LBP: Robust Rate Adaptation Algorithm for SVC Video Streaming

Anis Elgabli, *Student Member, IEEE*, Vaneet Aggarwal, *Senior Member, IEEE*, Shuai Hao, Feng Qian, and Subhabrata Sen, *Fellow, IEEE*

***Abstract*—Video streaming today accounts for up to 55% of mobile traffic. In this paper, we explore streaming videos encoded using scalable video coding (SVC) scheme over highly variable bandwidth conditions, such as cellular networks. SVC's unique encoding scheme allows the quality of a video chunk to change incrementally, making it more flexible and adaptive to challenging network conditions compared to other encoding schemes. Our contribution is threefold. First, we formulate the quality decisions of video chunks constrained by the available bandwidth, the playback buffer, and the chunk deadlines as an optimization problem. The objective is to optimize a novel quality-of-experience metric that models a combination of the three objectives of minimizing the stall/skip duration of the video, maximizing the playback quality of every chunk, and minimizing the number of quality switches. Second, we develop layered bin packing (LBP) adaptation algorithm, a novel algorithm that solves the proposed optimization problem. Moreover, we show that LBP achieves the optimal solution of the proposed optimization problem with linear complexity in the number of video chunks. Third, we propose an online algorithm (online LBP) where several challenges are addressed, including handling bandwidth prediction errors and short prediction duration. Extensive simulations with real bandwidth traces of public datasets reveal the robustness of our scheme and demonstrate its significant performance improvement as compared with the state-of-the-art SVC streaming algorithms. The proposed algorithm is also implemented on a TCP/IP emulation test bed with real LTE bandwidth traces, and the emulation confirms the simulation results and validates that the algorithm can be implemented and deployed on today's mobile devices.**

***Index Terms*—Video streaming, adaptive bit rate streaming, scalable video coding, combinatorial optimization, bandwidth prediction.**

I. Introduction

MOBILE video has emerged as a dominant contributor to cellular traffic. It already accounts for around $40 - 55$ percent of all cellular traffic and is forecast to grow by around 55 percent annually through 2021 [1]. While its popularity is on the rise, delivering high quality streaming video over cellular networks remains extremely challenging. In particular, the video quality under challenging conditions such as mobility and poor wireless channel is sometimes unacceptably poor. Almost every viewer at some point in time can relate to experiences of choppy videos, stalls, *etc*.

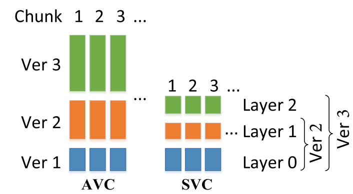

Fig. 1. AVC vs SVC Encoding.

Not surprisingly, a lot of attention from both research and industry in the past decade has focused on the development of *adaptive* streaming techniques for video on demand that can dynamically adjust the quality of the video being streamed to the changes in network conditions. Such a scheme has 2 main components:

• *Content Encoding*: On the server side, the video is divided into multiple *chunks* (segments), each containing data corresponding to some playback time (e.g., 4 sec), and then each chunk is encoded at multiple resolutions/quality levels (each with different bandwidth requirements).

• *Adaptive Playback*: During playtime, an entity (typically the player) dynamically switches between the different available quality levels as it requests the video over the network. The adaptation is based on many factors such as the network condition, its variability, and the client buffer occupancy *etc.*. This results in a viewing experience where different chunks of the video might be streamed at different quality levels.

In the predominant adaptive coding technique in use today, each video chunk is stored into L *independent* encoding versions, as an example of such a technique is H.264/MPEG-4 AVC (Advanced Video Coding) which was standardized in 2003 [2]. During playback when fetching a chunk, the Adaptive Bit Rate (ABR) streaming technique such as MPEG-DASH [3] (Distributed Dynamic Streaming over HTTP) needs to select one out of the L versions based on its judgement of the network condition and other aforementioned factors.

