GroupCast: Preference-Aware Cooperative Video Streaming with Scalable Video Coding

Anis Elgabli and Vaneet Aggarwal
Purdue University, West Lafayette, IN 47906
Email: {aelgabli,vaneet}@purdue.edu

***Abstract*—In this paper, we propose a Preference Aware Cooperative video streaming system for videos encoded using Scalable Video Coding (SVC) where all contributing users are interested in watching the same video on a single screen. Each user's willingness to cooperate is subjected to their own constraint such as user data plan (unlimited, 6GB, 2GB, $\cdots$, etc). Using SVC, each layer of every chunk can be fetched through only one of the cooperating users. We formulate the quality decisions of video chunks and fetching policy of the SVC layers subject to the available bandwidth, chunk deadlines, and cooperation willingness of the different users as an optimization problem. The objective is to minimize the re-buffering time as the first priority, maximize the average quality, and minimize the number of quality switches without violating any of the imposed constraints. We propose an offline algorithm to solve the non-convex optimization problem. This algorithm has a complexity that is polynomial in the video length and the number of cooperating users. Moreover, we propose an online algorithm for more practical scenarios where erroneous bandwidth prediction for a short window is used. Simulations driven by real SVC encoded video and real bandwidth traces of a public dataset reveal the robustness and high-performance achievement of our scheme. The results motivate our next step of implementing real test bed.**

***Index Terms*—Cooperative Video Streaming, Scalable Video Coding, Non Convex Optimization**

I. Introduction

In Scalable Video Coding (SVC [1]), each chunk is encoded into ordered *layers*: one *base layer* (BL) with the lowest playable quality, and multiple *enhancement layers*($E_1, .., E_N$) that further improve the chunk quality. For decoding a chunk up to enhancement layer i, a player must download all layers from 0 to i. Thus, adaptive SVC streaming can allow playback at a lower quality if all the enhancement layers of a chunk have not been fetched before its deadline. Moreover, for cooperative video streaming, when different users are willing to cooperate in order to improve the quality of the next chunk to fetch, different layers of the same chunk can be fetched through different contributors.

To motivate our problem, consider a scenario in which friends are gathered at a low speed internet place (*e.g.,* camp-site) and are interested in watching their favorite television show or movie on one screen at the highest quality possible. We assume that the videos are encoded using Scalable Video Coding (SVC [1]). The users can have data plans from different carriers, and with different limits (e.g, unlimited, 6GB, 3GB, etc.). With these assumptions, there will be preferences based on which a priority level of the user is decided. Users belong to a higher priority level (set) can help in fetching up to a higher layer. We further impose a maximum contribution limit for every user based on the remaining data on their plans. The maximum contribution indicates the maximum amount of data the user can download for the whole video. For example, $x_i\%$ of the remaining data of the i-th user plan can at most be used. Moreover, machine learning algorithms can be used to predict the maximum contribution of every user.

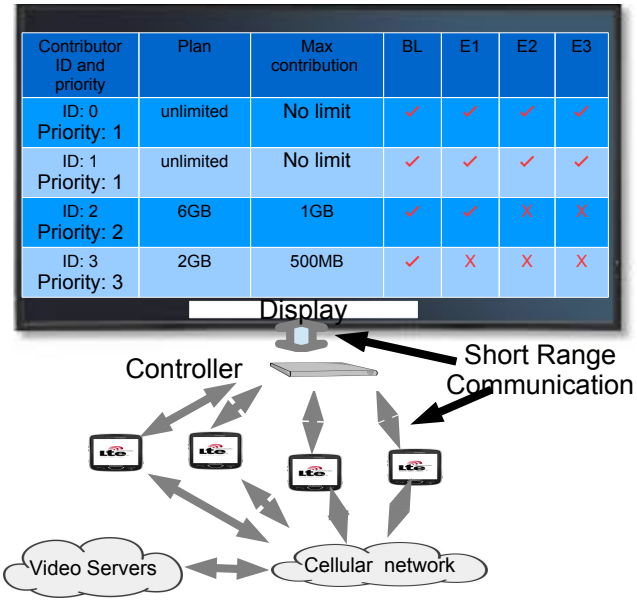

Fig. 1. Example to illustrate the priority level, amount of data different users can contribute, and the different layers they can help in fetching

As an example, Fig. 1 illustrates a setup with four users. The video is encoded into four layers, one base layer and three enhancement layers. The first two users (0 and 1) have unlimited data plans, so they can fetch all the layers, and have no maximum contribution limit. Thus, these two users are at the top priority level. User 2, at the second priority level, has limited data plan (6 GB/month). This user can contribute to increase the quality of the chunks up to the $1st$ enhancement layer quality if the first two users fail to do so due to their limited bandwidths. Even though User 2 helps in fetching chunks, she also has a maximum contribution limit of 1 GB.

