F.Live

Towards Interactive Live Broadcast FTV Experience

Shannon Chen, Zhenhuan Gao, and Klara Nahrstedt
University of Illinois at Urbana-Champaign
{cchen116, zgao11, klara}@illinois.edu

***Abstract*— Free-viewpoint television (FTV) is a visionary application that provides immersive experience to the audience with the freedom of changing viewpoint during the video playout. However, live broadcasting and user interaction do not coexist in existing FTV systems. In this paper, we propose F.Live, a framework of FTV content dissemination that supports user-initiated viewpoint changing for live broadcasting. Simulation result of a large-scale experiment, based on camera array settings of existing Nagoya systems and EyeVision System, shows that F.Live is capable of supporting 100,000 concurrent audiences with free-viewpoint low user interaction latency and feasible bandwidth requirements.**

Keywords—broadcasting, live streaming, FTV

I. INTRODUCTION

Free-viewpoint television (FTV) is a visionary application that provides immersive experience of a broadcasted physical event to the audience with the freedom of changing viewpoint during the video playout. Through depth estimation and interpolation among multiple raw video streams captured from different angles, FTV system can render a 3D space where the audience can view the event from arbitrary viewpoints as if they were physically in the same studio with the filmed objects. Large-scale camera array systems installed in outdoor stadiums and theaters are also possible for capturing sport events [1] and concerts [2] from a 360 degree view.

Yet two desirable characteristics: interactivity and live broadcasting fail to coexist in current FTV systems. Generally, the content delivery chains of existing FTV prototypes can be classified into two types. Type-1 is designed for broadcasting live performances (Fig. 1a). Under this scenario, raw streams captured by a number of cameras are interpolated to create virtual camera streams. These virtual and real streams altogether grant smooth transition between arbitrary viewpoints, hence create a free-viewpoint immersive experience. A commercialized system which implements this delivery chain has been deployed for live broadcasting of Super Bowl [7].

However, the term "free" in this type of delivery can be misleading because end users are deprived of interactively choosing their own viewpoint. The freedom of dynamic viewpoint selection is still in the hand of the service provider. A director of the program selects the viewpoint for the audience, so eventually only one view will be broadcasted at any given time.

Type-2 FTV delivery chain is suitable for pre-recorded programs (Fig. 1b). Under this scenario, raw video streams captured by different cameras are aggregated to create one huge free-viewpoint stream. The free-viewpoint stream is delivered to the audience as a whole. During the playout, the audience switches viewpoint arbitrarily by giving commands on-the-fly to their local display systems. The decoding module of the system only decodes necessary data for rendering scene related to user's viewpoint decision. Adopting this delivery chain, a working prototype comprised of 100 cameras has been developed by Tanimoto lab at Nagoya University [3].

However, delivery of the huge free-viewpoint stream makes bandwidth and computation requirement inevitably high. Thus, the delivery chain is not suitable for live broadcasting or any kind of low-buffer real-time streaming through modern data network. For the Nagoya FTV system to stream with HDTV resolution (1080p), the estimated required bandwidth is 1.6 Gbps [4][5], which largely surpasses regular networking capacity. With multi-view video coding (MVC) [6], the bitrate can be lowered by 20~30% but still in the gigabit magnitude.

In this work, we propose a new framework and delivery chain for live and interactive broadcast FTV, called F.Live. As we saw in the previous two types of FTV delivery chains, at any given time, a user can watch the free-viewpoint video from only one chosen view. The thick lines in Fig. 1 indicate the aggregated free-viewpoint stream which contains massive information. We see that the aggregated stream never reaches the end user, for she only needs content of one viewpoint at any given time. Thus, the earlier the decision of viewpoint is made, the more bandwidth and computation resource can be saved in the overall delivery. Based on this observation, we propose a new view-based delivery chain. The new chain introduces a session manager entity to coordinate the interactive user requests and distribution of raw video streams to different end users right after they are captured (Fig. 2). No aggregation is made throughout the delivery chain. The rendering module is moved to the audience side so that the users only have to receive minimum necessary streams to construct the scene they wish to see. Comparing to the bandwidth saving of MVC (20~30%), the view-based delivery chain can save at least 50%.

Since the aggregation module is obviated, dissemination of raw streams becomes more elastic. Each camera in the producer site can be seen as independent content producer entity in the system model. In addition, session manager also becomes an independent entity which sits on top of the dissemination network to control the content flows. Thus, content producers, audience sites, and the session manager form a distributed P2P delivery overlay which helps alleviate the transmission burden of producer site by content sharing. Due to its unique service