An alternative encoding scheme is Scalable Video Coding (SVC) which was standardized in 2007 as an extension to H.264 [4]. In SVC, a chunk is encoded into ordered *layers*: one *base layer* (Layer 0) with the lowest playable quality, and multiple *enhancement layers* (Layer i >0) that further improve the chunk quality based on layer $i - 1$. When downloading a chunk, an Adaptive-SVC streaming logic must consider

Manuscript received October 22, 2017; revised April 5, 2018; accepted May 22, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor T. Hou. Date of publication June 22, 2018; date of current version August 16, 2018. The work of A. Elgabli and V. Aggarwal was supported by the U.S. National Science Foundation under Grants CCF-1527486 and CNS-1618335. *(Corresponding author: Vaneet Aggarwal.)*

A. Elgabli is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: aelgabli@purdue.edu).

V. Aggarwal is with the School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: vaneet@purdue.edu).

S. Hao and S. Sen are with AT&T Labs Research, Bedminster, NJ 07921 USA (e-mail: haos@research.att.com; sen@research.att.com).

F. Qian is with the Department of Computer Science, Indiana University Bloomington, Bloomington, IN 47405 USA (e-mail: fengqian@indiana.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2018.2844123

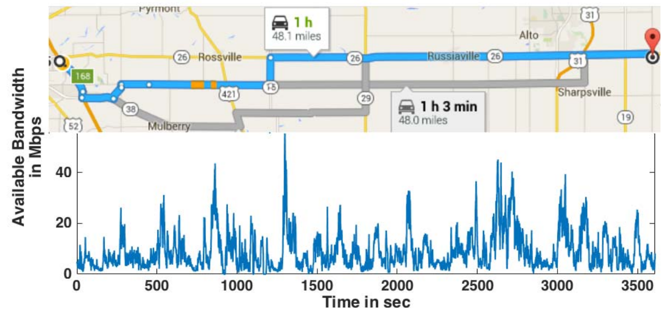

Fig. 2. Motivating example: network condition prediction can improve streaming quality under mobility.

fetching all layers from 0 to $i-1$ if layer i is decided to be fetched. In contrast, in AVC, different versions (*i.e.,* qualities) of chunks are independent, as illustrated in Fig. 1.

There are three typical modes of scalability, namely temporal (frame rate), spatial (spatial resolution), and quality (fidelity, or signal-to-noise ratio). The encoding however has an additional encoding overhead, which depends on the mode of scalability. For example [5] showed that there is minimal or no loss in coding efficiency using temporal scalability. Temporal scalability is also backward compatible with existing H.264 decoders, and is simple to implement as compared to other forms of scalability. However, there are some limitations for using temporal scalability such as being visually un-pleasing for low base layer rates which motivate the use of other scalability modes that require more overhead. Appendix A in the Supplementary Material describes some common scenarios where Adaptive-SVC streaming can be beneficial.

To motivate our problem, imagine a scenario where a mobile user starts a trip from point A to point B (see Fig. 2, anonymized with randomly chosen locations). As the user enters the destination location to the GPS application, she gets the route information, and the video player obtains the estimates of the bandwidth availability along the chosen path. The bandwidth estimation can be obtained using crowd-sourced information from measurements of other users who travelled the same route recently as we will show in Appendix B in the Supplementary Material. We will demonstrate that access to such information can help the player take significantly better informed decisions in its adaptation logic. For example, if the player is aware that it is about to traverse through a region with low bandwidth, it can switch to fetching the video at a lower quality to minimize the possibility of stalling. Another method to predict the future bandwidth that has been widely used in the literature is the harmonic mean based prediction [6], [7], which uses the harmonic mean of the past few seconds to predict the bandwidth for the next few seconds.