User 3, at the third priority level, can help in obtaining base layer if the other users are not able to obtain the chunks at that level. Further, the maximum contribution of user 3 is 500 MB. Using such preferences from all users as in Fig. 1, this paper proposes an algorithm to deliver the video at maximum possible quality without violating user imposed constraints. Therefore, this paper proposes optimized decision for each layer of every chunk to decide whether to fetch it or not and if fetched, from which user it is fetched. The objective is to minimize the stall duration as the first priority, and maximize the average quality as the next priority without violating user imposed constraints.

Cooperative video streaming for users who are interested in watching the same content when they are in proximity of each other has been investigated. For instance, [2], [3] proposed a P2P-TV system for single rate encoded video. [4] considers cooperative SVC video streaming for SVC encoded video with only two layers. The work in [5] proposed video assignment strategy based on the deficit round robin (DRR) for combined Scalable video Coding (SVC) and Multiple Description Coding (MDC) Co-SVC-MDC-Based Cooperative Video Streaming. The authors of [6] proposed SVC cooperative video streaming for vehicular networks. However, they consider only one relay node per receiver (two contributor at most), and they do not consider user preferences. In contrast to these, accounting for user preferences has not been considered, to the best of our knowledge. The contributions of this paper can be summarized as follows.

• We propose a Preference-aware cooperative video streaming system. To the best of our knowledge, this is the first work that provides video streaming system that integrates cooperative video streaming, SVC encoding, and preference awareness.

• We first, theoretically formulate preference-aware cooperative video streaming with perfect bandwidth prediction as a non-convex optimization problem, and we propose an offline algorithm that solves the optimization problem in polynomial time complexity.

• We propose an online algorithm (sliding window algorithm) for the more practical scenarios in which the bandwidth prediction is available for short time ahead and has errors.

• The proposed algorithms are evaluated using simulations with real SVC encoded videos and bandwidth traces. The evaluation demonstrates that our approach is robust to prediction errors, and works well with short prediction window.

The rest of the paper is organized as follows. Section II describes the system model. Section III describes the problem formulation. Further, polynomial-time algorithms for both offline and online scenarios are provided in section IV. Section V presents the trace-driven evaluation results with comparison to the different baselines. Section VI concludes the paper.

II. System Model

We assume that there are U users who wish to watch a video on a single display screen which is connected to a controller that manages the inputs of the different users to come up with streaming decisions. The screen and/or the controller can be stand alone, or can be chosen among the users. The SVC encoded video is divided into C chunks (segments), where every chunk is of length L seconds, is encoded in Base Layer (BL) with rate r_0 and N enhancement layers $(E_1, \cdots, E_N)$ with rates $r_1, \cdots, r_N$, respectively. Let $Y_n = L * r_n$ be the size of the nth layer, $n = 0, \cdots, N$.

The set of U users is divided into K sets, with set $k \in \{1, \cdots K\}$ having U_k users, $\sum_{k=1}^{K} U_k = U$. The users belong to set k can help in fetching the video till layer $N_k \in \{0, \cdots, N\}$. We also assume that $N_1 > N_2 > \cdots > N_K$. Further, each user $u_k \in \{1, \cdots, U_k\}$ in set k has a maximum download contribution $\eta^k_{u_k}$. We also assume strict priority in the sets in the sense that, user set k is used only to fetch chunks at layers from $\{0, \cdots, N_k\}$ if the user sets from 1 to $k-1$ are unable to fetch them without violating their deadlines.

