4D TeleCast: Towards Large Scale Multi-site and Multi-view Dissemination of 3DTI Contents

Ahsan Arefin, Zixia Huang, Klara Nahrstedt, *Pooja Agarwal
Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL
*Microsoft Corporation, Redmond, WA
{marefin2, zhuang21, klara}@illinois.edu, *pooja@microsoft.com

***Abstract*—3D Tele-immersive systems create real-time multi-stream and multi-view 3D collaborative contents from multiple sites to allow interactive shared activities in virtual environments. Applications of 3DTI include online sports, tele-health, remote learning and collaborative arts. In addition to interactive participants in 3DTI environments, we envision a large number of passive non-interactive viewers that (a) watch the interactive activities in 3DTI shared environments, and (b) select views of the activities at run time. To achieve this vision, we present 4D TeleCast, a novel multi-stream 3D content distribution framework for non-interactive viewers providing the functionality of multi-view selection. It addresses the following challenges: (1) supporting a large number of concurrent multi-stream viewers as well as multi-views, (2) preserving the unique nature of 3DTI multi-stream and multi-view dependencies at the viewers, and (3) allowing dynamic viewer behavior such as view changes and large-scale simultaneous viewer arrivals or departures. We divide the problem space into two: (1) multi-stream overlay construction problem that aims to minimize the cost of distribution of multi-stream contents, and maximize the number of concurrent viewers with sufficient viewer dynamism in terms of their resources and availabilities, and (2) effective resource utilization problem that aims to preserve the multi-stream dependencies in a view considering the heterogeneous resource constraints at the viewers. We evaluate 4D TeleCast using extensive simulations with 3DTI activity data and PlanetLab traces.**

I. Introduction

In current multimedia space, 3D Tele-immersive (3DTI) systems focus to provide an effective platform for multi-view immersive social interactions among geographically distributed parties (called sites). They are much more media enriched than traditional audio-video systems such as skype, PPLive [2], CoolStreaming [23], LiveSky [22] and YouTube. Examples of advanced TI systems include TEEVE [19] from UIUC, Halo from HP, and TelePresence from Cisco. Though such technologies have been available in the market for several years, their applications are limited to a small number of remote interactive participants (called *content producers*) due to the pixel space limitations of the virtual world projected on the physical display. However, we envision the presence of a large number of passive non-interactive *content viewers* that (a) watch the interactive activities of the content producers in 3DTI shared environments, and (b) select views of the activities at run time. Though, current IPTV solutions [2], [22], [23] provide efficient framework for a large scale dissemination, they do not consider any multi-party multi-stream contents with view change dynamics. Therefore, a live multi-stream multi-party 3D content distribution and routing framework supporting a large number of content viewers with functionality of view selection is yet to come.

3DTI content producers are active users performing collaborative activities in the virtual shared environment, e.g., dancers in collaborative dancing and players in immersive gaming. On the other hand, content viewers are passive users who view the joint performance of content producers without contributing in the virtual world with 3D contents. Examples of content viewers are audiences of virtual collaborative dancing or viewers of online exergaming. We envision hundreds-to-thousands of concurrent content viewers in a 3DTI session. The difference between the traditional TV viewers and the 3DTI viewers is that 3DTI viewers receive multiple streams at the same time from different content producers and the viewers are able to change views of the 3D contents. On the other hand, TV viewers are subscribed to only one stream at a time and they rely on the providers (e.g., ESPN) to change views. Figure 1 shows the interaction between the 3DTI content generation and viewing space. Each 3DTI producer site (Site-A, Site-B and Site-C in the figure) hosts multiple cameras that capture a local scene from various angles (represented as *streams*). Multiple adjacent streams (called *bundle of streams [5]*) compose a *view*. Content viewers request these views, which include subset of streams from each site. The requested

streams are then disseminated over the Internet, and aggregated (rendered together) at the viewer nodes (at the viewer tier) to construct an integrated 3D virtual scene (the detail models of 3DTI producers and viewers are given in Section II).

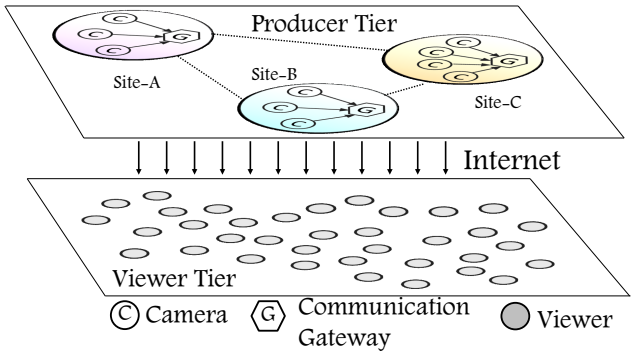


Fig. 1. 3DTI content generation and distribution spaces. Contents are genrated from the producer participants. Viewer participants only views the collaborative performance without participating in the virtual environment.

However, the large scale multi-stream and multi-view dissemination of 3DTI contents introduce several challenges. Firstly, bundles generated across the producer sites at any point in time are highly dependent; so are the streams inside a bundle, because they collaboratively represent a consistent virtual scene. Such time dependencies must also be preserved at the viewers due to the same reason. However, keeping the inter-bundle skew (delay difference between dependent bundles) or local inter-stream skew (delay difference between dependent streams inside a bundle) bounded in current Internet is very difficult due to the heterogeneity of dissemination paths taken by the dependent streams in a view. Often, in case of large inter-bundle skew or local inter-stream skew, the lagged streams are dropped to display a consistent scene, even though they consume network and system resources, which causes ineffective resource usage. Secondly, the large scale 3D multi-stream dissemination introduces a large demand on bandwidth. Each 3DTI stream consumes around 400Kbps to 5Mbps bandwidth, which is exacerbated due to the presence of multiple streams in each view as well as the need to stream multiple views to many viewers, one view from each content producer site. Without proper allocation of bandwidth, it is hard to support a large number of concurrent views as well as viewers in this dynamic environment. Finally, the 3DTI systems allow viewers to dynamically change their views during run time similar to TV channel switching. However, unlike TV channels, views may contain overlapping streams (details are show in Section II-B). Therefore, changing views changes the stream subscription of the viewers. Such view dynamics impacts the content dissemination topology and interfere ongoing transmission of already subscribed streams. Due to the nature of live 3DTI streaming, such inference should be low and restored quickly.

Therefore, there is a need to construct an efficient 3D content dissemination framework that allows viewers to subscribe to multiple streams while optimizing the bandwidth resources and preserving the stream dependencies. We present 4D TeleCast, a novel 3DTI content dissemination framework that scales to a large number of viewers and provides the functionality of view selection. We call it 4D TeleCast because we consider four dimensions in the tele-immersive space: (1) height, (2) width, and (3) depth of the tele-immersive video streams generated by individual producer sites, and (4) composite view of a viewer to watch these video streams from different locations in a virtual space.

4D TeleCast adapts the hybrid dissemination techniques from [20], [22] to balance the impact of centralized load of streaming and overhead of fully distributed management. A CDN (Content Distribution Network) is used to capture and distribute stream contents from the producers with P2P routing at the edges (among content viewers). An effective bandwidth allocation algorithm (in Section IV-B1) is considered at the P2P layer based on the stream *priority* (explained in Section II) to allocate inbound and outbound bandwidth at the viewers. A *degree push down* algorithm is used to finally build a P2P overlay that allows maximum number of concurrent viewers (given in Section IV-B2). Finally, to preserve the multi-stream dependencies, a *delay-based layering* is introduced below the CDN that bounds the differences in the *end-to-end delay* (the delay between the point when a frame is captured and the point when it is received at the viewer) incurred by the streams inside a requested view. It improves effective bandwidth usage in the system by using delayed receive for the streams with lower end-to-end delay.

The contributions of this paper include the following: (1) we provide a first step towards modeling a 4D content dissemination framework for a large number of widely distributed multi-stream viewers, (2) a stream priority based bandwidth allocation and overlay formation scheme is used to maximize the number of concurrently accepted views as well as viewers, (3) we introduce a delay-based layering hierarchy to preserve multi-party multi-stream dependencies at the viewers, which also helps to improve effective bandwidth usage in the system, and (4) finally we provide mathematical and experimental analyses showing the effectiveness of 4D TeleCast.

II. System Model and Assumption

A. 3DTI Component Model

3DTI systems impose heterogeneous demands on networks because of the hybrid nature of participants. The content producers (Site-A, Site-B and Site-C in Figure 1) require very tight real-time bound on end-to-end streaming delay due to the tight interactivity among the participants while we envision that content viewers can tolerate larger delays. According to [9], the time lag in PPLive viewers for 2D live streaming varies between 20 seconds to 2 minutes depending on the channel popularity.