In this paper, we first theoretically formulate the problem of adaptive-SVC video streaming with the knowledge of future bandwidth. We consider two streaming schemes: *skip based* and *no-skip based* streaming. The former is usually for real-time streaming in which there is a playback deadline for each of the chunks, and chunks not received by their respective deadlines are skipped. For no-skip based streaming, if a chunk cannot be downloaded by its deadline, it will not be skipped; instead, a stall (re-buffering) will incur, *i.e.,* the video will pause until the chunk is fully downloaded. In both variants, the goal of the proposed scheduling algorithm is to determine up to which layer we need to fetch for each chunk (except for those skipped in realtime streaming), such that the overall quality-of-experience (QoE) is maximized and the number of stalls or skipped chunks is minimized. The key contributions of the paper are described as follows.

- A novel metric of QoE is proposed for SVC streaming in both the scenarios (skip and no-skip). The metric is a weighted sum of the layer sizes for each chunk. Since the user's QoE is concave in the playback rate [8], the higher layers contribute lower to the QoE as compared to the lower layers. Thus, the weights decrease with the layer index modeling the diminishing returns for higher layers.
- We show that even though the proposed problem is a non-convex optimization problem with integer constraints, it can be solved optimally using an algorithm with a complexity that is linear in the number of chunks. The proposed algorithm, "Layered Bin Packing" (LBP), proceeds layer-by-layer, tries to efficiently bin-pack all chunks at a layer and provides maximum bandwidth to the next layer's decisions given the decisions of the lower layers of all the chunks.
- We propose an online robust adaptive-SVC streaming algorithm (Online LBP). This algorithm exploits the prediction of the network bandwidth for some time ahead, solves the proposed optimization problem to find the quality decisions for W chunks ahead, and re-runs every α seconds to adjust to prediction errors and find quality decisions for more chunks ahead.
- We considered two techniques of bandwidth prediction. First, harmonic mean based prediction which was widely used in the literature [6], [7] where the harmonic mean of the past few seconds is used to predict the bandwidth for few seconds ahead (typically 20 seconds ahead). Second, crowd-sourced erroneous bandwidth prediction where bandwidth profiles experienced by people travelled the same road recently are used to predict the bandwidth for the current user.
- Trace-driven simulation using datasets collected from commercial cellular networks demonstrates that our approach is robust to prediction errors, and works well with short prediction windows (*e.g.,* 20 seconds). The proposed approach is compared with a number of adaptation strategies including slope based SVC streaming [9], Microsoft's smooth streaming algorithm (adapted to streaming SVC content), and Netflix's buffer-based streaming algorithm (BBA-0) [10] (adapted to SVC).
- The results demonstrate that our algorithm outperforms the state-of-the-art by improving key quality-of-experience (QoE) metrics such as the playback quality, the number of layer switches, and the number of skips or stalls.
- In addition to the simulations, we built a testbed that streams synthetic SVC content over TCP/IP networks using real LTE traces. We then implemented our streaming algorithm on the testbed and evaluated it under challenging network conditions. The emulation outcome is very close to the simulation results and incurs very low run-time overhead, further confirming that our algorithm can be practically implemented and deployed on today's mobile devices.

II. Related Work

Video streaming has received a lot of attention from both the academia and industry in the past decade. We summarize some of the efforts devoted to streaming technologies that are based on Adaptive Bit Rate (ABR), Adaptive-SVC, and that rely on network bandwidth prediction.

ABR Streaming: The recent adoption of the open standards MPEG-DASH [3] has made ABR streaming the most popular video streaming solution. Commercial systems such as Apple's HLS [11], Microsoft's Smooth Streaming [12], and Adobe's HDS [13] are all ABR streaming algorithms. In recent studies, researchers have investigated various approaches for making streaming decisions, for example, by using control theory [6], [14], Markov Decision Process [15], machine learning [16], client buffer information [10], and data-driven techniques [17]–[19]. In this work, we use an optimization-based approach to design novel streaming algorithms for Adaptive-SVC streaming whose encoding scheme is very different from that of used for ABR streaming.