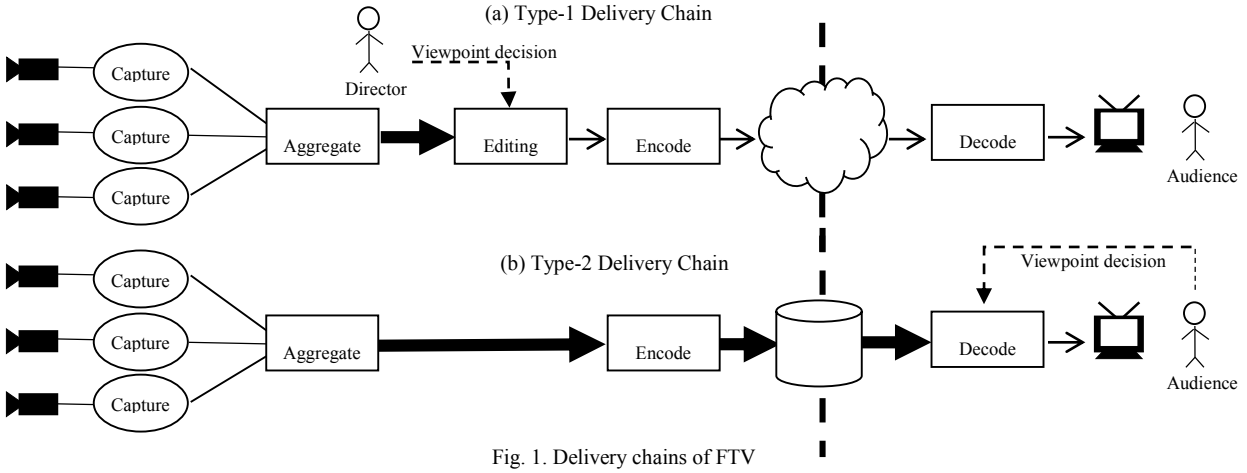

Fig. 1. Delivery chains of FTV

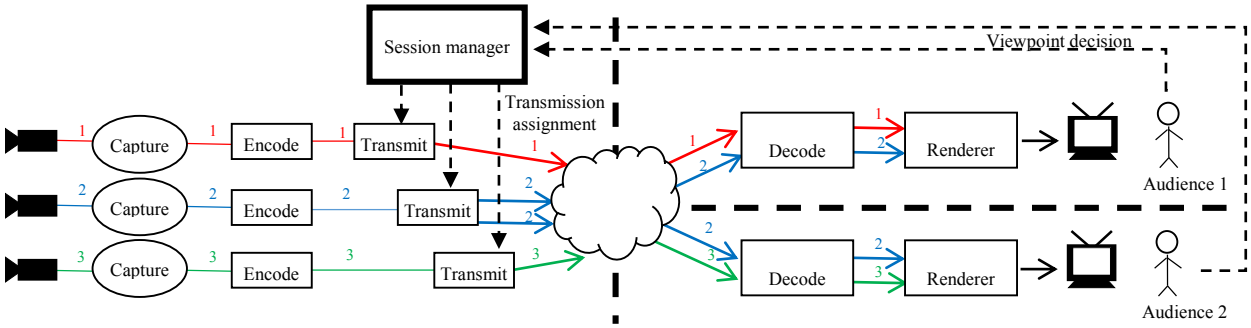

Fig. 2. View-based delivery chains of F.Live

type, the framework of F.Live poses a non-conventional multicast publish-subscribe problem [29] with additional cross-stream synchronization requirements.

With the objective of minimizing the bandwidth requirement while preserving interactivity in live broadcasting, we propose an overlay planning algorithm which lowers the synchronization overheads of contents downloaded from different peers down to 100ms scale with live broadcasting delay complies to commercial standard. Feasibility and performance of F.Live are evaluated by emulating existing camera array systems (Nagoya systems [3][16] and EyeVision [1]) on top of the proposed dissemination infrastructure with 100,000 concurrent audiences.

The rest of this paper is organized as follows. In §2 we review related works on IPTV and free viewpoint video multicasting. In §3 and §4 we introduce two existing FTV systems and setup the design agendas of F.Live. In §5 and §6, we introduce the distributed architecture of the proposed P2P framework along with the overlay planning algorithm. In §7 and §8, we evaluate the framework by simulating a large-scale audience group in F.Live. Finally in §9 we conclude the paper.

II. Related Works

Existing IPTV solutions [21][22] that work well for large scale dissemination do not consider view change dynamics with multi-party compositions. Since the content they disseminate is single-view 2D stream, the challenges are different from the scenario we are targeting. This is also true for other CDN-P2P based streaming solutions such as [17][23][24].

[26] and [27] provide detailed reviews on multicast routing algorithms. The authors further prove the NP-hardness of construction of dissemination tree that optimizes orthogonal metrics (e.g., latency and bandwidth). Thus, various heuristics have been proposed on application-layer multicasting in previous works. Among them, only few targets multi-view content dissemination [8][18][25][28].

[8] and [28] consider path delays of the requested streams for constructing the 3D multi-view 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. [18] proposes the 4D TeleCast framework which targets large-scale content multicasting. In order to accommodate the hundred-scale audience group, the authors propose to allocate the viewers into different classes with different delayed services and they do not target broadcasting of live events. [25] also aims at a multi-view multicasting system, to alleviate the bandwidth burden, audience's freedom of view is restricted to a certain region. The proposed system mimics the experience in an amphitheater, where audience can only turn their heads but not changing seats to adjust their viewpoints.

III. Existing FTV Platforms

A. FTV Broadcasting

From early 1990, the CMU robotic groups have been experimenting with multi-camera systems. The long-term goal of the Virtualized Reality [10] project was to develop a 4D (3D space plus time) event descriptor that covers a space with camera

array to virtualize real world events. During the pursuance, one of the systems developed was the EyeVision [1] replay system. The system was deployed in the Raymond James Stadium to cover Super Bowl XXXV. It used 30 high definition cameras, mounted apart at 80 feet above the football field, to cover 270 degree view of the stadium.

Depending on the viewpoint of interest, part of the 30 raw streams were interpolated to create a virtual stream shooting from an angle that no actual camera covered. Thus, the director could choose an arbitrary viewpoint to broadcast. Although EyeVision is used in broadcasting live events, it only served the purpose of game replay rather than live streaming [7]. The rendering complexity and the inferior interpolation quality made it infeasible for direct broadcasting. Most of the game therefore was still broadcasted like conventional TV service.

B. Interactive FTV

To enable interactive viewpoint selection, the FTV system, developed in Nagoya University [12], delivers to the end users a stream aggregation, called the ray-space data [11]. The ray-space data contains all the raw streams captured by the camera array. Thus, at the audience site, the end user can interact with the renderer by dynamically selecting her viewpoint of interest. However, with its highly sophisticated content, the decoding complexity becomes critical and the bitrate is inevitably high.

To tackle these problems, Fujii et al. adopt computer vision approaches to optimize capturing [15] and rendering [14] of content. Special purpose hardware for ray-space data acquisition [13] is developed to further accelerate data processing. Nevertheless, the multi-view video coding (MVC) techniques [6] they adopted only cut back bandwidth consumption by 20~30%. For the Nagoya FTV system, which comprised of 100 raw camera streams [3], the estimated bandwidth required to support 1080p HDTV service is still in 1.2~1.3 Gbps range with MVC [4][5].

Since the bandwidth requirement largely surpasses regular networking capacity. Ray-space data transmission is bound to be infeasible for live broadcasting or any kind of low-buffer streaming. Ray-space FTV content is hence only known suitable for store-and-play type of video services.

IV. Design Agendas of F.Live

Interactivity and live broadcasting are the two desired capabilities that ought to be coexisting in the next FTV system. Therefore we envision F.Live: a solution that includes capturing, dissemination, and rending of free-viewpoint broadcasting of live events. In order to achieve these goals, we propose two design agendas: feasible resource requirement and distributed sites.