We assume all time units are discrete and the discretization time unit is assumed to be 1 second. Let $Z_{n,i}$ denote the size of the n-th layer that has been fetched. Therefore, if the nth layer can be fetched $Z_{n,i} = Y_n$; otherwise $Z_{n,i} = 0$. Let $z^{(k)}_{n,u}(i,j)$ be the size of layer n of chunk i fetched through the u-th contributor of the k-th set at time slot j. Let $B^{(k)}_u(j)$ be the available bandwidth of the u-th link in the k-th set, and s be the startup delay. Let the total stall (re-buffering) duration from the start till the play-time of chunk i be $d(i)$. Therefore, the deadline of any chunk i is $(i-1)L + s + d(i)$. Note, the deadline of a chunk i is equal to $(i-1)L + s$ if the video has not run into re-buffering before fetching the i-th chunk.

III. Problem Formulation

In this section, we will describe the problem formulation for cooperatively streaming SVC videos. The key objectives of the problem are *(i)* minimization of the stall duration, *(ii)* maximization of the average playback rate of the video, and *(iii)* minimization of the quality changes between the neighboring chunks to ensure that perceived quality is smooth. We first describe the problem formulation assuming perfect prediction of the bandwidth for the whole period of the video (Offline) and infinite buffer size. We will relax these assumption when we describe our online algorithm in section. IV-C. In order to account for the these objectives, we consider the following objective function:

$$\sum_{k=1}^{K}\sum_{n=0}^{N}\lambda^k_n\sum_{u=1}^{U_k}\sum_{i=1}^{C}\sum_{j=1}^{deadline(C)}\left(z^{(k)}_{n,u}(i,j)\right) - \mu d(C) \quad (1)$$

In the above equation we maximize a weighted sum of the chunk layers fetched by different users. In order to prefer lower layers of chunks over higher layers and users belong to higher priority sets over the users belong to the lower priority sets, we introduced 2D weights (λ^k_n). With this, we give more weighs to lower layers and lower user-sets. Thus, the following two conditions must hold true:

$$\lambda^{(1)}_n > \lambda^{(2)}_n > ... > \lambda^{(K)}_n. \quad (2)$$

Further, for any set k, $k' \in \{(k+1)....K\}$, and $k'' \in \{1....k-1\}$

$$\lambda_a^{(k)} Y_a > C \cdot \Big(\sum_{n=a}^{N} \lambda_n^{k'} Y_n + \sum_{n=a+1}^{N} \lambda_n^{k''} Y_n \Big) \tag{3}$$

First, this choice of λ's implies that all layers higher than layer a have lower utility than a chunk at layer a for all a. For $a = 0$, it implies that all the enhancement layers have less utility than a chunk at the base layer. The use of λ helps in giving higher priority to fetch more chunks at nth layer quality over fetching some at higher quality in the cost of dropping the quality of other chunks to bellow the n-th layer quality. Second, it implies that the highest utility that can be achieved for every layer is when it is fetched by a user belong to the highest priority set. This will obey the priority order, and it will not use a lower priority user to fetch a layer that can be fetched by a higher priority user. Finally, the weight for the stall duration is chosen such that $\mu \gg \lambda_0^{(1)}$ since users tend to care more about not running into re-buffering over better quality. The problem is formulated as follows.

$$\textbf{maximize:} (1) \tag{4}$$

subject to

$$\sum_{j=1}^{(i-1)L+s+d(i)} \sum_{k=1}^{K} \sum_{u=1}^{U_k} z_{n,u}^{(k)}(i,j) = Z_{n,i} \quad \forall i, n \tag{5}$$

$$Z_{n,i} \leq \frac{Y_n}{Y_{n-1}} Z_{n-1,i} \quad \forall i, n > 0 \tag{6}$$

$$\sum_{n=0}^{N} \sum_{i=1}^{C} z_{n,u}^{(k)}(i,j) \leq B_u^{(k)}(j) \quad \forall k, u, \text{ and } j \tag{7}$$

$$z_{n,u}^{(k)}(i) z_{n,u'}^{(k')}(i) = 0 \forall n, i, \text{ and } (k,u) \neq (k',u') \tag{8}$$

$$\sum_{i,j} z_{n,u}^{(k)}(i,j) \leq \eta_u^k \forall u, k \tag{9}$$

$$z_{n,u}^{(k)}(i,j) = 0 \ \forall \{i : (i-1)L + s + d(i) > j\}, \forall u, k \tag{10}$$

