similar analysis can be applied to node Z. Node Y begins to receive the original stream at time t_{Y1}, it misses video $[t_0, t_{Y1}]$ due to service disconnection. The patching stream starts at $t_0 + \lceil \frac{t_{Y2}-t_0}{d} \rceil \times d$. Assume at time t_0, the client buffer has b_0 time unit of video content. During the time when the client is disconnected, node Y playout the video content from its buffer. If $b_0 \geq \lceil \frac{t_{Y2}-t_0}{d} \rceil \times d$, then the client's viewing process is not interrupted when the patching stream begins to feed the client. Otherwise, the client's may suffer a pause of $\lceil \frac{t_{Y2}-t_0}{d} \rceil \times d - b_0$. At time $t_0 + \lceil \frac{t_{Y2}-t_0}{d} \rceil \times d + t_{Y1}$, the patching period finishes and the patching stream is released.

3 Design Details

3.1 Time Shifting Video Server

A media server is the single source for the live video content, and is the root of the application layer multicast tree. In this section, we describe the time shifting video server design in our system.

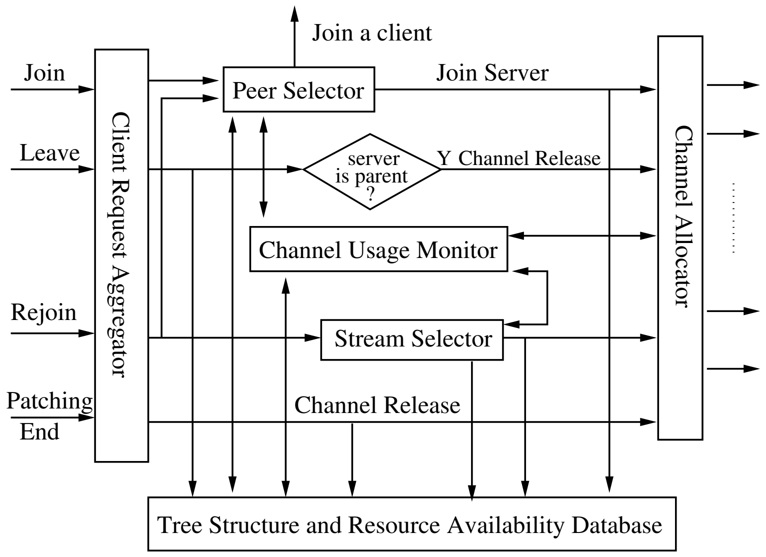


Figure 5: Video server Design

Figure 5 shows the structure of our server design. The *Client Request Aggregator* receives four kinds of client requests: *join, leave, rejoin*, and *patching end.* When the server receives a *join* request, it contacts the *Peer Selector* to find the appropriate parent node[3] to connect with. The *Peer Selector* obtains global topology and resource availability information from the centralized database. In the case of a graceful *leave*, the server signals the children of the leaving client to rejoin other parent nodes. The server also updates the *tree structure and resource availability* information upon a node leave. If a client experiences severe packet loss due to network congestion or ancestor node failure, it sends a *rejoin* message to the server. The server then assigns another parent node to forward the original stream to it. At the same time, the server also contacts the *Channel Usage Monitor*, and assigns a free video channel to transmit the time-shifted video stream to this client. The Stream Selector is responsible for assigning the client a stream with appropriate time shifting value. When the missed video portion has been fully received, it sends a *patching end* message to the server, the server then releases the corresponding patching channel.

3.2 Channel Allocation:

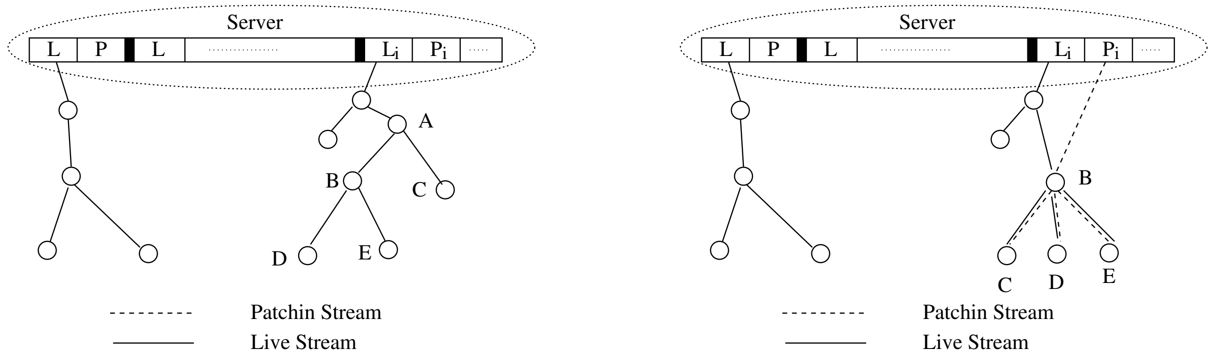
Channel allocation deals with whether the channel is used to transmit the original video stream or the time-shifted video stream. Our video server streams video content in two kinds of channels. The channel for the original stream is called *live channel*, and the channel for the time-shifted stream is called *patching channel.* The data rates for the live and the patching channels are the same. Video content that is transmitted on

[3]Parent node can be either the video server or a client which is currently in the application layer multicast tree

a particular channel is multicast to the clients in the application layer multicast tree associated with this channel. Allocating more channels to the original stream at the server leads to a shorter and wider tree. Reserving more channels for the time-shifted streams means that when the client's rejoin message reaches the server, there is a higher probability that the server has a free patching channel available. In this way, the client can start to receive the patching stream earlier.

3.2.1 1:1 Allocation

One way to allocate video channels is that whenever a server allocates one live channel to the client, it also reserves one patching channel for this client. We call this allocation scheme, *1:1 allocation.* This approach has the benefit that whenever there is a node failure in the application layer multicast tree of a live channel, there is a free patching channel available for the clients below the failed node to patch the missed video content.