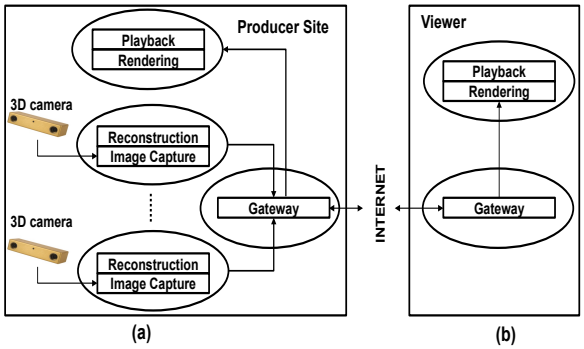


Fig. 2. Model of a 3DTI (a) content producer, and (b) content viewer.

Producer Model. Each 3DTI producer is equipped with multiple camera components to generate 3D contents. Figure 2(a) shows an architecture of a 3DTI producer. All cameras are connected to a Rendezvous Point (called gateway). Communications to/from remote producers or viewers are done only via these gateways. A renderer is connected to the gateway to render and display the 3D contents. Since the number of producers is limited and static in a single 3DTI session, the randomized dissemination algorithm [19] with adaptive end-point synchronization [10] works well for inter-producer communication.

Viewer Model. Viewers also contain the gateway for communication and the renderer for rendering and displaying the 3D contents. The architecture of a non-interactive viewer is shown in Figure 2(b). Buffering is done when the streams are received by the gateway. For playback smoothness, another level of buffering can be maintained inside the renderer, but for simplicity of analysis, we only consider buffering at the gateway. This assumption does not impact the correctness of our evaluation rather bounds the worst case performance.

B. Stream and View Model

Each 3DTI producer hosts multiple cameras; each captures a local scene from various angles represented as streams. The selection criterion of streams belonging to a view is calculated based on the stream orientation and the local view orientation. For example, if the viewer is currently looking at the front of a 3D object, streams generated by the back cameras are less important and can be dropped. [21] introduces a stream differentiation function to calculate the priority of a stream in a view, which indicates stream importance. Basically, let us denote $S.\vec{w}$ as the unit vector representing the spatial orientation of the stream S and $v.\vec{w}$ as the unit vector representing the view v of viewer u. Then the differentiation function (denoted as df) calculates the stream importance inside a producer site as $df(S, v) = S.\vec{w} \cdot v.\vec{w}$. Streams are prioritized depending on their importance. A threshold based cutoff scheme [21] over the value of df is used to remove the less important streams from a view. However, to compare priority of streams across different sites, we introduce another notation (η) that indicates the priority index of each stream inside the local site. If a site contains two streams S_1 and S_2, where $df(S_2, v) > df(S_1, v)$ for view v, then $\eta^v_{S_1} = 2$ and $\eta^v_{S_2} = 1$. The global priorities for streams for view v are then computed using $(\eta^v_{S_i} - df(S_i, v))$. Streams with lower $(\eta^v_{S_i} - df(S_i, v))$ have higher priorities.

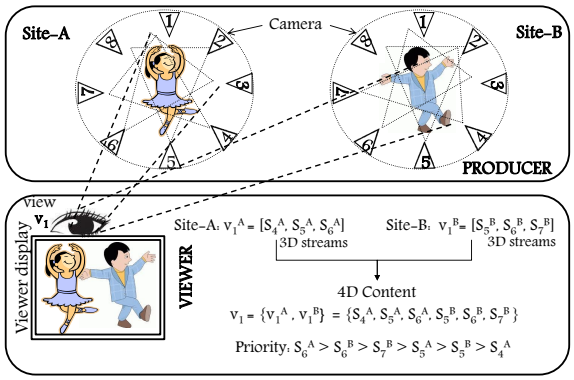


Fig. 3. Example of the 3DTI view and stream model with two producers and one viewers. A cut-off is used to remove the low priority streams.

When a viewer u requests a view v_i in the global virtual space, the request selects a single view (called *local view*[1]) from each content producer site. If there are two content producer sites, Site-A and Site-B, then $v_i = \{v_i^A, v_i^B\}$, where v_i^A and v_i^B are selected local views from Site-A and Site-B respectively. Let us consider that Site-A contains n streams $\{S_1^A, S_2^A, \cdots, S_n^A\}$ and for view v_i^A, their relative priorities are $S_1^A > S_2^A > \cdots > S_n^A$ (i.e., $df(S_1^A, v_i^A) > df(S_2^A, v_i^A) > \cdots > df(S_n^A, v_i^A)$). If df_{th} is the cut-off threshold, then v_i^A

[1]We use term "view" to indicate the global view. Local views are termed explicitly or can be inferred from the contexts.

can be represented as $v_i^A = \{S_1^A, S_2^A, \cdots, S_j^A\}$, where $df(S_j^A, v_i^A) \leq df_{th} < df(S_{j+1}^A, v_i^A)$ and j defines the number of streams having df below the cut-off value. Similarly, we can define v_i^B. Thus, each local view contains a composition of a subset of local 3D streams. The composition of all local views (one from each producer site) finally forms a global view, which we call the 4D content. Figure 3 shows an example of 4D composition. Considering the global view v_1 in Figure 3, the requested 3D streams from Site-A are S_4^A, S_5^A and S_6^A and from Site-B are S_5^B, S_6^B and S_7^B. The 4D content thus becomes $v_1 = \{S_4^A, S_5^A, S_6^A, S_5^B, S_6^B, S_7^B\}$.

C. View Change Model

4D contents introduce another unique scenario, called *view change*, where viewers change their views while watching 3D contents. A viewer only observes the visual information from one particular view at any given time and can therefore, change views any time during a live 3DTI session. View v_i is different from view v_j, i.e., $v_i \neq v_j$ if $\exists S_1 \in v_i \wedge S_2 \in v_j$ such that $S_1 \neq S_2$. Therefore, changing views causes a viewer to discard old streams and receive new streams of the new view.

D. Viewer Request Model

Once a viewer requests a view v_i, it is translated into multiple local views; one for each producer site in the running 3DTI session. Each local view contains multiple streams depending on their importance in the current view. Global priorities are computed using $\eta - df$ value. However, further cut-off can be done in run-time depending on the resource constraints (e.g., bandwidth limitation). Dropping of the lower priority streams only degrades the media quality (e.g., creates holes in the scene). However, a viewer request is accepted only if at least the highest priority stream of each local view can be delivered to the viewer. Since, the cut-off is done starting from the lower-priority stream, if at least one stream is served from a local view, it is always be the highest priority stream. Hence, if n is the number of content producer sites in a 3DTI session, the number of accepted stream (denoted by $N_{accepted}^u$) for viewer u must be bounded as follows: $N_{accepted}^u \leq n$.

E. Streaming Model

When a viewer requests for a stream from its parent, the contents of the stream, called 3D frames are transmitted to the viewer over the Internet. A stream S_i can be represented as a composition of incoming frames, e.g., $S_i = \{f_{t_1}^{(i,n_1)}, f_{t_2}^{(i,n_2)}, f_{t_3}^{(i,n_3)}, \cdots\}$, where t_1, t_2 and t_3 are capture timestamps and n_1, n_2 and n_3 are corresponding frame numbers. Each viewer maintains a list of these recent frames in the cache. Therefore, the requests for streaming eventually select the position in the cache at the parent viewers. Subsequent frames from that position are transmitted at the media rate.

III. System Architecture and Operation

4D TeleCast provides a hybrid overlay dissemination framework consisting of a CDN to transmit 4D contents from producers to content viewers, and P2P overlays to transmit them within the viewers. The purpose of using hybrid dissemination architecture is that using only CDN servers for streaming is very expensive in terms of deployment and maintenance. On the other hand, a P2P-based architecture requires sufficient number of seed supplying peers to *jump-start* the distribution process and offers less out-bound streaming rate compared to a CDN server [20]. Therefore, the commercial CDN-P2P-based live streaming system (e.g., LiveSky [22]) shows better performance than pure P2P-based live streaming systems [12].

A Global Session Controller (GSC) node is used to manage the live session of 4D TeleCast content dissemination. To scale GSC, we divide the geographical region into several region-based clusters and assign a Local Session Controller (LSC) to each cluster that manages the streaming requests of all the viewers inside its own cluster. The GSC also continuously monitors producers metadata (such as frame rate, frame number, and frame size for each stream), stream priorities of each viewer's request, and geographical location of the viewers. For finding the geo-location of the viewers, we use a location detector algorithm similar to [15]. All metadata information are available for the viewers upon query from the GSC. Figure 4 shows the architecture of 4D TeleCast with different functional components and their interactions. Below we explain them.

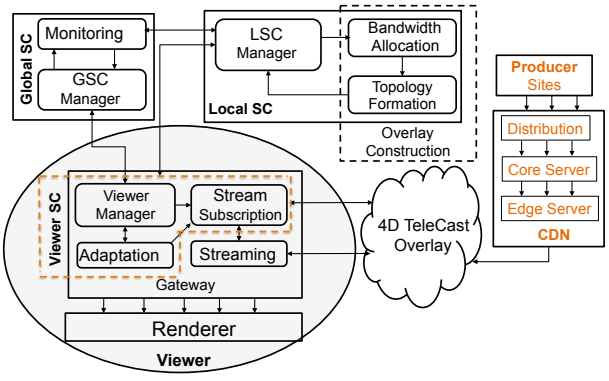


Fig. 4. 4D TeleCast system components and their interactions.