A. Feasible Resource Requirement

One of the causes that incur failure of preserving interactivity in live broadcasting is the high bandwidth demand of FTV. The aggregated free-viewpoint content, which grants interactive viewpoint changing, consumes too much bandwidth so it cannot be transmitted to the end users as a whole (Fig. 1a) otherwise the live streaming capability has to be sacrificed (Fig. 1b). Despite the effort to deliver the aggregated free-viewpoint content, the audience only focus on one chosen view at a given time. Therefore, in order to alleviate the bandwidth requirement, early viewpoint selection is the first alteration we propose in the new view-based delivery chain of F.Live.

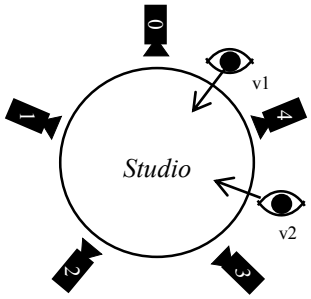

Fig. 3. Streams and user viewpoints

The view-based delivery chain (Fig. 2) obviates the aggregation module and replaces it by a session manager. The session manager receives viewpoint decision from an end user and assigns which raw streams should be transmitted to the user to fulfill this request. For example in Fig. 3, if user 1 and user 2 have chosen viewpoints v1 and v2, respectively, then the session manager will assign camera streams 0 and 4 to be sent to user 1; and streams 3 and 4 to be sent to user 2.

Although removing the aggregation module implies we can no longer benefit from multi-view video coding (MVC), the early viewpoint selection design actually brings us more bandwidth saving. The idea of MVC is to leave out similar graphical information between streams captured from similar viewpoints (e.g., stream 2 is similar to stream 1 and 3 in Fig. 3 due to adjacency). The compression gain is 20~30% more than frame-based compression [4]. Yet, the early viewpoint selection design in the new delivery chain brings more than 50% saving. Since the audience cannot see through objects, the streams they required is at most half of the total number. Any stream that covers the opposite side from the chosen viewpoint is unwanted (e.g., stream 1~3 for user 1). Therefore, from bandwidth aspect, removing the aggregation module is also a worthwhile decision.

B. Distributed Entities

Further bandwidth saving is done by P2P content sharing. Since the number of peers (audience sites) largely surpasses the number of contents (raw streams), substantial resource saving can be achieved by forming a content sharing network among the audience. In the previous example (Fig. 3), since stream 4 is required by both user 1 and 2, one user can receive it from the other to alleviate the outbound bandwidth of producer site.

However, the P2P-based delivery poses concern on interactive viewpoint selection. While the interactivity feature grants dynamic viewpoint changing initiated by end users, it also incurs dynamically changing structure of the P2P overlay. For example, in scenario of user 2 receiving stream 4 from user 1, if user 1 suddenly decides to switch her viewpoint and no long requires the stream, user 2 will be affected. This P2P churn could introduce latency to both users upon viewpoint changing if not properly handled. We will come back to the problem in §6.

Since raw streams need not to be assembled and aggregated in our view-based delivery chain, content producers (handler machines of each camera) become independent entities, and so

is the session manager. Communication between the audience and the session manager is unchanged. Yet, since the session manager does not need to be physically assembled with content producers, the assignment of stream transmission will go through the internet from session manager to camera handlers. Several session managers can coexist, each serving only its local audiences to distribute the communication burden. In a multi-manager scenario, content sharing is restricted by locality to avoid contention between session managers and to simplify the planning of transmission assignments.

V. Distributed System Model

In this section we introduce the system model of F.Live. As we mentioned in our design agendas, the system is distributed and is comprised of three types of entities: audience sites, content producers, and session manager.

A. Session Manager

Since a session manager does not touch the streams directly but only handles the control messages, it is the coordinator in the system model who plans the dissemination overlay. The hardware requirement is simple for it only needs a machine to 1) receive requests (viewpoint decisions) from the audience; 2) plan the P2P overlay and setup transmission assignments accordingly; and 3) send the assignments to the chosen sources.

Using Fig. 3 as an example, if an audience site *u* sends a viewpoint decision "v2" to session manager, the manager will translate it into subscription request of stream 3 and 4. Next, the manager finds sources that currently hold these streams. A source can be a content producer or another audience site since we adopt P2P sharing. After the manager finds the suitable sources, it sends transmission assignments to them, stating what streams the sources should send to audience *u*.

B. Audience Sites

The audience are consumers of the F.Live content. Thus, the hardware requirement of audience site includes a gateway machine, a display, and an input device that allows user to specify her viewpoint (e.g., a specialized TV remote control). The gateway machine has three tasks. The first one is stream sharing. It receives assignment from the session manager and executes reception and sharing of streams accordingly. The second task is handling user commands. Through the input device, user will specify her desired viewpoint. The information will be passed to the session manager by the gateway machine, so that the manager will plan the assignments accordingly. The last task of gateway is rendering. Since there might not be a stream which completely complies with the viewpoint selected by user (e.g., v1 in Fig. 3), adjacent streams will be interpolated to create the viewpoint of interest (e.g., stream 0 and 4).

C. Content Producers

Each content producer handles a camera in the camera array. The tasks of the producer are three-folds. First, the producer receives the stream captured by the camera and encodes it with frame-based compression codec (e.g., MPEG-TS). Second, it receives transmission assignment from session manager and sends the stream out to the specified audiences. Third, it stores the captured frames in a local storage for record.

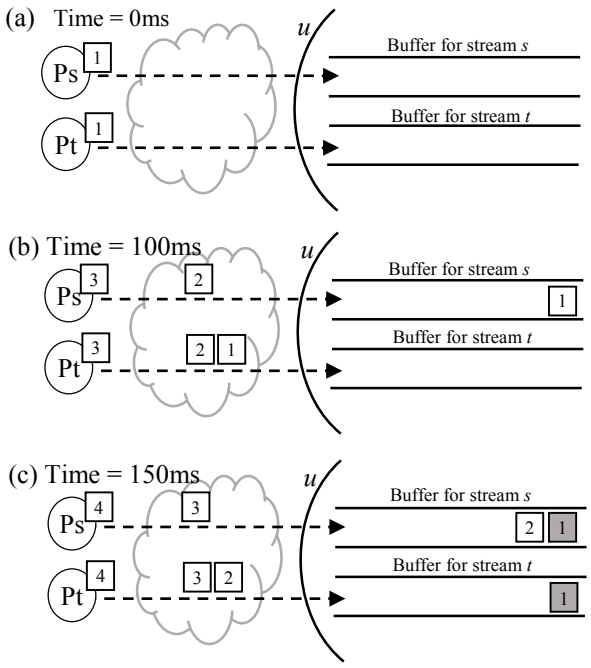

Fig. 4. Stream synchronization.