(a)Tree Structure (b) Patching Structure

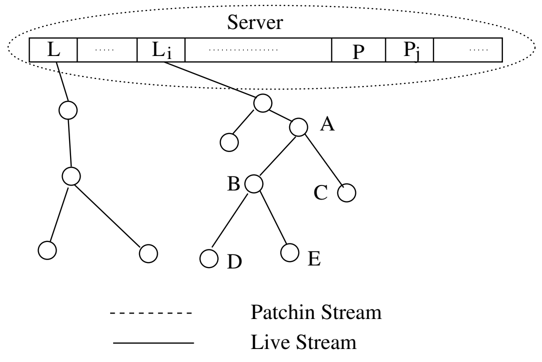
Figure 6: 1:1 channel allocation

Figure 6 shows an example of the *1:1* allocation. As shown in Figure 6(a), node A in the multicast tree of channel L_i leaves the group. The subtree of node A manages to rejoin the patching channel P_i for the time-shifted video stream, as well as rejoin the multicast tree for the original stream. The problem with this channel allocation scheme is that each patching channel is bound with a live channel. If at time t, there are more than one patching channel requests from live channel L_i, only one of them can be served, even though the patching channels for other live channels are left unused.

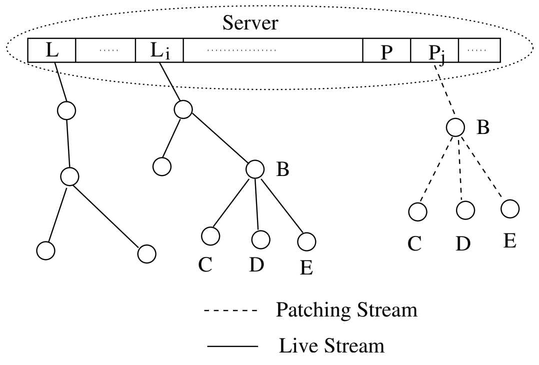
3.2.2 Static Multiplexing

To overcome the inefficiency of *1:1* allocation, we propose the *static multiplexing* allocation scheme. In this scheme, m of the C video channels are allocated as live channels, while the other $n = C - m$ channels are allocated as patching channels. There is no fixed binding in this scheme, the patching channel can be used to patch the disconnected clients from any live channel.

Figure 7 shows how this channel allocation scheme works. Node A of channel L_i leaves the group,



(a)Tree Structure



(b) Patching Structure

Figure 7: Static multiplex channel allocation

causing service disconnection for all its children. These nodes receives the patching stream by rejoining one of the free patching channel P_j. Obviously, this scheme is more efficient than the *1:1* scheme, since the video patching request will be served as long as there is a free patching channel available.

3.2.3 Dynamic Multiplexing with Channel Merging

In *static multiplexing* scheme, the value of m and n is pre-determined and fixed throughout the video transmission process. A more flexible scheme is to assign the number of live and patching channels dynamically. In this section, we propose a *lifetime-based allocation* scheme. The lifetime of an original stream is the remaining playback time of the video program. The lifetime of the patching stream is the duration of the patching period. For a video program of length T, at time t_0, the lifetime of the original stream is $T - t_0$. As to the patching stream, the life time is the time shifting value of this stream; the lifetime for stream s_k is $d \times k$. Our allocation algorithm works as follows: For the patching stream, at time t_0, the number of patching channels reserved is:

$$|P| = \lceil \frac{\sum_{i=1}^{m}(i \times d)}{T - t_0 + \sum_{i=1}^{m}(i \times d)} \times C \rceil$$

For the original stream, at time t_0, the number of video channels allocated is:

$$|L| = C - |P|$$

The number of patching channels $|P|$ is monotonically increasing as the video program proceeds. It is possible that when the server needs to allocate more patching channels, there is no free channel available. We introduce a channel merging scheme to deal with this problem. In this solution, when the server's request for more patching channels can not be satisfied, two live channels that have the fewest number of clients in their multicast trees are merged, and the freed channel is used as a patching channel. To merge two live channels, just connect the root node of the channel with fewest members to the highest possible level of the other channel.

3.3 Patching Stream Selection

The *Stream selector* in the video server determines which time-shifted stream should be used to patch the missed video portion for a certain client. When a client is disconnected from the group, it records the video playback time t_d, and sends a node rejoin request to the server. Assume when the server receives this node rejoin request, the live video playback time is t_1.

If the server has an available channel at this time, it estimates the connection set up time as $t_1 - t_d$. This is the time for the rejoin signal to travel from the client to the server. Thus, stream s_i is selected, while i is calculated by the formula below:

$$i = \lceil \frac{2 \times (t_1 - t_d)}{d} \rceil$$

If there is no available channel when the rejoin message reaches the server, this rejoin message will be held until a free patching channel is available. For example, at time t_2. Then the stream s_j should be selected, while j is determined by the formula below:

$$j = \lceil \frac{(t_2 - t_d) + (t_1 - t_d)}{d} \rceil$$

```
Join (C, S) {
1:  joined = false;
2:  Push the server IP address $S$ in current list $L_c$;
3:  while (joined==false and $L_c \neq \emptyset$ ) {
4:    repeat
5:      Probe the next IP address $p_n$ in $L_c$;
6:      if ($p_n$ has enough resource for new client) {
7:        $C$ join the tree and become $p_n$'s child;
8:        joined ==true;
9:        }
10:     Push the children of $p_n$ into $L_n$;
11:   until($L_c$ is visited or joined == true)
12:   $L_c = L_n$;
13:   }
14:  return joined;
15: }
```

Figure 8: Node Join Algorithm

3.4 Node join algorithm

A "well shaped" application layer multicast tree should be wide and short. A shorter multicast tree means a smaller number of overlay hops from a client to the server, thus a smaller average stretch. Stretch is the ratio of the latency along the overlay to the latency along the direct unicast path [2, 5] . The stretch of the application layer multicast tree is the average stretch value over all the clients in the tree. Furthermore , by reducing the number of intermediate clients, the chance that a client suffers service disconnection due to ancestor nodes leave is also reduced. In this section, we design a level-based scheme to handle client join. In this scheme, a newly arriving client joins the node whose overlay path distance to the server is shortest, and has enough available bandwidth resource to accomodate a new client. The join algorithm is shown in Figure 8:

```
$BWJoin(c, S, bw_c)$ {
01:    joined = false;
02:    Push the server IP address $S$ in current list $L_c$;
03:    **while** ($L_c \neq \emptyset$ ) {
04:      Push the children of members in $L_c$ into $L_n$.
05:      **repeat**
06:        probe the bandwidth of next host $p_n$ in $L_n$;
07:        **if** ($p_n$'s bandwidth $< bw_c$)
08:          **if** (Join_Level($L_c$) == true) **return**(true);
09:        **else**
10:          $c$ becomes $p_m$'s parent's child;
11:          $p_m$ becomes the child of $c$;
12:          **return**(true);
13:        **if** ($p_n$'s bandwidth == $bw_c$)
14:          **if** (Join_Level($L_c$)==true) **return**(true);
16:          **else**
17:            **if** (Join_Level($L_n$)== true) **return**(true);
19:      **until**($L_c$ is visited)
20:      **if** ($L_n = \emptyset$)
21:          **if** (Join_Level($L_c$)==true) **return**(true);
23:      $L_c = L_n$;
24:      }
25:    **return** joined;
26: }
```