Adaptive-SVC Streaming: SVC encoding received the final approval to be standardized as an amendment of the H.264/MPEG-4 standard in 2007 [4]. Although much less academic research has been conducted on Adaptive SVC streaming compared to ABR streaming, there exist some studies of using SVC encoded videos to adapt video playback quality to network conditions. A prior study [20] proposed a server-based quality adaptation mechanism that performs coarse-grained rate adaptation by adding or dropping layers of a video stream. While this mechanism was designed to be used over UDP with a TCP-friendly rate control, more recent research has explored techniques that use Adaptve-SVC streaming over HTTP. A study [21] compared SVC with regular H.264 encoding (H.264/MPEG). Their results suggest that SVC outperforms H.264/AVC for scenarios such as VoD and IPTV through more effective rate adaptation. The work [22] published the first dataset and toolchain for SVC. Some prior work [23], [24] proposed new rate adaptation algorithms for Adaptive-SVC streaming that prefetch future base layers and backfill current enhancement layers. Our work differs from the above in that we develop low-complexity algorithms that explicitly and strategically leverage the future knowledge of network conditions for better rate adaptation.

Streaming That Exploits Network Condition Prediction: The knowledge of the future network conditions can play an important role in Internet video streaming. A prior study [25] investigated the performance gap between state-of-the-art streaming approaches and the approach with accurate bandwidth prediction for ABR. The results indicate that prediction brings additional performance boosts for ABR, and thus motivates our study. Prior studies [26], [27] proposed ABR streaming mechanisms that use pre-collected geo-tagged network bandwidth profiles. Our work also exploits the predictable nature of future network conditions, but provides an optimization based framework in the context of SVC-based encoding. In Appendix B in the Supplementary Material, we show more evidence of network predictability in the context of cellular networks. We note that even though there is a broad interest in the bitrate adaptation algorithms, a principled understanding of algorithms is limited. One of the key fundamental approachs to formulate the optimization problem was given in [6]. However, the proposed algorithm in [6] is computationally hard and thus a lookup table is hard coded based on solving the optimization problem offline for a given set of encoding rates. To make the table size small, the offline solution is divided in coarse bins thus giving an approximate solution. [19] proposed crowd-source based bandwidth prediction and used the streaming algorithm proposed in [6] to make the bit rate decision per video's chunk. Moreover, [28] gives feasible solution by relaxing the integer constraints in the streaming optimization problem. Further, [29] considers prediction-based formulation while giving heuristics to solve the problem.

In contrast to [6], we propose an online algorithm that solves the optimization problem optimally in linear complexity and can run on the fly. Thus, the proposed approach does not need to hard code information for different encoding rates. Moreover, the offline algorithm is shown to be also optimal and is solvable in linear time complexity. Therefore, we provide a theoretic upper bound to our formulation. Finally, we do not relax any of the constraint, we consider both skip and no-skip based streaming scenarios, and we show optimality in both cases.

III. System Model

We consider the problem of adaptively streaming an SVC video. An SVC encoded video is divided into C chunks (segments) and stored at a server. Every chunk is of length L seconds, and is encoded in Base Layer (BL) with rate r_0 and N enhancement layers $(E_1, \cdots, E_N)$ with rates $r_1, \cdots, r_N \in \mathcal{R} \triangleq \{0, r_0, r_1, \cdots, r_N\}$. We assume that each layer is encoded at constant bit rate (CBR). In other words, all chunks have the same nth layer size. Let the size of the n-th layer of chunk i be $Z_{n,i} \in \mathcal{Z}_n \triangleq \{0, Y_n\}$, where $Y_n = L \times r_n$. Let the size of a chunk that is delivered at the n-th layer quality be $X_n(i)$, where $X_n(i) = \sum_{m=0}^{n} Y_m$.

Let $z_n(i,j)$ be the size of layer n of chunk i that is fetched at time slot j, and $x(i,j)$ be what is fetched of all layers of chunk i at time slot j, i.e., $x(i,j) = \sum_{n=0}^{N} z_n(i,j)$. Further, let $B(j)$ be the available bandwidth at time j. For the offline algorithm, we assume the bandwidth can be perfectly predicted. Also let s be the startup delay and B_m be the playback buffer size in time units (*i.e.,* the playout buffer can hold up to B_m seconds of video content). We assume all time units are discrete and the discretization time unit is assumed to be 1 second (which can be scaled based on the time granularity). Since the chunk size is L seconds, the buffer occupancy increases by L seconds when chunk i starts downloading (we reserve the buffer as soon as the chunk start downloading).