A. Server Side Distribution

Content producers and CDNs are acted as 4D TeleCast media and distribution servers. Several CDN providers are available in market (e.g., Akamai [6] and Amazon CloudFront [1]). Though they do not provide any real-time service guarantee on content delivery, 4D TeleCast aims to use those commercial CDNs for live 4D contents streaming due to the non-immersive nature of the viewers.

The CDN architecture usually contains a storage (called *distribution* in Figure 7), where 3DTI contents (3D media frames) are uploaded from the producers' gateway. The *core servers* distribute the contents to different *edge servers*. In addition, CDN internally handles load distribution, data replication and data availability across the edge servers so that the viewer requests can be served quickly [3]. Note that, 4D TeleCast uses the CDN as a storage and first layer distribution server and does not require any changes in architecture or software modification inside the CDN.

B. Client Side Distribution

4D TeleCast clients are viewers. Each viewer requires a viewer software that includes both the gateway and the renderer (as shown in Figure 3). The gateway is further divided into data plane (streaming) and control plane (viewer SC). The data plane is only resonsible for streaming using the session *routing table*. The control plane helps building the routing table and keeps it updated in case of network and viewer dynamisms.

The structure of session overlay routing table at the viewer is shown in Table I. The session routing table contains following fields: *matching field* (usually stream ID and corresponding parent ID), *forwarding address* (list of child addresses separated by comma), *action list* (one for each forwarding address separated by comma), and *subscription points* (one for each forwarding address separated by comma). When a frame of a stream arrives at a viewer, it is placed in the viewer buffer and matched against the matching field in the routing table to get the matching routing table entry. For each forwarding address in the matching entry, a forwarding frame is picked from the viewer's buffer and cache depending on the corresponding subscription point defined in the table if the action list is "forward". No forwarding is done in case of action is "drop". Below we discuss how the session routing table is populated when a viewer joins and how the streaming is done at the client side. In this paper, we always assume to have "foward" action in the action field. However, the future extension may define specific compression, encoding or rate control as different actions for the forwaring streams.

To join, the *viewer manager* inside the control plane requests a join to the *GSC manager* at the Global Session Controller. GSC then finds the appropriate LSC from the *monitoring* component based on the viewers geographical region, and forwards the viewer's join request to the *LSC manager* to handle the join process. When LSC replies to the viewer, the viewer sends its view request with its inbound and outbound capacity information. Figure 5 shows the session join protocol.

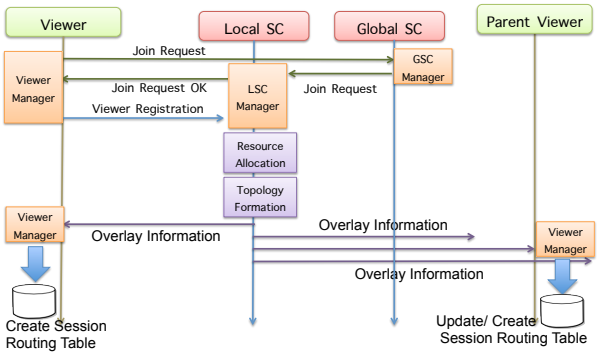


Fig. 5. Communication protocol in viewer join in 4D TeleCast.

Depending on the view information, LSC management computes the list of requested streams and their priorities in the current view, and forwards them to a *bandwidth allocation* component along with viewer's inbound and outbound capacity information. The bandwidth allocation component allocates viewer inbound and outbound capacity for the requested streams. Due to the bandwidth insufficiency, some of the lower priority stream requests may be dropped. The list of accepted streams in the view is sent to the *topology formation* component. It creates the final overlay tree (i.e., defines the parents and children) for each accepted stream considering the bound on end-to-end delay.

Topologies are formed separately for each view group, i.e., the topology formation component groups the viewers depending on the view request. The grouping ensures that the popular view creates enough resources (or seeds) to share and support other viewers with the same view compared to the non-popular views, and does not get interfered by the non-popular views. The algorithm first tries to provision a viewer request from the available viewers watching the same view, if failed, the request is provisioned from the CDN (provided the CDN has unused outbound capacity). The overlay information is then sent to the viewer and to the parents of the viewer to create or modify their overlay session routing tables. The goal of bandwidth allocation and topology formation (compositely we call them as overlay construction) is to

TABLE I
FIELD OF SESSION ROUTING TABLE

Match Field	Forwarding Address	Action	Subscription Point
Parent Viewer:Stream ID	Child Viewers	drop/forward/encoding/rate	Position in buffer and cache

build an optimal overlay that maximizes the number of accepted streams in the overlay and minimizes the CDN usage. The detail algorithms for overlay construction are given in Section IV.

Once, the joining viewer receives the topology information (list of parents and children) with the list of accepted and forwarded streams from the LSC, it updates its session routing table. One entry is created for each forwarded stream. Therefore, the matching field contains the forwarded stream ID and the corresponding parent ID. The forwarding address field is populated with the corresponding child addresses.

Though the overlay structure ensures the delivery of all accepted streams in the view request, it does not bound the inter-stream delay inside the view, which creates the *view synchronization* problem. The phenomena is explained in Figure 7(a), where the viewer u (shown in Figure 3) is provisioned to receive streams S_6^A, S_7^B, S_6^B by it LSC. Here, d_{buff} defines the time a frame of a stream stays in the buffer after it is received. As we see in the figure, when a frame of S_6^B is received, the correlated frame (captured at the same time) of S_6^A is already been discarded from the buffer. Since the end-to-end delay between S_6^B and S_6^A is higher than d_{buff}, viewers can not display them together at the renderer. This creates a lower quality view at u even though the network resources are consumed for the higher quality view. Therefore, the effective bandwidth utilization becomes low. This unused but allocated bandwidth can be reallocated to another viewer to generate high quality video at that viewer.

To solve this view synchronization problem, the *stream subscription* component in the viewer gateway computes the *subscription point* for each accepted stream. The subscription point of a stream defines the position at the cache of the parent viewers from where the parents should forward the stream. It eventually introduces *delayed receive* for some of the streams so that the dependent streams are always present in the buffer. To understand this stream subscription point in the overlay in terms of delay, we introduce a delay layer hierarchy. The details of delay layer hierarchy architecture and a stream subscription algorithm to solve the view synchronization problem are shown in Section V. However, if the delayed receive violates the maximum allowed end-to-end delay for a stream, the request for that stream is dropped and the resources are made available to use by the other viewers.

After the view synchronization is computed, the stream subscription component at the viewer sends the subscription point of the accepted streams to the corresponding parents and children. When the child viewers receive the subscription information, they update their synchronization point accordingly. When the parent viewer receives the stream subscription requests, it updates the subscription point in the routing table for each forwarded stream and starts streaming from the subscription point of its cache for each requested stream. Figure 6 shows the stream subscription protocol to update the subscription point in the session routing table. For data plane communication, we use an extension of RTP called S-RTP proposed by our previous work [4].

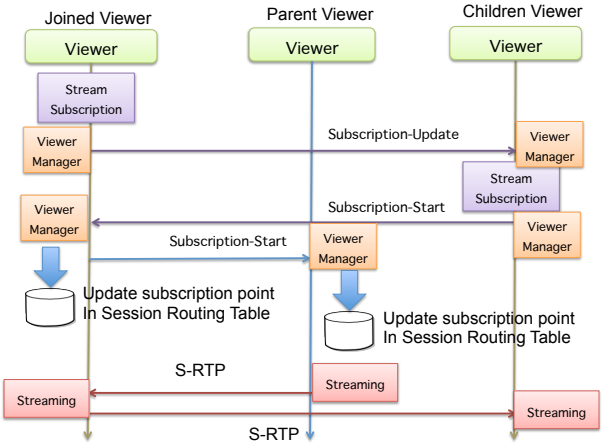


Fig. 6. 4D TeleCast stream subscription communication protocol.

However, due to the frame loss, congestion or other network events, the inter-stream delay may change. Therefore, an *adaptation management* component is used at the viewer's that periodically collects monitoring information from the GSC monitor (via the viewer management component) and updates the delay layer hierarchies and the stream subscription layers. It also handles overlay fault tolerance due to the viewers arrivals and departures. Due to the space limitation, we explain its functionality briefly in Section VI.