Figure 9: Bandwidth First Node Join Algorithm

The patching scheme in our design requires extra bandwidth for each client, we now analyze how this scheme influences the overall structure of the tree. Assume the bandwidth of a video stream is b, and the average bandwidth of each host is $k \times b$. The height of a tree with N hosts should be $h_1 \approx log_k N$. Because of the use of the video patching scheme, each host has to allocate twice the bandwidth of a video stream: one for the original stream, the other is for the patching stream. Thus, the height of the tree is $h_2 \approx log_{\frac{k}{2}} N$. We now have:

```
Join_Level($L_c$) {
1:    repeat
2:      Probe the next host $p_a$ in $L_c$;
3:      if($p_a$ has enough bandwidth)
4:          c join the tree and become $p_a$'s child;
5:          return true;
6:    until ($L_c$ is visited)
7:    return(false);
8: }
```

Figure 10: Level Join Algorithm

$$\frac{h_2}{h_1} \approx \frac{log_{\frac{k}{2}} N}{log_k N} = \frac{logk}{logk - 1}$$

To further reduce the height of the tree, we design a *high-bandwidth-first* tree join algorithm. The key idea is to push the client with larger bandwidth up to the higher level of the tree. The multicast tree built by this scheme has the feature that clients in the lower level of the tree do not have more bandwidth capacity than the clients in the upper level of the tree. Figure 9 and Figure 10 shows the *high-bandwidth-first* join algorithm.

3.5 Node Leave

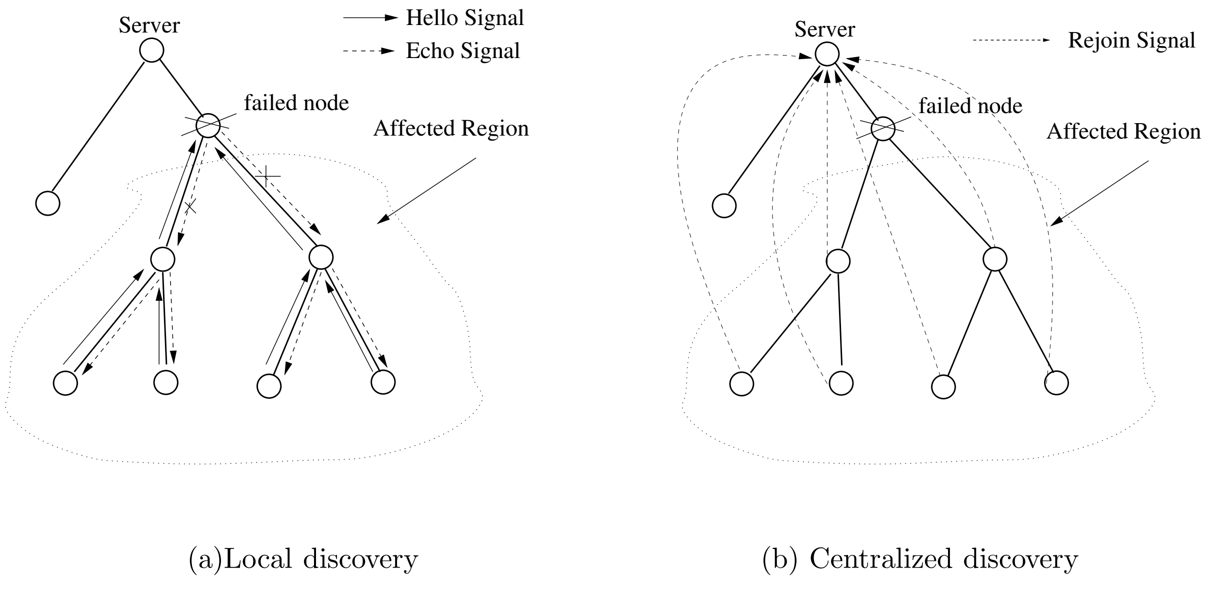


Figure 11: Failed node discovery process

Clients in the application layer multicast tree may leave the group at any time. A leaf node leaving the group does not have much impact on the application layer multicast tree. An internal node leaving the group, on the other hand, will cause service disconnection for its children. There are two kinds of leave: graceful leave and node failure. With a graceful leave, the client first informs the server and its children. Its children then manage to rejoin the tree by connecting to other parent nodes. It leaves the tree after the adaptation of the tree is finished. In the case of a node failure, the client leaves the group without informing any other hosts. The tree recovery operation due to a node failure is a two step process: First, the failed node and affected region detection.[4] Second, disconnected nodes rejoin the tree, this step follows the same operation as the rejoin process under a graceful node leave.

```
**Leave** $(l, S)$ {
    **if** $l$ is a leaf node, **return**;
    Store the address of $l$'s children in $L_c$.
    **repeat**
        Probe the next IP address $p_n$ in $L_c$;
        **if** ($p_n$ can accommodate all other hosts in $L_c$)
          {
          $p_n$ connects as $l$'s parent's child;
          All other hosts in $L_c$ rejoin $p_n$.;
          **return**;
          }
    **until**($L_c$ is visited)
    **repeat**
        Probe the next IP address $p_n$ in $L_c$;
        Join($p_n$, S);
    **until**($L_c$ is visited)
}
```

Figure 12: Node Leave Recovery Algorithm

There are two approaches to discover the failed node and the affected region, as shown in Figure 11. First approach is through localized detection. A client that detects a heavy packet loss sends a *hello* message to its parent node. If the parent node is experiencing the same problem, it sends an *echo* message back to the sender, and sends another *hello* message to its parent. If the parent node does not suffer packet loss, then it is the link congestion between the child and the parent causing the problem. This process repeats until a client does not receive *echo* message from its parent, then either this parent node is failed or the link between this client and its parent is disconnected. This approach does not require global topology knowledge, and detection does not exert extra load on the media server. The problem of this approach is that it might be slow in detecting the affected region. The other approach is to use the central video server. The video server maintains the topology of the application layer multicast tree. Each client that suffers service disconnection reports to the server, the server then figures out which node or link is failed, and the corresponding affected region.