The optimization framework can run at either the client or the server side as long as the required inputs are available. A setup where the algorithm is run at the client side is depicted in Fig. 3. The algorithm takes as an input, the predicted bandwidth for the time corresponding to the next C chunks, layer sizes $(Y_0, \ldots, Y_N)$, startup delay (s), and maximum buffer size B_m, and outputs the layers that can be requested for the next C chunks $(Z_{n,i}, i \in \{1, ..C\}, n \in \{0, \ldots, N\})$. The video chunks will be fetched according to the requested policy and in order. For the online algorithm, this process repeats every α seconds, and decisions can be changed on fly since the proposed algorithm adapts to the prediction error.

We consider two scenarios: skip based streaming and no-skip based streaming. For skip streaming, the video is played with an initial start-up (*i.e.,* buffering) delay s seconds and there is a playback deadline for each of the chunks where chunk i need to be downloaded by time $deadline(i)$. Chunks not received by their respective deadlines are skipped. For no-skip streaming, it also has start-up delay. However, if a chunk cannot be downloaded by its deadline, it will not

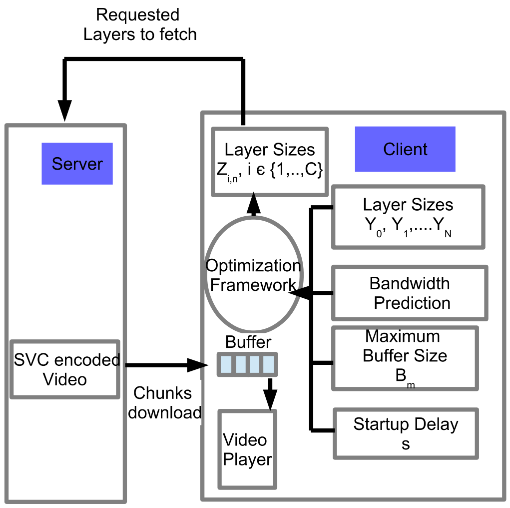

Fig. 3. System Model.

be skipped. Instead, a stall (*i.e.,* rebuffering) will occur *i.e.,* the video will pause until the chunk is fully downloaded. In both scenarios, the goal of the scheduling algorithm to be detailed next is to determine up to which layer we need to fetch for each chunk (except for those skipped), such that the number of stalls or skipped chunks is minimized as the first priority, the overall playback bitrate is maximized as the next priority, the number of quality switching between neighboring chunks is minimized as the third priority. Similar to many other studies on DASH video streaming [6], [29], [10], this paper does not consider mean opinion score (MOS) metric since obtaining MOS ratings are video-dependent and are time-consuming and expensive as they require recruitment of human assessors. A table of notations used in this paper is included in Appendix C in the Supplementary Material.

IV. Adaptive SVC Streaming

We now detail the adaptive SVC streaming algorithms. We describe the basic formulation for skip-based streaming in §IV-A. We then identify the particular problem structure in our formulation and strategically leverage that to design a linear-time solution in §IV-B and §IV-C. We prove the optimality of our solution in §IV-D. An example of the algorithm is given in Appendix E in the Supplementary Material, and detailed proofs are in Appendix F in the Supplementary Material. We then extend the basic scheme to its online version in §IV-E and to no-skip based streaming in §IV-F (with detailed algorithm in Appendix I in the Supplementary Material, example in Appendix J in the Supplementary Material, and proofs in Appendix K in the Supplementary Material).