An example of 4D TeleCast distribution structure in shown Figure 7(b). Viewers u_1, u_2 and u_3 request for view v_1, which includes streams S_1, S_2 and S_3 (with priority $S_2 > S_1 > S_3$), and viewer u_4 and u_5 request for view v_2, which includes streams S_2, S_3, S_4 (with

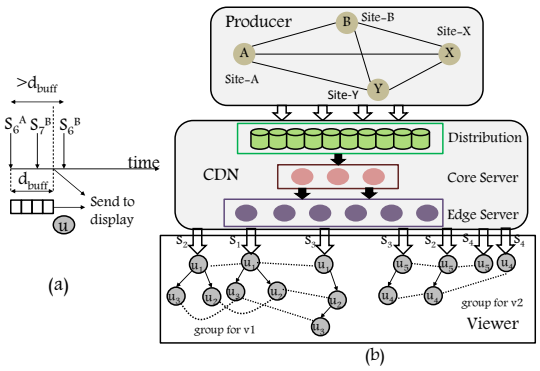


Fig. 7. Examples of (a) view synchronization problem, (b) 4D TeleCast dissemination and streaming architecture, where u_1, u_2 and u_3 request $v1 = [S_1, S_2, S_3]$ and u_4 and u_5 request $v2 = [S_2, S_3, S_4]$.

priority $S_3 > S_2 > S_4$). Depending on the view request, 4D TeleCast groups u_1, u_2, and u_3 in the same group, and u_3 and u_4 in a different group. Inside each group, individual streaming trees are created for each stream included in the corresponding view.

View Change. When a viewer requests a view change, to respond quickly, 4D TeleCast supports the streams in the new view instantaneously from the CDN. Therefore, the bandwidth allocation and topology formation component (in Figure 4) at the LSC, when detect a view change, initiate two parallel join processes. The first process serves all streams in the request directly from the CDN and the second process initiates a normal join explained above. Once the second process is done, the viewer is switched to the overlay constructed by the second process. Since the second process runs on background, the overall approach reduces the response time for a view change. Similar algorithm is used to re-attach the *victim viewers* (who get disconnected from the streaming tree due to the view changes) into the streaming overlay. We explain the view change process in Section VI.

IV. Multi-stream Overlay Construction

A. Overlay Construction Problem

The overlay construction problem considers three constraints (bandwidth, delay and number of accepted stream constraints), and one optimization goal.

Bandwidth Constraint. Each viewer u has limited total inbound bandwidth (C^u_{ibw}) and limited total outbound bandwidth (C^u_{obw}). The CDN also defines a bounded inbound and outbound capacity (C^{cdn}_{ibw} and C^{cdn}_{obw}, respectively) that can be used in the 3DTI session. Due to the limited number of producers sending content to the CDN, we assume C^{cdn}_{ibw} bound is always met.

Delay Constraint. Even though, the viewers are not immersed into the 3DTI space, they impose a delay bound for live streaming. d_{max} defines the maximum delay from the time when a 3D stream is captured at the producer site until the time when it is displayed at the viewer's display.

Number of Accepted Stream Constraint. As we mentioned before, to accept a viewer request, at least one stream (i.e., the most priority stream) from each producer site included in the view should be served to the viewer, i.e, for n number of producer sites, $N^u_{accepted} \geq n$.

Optimization Goal. Due to the delay and bandwidth constraints listed above, we cannot guarantee that all stream requests can be satisfied. The metric we wish to maximize first is the *acceptance ratio (ρ)* for all requests in the system. Suppose, the number of streams that viewers request is N_{total}; among which $N_{accepted}$ are accepted due to the resource constraints, then we have $\rho = \frac{N_{accepted}}{N_{total}}$. However, we also want to minimize the *CDN outbound capacity usage (obw_{cdn})* in the system so that cost of distribution becomes less (the use of 1GB traffic in Amazon CloudFront [1] CDN costs \$0.18). Formally, the overlay construction problem aims to build a dissemination architecture from producers to the viewers that maximizes the total acceptance ratio with minimum CDN outbound capacity usage such that $N^u_{accepted} \geq n$, $ibw_u \leq C^u_{ibw}$, $obw_u \leq C^u_{obw}$, $d^u_{S_i} \leq d_{max}$ and $ibw_{cdn} \leq C^{cdn}_{ibw}$ $\forall u \in U$, where ibw_u and obw_u are the inbound and outbound bandwidth usage of u, $d^u_{S_i}$ is the end-to-end delay of stream S_i from the producer to u and U is the set of all connected viewers in the overlay.

We provide a heuristic solution since an optimization problem in multicast with two or more constraints is NP-C [17]. We divide the solutions into two parts: *bandwidth allocation* and *overlay formation*.

B. Proposed Solution

1) Viewer Bandwidth Allocation: The bandwidth allocation should consider both the inbound and outbound bandwidth allocation. Below we discuss them.

Inbound Bandwidth Allocation. The process starts by performing bandwidth allocation on the viewer's inbound capacity. Streams are assigned required inbound bandwidth at the viewer in the order of their priorities provided that two conditions are met; 1) there is enough inbound bandwidth left for allocation at the viewer, and 2) the P2P layer or CDN has enough outbound bandwidth to support the stream. If any of these two conditions are violated, the lower priority streams are not assigned any bandwidth and they are removed from the view request. Suppose, a viewer u requests for view $v_m = \{S_1, S_2, \cdots S_n\}$ containing n stream from the

participating producer sites with priority $S_1 > S_2 > \cdots > Sn$, $abw_{Si}^{v_m}$ is the current available outbound bandwidth for stream S_i for v_m and bw_{S_i} is its required network bandwidth. To allocate viewer's inbound bandwidth, LSC assigns bandwidth bw_{S_i} to stream S_i if $bw_{S_i} \leq abw_{S_i}^{v_m}$ for first j streams in the order of their priorities such that $\sum_{i=1}^{j} bw_{S_i} \leq C_{ibw}^{u} < \sum_{i=1}^{j+1} bw_{S_i}$. Hence, the list of accepted streams in v_m becomes $S_1, S_2, \cdots S_j$. If $j = n$, then all streams are accepted.

If the number of accepted streams is less than the total number of participating site (i.e., at least one stream from each site), then the viewer's request is rejected. Otherwise, the LSC performs bandwidth allocation on the viewer's outbound capacity.

Outbound Bandwidth Allocation. Once the inbound bandwidth is allocated, the LSC allocates outbound bandwidth only for the set of streams accepted in the previous step. The outbound allocation is critical because if we assign outbound bandwidth to only the highest priority stream of each site, we can support maximum number of viewers but with lower media quality. However, if we assign bandwidth equally to all streams, we get limited number of viewers but with better media quality. The phenomena is shown in Figure 8(a). Overlay-1 supports less number of viewers but with better view quality (higher number of accepted streams), while Overlay-2 suports higher number of viewers with lower media quality (lower number of accepted streams). Therefore, we need a tradeoff in the outbound bandwidth assignment, where we can support sufficient number of viewers with good quality. Figure 8(b) shows the corresponding tradeoff among media quality, number of concurrent accepted viewers, outbound assignment and the number of concurrent accepted streams. Our goal is to maintain the overlay in the middle of the tradeoff lines (shown dotted). Though, the problem is complicated, It truns out that using a round-robin allocation of outbound bandwidth in the order of stream priority can solve it. Below we discuss the algorithm for outbound bandwidth allocation.

The outbound allocation starts with allocating the bandwidth for S_1 and continues until S_j similar to inbound allocation, but it rounds up, i.e., it starts allocating from S_1 again if there is enough bandwidth left after assigning the outbound bandwidth to other lower priority streams. It ensures that even with the bandwidth limitation, the highest priority streams in a view has higher probability to be served compared to the lower priority streams. Basically, if $S_i > S_j$ for a view v_m, then at any point in time $abw_{S_i}^{v_m} \geq abw_{S_j}^{v_m}$.

If $obw_{S_i}^{u}$ is the allocated outbound bandwidth for

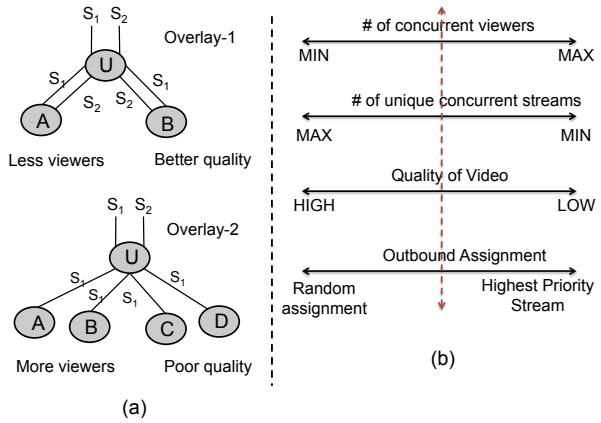


Fig. 8. (a) Example of different outbound allocations, (b) Tradeoffs among different paramters impacted by the outbound bandwidth allocation.

stream S_i at viewer u, the out-degree link for S_i at u is computed by $oDeg_{S_i}^{u} = \lfloor \frac{obw_{S_i}^{u}}{bw_{S_i}} \rfloor$. After the bandwidth allocation, the allocated out-degree is used to connect the viewers to the streaming tree.