[4]Affected region of a client c denotes the set of all nodes that are disconnected due to node failure of c.

4 Client side requirement analysis

Video server sends out $m+1$ streams, named stream $0, 1, \ldots, m$. The inter-stream delay among these m streams is d time units. The video serving time is T, and $T >> m \times d$. During a client's viewing process, we assume its service will be interrupted n times due to some ancestor nodes leave the group. Each time, the repair time is $r_i (i = 1, \ldots, n)$.

We are interested in the maximum buffer requirement at the client, and the maximum viewing delay the client will experience.

4.1 Non-Consecutive Failure Analysis

In this section, we analyze the client viewing behavior and resource requirement under the case of non-consecutive node failures. Non-consecutive failure means that, for a client, the next service disconnection does not happen until the client is recovered from previous node failure, and the missing video is fixed up.

During a client's viewing process, we assume its service will be interrupted n times. Each time, the node repair time (or rejoin time) is $r_i (i = 1, \ldots, n)$. We are interested in the following aspects: (1) What is the maximum buffer requirement for the client? (2) What is the overall delay the client will experience? (3) What's the actual client viewing time due to service disconnection? (4) What is the duration and frequency of the freezing period? We draw the following conclusion:

Conclusion *The overall client side viewing delay, and the maximum client side buffer requirement under non-consecutive loss is* $max(\lceil \frac{r_j}{d} \rceil \times d)$, $j = 1, 2, \cdots, n$

Proof:

We prove by induction.

i) For $n = 1$, which means there is only one service disconnection during the viewing process. Assume the client disconnects at time t, and rejoins at time $t + r_1$. From time $t + r_1$, the client begin to receive the original stream, and at time $t + \lceil \frac{r_1}{d} \rceil \times d$, the patching stream begins to serve the client. Thus, the client experience a delay of lceil $\lceil \frac{r_1}{d} \rceil \times d$, and the buffer usage is also $\lceil \frac{r_1}{d} \rceil \times d$. Our conclusion is true for $n = 1$.

ii) Assume the conclusion is true for the case of n,

iii) Let's consider the case of $n+1$. Assume the overall delay for first n disconnections, as well as the maximum buffer usage is $\lceil \frac{r_x}{d} \rceil \times d$. When the client is disconnected from the tree at time t for the $n + 1_{th}$ time. The repair time for this service disconnection is r_y, and the selected stream has a delay of $\lceil \frac{r_y}{d} \rceil \times d$. Three cases should be considered according to the relationship between r_x, and r_y, they are shown in Figure 13. We denote $k_x = \lceil \frac{r_x}{d} \rceil$, and $k_y = \lceil \frac{r_y}{d} \rceil$.

• *case 1:* $k_y \times d \leq k_x \times d$. At $t_y + r_y$, original stream begin to feed the client's buffer. Stream k_y is selected as the patching stream. At time $t_y + k_y \times d$, the client begins to receive the patching stream from server. Note that at time $t_y + k_y \times d$, the client is still playing from the buffer, so there is no extra delay introduced, and there is no freezing period during this time interval. At time $t_y + k_y \times d + r_y$, the missed portion is fixed. The buffer usage is: $k_x \times d - r_y + 2 \times r_y - r_y = k_x \times d$.

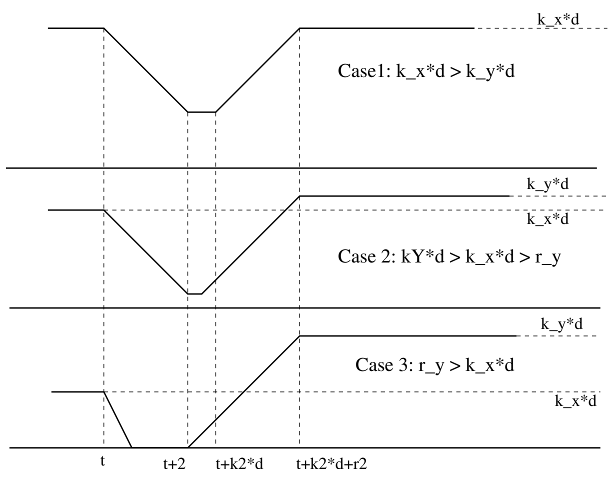


Figure 13: Client Side Buffer Size Under General Case

- *case 2:* $k_y \times d \geq k_x \times d \geq r_y$. At $t_y + r_y$, client is playing from the buffer. Original stream begin to feed the client buffer. At time $t_y + k_y \times d$, the client begin to receive the patching stream from server. The client buffer becomes empty before $t_y + k_y \times d$. Thus, the client suffer an extra delay of $k_y \times d - k_x \times d$. And the freezing period is $k_y \times d - k_x \times d$. The patching stream is played out while the original stream is buffered at the client. At time $t_y + k_y \times d + r_y$, the missed portion is fixed. The overall delay at the client is: $k_x \times d + k_y \times d - k_x \times d = k_y \times d$ latency. The buffer size used is $k_y \times d$.

- *case 3:* $r_y \geq k_x \times d$. At $t_y + r_y$, client buffer is empty. Original stream begin to feed the client buffer. At time $t_y + k_y \times d$, the client begin to receive the patching stream from server. The patching stream is played out while the original stream is buffered at the client. At time $t_y + k_y \times d + r_y$, the missed portion is fixed. The client suffer and extra delay of $k_y \times d - k_x \times d$, as well as the freezing period. The overall latency is $k_x \times d + k_y \times d - k_x \times d = k_y \times d$ The buffer size used is $k_y \times d$.

Thus, our conclusion is true for the case of $n + 1$.