A. Skip Based Streaming: Offline Problem Formulation

Given the settings described in §III, we first formulate an offline optimization problem. It jointly *(i)* minimizes the number of skipped chunks, *(ii)* maximizes the average playback rate of the video, and *(iii)* minimizes the quality changes between the neighboring chunks to ensure the perceived quality is smooth. We give a higher priority to *(i)* as compared to *(ii)*, since skips cause more quality-of-experience (QoE) degradation compared to playing back at a lower quality [6]. Further, *(iii)* is the lowest priority among the three objectives. The proposed formulation maximizes a weighted sum of the layer sizes. The weights are along two directions. The first is across time where the layers of the later chunks are weighed higher using a factor $\beta \geq 1$. The second is across the layers where fetching the n-th layer of a chunk achieves a utility that is $0 < \gamma < 1$ times the utility that is achieved by fetching the $(n-1)$-th layer. Thus, the objective is given as $\sum_{n=0}^{N} \gamma^n \sum_{i=1}^{C} \beta^i Z_{n,i}$. We further assume that

$$\gamma^a r_a > \sum_{k=a+1}^{N} \gamma^k r_k \sum_{i=1}^{C} \beta^i \quad \text{for } a = 0, \cdots, N-1. \tag{1}$$

This choice of γ implies that all the higher layers than layer a have lower utility than a chunk at layer a for all a. For $a = 0$, this implies that all the enhancement layers have less utility than a chunk at the base layer. Thus, the avoidance of skips is the highest priority. The use of γ helps prioritize lower layers over higher layers and models concavity of user QoE with playback rate. Due to this weight, the proposed algorithm will avoid skip as the first priority and will not use the bandwidth to fetch higher layers at the expense of base layer. Similar happens at the higher layers. The combination of the two weights help minimize multi-layer quality switches between neighboring chunks since the use of γ discourages getting higher layers at the expense of lower layers. We assume $\beta = 1+\epsilon$ where $\epsilon > 0$ is very small number (*e.g.,* 0.001). The use of $\beta = 1+\epsilon$ helps in three aspects, (i) makes optimal layer decisions for different chunks unique, (ii) better adaptability to the bandwidth fluctuations by preferring fetching higher layers of later chunks, and (iii) reduction of quality variations. Indeed, if the playback buffer is not limited, there will ideally be a few jumps of quality increases and no quality decrease in the playback of the chunks using this metric. An example to further explain the objective and the above mentioned points for γ and β is provided in Appendix D in the Supplementary Material.

Overall, the SVC layer scheduling problem with the knowledge of future bandwidth information can be formulated as follows, where $\mathbf{I}(.)$ is an indicator function which has the value 1 if inside expressions holds and zero otherwise.

$$\textbf{Maximize:} \left(\sum_{n=0}^{N} \gamma^n \sum_{i=1}^{C} \beta^i Z_{n,i} \right) \tag{2}$$

$$\text{subject to} \sum_{j=1}^{(i-1)L+s} z_n(i,j) = Z_{n,i}, \quad \forall i, n \tag{3}$$

$$Z_{n,i} \leq \frac{Y_n}{Y_{n-1}} Z_{n-1,i}, \quad \forall i, n > 0 \tag{4}$$

$$\sum_{n=0}^{N} \sum_{i=1}^{C} z_n(i,j) \leq B(j)$$

$$\forall j, \tag{5}$$

$$\sum_{i,(i-1)L+s>t} \mathbf{I}\left(\sum_{j=1}^{t} \left(\sum_{n=0}^{N} z_n(i,j) \right) > 0 \right)$$

$$L \leq B_m \quad \forall t \tag{6}$$

$$z_n(i,j) \geq 0 \quad \forall i \tag{7}$$

$$z_n(i,j) = 0 \quad \forall i, j > (i-1)L + s \tag{8}$$

$$Z_{n,i} \in \mathcal{Z}_n \quad \forall i, n \tag{9}$$

$$\text{Variables: } z_n(i,j), Z_{n,i} \quad \forall i = 1, \cdots, C,$$
$$j = 1, \cdots, (C-1)L + s, \; n = 0, \cdots, N$$