2) Topology Formation: For building the overlay topology for each accepted stream in a view, LSC only considers the viewers with the similar view request as the peers of the P2P layer. A degree push down algorithm is used over the allocated inbound and outbound bandwidth. The goal is to improve the depth of the tree by constructing a flatter tree. Also, it maximizes the number of viewer nodes that can be accepted in the tree within the fixed height by pushing higher out-degree viewers towards the root (flatter trees).

Algorithm 1 Degree Push Down Algorithm

Input: u, $oDeg_{S_i}^{u}$
Algorithm:
set $Q1 = \{root\}$;
repeat
  $Q2 = Q1$;
  for all $z \in Q2$ **do**
    if ($oDeg_{S_i}^{u} > oDeg_{S_i}^{z}$) or ($oDeg_{S_i}^{u} == oDeg_{S_i}^{z}$ and $C_{obw}^{u} > C_{obw}^{z}$) **then**
      replace z by u;
      $\boldsymbol{Child_{S_i}^{u}} = \boldsymbol{Child_{S_i}^{u}} \cup z$.; return 1;
    end if
    $Q1.dequeue(z)$;
    for all each child $ch \in \boldsymbol{Child_{S_i}^{z}}$ **do**
      $Q1.enqueue(ch)$
    end for
  end for
until ($Q1$ is empty)
return 0;

Algorithm 1 explains the 4D TeleCast degree push down algorithm. The algorithm is executed for each

accepted stream in the view request. It uses the viewer id u and its out-degree $oDeg^u_{S_i}$ (computed in the previous section) as inputs. We use two priority queues $Q1$ and $Q2$ that store the viewers in ascending order of their out-degrees at each level of the streaming tree. $\boldsymbol{Child^u_{S_i}}$ defines the set of child viewers for stream S_i at u. For empty child we put out-degree -1. Starting from the root, the algorithm looks for viewers z such that ($oDeg^u_{S_i} > oDeg^z_{S_i}$) or ($oDeg^u_{S_i} == oDeg^z_{S_i}$ and $C^u_{obw} > C^z_{obw}$). If found, it is replaced by u (which also updates $\boldsymbol{Child^u_{S_i}}$) and the replaced viewer is added as another child of u. If the algorithm returns 0 for a stream, the stream is requested from the CDN provided that current usage of the CDN is less than the maximum capacity allowed, otherwise the stream is rejected. An example of 4D overlay formation is shown in Figure 9.

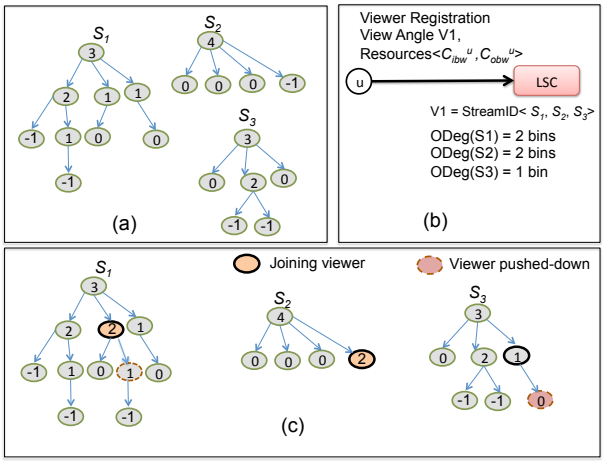


Fig. 9. (a) Multi-stream overlay before viewer u joins, (b) u sends join request to LSC and LSC computes the out-degree for the accepted streams, and (c) finally the overlay is formed by pushing lower-degree nodes down in the overlay.

The overlay ensures that viewers with higher outbound bandwidth receive streams with lower end-to-end delay, which provides an incentive to the viewers to engage more bandwidth in their session. The property can be described as follows.

Overlay property: *If viewers $u1$ and $u2$ request for the same view $v = [S_1, S_2, \cdots S_n]$ under the same LSC, then if $u1$ is in higher layer than $u2$ in the same streaming tree for one stream, then it is in higher layer compared to $u2$ for all other streams.*

We can easily prove it. Since, the outbound bandwidth is allocated in a round-robin manner in order of stream priorities and the priorities are always fixed inside a group (defined by view), the viewer with higher bandwidth always assigns higher outbound bandwidth to a stream compared to the other viewers with lower bandwidth inside the same view group. Moreover, the degree push down algorithm always puts the viewer with higher bandwidth closer the root. Therefore, for a certain view group, the viewers with higher out-degree becomes closer to the root compared to the viewers with lower out-degree and this is true for all the streaming trees they are subscribed to.

Once the topology construction is completed for all accepted streams, LSC sends the set of accepted streams and the overlay information back to the viewer. The overlay information includes the parent ($Parent^u_{S_i}$) and children ($\boldsymbol{Child^u_{S_i}}$) IDs along with their end-to-end delay and layer information for each accepted stream.

V. View Synchronization

A. View Synchronization Problem

The goal of view synchronization is to preserve the multi-stream dependencies at the viewer with the given bandwidth constraints. If the dependencies can not be preserved for a stream, then the bandwidth used by that stream should be made available to be used by other viewers. Suppose, at viewer u, the set of accepted stream in the request v_m is $\{S_1, S_2, \cdots S_j\}$. $d^u_{S_i}$ is the end-to-end delay of stream S_i at u perceived by the overlay structure. Though, the overlay construction problem bounds $d^u_{S_i}$ (where $d^u_{S_i} << d_{max}$), it does not provide any bound on $|d^u_{S_i} - d^u_{S_k}|$ $(1 \leq i, k \leq j)$. If the viewer maintains a buffer of length d_{buff} for each requested stream, to represent a synchronous view, the following constraint must be fulfilled: $|d^u_{S_i} - d^u_{S_k}| \leq d_{buff} + d_{skew}$, where d_{skew} defines visually allowed maximum inter-stream skew (i.e., the maximum unnoticeable inter-stream skew at the display). For ease of explanation, we use $d_{skew} = 0$.

B. Proposed Solution

To solve view synchronization problem, viewers first need to understand the stream end-to-end delay for the list of accepted streams after joining into the streaming overlay. Therefore, we introduce delay layer hierarchy in the P2P dissemination layer. We modify the viewer buffer architecture to support this layer hierarchy, and finally we show how we use stream subscription to solve the view synchronization problem.

1) Delay Layer Hierarchy: The purpose of delay layering is to understand the streams' position at the viewer in terms of end-to-end delay in the overlay. Each layer is identified by the delay value and bounded by a delay duration called τ. We define $\tau = \frac{d_{buff}}{\kappa}$, where $\kappa \geq 2$. κ defines the layer width. Viewers at the higher layers (with lower layering index) for a stream receive frames with lower delay and the viewers at the lower layers (with higher layering index) receive frames with

higher delay. A viewer can be in multiple layers; one layer for each requested stream. Therefore, the layering architecture can be represented as several concentric circles (one for each layer) with the producer and the CDN in the center as shown in Figure 10(a).

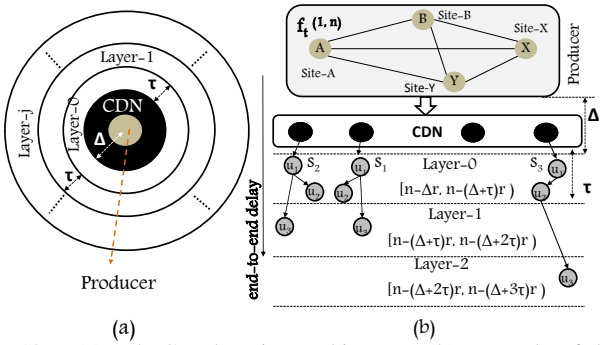


Fig. 10. (a) Tele-Cast layering architecture, (b) Example of delay layering showing the distribution of frame numbers at different layers at time t.

Definition of Delay Layer. To define formally, suppose, Δ indicates the maximum delay a frame takes to get delivered at any viewer via CDN after it is captured at the producer. Therefore, according to the delay layer architecture, the viewers at Layer-y receive streams with end-to-end delay bounded by $[\Delta + y\tau, \Delta + (y+1)\tau)$, where $y \geq 0$. Continuing the example shown in Figure 7, an example of delay layer hierarchy for viewers u_1, u_2, and u_3 requesting v_1 is shown in Figure 10(b). Depending on the end-to-end delay, u_3 is in Layer-1 for stream S_1 and S_2, but in Layer-3 for stream S_3.

If the frame rate is r for stream S_1^A, according to the layer architecture, at time t, viewers at Layer-y receive frames with frame numbers between $[n1, n2)$, where n is the latest captured frame number at the producer for stream S_1^A, $n1 = n-(\Delta+y\tau)r$, $n2 = n-(\Delta+(y+1)\tau)r$. Figure 10(b) shows the corresponding layering hierarchy for $y \in \{0, 1, 2\}$.