4.2 Analysis on Consecutive losses

Consecutive loss means the client is disconnected again before the patching period finishes. Figure 14 shows different stages for a client to rejoin the tree after disconnection. Next disconnection can happen at any time after t_1. For the case of non-consecutive loss, next disconnection happens after $t_1 + k_1 \times d + r_1$, The case where next disconnection happens during $[t_1 + r_1, t_1 + k_1 \times d + r_1]$ is consecutive loss.

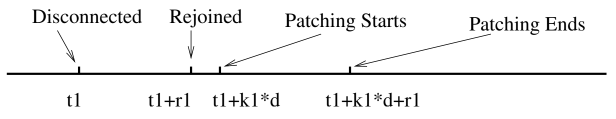


Figure 14: Stages of Client Recovery Process

The worst case scenario is every time whenever the client rejoined the tree, it is disconnected. In this case, the overall delay for the client will be $\sum_{i=1}^{n} r_i$. To achieve lossless transmission, the client should have a buffer size of $\sum_{i=1}^{n} r_i$.

In this section, we briefly analyze the buffer requirement under different node failure pattern.

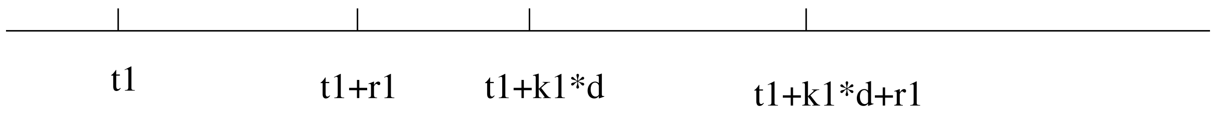


Figure 15: Consecutive Loss

As shown in Figure 15, the second node failure happens at time t_2:

i. $t_2 < t_1 + r_1$: this means that the parent node leaves the group before the node rejoins the tree, this is impossible. Since we do not have a parent node yet.

ii. $t_1 + r_1 < t_2 < t_1 + k_1 \times d$ and $t_2 + r_2 > k_1 \times d$: this means that the patching stream has not started before the parent leaves the group. we consider several cases of remaining buffer sizes: The missing portion is $[t_1, t_1 + r_1]$, and $[t_2, t_2 + r_2]$. To patch the first portion, stream $w_1 = \lceil \frac{t_2+r_2-r_1}{d} \rceil$ is selected. The buffer size at time $t_2 + r_2$ is $b_1 = max(0, b - (t_2 + w_2 \times d - t_1))$ at that moment. If $b_1 > w_1 \times d$, then no waiting time is needed. And the buffer size is b_1 at the end of the first patching. Otherwise, the buffer size is $w_1 \times d$. The content that is contained in the client buffer is $[t_1 - b + (t_2 + w_2 \times d - t_1), t_1 + r_1] \cup [t_2 + r_2, t_2 + w_1 \times d]$. Now the time is $t_2 + w_1 \times d + r_1$, the stream $w_2 = \lceil \frac{t_2+w_1 \times d+r_1-t_2}{d} \rceil$ is selected. The effective buffer is the first portion of the two, and when the second patching is fixed, the effective patching is continuous. And the size will be something like $Max(effectivepatching, w_2 \times d) - [t_2 - t_1 - r_1, , t_2] + w_1 \times d + w_2 \times d$.

iii. $t_1 + k_1 \times d < t_2 < t_1 + k_1 \times d + r_1$: this means that the patching stream has already started but is disconnected in the middle. The missing portion should be $[t_1 + (t_2 - t_1 - k_1 \times d), t_1 + r_1] \cup [t_2, t_2 + r_2]$.

iv. $t_2 > t_1 + k_1 \times d + r_1$: this is the non-consecutive case, which we have analyzed in previous section.

5 Performance Evaluation

5.1 Simulation Environment Setup

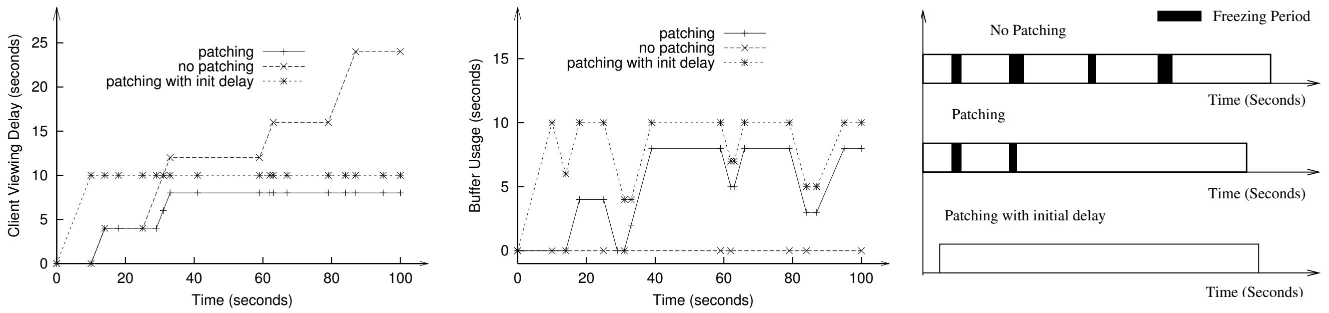
We use the GT-ITM transit-stub model to generate a network topology [4] of about 1400 routers. The average degree of the graph is 2.5, and core routers have a much higher degree than edge routers. The media server and the end hosts are connected with the edge routers. We categorize the bandwidth capacity of the end hosts into three levels. 1) Narrow bandwidth: with $1.5Mbps$ bandwidth, 70% of the end hosts are in this category. This bandwidth capacity is only enough for receiving video streams. The hosts with such bandwidth capacity can only be leaf nodes in the tree. 2) Medium bandwidth: with $10Mbps$ bandwidth, 20% of end hosts belong to this category. 3) High bandwidth: with $100Mbps$ bandwidth, only 10% of hosts have such high-speed connection.

There is one fixed media server in the topology, it has $100Mbps$ high speed connection. The time shifting value between stream s_i and s_{i+1} is 4 seconds. The server processing capability and I/O capacity is not a bottleneck compared to the server side bandwidth. We further assume links between the routers are never a bottleneck.