All content producers reside in the same producer site (e.g., Fox studio or Raymond James Stadium). The networking capability of a producer site is expected to be at gigabit magnitude. Since producer sites in our use case are cooperation studios, it is financially practical to assume higher end network infrastructure. Note that even with the high networking capacity, bandwidth saving via P2P architecture is still necessary. The reason is again financial. Renting dedicated gigabit carrier can be a substantial expense for F.Live content provider. Every bit per second reduced from the outbound bitrate is a penny saved.

VI. P2P Overlay Planning

Overlay planning happens with user churn (audience join and leave) and with user subscription changes (viewpoint changing). When these events happen, the structure of the content dissemination trees change. A proper overlay planning algorithm should reshape the trees to maintain the system performance with respect to the following objectives.

1) Interactivity. The performance of interaction is evaluated by the response time of our system to user's commands, including new user join and viewpoint changing.

2) Content freshness. Since F.Live is targeting live broadcasting, the time difference between when an incident is captured by a camera and when the audience actually sees it at their sites is an important metric.

3) Producer bandwidth consumption. The outbound bitrate of producer site is related to the financial cost for content provider to sustain F.Live. Thus, we have to balance the number of users that download directly from the producer site and users who share content with each other.

A. Preliminaries

Before we dive into details of the algorithms, a few notations and concepts must be introduced for the ease of explanation.

First, we define notation $u_1 \xrightarrow{s} u_2$ to denote that user 2 is downloading stream *s* from user 1. Note that, this relationship implies user 2 will never get a particular frame of stream *s* before user 1 does because each user only downloads a stream from one source. In the rest of the paper, we use users and audiences interchangeably, both refer to the audience sites in the system.

Second, due to propagation delay, a user cannot get the newest frame captured by the producer instantaneously. Thus, there exists a "frame elapse" between the newly captured frame and the frame being played out at the audience site. For example, in Fig. 4, a frame of stream *s* takes 100ms to travel from its producer (P_s) to user *u*; a frame of stream *t* takes 150ms to travel from its producer (P_t) to user *u*. If new frames are captured every 50ms (i.e., 20fps), it means that when the first frame of stream *s* finally arrives at *u*, that frame is actually captured two frames before the newest frame. Hence, we define frame elapse of stream *s* to user *u* as :

$$\mathrm{E_s}(u) = \text{newest\_frame\_number} - \text{received\_frame\_number}$$

Thus, $\mathrm{E_s}(u) = 3 - 1 = 2$ in Fig. 4b. Note that all subscribed streams have to be synchronized at the audience site so that they can be rendered together. In Fig. 4, user *u* subscribes to stream *s* and *t*. Consequently, user *u* has to wait until frames of *s* and *t* that were captured at the same time to arrive in its buffer before it can start rendering (Fig. 4c). Hence, we define the frame elapse of user *u* as:

$$\mathrm{E}(u) = \max_{s \in S'} \{\mathrm{E_s}(u)\}$$

where *S'* is the set of streams subscribed by user *u*. Thus, $\mathrm{E}(u) = \max\{\mathrm{E_s}(u), \mathrm{E_t}(u)\} = \max(2,3) = 3$ in Fig. 4.

Now we are ready to introduce the planning algorithm. The algorithm is three-folds, each handling audience join, audience leave, and viewpoint changing events. Pseudo codes of the algorithms are provided in the appendix. In the following we introduce the rationales of our designs.

B. Audience Join

From the example illustrated in Fig. 4, we know that the delay between a join request and a frame is finally rendered consists two parts. The first part is propagation delay (0~100thms), which is inevitable and not controllable from the application layer. The second part is synchronization delay (100~150thms), which is the waiting time for the same frames of all subscribed streams to arrive. The synchronization delay is occurred by the multi-subscription property of FTV and can be minimized with proper assignment of content sources. In formal form, the synchronization delay is determined by

$$U = \{u_s \mid u_s \xrightarrow{s} u \;\; s \in S'\}$$
$$sync\_delay = \{\max_{u_s \in U} \mathrm{E}(u_s) - \min_{u_s \in U} \mathrm{E}(u_s)\}/frame\_rate$$

where *u* is the newly joined user, *S'* is the set of streams it subscribes to, and *U* is the set of sources that will provide the streams. Hence, the objective of the join algorithm is to minimize the difference between frame elapses of chosen sources. A naive way would be to let the new user always download streams directly from the producers. This way, user will always get the freshest frames and the frame elapses of sources are likely to be similar. Yet this method will soon saturate the outbound bandwidth of producer site, which violates our objective. Thus, the new user should always download a stream from an existing user if possible.

The algorithm is comprised of two phases. The objective in the first phase is to ensure the freshness of content. We want the new user to be downloading the freshest content possible. We first select the user who holds the freshest content in each dissemination tree of each stream that the new user intends to subscribe. For example, in Fig. 5a, $S'=\{s_a, s_b, s_c\}$ and the dissemination trees are illustrated. The roots are producers and the other nodes are existing users. In this case, the users who hold the newest content in each tree would be u_1 (holding s_a), u_2 and u_5 (holding s_b), and u_3 (holding s_c). Note that the selected users must have available bandwidth to send stream to the new user. If not, we select the one with the second freshest content and so on. If there is no user in a tree who has available bandwidth to share, then the new user will directly download from the producer. Assume that in Fig. 5a the bandwidth of u_5 is already saturated, thus the list becomes $\{u_1, u_2, u_3\}$.

The objective of the second phase is to minimize the synchronization delay. By definition, this equals to minimizing the difference between the largest and the smallest frame elapses among the chosen sources. Hence, we start from the source with the largest elapse in the list provided by the first phase, and gradually include new sources that minimize current synchronization delay until all the subscription requests are fulfilled. Using Fig. 5a as an example, the list provided by phase one is $\{u_1, u_2, u_3\}$. We start from choosing u_3 as the source of s_c because $\mathrm{E}(u_3) = 10$ is the largest elapse in the list. Second, we choose u_3 also as source of s_b because this minimizes the synchronization delay (10-10=0). Last, we choose u_4 to be the source of s_a because it is the subscriber of s_a that has an elapse closest to the other chosen sources. Thus, the final synchronization delay is (10-5)/*frame_rate* = 250ms. Pseudo code of the join algorithm is provided in the appendix.