How to Compute Delay Layer for a Stream. When a viewer is added into the dissemination overlay, the layer (i.e., the lowest layer index) of the viewer for a requested stream (interchangeably we term it as the layer of the requested stream) is computed using the end-to-end delay of that stream at its *parent* in the overlay (from which the viewer receives the requested stream), the processing delay inside the parent (due to internal processing and buffering) and the network propagation delay from the parent to the viewer. If d_{prop} defines the network propagation delay between u and its parent ($Parent^u_{S_i}$) for stream S_i, $d_{S_i}^{Parent^u_{S_i}}$ is the end-to-end delay of stream S_i at $Parent^u_{S_i}$, and δ indicates the processing delay at the parent, then the layer of viewer u for stream S_i ($Layer^u_{S_i}$) is computed as follows:

$$Layer^u_{S_i} = \lfloor \frac{d_{S_i}^{Parent^u_{S_i}} - \Delta + d_{prop} + \delta}{\tau} \rfloor \quad (1)$$

In our evaluation, we assume $d_{S_i}^{CDN} + d_{prop} + \delta = \Delta$ for the viewers with CDN parents, i.e., the summation of end-to-end delay from the producers to the CDN and the CDN to its first children take approximately the constant time for each stream and it is equal to Δ. Therefore, the viewers who receive streams directly from the CDN can always achieve the highest layer, Layer-0. However, this constraint can be easility relaxed by considering separete Δ value for each producers' stream. We skip the details here.

How to Modify Delay Layer for a Stream. Layer modification at the viewer for a particular stream can be done simply by requesting frames back in time or ahead in time from the parents. However, due to the bound on the propagation and processing delay from the parents, the viewers cannot decrease the layer indexes (i.e., move to the higher layer) from the value measured by Equation 1. If a viewer is at Layer-y for stream S_i, at any time t, it can switch to Layer-x by requesting the streams starting from the frame number n' from its parent, where n' is defined as follows:

$$n' = n - (\Delta + (x+1)\tau)r + (d_{prop} + \delta)r + d_{prop}r + \Re \quad (2)$$

Here n is the latest frame number at the producer at time t (collected from the GSC monitoring), r defines the frame rate (also collected from the GSC monitoring component) and $\Re$ generates an offset between $[0, \tau r]$. Using $\Re$, we can chose viewers' position inside the desired layer boundary. The term $(d_{prop}+\delta)r$ in Equation 2 considers the total propagation delay from the parent and the third term $d_{prop}r$ is added since the request to update the stream layer takes additional d_{prop} time to get delivered to the parent. The parent then streams at the media rate starting from n'.

Since, the change in delay layer requests frames of different time period from the parents of the overlay tree, viewers needs to modify their buffering structure to cache 3D frames. Below we discuss the 4D TeleCast viewer buffer architecture.

2) Extension of Viewer Buffer: We extend the single-stream based buffering architecture [8], [9], [11] used in PPLive and CoolStreaming for multi-stream scenario. Each viewer maintains a local buffer at the gateway. For simplicity, we consider separate local buffers for different streams. The architecture of a viewer's local buffer is shown in Figure 11, where the viewer is subscribed to a view with two media streams: S_1, and S_2. At the *Media Playback Point (MPP)* in the Figure, the renderer picks up the synchronized frames (where the difference between the origin timestamps is less than d_{skew}) of S_1 and S_2 from the respective buffers (between MPP and buffer end) and sends them to the display. There can be a separate playback buffer inside

the display for smooth playback, which we ignore in our analysis. This is a rather conservative assumption that underestimates the system performance. For the rest of the paper, we use the term "buffer" to indicate the part of the local buffer from buffer end to the MPP and "cache" to indicate the part from MPP to the buffer head. d_{cache} is the caching delay and $d_{buff} << d_{cache}$.

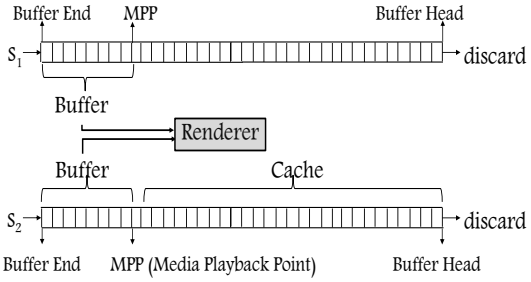


Fig. 11. Viewer's local buffer architecture in 4D TeleCast.

The frames stored in the buffer and cache are both available to support other viewers, while the frames stored in the buffer are only used in local media playback at the renderer. At this point, we propose our first property about the delay layer hierarchy.

Layer Property 1: *A viewer u with end-to-end delay $d^u_{S_i}$ for stream S_i can share streams of Layer-$\lfloor \frac{d^u_{S_i}-\Delta+d_{prop}+\delta}{\tau} \rfloor$ to Layer-$\lfloor \frac{d^u_{S_i}-\Delta+d_{prop}+d_{cache}+d_{buff}+\delta}{\tau} \rfloor$ to any child viewer at distance d_{prop}.*

In case of CDN parents, the distribution storage is very large and hence we assume that the CDN can share any layers of streams to its direct children. This layer property can be easily proved by the definition of delay layer hierarchy.

For simplicity, in our current work, we assume $d_{cache} = d_{max} - \Delta - d_{buff}$. Since, d_{max} is the maximum possible end-to-end delay at the viewers, the value of maximum acceptable layer index is bounded by $\frac{d_{max}-\Delta}{\tau}$. Therefore, this value of d_{cache} ensures that any viewer can support any acceptable layers of its child viewers into the TeleCast overlay streaming tree.

3) Stream Subscription: The stream subscription process works locally at each viewer after it joins into the overlay. It includes two steps: 1) finding the layer index for each accepted stream, and 2) bounding the differences in layer indexes so that the delay differences are bounded by d_{buff}. At this point, we present our second property about the delay layer hierarchy.

Layer Property 2: *A viewer (u) can render dependent streams $(S_1, S_2, \cdots S_n)$ synchronously at the display if the differences in the layering indexes for those streams are less than or equal to κ, i.e., $|Layer^u_{S_i} - Layer^u_{S_k}| \leq \kappa$, where $i, k = 1, 2, \cdots n$.*

We can easily prove this property. According to the layer definition, if $|Layer^u_{S_i} - Layer^u_{S_k}| \leq \kappa$, the difference between the end-to-end delay of stream S_i ($d^u_{S_i}$) and the end-to-end delay of stream S_k ($d^u_{S_k}$) at u must be bounded by $\kappa\tau$ (where τ is the size of each layer). Also, we define $\tau = \frac{d_{buff}}{\kappa}$. Therefore, by combining these two equations,

$$|d^u_{S/-i} - d^u_{S_k}| \leq \kappa\tau \leq d_{buff}$$

As we mentioned before, if the inter-stream delay can be bounded by d_{buff}, then the renderer can pick the dependent frames from the respective buffers and display a consistent virtual space.

After a viewer joins the overlay structures for j streams, it computes the minimum layer indexes $Layer^u_{S_i}$ for each stream S_i using Equation 1 ($1 \leq i \leq j$). If $|Layer^u_{S_i} - Layer^u_{S_k}| > \kappa$, then $|d^u_{S_i} - d^u_{S_k}| > d_{buff}$, which violates the view synchronization constraint. To bound the differences by κ, the viewer first finds the maximum layer index $Layer^u_{min} = max(Layer^u_{S_1}, Layer^u_{S_2}, \cdots Layer^u_{S_j})$. The updated layer index of each stream S_i is then computed as: $Layer^u_{S_i} = max(Layer^u_{S_i}, Layer^u_{min} - \kappa)$. We call this process as a *layer push-down*. In case of the layer push-down of a stream, the request for streaming are sent to the parent by computing the frame number using Equation 2, otherwise the parents are requested to send frames from their buffer end.

Next, the viewer sends the end-to-end delays to each child in $\boldsymbol{Child^u_{S_i}}$ for each forwarded stream S_i. Once the child viewer ch receives this information, it computes the delay layer index (x) that it can achieve using Equation 1. If $x > Layer^{ch}_{min}$ (the maximum layer index in ch), a new subscription process is started since the delay bound may be violated. However, if $x \leq Layer^{ch}_{min}$, then no layer subscription process is required, beacuse the parent viewer is still able to support ch with its current layer index of the stream.

It is important to note that the layer modification (if any) in the push-down process at the child viewer also needs to be propagated to its children, who may initiate another subscription process. Therefore, when a new viewer joins, the overlay may initiate a chain of subscription processes. Figure 12 shows an example of layer modification in viewer u_2 when a new viewer u_4 joins. Due to the layer modification, the layer bound ($\kappa = 2$) is violated. So, a layer push-down is required for stream S_2 in u_2. However, we can easily prove that due to the nature of overlay construction, it will not create any *cyclic impact*, which means that if a viewer u_1 starts the initial subscription process in the chain, it does not receive the layer update request along this chain.

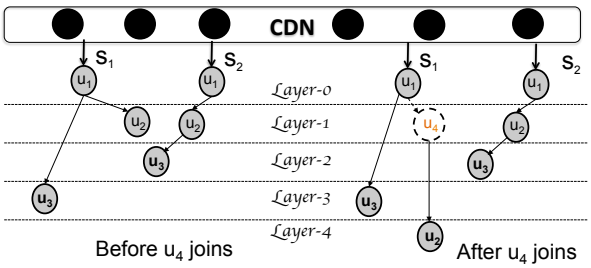


Fig. 12. Example of layer push-down in u_2 when a new viewer u_4 joins.