5.2 Evaluation of client video reception performance

In this section, we study the client video reception performance in our system. We are interested in the following performance metrics: *client viewing delay, maximum buffer usage, video continuity,* and *video completeness*. *Client viewing delay* means the delay between the client side playback time and the playback time of the original stream. *Maximum buffer usage* records the maximum buffer usage throughout a client's video reception process. *Video continuity* for a client is evaluated by the duration and frequency of freezing period. *Video completeness* refers to the rate between the received video and the transmitted video.

5.2.1 Effect of video patching



(a)Viewing Delay (b) Buffer Usage (c) Video Continuity

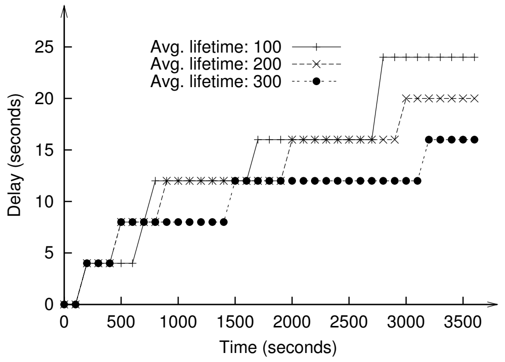
Figure 16: Video patching on client video reception

We first compare the video reception performance of a client with or without video patching scheme. In this experiment, we record a client's viewing behavior from the time it first joins the application layer multicast tree until it leaves the group. We assume infinite client buffer size in this simulation.

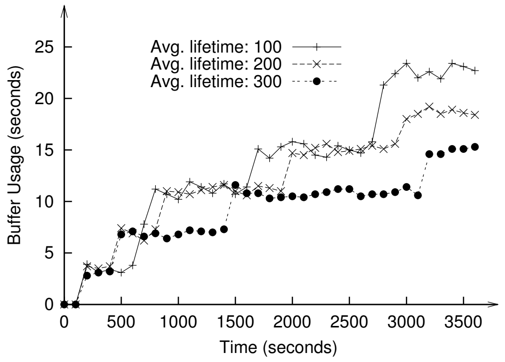
Figure 17 shows the simulation result. Figure 17(a) shows the client viewing delay. Without video patching, the client suffers the longest viewing delay. Video patching scheme can greatly reduce the viewing delay. As to the buffer usage, video patching scheme requires some buffer space to store the video packets that is not being played out. The no patching scheme does not use any buffer space, as shown in Figure 17(b). Figure 17(c) shows the video continuity during the viewing process. Without video patching scheme, the client suffers freezing period every time it is disconnected. With video patching, the freezing period is significantly reduced, since the client can still playout the previously buffered video content when the subsequent service disconnection happens. We also find that, if the client starts playout the video after an appropriate initial delay, the client can receive continuous video stream.

5.2.2 Adaptation to different traffic patterns

We now study the video repection performance of a client under different traffic patterns. In this experiment, the clients join the multicast tree according to a *Poisson* process, their average time in the group varies from 100s to 300s. A client joins the group and does not leave throughout the simulation process. The simulation time is one hour, we sample the viewing delay every 100 seconds, and calculate the average buffer usage for every 100 second interval.



(a)Client Viewing Delay



(b) Buffer Usage

Figure 17: Video Patching on viewing performance

Figure 18 shows the simulation result. The shorter the average client lifetime, the longer the client viewing delay. Since as the clients leave the group more frequently, it is more possible that the client is disconnected again before the patching period finishes. In this way, the client has to join later and later time-shifted streams each time it rejoins the multicast tree. Correspondingly, as the clients keep joining the stream with longer time shifting value, the patching period also becomes longer, thus accumulating more data in the client buffer.

5.2.3 Overall Distribution

In this section, we focus on the overall system performance for all the clients in our system design.

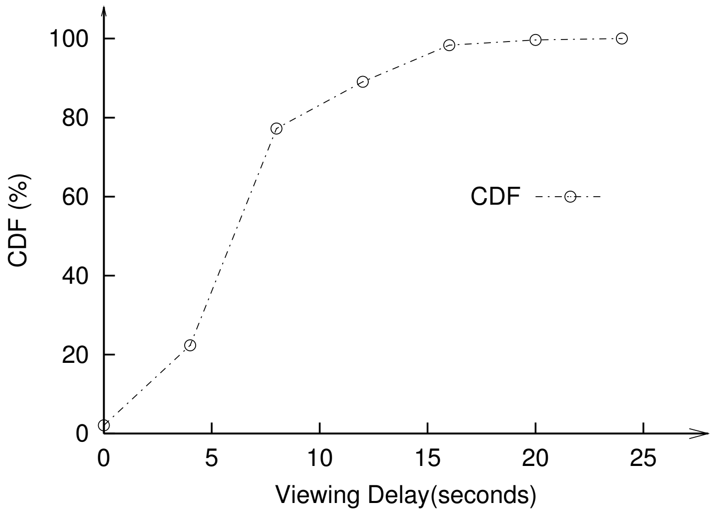
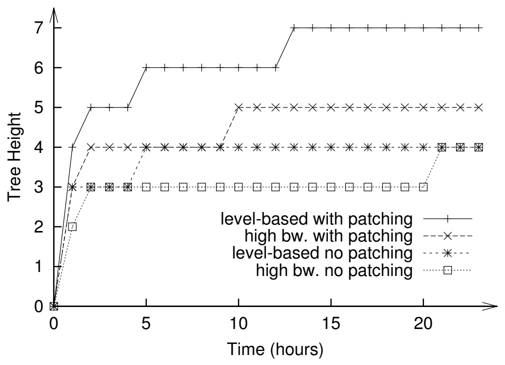


Figure 18: Viewing delay distribution

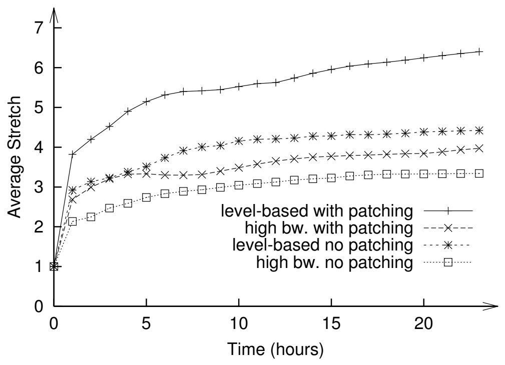
Figure 19 shows the distribution of client viewing delay. The client buffer usage demonstrates very similar distribution. The maximum client viewing delay value is 24 seconds. Most of the clients' viewing delay is between 4 and 16 seconds. The factors that determine the viewing delay are: the client rejoin latency, and the server video channel usage condition. The client rejoin latency is mostly determined by its location in the application layer multicast tree. There are some clients directly connected with the video server, these clients suffer no viewing delay at all. Those clients further away from other clients tend to need longer time to rejoin the application layer multicast tree. Another factor that influences the viewing delay is the server channel usage condition. If the server does not have a free patching channel available when the client rejoins the tree, the patching stream has to be delayed, as well as the client viewing delay. Clients that suffer longer viewing delay also need more buffer space, since the patching period tends to be longer.