C. Audience Leave

Audience leave event can be classified into normal leave or abnormal leave. Normal leave is when the leaving user notifies the session manager before it disconnects, so that the manager can handle the children of the leaving user in the dissemination trees. Abnormal leave is caused by unexpected termination of the audience site. In this case, the children have to detect the incident and notify the session manager to be reassigned a new parent. Detection of parent failure is done by standard heartbeat approach by treating the incoming frames as keep-alive messages.

For orphan reassignment, we want the transition to its new parent to be seamless without any interruption in the playout. Thus, we have to find a new parent who is holding content that, after propagation delay between orphan and itself, can continue with the same frame elapse as the orphan. When the audience group is small, the probability of finding such new parent is small. In the case that no suitable new parent exists, the orphan is reassigned to the producer. As we mentioned in §5B, producer

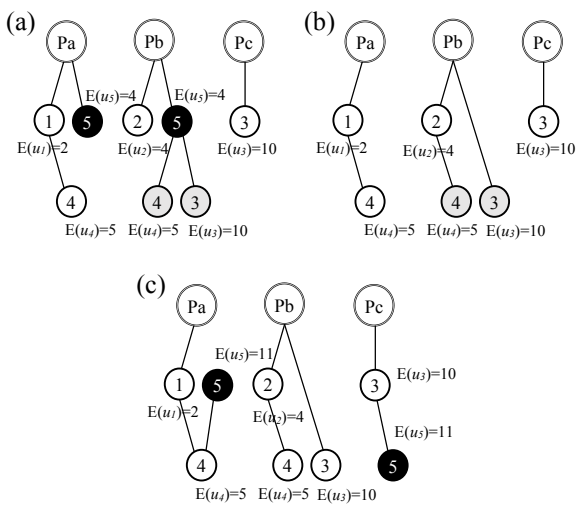

Fig. 5. Overlay planning

stores all the frames captured during the broadcast. Thus, it can simulate any frame elapse by providing old content to the orphan.

Transition of orphan reassignment is illustrated from Fig. 5a to 6b. Assume propagation delay will cause the frame elapsed to be increased by one (i.e., if $u_x \xrightarrow{s} u_y$ then $\mathrm{E_s}(u_y) = \mathrm{E_s}(u_x) + 1$.) In Fig. 5a, if u_5 leaves, it makes u_3 and u_4 orphans. After u_3 requests to be reassigned, it will download s_b directly from the producer since no existing user can continue its frame elapse. After reassigning u_4, it will be downloading s_b from u_2 (Fig. 5b).

D. Viewpoint Changing

The last part of the algorithm handles viewpoint changing. The procedure is straightforward for it is similar to leaving and then rejoining with a different subscription request.

However, direct invoking of previous leave and join algorithms can cause user to miss part of the content during viewpoint changing. For example, say before the viewpoint change event, the user has a frame elapse of 10 (i.e., user is watching content happened 500ms before). Since the objective of join algorithm is to deliver the freshest content possible, after the viewpoint change algorithm calls leave and join functions, the new elapse might be smaller than 10 (e.g., content happened 50ms before) which can cause user to miss at most ten frames. Thus, in our viewpoint change algorithm we only select sources that have larger elapses than the user who requests viewpoint change. This restricts the candidate sources to those who hold content that are older than or concurrent to the user's content.

Using Fig. 5a through 5c as an example, say u_5 in Fig. 5a issues a viewpoint changing request and wants to change her subscription from $\{s_a, s_b\}$ to $\{s_a, s_c\}$. The planning will happen as follows. Similar as the new user join procedure, users that hold the newest content in tree of s_a and tree of s_c in Fig. 5c are selected. However, this time we only select users who have elapses larger than u_5 before she leaves (i.e., $\mathrm{E}(u_5) = 4$ in Fig. 5a). Thus, for source of s_a, u_4 is selected instead of u_1. For source of s_c, u_3 is selected (Fig. 5c). In result, the frame elapse of u_5 after viewpoint changing is $\max\{\mathrm{E}(u_3), \mathrm{E}(u_4)\} + 1 = 11$.

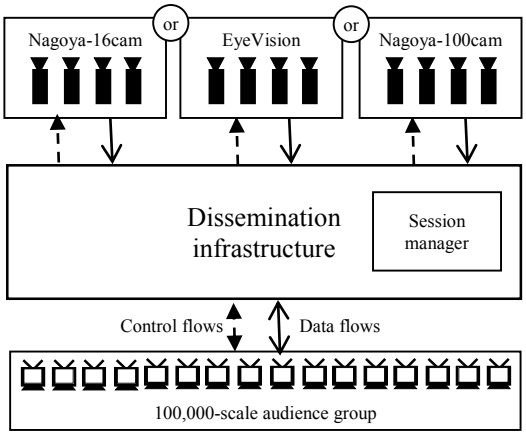

Fig. 6. Simulation architecture

VII. Simulation Settings

To evaluate the performance of our planning algorithms, we simulate an infrastructure that carries data (streams) and control messages (viewpoint) among audience sites, content producers, and session manager (Fig. 6). The infrastructure implements the view-based delivery chain proposed and the session manager handles overlay planning with our proposed algorithms.

Network Settings. We adopt real-world topology from Netmap [19] as the testbed of our simulation. Among the real hosts distributed around the world in the Netmap database, we randomly picked them to be our participating audience sites. The host-to-host delay is estimated based on the geographic distance between them [24].

Producer Site Settings. We emulate the producer site in F.Live with system requirements of the Nagoya-16cam system [16], EyeVison system [1], and Nagoya-100cam system [3] in three simulation scenarios. The video quality and bandwidth requirements are listed in Table I. Considering the advance of consumer electronics, we upgrade the video qualities of these system to HDTV standard in our settings.

TABLE I. Producer Site Settings

	Nagoya-16cam [16]	EyeVision [1]	Nagoya-100cam [3]
Cameras	16	30	100
Resolution	1080p	1080i	SDTV
Framerate	30 fps	30 fps	30 fps
Bitrate	12 Mbps	6 Mbps	2 Mbps

Simulation Scenario. The simulation scenario is a live FTV performance broadcasted by a producer site with hardware listed in Table I. During the performance, new users continuously join the audience group with random arrival time. We simulate the join events of the first to the 100,000th audience and random viewpoint changing events of joined audience.