Also during the layer push-down using Equation 2, we apply $\Re = \tau r$ (i.e., position the children at the top of the modified layer boundary) so that the push-down fades out (reduce the number of layer modification) in the subsequent children.

VI. System Adaptation

View Change Adaptation. When a viewer requests for a view change, there is a change in the list of streams the viewer requested before. Let us continue the example shown in Figure 7(b). In case of a view change from v_1 to v_2 by u_2, the viewer u_2 needs to leave the streaming tree of S_1, S_2 and S_3 inside the group for v_1 and join the streaming tree of S_2, S_3, and S_4 inside the group for v_2 under the same LSC. Since, the joining process takes time (up to several seconds), to allow quicker view change, TeleCast streams the new requests from the CDN (i.e., after the view change request, u_2 receives S_2, S_3 and S_4 from CDN inside the group for v_2), while the traditional joining process (bandwidth allocation + overlay formation + stream subscription) is performed in background. Once the joining process is done, the streams are served according to the join overlay structure. Similar algorithm is used in LiveSky [22] for changing channels in live streaming.

However, the view change may create victim viewers. For example, in Figure 7(b), when u_2 switches view, u_3 becomes victim for stream S_3. To recover quickly, the victim viewers are also supported from the CDN at their current delay layer, while the LSC finds the positions for them into the overlay using the degree push down algorithm. The other portions of the overlay remains same (e.g., if u_3 has a child viewer in stream tree S_3, it will still be connected as the child of $u3$). 4D TeleCast uses same algorithm to recover the victim viewers in case of failures (due to viewers departure or failures).

Delay Layer Adaptation. Before any layer push-down happens, if the layer index of any stream at the viewer is higher than the maximum allowed layer index in the system, the viewer drops the subscription of that stream provided that the parent of the stream is CDN. However, if the parent is another viewer, then LSC first tries to provision the stream from the CDN provided that the CDN has available bandwidth.

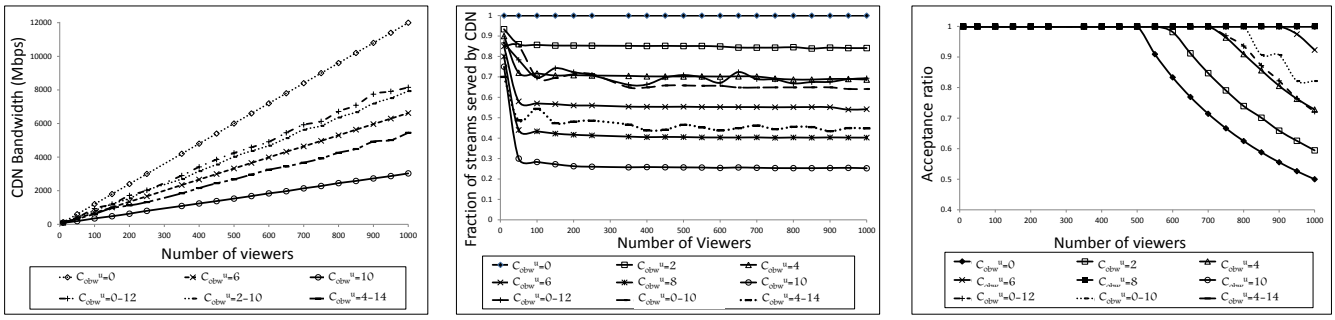
Due to the network dynamism, viewers also periodically monitor the end-to-end delay of all streams in the requested view and updates their layer indexes accordingly using Equation 1. If κ bound is violated at any point in time, the stream subscription process is initiated to bound the view synchronization. Each viewer also continuously monitors the layers of its parents for all accepted streams. If the parent layers for all streams move up, the viewer also move up the subscription of all streams. This ensures that the viewers continue to watch requested streams as quickly as possible keeping the dependencies preserved.

VII. Evaluation of 4D TeleCast

We evaluate 4D Tele-Cast using a discrete event simulator. For the experimental setup, we use configurations of system and application components from TEEVE [19], an advanced 3DTI system. We use 2 producers with 8 camera streams at each site representing the content producers. We simulate a CDN that creates minimum e2e delay of 60sec (i.e., $\Delta = 60$sec) from producers to the viewers. The number of viewers are varied from 10 to 1000. The delay between them are obtained from $4-$hours PlanetLab traces [14].

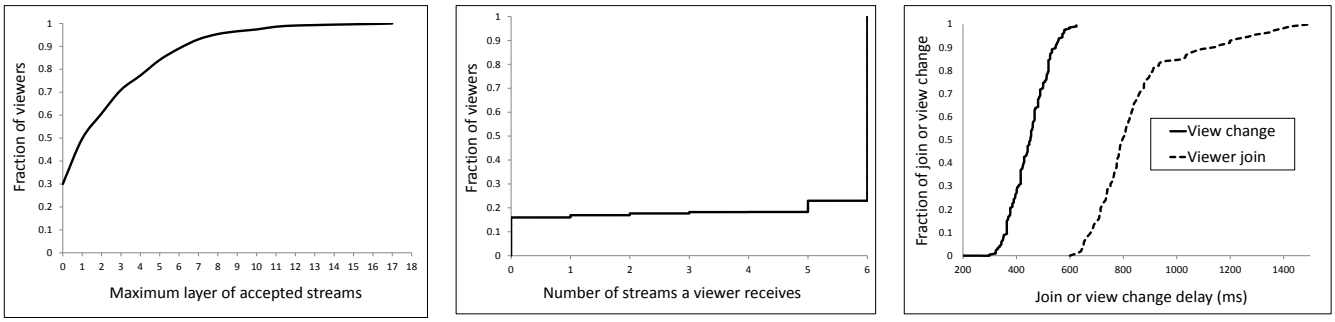
For each producer stream, we use traces collected from a TEEVE session [18], where two remote participants virtually fight with each other using light sabers. Each stream is bounded by 2Mbps bandwidth requirement. Each viewer requests a view that includes 6 streams; 3 from each producer. We assume each viewer has 12Mbps inbound bandwidth, but the outbound bandwidth (C^u_{obw}) varies from 0 to 14Mbps. The acceptable end-to-end delay at the viewer is bounded by 65 sec (i.e., $d_{max} = 65$sec). The buffer size is 300msec and the cache size to allow peer sharing is 25sec. We fix $\kappa = 2$.

Performance of Overlay Construction. The goal of solving overlay construction problem is to maximize the acceptance ratio (ρ) and minimize the CDN usage. We change the viewers outbound bandwidth from 0 to 14Mbps. For different values of outbound bandwidth, Figure 13(a) shows the CDN bandwidth requirements to support all requested streams (i.e., to achieve $\rho = 1$). When the viewers do not provide any outbound bandwidth, all requests are served from the CDN (case when $C^u_{obw} = 0$). Most of the cases, the joining viewers provide some bandwidth to allow forwarding. As the figure shows, even if the viewers contribute about 0 to 12Mbps bandwidth uniformly, the required bandwidth from the CDN to support all requests using the hybrid



(a) CDN bandwidth requirements (b) Fraction of requests served by CDN (c) Request acceptance ratio

Fig. 13. Performance of 4D TeleCast overlay construction and content distribution. CDN capacity is bounded to 6000Mbps in (b) and (c).



(a) Layers of viewers (b) Number of accepted streams (c) 4D Telecast overhead

Fig. 14. (a) The distribution of layers of accepted streams at the viewers, (b) The distribution of the number of accepted streams at the viewers, and (c) The overhead of 4D TeleCast for viewer join and view change.

structure is around 6000Mbps. We allocate this amount of outbound bandwidth to the CDN for rest of the experiments.

To understand how much savings in cost we make in 4D TeleCast due to the priority based bandwidth allocation and degree push down based overlay construction, we plot the fraction of requests served by CDN while varying the viewers outbound bandwidth in Figure 13(b). When the viewers' bandwidth varies between 4 to 14Mbps or each viewer allocate at least 8Mbps outbound bandwidth, about 55% or higher requests are served from the P2P.

If we bound the CDN capacity, some of the viewer requests may not go through due to the lack of bandwidth availability. To understand the performance of the 4D TeleCast in terms of the number of accepted requests with $C_{obw}^{cdn} = 6000$Mbps, we plot the acceptance ratio in Figure 13(c). When the viewers do not provide any outbound bandwidth the acceptance ratio becomes low. When the viewers contribute outbound bandwidth, in most of the cases, the acceptance ratio is very high. The system achieves perfect acceptance ratio when the viewers' bandwidth varies between 4 to 14Mbps or each viewer allocates at least 8Mbps outbound bandwidth.