5.3 The effect of video patching on tree structure

In this section, we study the influence of the patching scheme on the structure of the the application layer multicast tree, and the average stretch of the end hosts. In this experiment, the clients join the multicast group in a *Poisson* process, and stays in the group throughout the experiment. The number of clients in the tree is monotonically increasing. The number of clients in the tree increases from 1 to about 100000 in our simulation. The simulation time is 1 day in this experiment.



(a)Level-Based Scheme



(b) High-BW First Scheme

Figure 19: Influence of video patching on tree structure

We compare four tree join algorithms in this experiment: *level-based* algorithm, with and without video patching; *high-bandwidth-first* algorithm, with and without video patching. Figure 20(a) shows the height of the tree. The level-based tree join algorithm with video patching generates the highest multicast tree. The high-bandwidth-first algorithm promotes the clients with high bandwidth to the upper level of the multicast tree, thus increases the fan out of the tree in the higher level. The height of the tree for the high-bandwidth-first algorithm is significantly smaller than the level-based scheme. Furthermore, for the high-bandwidth-first algorithm, the height of the tree with video patching does not increase much compared to the scheme without video patching.

Figure 20(b) shows that the level-based tree join algorithm with video patching has the worst stretch performance, and is much worse than the same tree join algorithm without video patching. For the high-bandwidth-first scheme, the shape of the tree is optimized, and the height of the tree is comparatively shorter, the stretch performance with or without video patching is close to each other.

5.4 Server Channel Allocation Scheme

The server side bandwidth resource (video channel) availability determines the shape of the multicast tree, and the starting time of the patching stream. How to effectively assign server channels is important to the overall system performance, and to the satisfaction of client viedeo reception. We have proposed three video channel allocation scheme: 1:1, static multiplexing, and dynamic multiplexing. In this section, we evaluate two aspects of video channel allocation scheme: 1) video channel utilization 2) client queuing delay for the patching channel.

5.4.1 Server channel utilization

Server channel utilization means the percentage of the number of channels in use to the number of all the video channels. In this experiment, we assume the server can support 100 video channels simultaneously. We run the simulation for 1 hour, and record the average channel utilization value every 100 seconds. Figure 21 shows the simulation results.

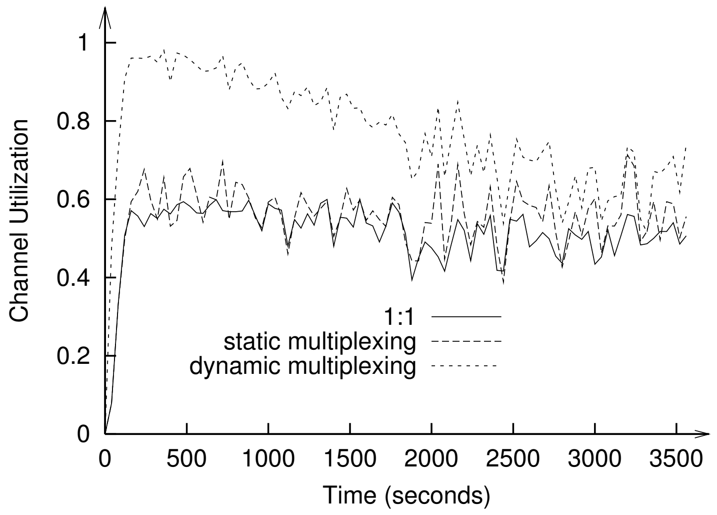


Figure 20: Video Server Channel Utilization

For *1:1* channel allocation scheme, the channel utilization value is around 50%. In this scheme, 50% of the video channels are allocated as live channel, and each live channel is bound with a patching channel. A patching channel is used only after a service disconnection in the application layer tree of the associated live channel, otherwise, it is free. And the life time of a patching channel is short compared to the live video program time. Furthermore, in 1:1 scheme, even if there are multiple rejoin requests in the application layer multicast tree of a live channel, the associated patching channel can only serve one request at a time, other requests have to be put into a queue until this patching channel is free.

In static multiplexing scheme, we use half of all the video channels as live channel, and reserve the other half as patching channel. The channel utilization value is better than the *1:1* scheme. And the channel utilization has larger variations, since when there are multiple rejoin reuqests, all the free patching channels can be put into use, thus increasing the channel utilization value. Dynamic multiplexing with channel merging performs the best in channel utilization. Since in the beginning stage, most of the video channels are used as live channel. Thus, at the early stage, the channel utilzation value is close to 1. As time proceeds, the server merge the live channels with fewest clients, and reserve more channels for the patching channel, causing a degradation in channel utilization.

5.4.2 Client queueing delay

Client queueing delay means the time when the server begins to receive the video patching request until the time its requests is being served. In this section, we compare the client queueing delay under the three video channel allocation schemes. We use the same configuration as last section, and Figure 22 shows the simulation result.