$$d(i) \geq d(i-1) \geq 0 \ \forall i > 0 \tag{11}$$

$$Z_{0,i} = Y_0 \tag{12}$$

$$z_{n,u}^{(k)}(i,j) \geq 0 \ \forall u, \forall i \tag{13}$$

$$Z_{n,i} \in \mathcal{Z}_n \triangleq \{0, Y_n\} \quad \forall i, n \tag{14}$$

In the above formulation, λ's are constrained by (2) and (3). Constraints (5)and (14) ensure that what is fetched for layer n of chunk i over all links and times to be either zero or Y_n. The constraint (6) ensures that n-th layer of any chunk cannot be fetched if the lower layers have not been fetched. (7) imposes the bandwidth constraint at each time slot j. Constraint 8 enforces a layer of a chunk to be fetched only over one of the paths. Constraint (9) limits the maximum contribution of the u-th user in the k-th set to η_u^k bytes over the whole period of the video. Constraint (10) imposes the deadline constraint (no chunk is fetched after its deadline). Recall that s is the initial startup delay. Constraint 12 enforces fetching the base layer of every chunk (i.e base layers can't be skipped). Constraint(13) imposes the non-negativity of the download of a chunk

IV. Efficient Algorithm

In this section, we describe our proposed algorithm. First, we describe our offline algorithm for the scenario when all users have the same priority level, so all users can be used equally likely to fetch all layers. We call this algorithm "Offline No preference GroupCast" (Offline No-Pref GroupCast). Then we describe an offline algorithm (Offline Pref GroupCast) for the more general case in which some users can cooperate in fetching up to a certain layer. Finally, we propose an efficient online algorithm (Online No-Pref/Pref GroupCast) in which more practical assumptions such as short bandwidth prediction with prediction error and finite buffer size are considered

A. Offline No-pref GroupCast

bf Base layer decision: In the No-preference scenario, there is only one set consists of U users. The offline No-Pref GroupCast algorithm (Algorithm 1) initially runs initial forward scan with the objective of checking if all chunks can be fetched at least at base layer quality with the current startup delay. The forward algorithm finds the maximum number of chunks that can be fetched before the deadline of every chunk i ($V(i)$). The maximum number of chunks that can be fetched before the deadline of the i^{th} chunk is: $V(i) = \sum_{u=1}^{U} \lfloor r^u(deadline(i) \rfloor$(line 5). Therefore, if $V(i) < i$ at the deadline of the i^{th} chunk, the algorithm increments the deadline of every chunk $\geq i$ by 1 and resumes the forward scan. The algorithm does not proceed to the next chunk till the condition of $V(i) \geq i$ is satisfied. At the end, the algorithm sets the final deadline of every chunk i to be: $deadline(i) = (i-1)L + s + d(C)$. Note that, if the maximum contribution of all users is reached and not all chunks are fetched, the algorithm will not consider fetching more chunks, so the users have to increase their contribution if they want to watch more of the video content.

In the second, step, the algorithm performs backward scan per chunk starting from the first chunk that was decided to be fetched by calling Algorithm 2 (line 21). The backward scan simulates fetching every chunk i starting from its deadline and by every user that has $r^{(u)}(deadline(i)) \geq Y_0$(Line 21). For all choices of links, the algorithm computes the residual total bandwidth that will remain. The link choice that maximizes the residual total bandwidth is chosen to fetch chunk i as shown

in Algorithm 2. Note that the link over which the chunk i is fetched is the one that gives the maximum amount of total available bandwidth over all links before the deadline of chunk numbered $i-1$. For example, consider that there are only two links. Also, consider that fetching the i^{th} chunk by user 1 results in using x amount of the bandwidth before the deadline of $i-1^{th}$ chunk while fetching the i^{th} chunk by user 2 results in using y amount of the bandwidth before the deadline of $i-1^{th}$ chunk. Then, the first user will be chosen to fetch the chunk i if $x < y$. The objective is to free as much as possible of early bandwidth since it will help more chunks to fetch their higher layers because it comes before their deadlines.