Performance Metrics. The performance of F.Live is evaluated from three aspects, which correspond to the objectives of the overlay planning algorithm. First, we evaluate the interactivity of the user commands by measuring the synchronization delay. As we mentioned in the algorithms, synchronization of streams happens with user join and viewpoint changing, and it contributes to delay caused by these

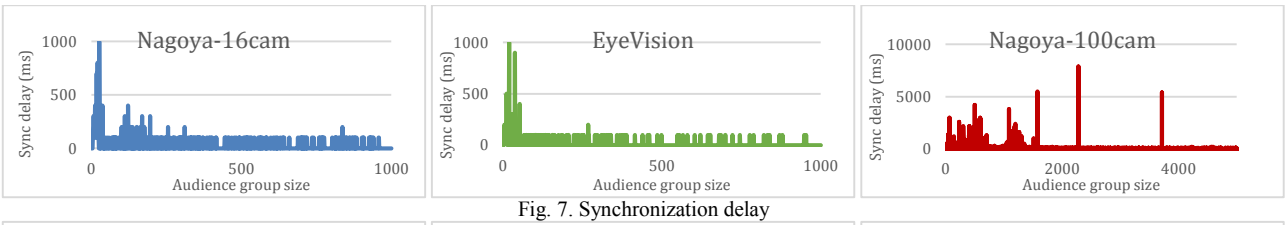

Fig. 7. Synchronization delay

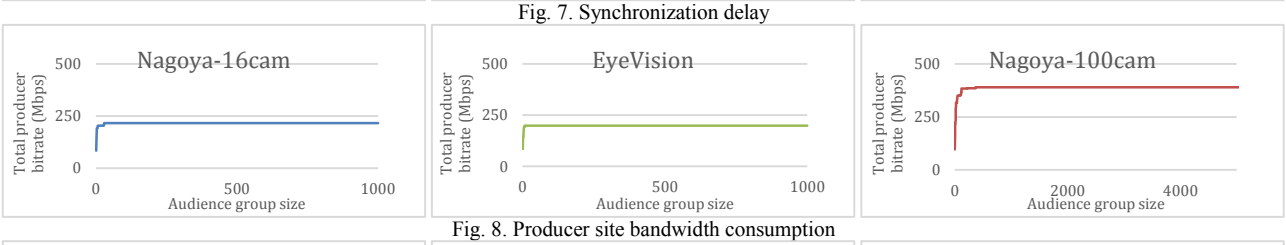

Fig. 8. Producer site bandwidth consumption

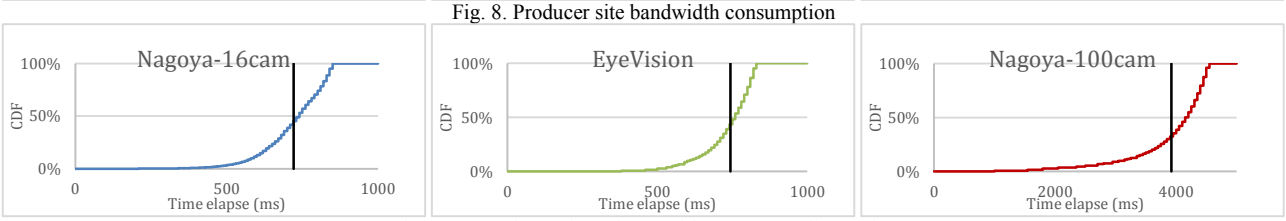

Fig. 9. CDF of user time elapse with 100,000 audience size

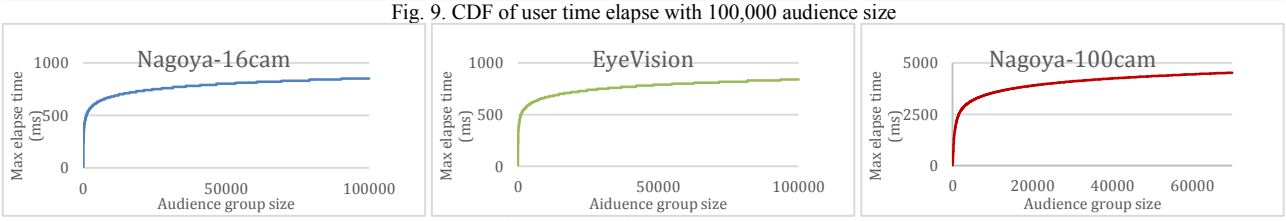

Fig. 10. Maximum elapse delay among 100,000 audiences

interactive commands. Second, the bandwidth requirement for the content provider to sustain F.Live is measured. We record the sum of outbound bitrate of all content producers to measure the gain of the adoption of P2P content sharing. The required total bitrate should be less than our gigabit capacity assumption of the content provider's outbound network. Last, since F.Live is targeting live broadcasting service, we want to know the freshness of content rendered by the audience. Content freshness is determined by the time elapse between the capturing of a scene and its rendering at the audience site.

VIII. Evaluation

The evaluation of our framework is two-fold. First, we report the simulation results of the three producer site settings regarding interactive-ness, producer bandwidth consumption, and content freshness. Second, we compare the proposed view-based delivery chain with other possible delivery frameworks for interactive live broadcasting of FTV.

A. *Interactive-ness*

The synchronization delay of user command is plotted in Fig. 7. The x-axis of the plot is the number of audiences in the system when user command is issued. The y-axis is the average synchronization delay of the commands. As we can see in the plots, in early stage when the audience group size is not big enough, delays are longer and unstable. This is due to the policy that users choose their peers over the producers as sources when joining/rejoining the system. In early stage, the number of users is small so a joining/rejoining user has a smaller chance to find a group of source users that hold in-synced contents. After the system gain more users, the synchronization delay drops drastically and is stable below 200ms.

Comparing the delays across different producer site settings, we see that the Nagoya-100cam system takes longer time to stabilize. In addition, during the stable period, a few anomalies happens occasionally with noticeable delays (> 3sec). This happens when the dissemination tree grows tall and the available sources in distant tree layers hold highly asynchronous contents. One solution is to set a threshold for source selection. If the synchronization delay should exceed the threshold, then the user will directly download from the producers. Fig. 11 shows the synchronization delay of Nagoya-100cam after applying this amendment with a 3-second threshold.

B. *Bandwidth Consumption*

In Fig. 8 we show the bandwidth consumption of producer site with x-axis as the number of audiences and the y-axis as the sum of out-going bitrate of the content producer.