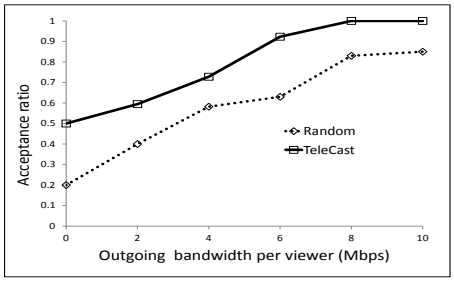
Performance of Stream Subscription. In this section, we show the performance of view synchronization using the delay layer hierarchy. We uniformly assign each viewer an outbound bandwidth between 0 to 12Mbps. The minimum subscription layer of the accepted streams at each viewer is shown in Figure 14(a). About 30% of the viewers are in Layer-0 and 80% of the viewers are watching the contents in Layer-4 or less. However, not all viewers receive 6 streams requested in the view. In case of bandwidth limitation, the lower priority streams are dropped. However, the inter-stream delay among the accepted streams are always less than 300msec due to the bound on the layering.

To evaluate the quantity of the accepted streams at each viewer, we plot the CDF of the viewers in terms of the number of accepted streams with similar setup. The result is shown in Figure 14(b). As the figure shows, most of the viewers (above 70%) receive all streams requested in a view with CDN capacity of 6000Mbps. Only 15% viewers do not receive any requested streams due to the bandwidth limitation.

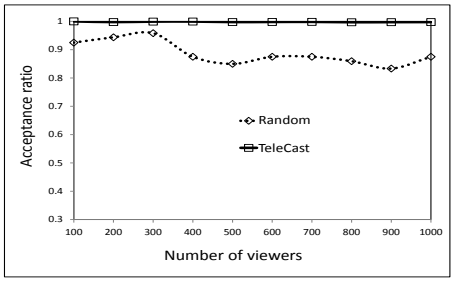
4D TeleCast Overhead. The overhead of 4D TelCast comes from the overlay construction and the stream subscription processes. The join of a viewer goes through three steps: bandwidth allocation, overlay construction and stream subscription. Hence, the joining delay includes the network delay and the processing delay at each of these steps. Figure 14(c) shows the joining delay in terms of the CDF of the number of viewers. The value varies up to 1.5sec. The value shown in the Figure 14(c)

does not include the buffering delay.

4D TeleCast improves the view change latency by serving the new streams due to the view changes directly from the CDN, while in the background, the new requests are added into the overlay using normal join steps. Therefore, the view change is satisfied quickly within 500msec.



(a)



(b)

Fig. 15. Comparison of TeleCast with Random routing scheme: (a) Varying outbound bandwidth per viewer, and (b) Scaling number of viewers.

Comparison with Random Dissemination. We evaluate our scheme against *random* routing scheme [19], which works well inside the communication among the producers. In Random routing, a joining node is randomly attached to another node, which can serve the request of the joining node. No clustering or pre-allocation of outgoing bandwidth of the node is done in the random routing scheme. We perform two sets of experiments. In the first experiment, we vary the amount of total outgoing bandwidth at each viewer node from 0 to 10Mbps for 1000 viewers, and in the second set, we increase the number of viewers joining the system having 2-14Mbps bandwidth. The results are shown in Figure 15. It can be seen from Figure 15(a) that our approach increases the acceptance percentage by 20% as compared to the random scheme. Also from Figure 15(b), TeleCast achieves very high acceptance ratio even under large number of viewers, 98% to 99% as compared to Random routing scheme (typically 80% to 88%)

VIII. Related Work

Nahrstedt *et al.* [13] first envisioned an *Internet Interactive Television*, where viewers can select multiple contents they want to watch together in a smart room and the contents are generated from different heterogeneous sources (e.g. mobile device, TV camera, etc.) by different entities distributed over the world. Each content is a single 2D stream and may not be semantically correlated to each other. However, in 4D TeleCast, single 4D content includes multiple 3D streams and they are highly correlated; therefore, needs to be synchronized to display a consistent high quality view. There are some prior works on 3DTI content dissemination. [21] considers the communication among producer sites. The contents usually are distributed in a mesh. Therefore, it supports very limited number of content producers. Later, Wu *et al.* [19] solves this problem using several tree based heuristic algorithms. However, the solutions consider that all producers (or viewers) are available when the session starts. This assumption is violated in case of widely distributed content viewers.

There are many IPTV solutions [2], [23] that work well for large scale dissemination, but none of them consider view change dynamics with multi-party compositions. Therefore, the challenges are different. This is also true to for other CDN-P2P based streaming solutions such as [9], [20] and [22].

Multicast routing algorithms have been well studied in [16]. [16] and [17] also point out that constructing a tree that optimize multiple metrics is an NP-complete problem. However, conventional tree based multicast may lead to a lower acceptance ratio([19], [7]). [10] considers path delays of the requested streams for constructing the 3DTI dissemination overlay. The solution works well for preserving the interactivity among the content producers, but does not create optimal overlay for supporting large number of concurrent viewers.

IX. Conclusion

We propose 4D TeleCast, a novel multi-stream content dissemination framework. It supports a large number of concurrent views as well as viewers while preserving the unique nature of 3DTI multi-stream dependencies by ensuring maximum effective resource utilization and view change capabilities. Our research results are significant for next-generation multi-stream and multi-view content distribution.

Acknowledgement. This material is based upon work supported by NSF Grant NetTS 0964081, and NeTSE 1012194.

REFERENCES

[1] Amazon CloudFront. http://aws.amazon.com/cloudfront/.

[2] PPLive. http://www.pptv.com/.

[3] M. Afergan, J. Wein, and A. LaMeyer. Experience with some principles for building an internet-scale reliable system. In *Proc. of WORLDS*, 2005.

[4] P. Agarwal, R. Rivas, W. Wu, and A. Arefin et al. SAS Kernel: streaming as a service kernel for correlated multi-streaming. In *Proc. of ACM NOSSDAV*, 2011.

[5] P. Agarwal, R. Rivas, and W. Wu et al. Bundle of streams: Concept and evaluation in distributed interactive multimedia environments. In *Proc. of ISM*, 2011.

[6] Akamai. http://www.akamai.com/.

[7] M. Castro, P. Druschel, and A. Kermarrec et al. SplitStream: high-bandwidth multicast in cooperative environments. In *Proc. of SOSP*, 2003.

[8] Y. Chen, C. Chen, and C. L. A measurement study of cache rejection in P2P live streaming system. In *Proc. of ICDCS*, 2008.

[9] X. Hei, C. Liang, and J. Liang et al. Design and deployment of a hybrid CDN-P2P system for live video streaming. In *IEEE Transactions on Multimedia*, 2007.

[10] Z. Huang, W. Wu, and K. Nahrstedt et al. Tsync: a new synchronization framework for multi-site 3d tele-immersion. In *Proc. of ACM NOSSDAV*, 2010.

[11] J. Kuo, C. Shih, and Y. Chen. A dynamic self-adjusted buffering mechanism for peer-to-peer real-time streaming. In *Journal of Intelligent Systems and Application*, 2011.

[12] B. Li, Y. Qu, Y. Keung, and S. Xie et al. Inside the new CoolStreaming: principles, measurements and performance implications. In *Proc. of INFOCOM*, 2008.

[13] K. Nahrstedt and B. Yu et al. Hourglass multimedia content and service composition framework for smart room environments. In *Elsevier Journal on Pervasive and Mobile Computing*, 2005.

[14] PlanetLab 4hr traces. http://www.eecs.harvard.edu/syrah/nc/sim/pings.4hr.stamp.gz.

[15] S. Ratnasamy and Mark Handley et al. Topologically-aware overlay construction and server selection. In *Proc. of Infocom*, 2002.

[16] B. Wang and J. Hou. Multicast routing and its QoS extension: problems, algorithms, and protocols. In *IEEE Network*, 2000.

[17] Z. Wang and J. Crowcroft. Quality of service routing for supporting multimedia applications. In *IEEE Journal Selected Areas in Communications*, 1996.

[18] W. Wu, A. Arefin, and Z. Huang et al. I'm the Jedi! - A case study of user experience in 3D tele-immersive gaming. In *Proc. of ISM*, 2011.

[19] W. Wu, Z. Yang, and K. Nahrstedt et al. Towards multi-site collaboration in 3D tele-immersive environments. In *Proc. of ICDCS*, 2008.

[20] D. Xu, S. S. Kulkarni, and C. Rosenberg et al. A CDN-P2P hybrid architecture for cost-effective streaming media distribution. 2004.

[21] Z. Yang, W. Wu, and K. Nahrstedt et al. Enabling multi-party 3D tele-immersive environments with viewcast. In *ACM Transactions on Multimedia Computing, Communications and Applications*, 2010.

[22] H. Yin, X. Liu, and T. Zhan et al. Design and deployment of a hybrid CDN-P2P system for live video streaming: Experiences with LiveSky. In *Proc. of ACM MM*, 2009.

[23] X. Zhang, J. Liu, and B. Li et al. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In *Proc. of INFOCOM*, 2005.