Enhancement layer modifications: The algorithm proceeds in performing forward-backward scan per layer in order. However, the deadline is final and any enhancement layer can't be fetched before the deadline of the chunk will be skipped(lines 18-24). Moreover, the bandwidth is now modified to be the remaining bandwidth after excluding whatever has been reserved to fetch lower layers (line 7, Algorithm 2). Also note that nth layer of a chunk is not fetched if its $(n-1)$th layer was not fetched (line 20, Algorithm 1).

The overall complexity of the algorithm is $O(U \cdot N \cdot C^2)$, and the detailed complexity analysis is omitted for the lack of space.

B. Offline Pref GroupCast

In this section, we consider K sets in which the users (links) belong to the first set ($k = 1$, users 1 to u') have unlimited plan and can contribute as much as their bandwidths allow, but users from $u'+1$ to U have different preferences, and they could belong to different sets. For example, some of them can contribute in fetching up to the n-th enhancement layer, and some others can only help in avoiding stalls due to their limited plans. The later ones will belong to the set numbered K since sets are ordered in their priority levels (willingness to contribute). Therefore, offline Pref GroupCast algorithm is proposed (Algorithm 3): First the offline No-Pref-GroupCast is run considering the lowest set of layers that all users are willing to fetch. Let's denote these layers by layers 0 to m. The objective of the first call of Offline No-Pref-GroupCast is to find the maximum layer $\leq m$ that can be fetched for every chunk. Therefore, the outcome (the initial fetching policy) of this run is not final since the preference has not been taken into consideration. In the second step, Offline No-Pref-GroupCast is re-run for layers 0 to m that were initially decided to be fetched by the users belong to the lowest priority set (set K) but after excluding this set and the bandwidth that is reserved for fetching other layers, so the objective is to move as much as possible of these layers to the higher priority links. In final step, the links that can't help in fetching beyond the m-th layer are excluded, and the same process repeats for the remaining sets and layers. Therefore, the next run considers up to the lowest layer that all remaining links are willing to fetch (layer $m+1$, line 10). Finally, No-Pref-GroupCast Algorithm consider only the users 1 to u' (users in set 1) with their remaining bandwidths after all possible lower layers were fetched.

Algorithm 1 Offline No-Pref GroupCast Algorithm

1: **Input:** $\mathbf{Y} = \{Y_n \forall n\}$, L, s, C, U, $\mathbf{B}_u = \{B_u(j) \forall j\}$, $u = 1,..,U$.
2: **Output:** $I_n^{(u)}$, $n = 0,...,N$: set containing the indices of the chunks that can have their nth layer fetched by user u.
3: $d(i) = 0$, $deadline(i) = (i-1)L + s, \quad \forall i$
4: $r^{(u)}(j) = \sum_{j'=1}^{j} B_u(j')$, $j = 1,....deadline(C)$, $u = 1,..,U$
5: $V_i = \sum_{u=1}^{U} \lfloor r^{(u)}(deadline(i) \rfloor$: The total number of chunks that can be fetched before the deadline of the i-th chunk.
6: **for** each layer $n = 0, \cdots, N$ **do**
7: **if** $n = 0$ **then**
8: **for** $i = 1 : C$ **do**
9: **If** $i > 1$ **then** $d(i) = d(i-1)$
10: **while** $(V(i) < i)$ **do**
11: $d(i) = d(i) + 1$
12: $deadline(i) = (i-1)L + s + d(i)$
13: $V_i = \sum_{u=1}^{U} \lfloor r^{(u)}(deadline(i) \rfloor$
14: **end while**
15: **end for**
16: $deadline(i) = (i-1)L + s + d(C), \quad \forall i \; i' = 1$
17: **else**
18: $skip = 0$
19: **for** $i = 1 : C$ **do**
20: **if** $V(i) < i$ **then**
21: $skip = skip + (i - V(i))$
22: **end if**
23: **end for**
24: Skip the first $skip$ chunks.
25: i' =the index of the first chunk to fetch
26: **end if**
27: **for** $i = i' : C$ **do**
28: $j_u = deadline(i), \forall u = 1....U$
29: $r^u(j_u) = \min(r^u(j_u), \eta_u), \forall u = 1....U$
30: **if** ($n = 0$ or $i \in \{I_{n-1}^{(1)} \cup I_{n-1}^{(2)} \cup .... \cup I_{n-1}^{(U)}\}$) **then**
31: **if** $max(r^{(1)}(j_1), ..., r^{(U)}(j_U)) \geq Y_n$ **then**
32: **if** $r^u(j_u) > Y_n$ **then**
33: $[R^{(u)}, B2_u] = Backward(u, i, j_u, \{r^{(u)}(j) \forall j\}, \mathbf{B}_u, Y_n, \eta_u^k), \forall u$
34: $tmp(u) = R^{(u)}(deadline(i-1)) + \sum_{u'=1, u' \neq u}^{U} r^{(u')}(deadline(i-1)), u = 1, 2, ...U$
35: **else**
36: $tmp(u) = 0$
37: **end if**
38: $u_1 = \arg\max(tmp)$
39: **if** $(r^{(u_1)}(deadline(i)) \geq Y_n)$ **then**
40: $I_n^{(u_1)} = I_n^{(u_1)} \cup i$, $\mathbf{B}_{(u_1)} = B2_{(u_1)}$
41: $r^{(u_1)} = R^{(u_1)}$, $Y_n = 0$
42: **end if**
43: **end if**
44: **end if**
45: **end for**
46: **end for**