From the plots, we see that the bandwidth consumption rises rapidly in the beginning and then stays in a stable state. This phenomenon is caused by the P2P stream sharing in content dissemination. In the beginning, newly joined users have no choice but to download the streams directly from their producers. After the streams are held by some audiences, new comers can download from them to alleviate the burden of producer site. The stable outbound bandwidth requirement implies good scalability of our distributed system. In their stable stage, the bandwidth requirements are below 400 Mbps for all three

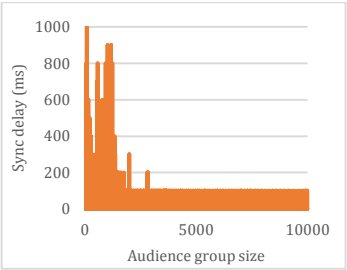

Fig. 11. Synchronization delay with threshold

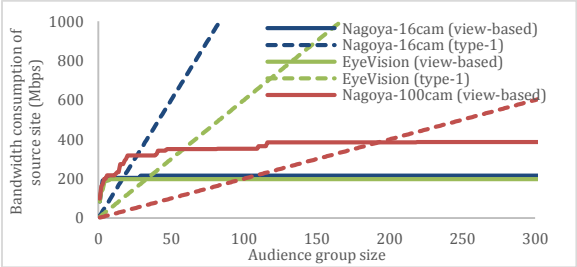

Fig. 12. View-based delivery chain vs. Type-1

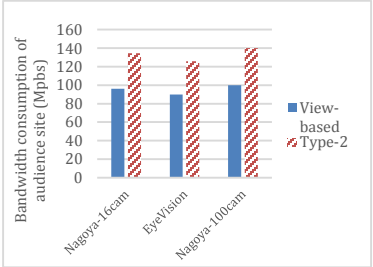

Fig. 13. View-based delivery chain vs. Type-2

producer site settings, which is well-manageable by the gigabit network infrastructure of content providers we expected.

C. Content Freshness

To determine whether F.Live is suitable for delivering live content, we plot the time elapses of each audience when there are 100,000 concurrent audiences in the system in the CDF graph in Fig. 9. We see that for medium-scale camera array systems (Nagoya-16cam and EyeVision), the time elapses are bounded under 1 second. As for large scale camera array as Nagoya-100cam, the elapse is bounded under 4.5 seconds. This means that in the worst case, the actions seen by the end users can actually happen 4.5 seconds ago. Comparing to the broadcasting standard of TV industry, 4.5 seconds elapse is still within an acceptable range. As a reference, the tolerable delay for live broadcasting of CBS TV network is set at 5 seconds [20].

From the distribution of users' time elapses, we see only few users have short elapses. 55~70% of the end users experience elapse longer than average (marked by black lines in Fig. 9). This is due to the tree structure of our dissemination overlay. Having more nodes reside in the lower layers is a natural property of tree topologies. This raises the concern of ever growing time elapse when the audience group grows, which could be a performance defect on scalability. Thus, we plot the growth of maximum time elapse along the increasing audience group size in Fig. 10. From the plots, we see that the growth curves of elapse time are sub-linear. This indicates that the elapse time scales well with the size of audience group. Take the result of Nagoya-16cam system, based on its regression curve, we can expect the time elapse still be bounded under 1 second even when the audience group size is doubled from 100,000.

D. Comparison to Other Delivery Chains

In this section, we simulate the two existing delivery chains introduced in §1 and compare their resource consumptions with our view-based delivery chain. As we mentioned in §3, these two existing frameworks are only capable of providing broadcasting service or interactive service, but not both. For the ease of discussion, we call them type-1 and type-2 delivery chains, as illustrated in Fig. 1.

First, for type-1 delivery chain (Fig. 1a) to be able to deliver interactive content, we assume that there is a channel between users and the producer site which users can use to specify their viewpoints. The producer site then create customized interpolated views according to each and every viewpoint requests and send them to each user. In Fig. 12, we see the bandwidth consumption of the producer site. The solid lines present the bandwidth consumption of our view-based delivery, and the dashed lines are the consumption of type-1 delivery chain. Since every user receive a unique, customized stream tailored for their viewpoints, stream sharing becomes infeasible. Thus, the bandwidth consumption linearly grows with the size of audience group. After the audience group size passes 200, our view-based delivery chain achieves significantly more bandwidth saving comparing to type-1 delivery.

Second, for type-2 delivery chain (Fig. 1b) to be able to do live broadcasting, the producer site sends all the streams captured by its camera array to every user and let the users choose their viewpoint locally. Note that MVC is applicable in this case since streams are aggregated at the producer site. Tanimoto *et al.* [4] report 20~30% compression ratio of MVC on the Nagoya systems hence we assume 30% bitrate saving for type-2 framework in our experiment. In Fig. 13 we plot the average bandwidth consumption of audience sites. The blue bars are the consumption of view-based delivery chain and the red bars are type-2 delivery chain. From the figure we see even with the optimistic 30% saving assumption, the type-2 framework still consumes 30~40 Mbps more bandwidth than ours.

IX. Conclusion

In this paper we propose a new delivery infrastructure for live FTV content broadcasting. Supported by the result of large-scale simulation, the proposed F.Live framework preserves interactivity and live broadcasting at the same time, which is a desirable improvement from existing FTV systems.

References

[1] EyeVision. www.ri.cmu.edu/events/sb35/tksuperbowl.html

[2] T. Horiuchi, H. Sankoh, T. Kato, *et al.*, "Interactive Music Video Application for Smartphones Based on Free-Viewpoint Video and Audio Rendering", ACM MM, 2012.

[3] T. Fujii, K. Mori, K. Takeda, *et al.*, "Multipoint Measuring System for Video and Sound - 100-Camera and Microphone System", IEEE ICME, 2006.

[4] M. Tanimoto, M. P. Tehrani, T. Fujii, *et al.*, "Free-viewpoint TV: A Review of the Ultimate 3DTV and its Technologies", IEEE Signal Processing Magazine, 2011.

[5] Huawei, "Technical White Paper for HDTV Bearer Network", www.huawei.com/es/static/hw-076764.pdf, 2010.

[6] ISO Standard, "Introduction to Multi-View Video Coding", ISO/IEC JTC1/SC29/WG11, N7328, 2005.