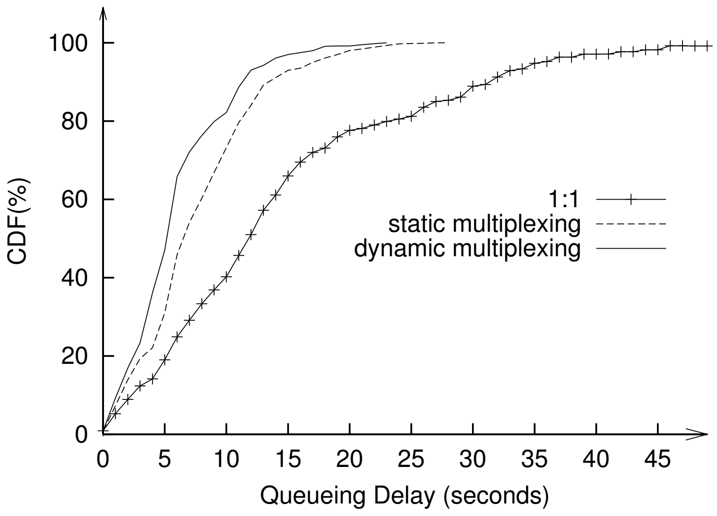


Figure 21: Client Requests Queueing Delay

We find out that the queueing dealy for *1:1* scheme is significantly longer than the other two schemes. Since the patching channels are not shared in this scheme. If a node with many children leaves the group, it could produce many rejoin requests at the same time. Since *1:1* can serve only 1 request a time, the queueing delay can be very long. For the other two schemes, because they use channel multiplexing, so that the rejoin requests can be served in a global basis, thus the queuing delay is significantly reduced. For the dynamic multiplexing case, when the patching channel is not enough to handle the rejoin requests, some live channels are merged and used as patching channel. Thus, dynamic multiplexing can further reduce the queueing delay.

5.5 Protocol Overhead Analysis

In this section, we consider several aspects of complexity in our system design. • Message processing overhead: in our system design, the server has to process four kinds of messages: *join, leave, rejoin* and *patching end*. We assume the weight for processing these messages is the same. Our node rejoin message is composed of *join live* and *join patching* messages. The *patching end* message is coupled with a *join patching* message. Assuming there are N node join message, and M node rejoin messages throughout the video streaming process, the number of messages should be: $2 \times N + 3 \times M$. For those no-shifting, no patching solutions, the number of messages is: $2 \times N + M$. For a CoopNet solution with m descriptions, each *join, leave, rejoin* message involves operations over m trees, the number of messages should be: $(2 \times N + M) \times m$.

• Tree management overhead: our server maintains one tree for the original stream, and multiple trees for the patching stream. The patching tree has considerably smaller size and shorter life time than the original tree.

• Bandwidth overhead: although the MDC codec used in CoopNet introduces some bandwidth overhead [1, 7], our solution is more bandwidth intensive. Since we need to reserve same amount of bandwidth for each original stream allocated. But the adoption of the MDC approach brings extra overhead such as coding/decoding complexity, synchronization of multiple streams, etc.

6 Concluding Remarks

In this paper, we deal with the problem of continuous live video streaming to a group of cooperative clients. Our solution is centered around a time-shifting video server, and a video patching scheme. The time-shifting video server sends multiple video streams with different time shifting value. Clients in the application layer multicast tree can retrieve the missed video content by joining the time-shifted video stream. To avoid *indefinite time-shifting* due to multiple service disconnection during the video reception process, we introduce the video patching scheme. During the patching period, a client is receiving the time-shifted video stream as well as the original stream. The video content that is not being played out immediately is stored in a client buffer. When a subsequent service disconnection occures, a client can play the video content from the buffer while rejoining the group. In this way, the client can receive the complete video program even though the forwarding infrastructure is unreliable. Continuous video streaming is achieved if the client starts video playout after some *initial delay*.

Our design has following features: 1) lossless video reception: by allowing clients rejoin the time-shifted video stream, the client can receive the whole video content from the point it first joined the group. 2) stable video quality: the client receives full quality video throughout the video reception process. 3) continuous video streaming: continuous video streaming can be achieved by sacrificing initial video access delay. 4) Compared to CoopNet's MDC-based system, our system has the advantage that it can use standard-based single description encoded streams. 5) Moderate complexity: the overhead of message processing and tree management is at the same level with a no-shifting, no-pacthing solution.

References

[1] J. Apostolopoulos. Reliable Video Communication over Lossy Packet Networks using Multiple State Encoding and Path Diversity . In *Proceedings of VCIP 2001.*

[2] S. Banerjee, B. Bhattacharjee, C. Kommareddy Scalable Application Layer Multicast In *Proceedings of ACM SIGCOMM*, August 2001.

[3] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, S.Khuller Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications To appear in *Proceedings of Infocom*, 2003

[4] K. Calvert, M. Doar, and E. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, June 1997.

[5] Y.H. Chu, S.G. Rao and H. Zhang A Case For End System Multicast In *Proceedings of ACM SIGMETRICS*, June 2000,

[6] Y.H. Chu, S.G. Rao and H. Zhang Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture In *Proceedings of ACM SIGCOMM*, August 2001.

[7] V.K. Goyal, J. Kocevic, R. Arean, and M. Vetterli. Multiple Description Transform Coding of Images In *Proceedings of ICIP*, 1998

[8] D. Hrishikesh; B. Mayank; G. Hector Streaming Live Media over a Peer-to-Peer Network. Technical Report, Stanford, 2001

[9] K.A. Hua, Y. Cai, S. Sheu. Patching: a multicast technique for true video-on-demand services . In *Proceedings of the sixth ACM international conference on Multimedia*, 1998.

[10] M. S. Kim, S. S. Lam, D.Y. Lee Optimal Distribution Tree for Internet Streaming Media. *Technical Report* , U.T. Austin, 2002

[11] V. N. Padmanabhan, H. J. Wang, P. A. Chou, K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. In *Proceedings of NOSSDAV 2002.*

[12] V. N. Padmanabhan, H. J. Wang, P. A. Chou. Resilient Peer-to-Peer Streaming In Microsoft Research Technical Report, March, 2003. http://research.microsoft.com/projects/coopnet/papers/msr-tr-2003-11.pdf

[13] D. A. Tran, K. A. Hua, T. T. Do Peer-to-Peer Streaming Using A Novel Hierarchical Clustering Approach To appear in *Proceedings of ICDCS*, 2003

[14] D.Y. Xu, M. Hefeeda, S. Hambrusch, B. Bhargava On Peer-to-Peer Media Streaming. In *Proceedings of ICDCS*, 2002

[15] L. Zou, E.W. Zegura, M.H. Ammar The Effect of Peer Selection and Buffering Strategies on the Performance of Peer-to-Peer File Sharing Systems. In*Proceedings of MASCOTS 2002*, October 2002.