C. Online Pref/No-Pref GroupCast

For the algorithm described in Section IV, we assumed that perfect bandwidth prediction, and client (display) buffer capacity is unlimited. However, practically, the prediction will not be perfect, and the client buffer might be limited. In fact, even if the buffer is unlimited the video content itself may not be available after few chunks ahead. Therefore, the buffer

Algorithm 2 Backward Algorithm

Input: u, i, j, r, B, Y_n
Output: R is the residual cumulative bandwidth at different times after fetching chunk i, $B2$ is the residual bandwidth after accounting for fetching of chunk i
Initialization:
$i = C$
while $(Y_n > 0)$ **do**
  $fetched = min(B(j), Y_n)$
  $B(j) = B(j) - fetched$, $r(j) = r(j) - fetched$
  $Y_n(i) = Y_n(i) - fetched$
  if $(B(j) = 0)$ **then** $j = j - 1$
  $R = r$, $B2 = B$
end while

Algorithm 3 Offline Pref GroupCast Algorithm

$\chi = \{1, ..., K\}$: set of all user sets
$lowest = K$
$p = 0$
while set $2 \in \chi$ **do**
  m=max layer index in which users belong to set $lowest$ can fetch
  Run offline No-Pref GroupCast for layers p to m to extract the initial fetching policy
  Run offline No-Pref GroupCast for layers p to m, sets 1 to $lowest - 1$, and only chunks fetched by users belong to set $lowest$.
  $\chi = \chi - \{lowest\}$
  $lowest = lowest - 1$
  $p = m + 1$
end while
Run offline No-Pref GroupCast for all the remaining layers using only users belong to set 1.

constraint is considered to account for both scenarios (limited client buffer, availability of the future chunks). In this section, we will use an online algorithm that will obtain prediction per user for a window of size W chunks ahead and make decisions based on the prediction. There are multiple ways to obtain the prediction. Our approach is to use the harmonic mean of the past β seconds to predict the future bandwidth [7], [8]. The decisions are re-computed for the chunks that have not yet reached their deadlines periodically every α seconds.To account for the buffer, we consider that W is no more than the buffer duration from the chunk that has the current time as its deadline.

For the prediction window W, the algorithm in Section IV-A or IV-B is run to find the quality using the predicted bandwidth, then the W chunks start to be fetched according to the algorithm decision. If all W chunks are fetched before next re-computation time, the current time slot is set as re-computation time. Finally, at the start of the download, only users at the highest priority set are assigned chunks to fetch at base layer quality since there is no bandwidth prediction available yet, and the users in the next priority set are triggered whenever needed, so the algorithm starts to know about their bandwidths and include them in the optimization framework. Finally, the maximum contribution of every user starts as x percent of the total contribution and increases gradually to avoid being optimistic at the beginning and run into re-buffering and low quality later on.

V. Simulation Results

In this section, we evaluate our algorithms using simulation.