[7] Best of EyeVision. youtube.com/watch?v=ohdhYEcCGVo

[8] 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

[9] S. Chen, Z. Gao, K. Nahrstedt, *et al.*, "3DTI Amphitheater: Towards 3DTI Broadcasting", ACM TOMM, 2015.

[10] Takeo Kanade and P.J. Narayanan, "Virtualized Reality: Perspectives on 4D Digitization of Dynamic Events", IEEE Computer Graphics and Applications, Vol 27, Issue 3, pp. 32-40, 2007.

[11] Toshiaki Fujii, Tadahiko Kimoto, and Masayuki Tanimoto, "A New Flexible Acquisition System of Ray-Space Data for Arbitrary Objects", IEEE Trans. on Circuits and Systems for Video Technology, Vol 10, Issue 2, pp. 218-224, 2000.

[12] M. Tanimoto, M. P. Tehrani, T. Fujii, et al., "FTV for 3-D Spatial Communication", Proceedings of the IEEE, Vol 100, Issue 4, pp. 905-917, 2012.

[13] T. Fujii and M. Tanimoto, "A Real-time ray-space acquisition system", SPIE Electronic Imaging, 2004.

[14] Norishige Fukushima, Tomoh iro Yendo, Toshiaki Fujii, *et al.*, "Real-time Arbitrary View Interpolation and Rendering System using Ray-Space", SPIE Three-Dimensional TV, Video, and Display, Vol 6016, pp. 250-261, 2005.

[15] Toshiaki Fujii, Tomohiro Yendo, and Masayuki Tanimoto, "Ray-Space Transmission System with Real-Time Acquisition and Display", Proc. The 20th Annual Meeting of the IEEE Lasers and Electro-Optics Society (LEOS), 2007.

[16] Purim Na Bangchang, Toshiaki Fujii, and Masayuki Tanimoto, "Experimental System of Free Viewpoint Television", Proc. IST/SPIE Symp. Electronic Imaging, vol. 5006-5066, pp. 554-563, 2003.

[17] Hao Yin, Xuening Liu, and Tongyu Zhan et al., "Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky," Proc ACM Multimedia (MM), 2009.

[18] A. Arefin, Z. Huang, K. Nahrstedt et al., "4D TeleCast: Towards Large Scale Multi-site and Multi-view Dissemination of 3DTI Contents," Proc. IEEE ICDCS, 2012.

[19] Netmap. www.caida.org/tools/visualization/mapnet

[20] CBS broadcast delay. http://us.cnn.com/2004/SHOWBIZ/TV/02/03/grammys.tape. delay/index.html

[21] PPTV. http://www.pptv.com/.

[22] PPS. http://www.pps.tv/.

[23] 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

[24] D. Feldman and Y. Shavitt, "An optimal median calculation algorithm for estimating internet link delays from active measurements," IEEE E2EMON, 2007

[25] S. Chen, K. Nahrstedt, and I. Gupta, "3DTI Amphitheater: A Manageable 3DTI Environment with Hierarchical Stream Prioritization", Proc. ACM Multimedia Systems, 2014

[26] B. Wang and J. Hou, "Multicast Routing and its QoS Extension: Problems, Algorithms, and Protocols", In IEEE Network, 2000.

[27] Z. Wang and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications", In IEEE Journal Selected Areas in Communications, 1996.

[28] W. Wu, Z. Yang, and K. Nahrstedt et al., "Towards Multi-Site Collaboration in 3D Tele-Immersive Environments", In Proc. of IEEE ICDCS, 2008.

[29] M. Castro, P. Druschel, and A. Kermarrec et al., "SplitStream: High-Bandwith Multicast in Cooperative Environments", SOSP 2003.

APPENDIX

Here we list the pseudo codes of the overlay planning algorithms. Part of the codes is modified from previous work [9].

Algorithm 1: $UJoin(u, S')$
Input: *new user u, set of stream S' to subscribe*

```
1:  // Phase 1
2:  C ← ∅
3:  For each s ∈ S' do
4:      Find (u_s, s) such that
5:          (0)  E(u_s) + ε ≥ E(u)
                 // effective for rejoined user (view change) only.
                 E(u) of newly joined user is set to be -∞
                 // ε is additional elapse due to propagation
6:          (1) u_s subscribes to stream s
7:          (2) u_s has available bandwidth
8:          (3) u_s has the smallest E(u_s) under (0)(1)(2)
9:      If no such (u_s, s) exists do
10:         Let user u download stream s from the producer
11:         Remove s from S'
12:     Else do
13:         Add (u_s, s) to C
14:     End if
15: End for
16: If C = ∅ do
17:     Return // u downloads all streams from producers
18: End if
19: (ŭ_s, š) ← user ŭ_s has the largest E(ŭ_s) in C
20: Let user u download stream š from ŭ_s
21: Remove š from S'
22: I ← [E(ŭ_s), E(ŭ_s)]  //tight interval of sources' elapses
23: // Phase 2
24: C ← ∅
25: While S' ≠ ∅ do
26:     For each s ∈ S' do
27:         Find (u_s, s) such that
28:             (0) E(u_s) + ε ≥ E(u) // effective on view change
29:             (1) u_s subscribes to stream s
30:             (2) u_s has available bandwidth
31:             (3) E(u_s) is closest to interval I under (0)(1)(2)
32:         If no such (u_s, s) exists do
33:             Let user u download stream s from the producer
34:             Remove s from S'
35:         Else do
36:             Add (u_s, s) to C
37:         End if
38:     End for
39:     (ŭ_s, š) ← user ŭ_s has the smallest E(ŭ_s) in C
40:     Let user u download stream š from ŭ_s
41:     Remove š from S'
42:     Include E(ŭ_s) to I
43: End While
```

Algorithm 2: $Reassign(o, S'_o)$
Input: *orphan user o, set of streams S_o' without source*

```
1:  For each s ∈ S'_o do
2:      Find (u_s, s) such that
3:          (0) E(u_s) + ε = E(o)
4:          (1) u_s subscribes to stream s
5:          (2) u_s has available bandwidth
6:      If no such (u_s, s) exists do
7:          Let user o download stream s from the producer
8:      Else do
9:          Let user o download stream s from u_s
10:     End if
11: End for
```

Algorithm 3: $VChange(u, S')$
Input: *view changing user u, new set of stream S' to subscribe*

```
1:  O ← Children of user u
2:  For each o ∈ O do
3:      Reassign(o, S'_o)
4:  End for
5:  User u terminates all the downloading
6:  UJoin(u, S')
```