TABLE I
SVC encoding bitrates used in our evaluation

layer	BL	EL1	EL2	EL3
nominal rate (Mbps)	1.5	2.75	4.8	7.8

Simulation Setup. We simulate our algorithm assuming a system of 4 users who are interested in watching the same video in a single screen. To make our simulation realistic, we choose the SVC encoding rates of an SVC encoded video "Big Buck Bunny", which is published in [9]. It consists of 299 chunks, and the chunk duration is 2 seconds. Table I shows the cumulative nominal rates of each of the layers. In the table, "BL" and "EL$_i$" refer to the base layer and the cumulative (up to) ith enhancement layer size, respectively. For example, the exact size of the ith enhancement layer is equal to EL$_i$-EL$_{(i-1)}$. For all algorithms, we assume at any given time, only the next 10 chunks from the chunk with the current time being its deadline can be fetched ($W = 10$ chunks). Moreover, we re-consider the decisions every 2 seconds ($\alpha = 2s$). Finally, the startup delay is 5 seconds.

Bandwidth traces. For bandwidth traces, we used a public datasets consists of continuous 1-second measurement of throughput of a moving device in Telenor's mobile network in Norway [10]. The dataset has been post-processed in [11] to give 1000 traces, each of 6-minute length which will be used in this paper for evaluations. We assign 250 traces to each of the users. We note that since the "Big Buck Bunny" is 299s, so it is longer than the bandwidth traces, we cut the video at 175 chunks.

We use the harmonic mean of the bandwidth of the last 5 seconds as a predictor of the future bandwidth. We assume a system in which users numbered 1, and 2 have unlimited plan and are willing to cooperate without imposing constraints. However, users numbered 3 and 4 are only willing to help in avoiding stalls, so they are used to fetch base layers when 1 and 2 can't meet the deadlines. Moreover, non of the users is imposing maximum contribution constraint. We compare our proposed online algorithm with the following baseline algorithms.

Buffer-based Preference-aware Cooperative streaming Approach (Base 1): Base 1 uses similar strategy to BBA-0 [12] to make a quality decision per chunk, and then select randomly user 3 or 4 only if the base layer has not yet been downloaded within a pre-defined threshold 2 seconds from the deadline. BBA-0 quality decision depends on two thresholds. If the buffer occupancy is lower (higher) than the lower (higher) threshold, chunks are fetched at the lowest (highest) quality. If the buffer occupancy lies in between the two thresholds, the buffer-rate relationship is linear. We use the 8 and 16s as the lower and upper thresholds.

Prediction-based Cooperative streaming Approach (Base 2): This approach combines the predicted bandwidths of the

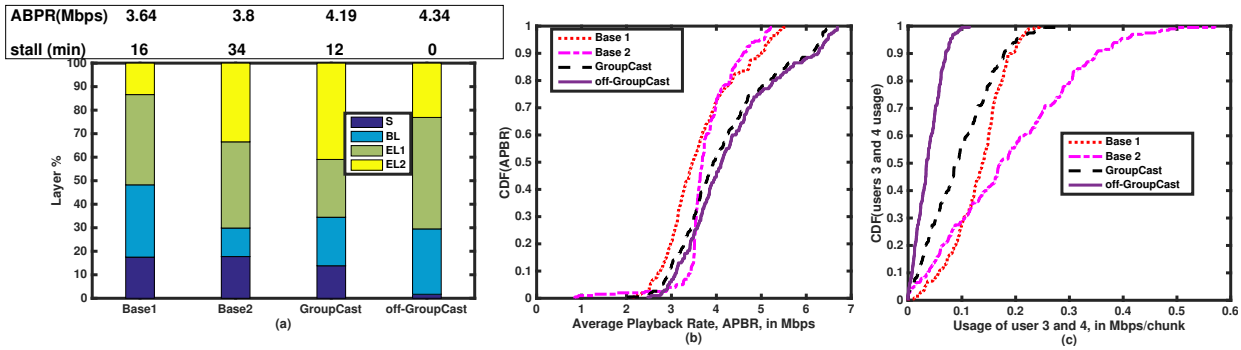

Fig. 2. Simulation results: (a) layer distribution, (b) CDF of Average Playback Rate (APBR), and (c) CDF of users 3 and 4 bandwidth usage.

users 1 and 2, considers only 85% of the the predicted bandwidth, decides the quality of the next chunk and split layers between the 2 users. It only uses users 3 and 4 if the base layer has not yet been downloaded within 2 seconds from the deadline.

We will now evaluate the proposed algorithm (GroupCast) and compare it to the two baseline approaches described in Section V. The results are shown in Fig. 2. We use GroupCast and off-GroupCast to refer to online and offline Pref GroupCast respectively.

Fig. 2-a shows the probability mass function of the number of chunks fetched at the different qualities. We see that GroupCast achieves the lowest re-buffering (stall) duration (12 minutes) yet it still achieves 16% and 11% higher average playback rate than Base1 and Base2 respectively. Moreover, Fig. 2-b clearly shows that GroupCast achieves the highest average playback rate in almost every single bandwidth trace.

However, when users 1 and 2 are preferred over 3 and 4, not only the average playback rate and the stall duration are important, but the bandwidth usage of less-preferable users (users 3 and 4) is also important. Thus, in Fig. 2-c, we plot the CDF of users 3 and 4 bandwidth usage. We clearly see that GroupCast achieves the minimum usage of users 3 and 4 as compared to Base 1 and 2. GroupCast uses users 3 and 4 to fetch less than 0.1 Mbps/chunk for 50% of the bandwidth traces. In contrast, Base 1 and 2 have used both users 3 and 4 in total, 1.5 and 2 times higher than GroupCast respectively. Thus, the proposed algorithms achieves less stall duration, and gives higher average qualities than the two baselines with significantly lower usage of users 3 and 4. The integration of the chunk deadlines, prediction, and user preferences into its optimized decisions makes GroupCast adaptive and robust. Finally, off-GroupCast, provides a theoretic upper bound that can be achieved for perfect bandwidth prediction for the whole video duration and infinite buffer size. We see that GroupCast is the closest one to the upper bound but there is still room for improvement. Bandwidth prediction technique is a key in improving the online algorithm.

VI. Conclusion and Future Work

GroupCast, a system for Preference-Aware cooperative video streaming is proposed. The system problem is formulated as an optimization problem. Low complexity novel algorithms were proposed to solve the problem. Currently, we are working on real implementation of the system and investigating other bandwidth prediction techniques such as geo-location and neural network based prediction techniques.

References

[1] "Scalable Video Coding," https://goo.gl/I5C9RL, 2017, online; accessed 14 July 2017.

[2] M. Stiemerling and S. Kiesel, "A system for peer-to-peer video streaming in resource constrained mobile environments," in *Proceedings of the 1st ACM workshop on User-provided networking: challenges and opportunities*. ACM, 2009, pp. 25–30.

[3] A. Le, L. Keller, H. Seferoglu, B. Cici, C. Fragouli, and A. Markopoulou, "Microcast: Cooperative video streaming using cellular and d2d connections," *arXiv preprint arXiv:1405.3622*, 2014.

[4] E. Yaacoub, F. Filali, and A. Abu-Dayya, "Svc video streaming over cooperative lte/802.11 p vehicle-to-infrastructure communications," in *Computer and Information Technology (WCCIT), 2013 World Congress on*. IEEE, 2013, pp. 1–5.

[5] C.-H. Lee, C.-M. Huang, C.-C. Yang, and T.-H. Wang, "Co-svc-mdc-based cooperative video streaming over vehicular networks," *The Computer Journal*, vol. 55, no. 6, pp. 756–768, 2011.

[6] R. An, Z. Liu, and Y. Ji, "Svc-based cooperative video streaming in highway vehicular networks," in *Advanced Information Networking and Applications Workshops (WAINA), 2017 31st International Conference on*. IEEE, 2017, pp. 216–221.

[7] Y.-C. Chen, D. Towsley, and R. Khalili, "Msplayer: Multi-source and multi-path video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 8, pp. 2198–2206, 2016.

[8] A. Elgabli, V. Aggarwal, S. Hao, F. Qian, and S. Sen, "DBABP: Robust Rate Adaptation Algorithm for SVC Video Streaming," *Submitted to IEEE/ACM Transaction in Networking*, 2017.

[9] C. Kreuzberger, D. Posch, and H. Hellwagner, "A Scalable Video Coding Dataset and Toolchain for Dynamic Adaptive Streaming over HTTP," in *Proc. ACM MMSys*, 2015.

[10] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 2013, pp. 114–118.

[11] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 325–338, 2015.

[12] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. ACM SIGCOMM*, 2014.