Constraints (3) and (9) ensure that what is fetched for any layer n of a chunk i over all times to be either zero or the n-th layer size. The decoder constraint (4) enforces that the

nth layer of a chunk cannot be fetched if the lower layer is not fetched since this layer will not be decoded because of the layer dependency. (5) imposes the available bandwidth constraint at each time slot j and (6) imposes the playback buffer constraint so that the content in the buffer at any time does not exceed the buffer capacity (given in time units) B_m. Constraint (7) imposes the non-negativity of the chunk download sizes, and (8) enforces not to fetch a chunk after its deadline. The deadline of chunk $i \in \{1, \cdots, C\}$ is $deadline(i) = (i-1)L + s$.

B. Optimization Problem Structure

The problem defined in §IV-A has integer constraints and has an indicator function in a constraint. This problem is in the class of combinatorial optimization [30]. Some of the problems in this class are the Knapsack problem, Cutting stock problem, Bin packing problem, and Travelling salesman problem. These problems are all known to be NP hard. Very limited problems in this class of combinatorial optimization are known to be solvable in polynomial time. Some typical examples being shortest path trees, flows and circulations, spanning trees, matching, and matroid problems. The well known Knapsack problem optimizes a linear function with a single linear constraint (for integer variables), and is known to be NP hard. The optimization problem defined in this paper has multiple constraints, and does not lie in any class of known combinatorial problems that are polynomially-time solvable to the best of our knowledge. In this paper, we will show that this combinatorial optimization problem can be solved optimally in polynomial time.

Algorithm 1 Layered Bin Packing Adaptive Algorithm

Input: Y_n, $deadline(i)$, s, B_m, C, $B(j)$: available bandwidth at time j,
Output: $X(i) \forall i$: The maximum size in which chunk i can be fetched, I_n: set contains the indices of the chunks that can be fetched up to layer n quality.
Initialization:
$X_n = \sum_{m=0}^{n} Y_m$ cumulative size up to layer n
$c(j) = \sum_{j'=1}^{j} B(j')$ cumulative bandwidth up to time $j, \forall j$
$t(i) = 0, \forall i$, first time slot chunk i can be fetched
$a(i) = 0, \forall i$, lower layer decision of fetched amount of chunk i at its lower deadline time $t(i)$
$e(j) = B(j), \forall j$, remaining bandwidth at time j after all non skipped chunk are fetched according to lower layer size decisions
$X(i) = 0, deadline(i) = (i-1)L + s \quad \forall i$
$bf(j) = 0, \forall j$, buffer length at time j
For each layer, $n = 0, \cdots, N$
$[X, I_n] = backwardAlgo(B, X, X_n, C, L, deadline, B_m, bf, t, c, a, e)$
$[t, a, e] = forwardAlgo(B, X, C, deadline, B_m, bf, I_n)$

C. Optimal Linear-Time Solution

We now show the proposed problem in (2-9) can be solved optimally with a complexity of $O(CN)$. We call our proposed algorithm "Layred Bin Packing Adaptive Algorithm" (LBP), which is summarized in Algorithm 1. At a high level, our algorithm works from the lowest (*i.e.,* the base) to the highest enhancement layer, and processes each layer separately. It performs backward and forward scans (explained below) at each layer given the decisions of the previous layers.

Algorithm 2 Backward Algorithm

1: **Input:** $B, X, X_n, C, L, deadline, B_m, bf, t, c, a, e$
2: **Output:** $X(i)$ size of chunk i, I_n: set contains chunks that can be fetched in quality up to n^{th} layer.
3: **Initilization:**
4: $i = C$, $j = deadline(C)$
5: initialize $bf(j)$ to zeros $\forall j$.
6: **while** ($j > 0$ and $i > 0$) **do**
7:   **if** $j <= deadline(i)$ **then**
8:     **if** $(bf(deadline(i)) = B_m)$ **then** $i = i - 1$
9:     **if** j is the first time to fetch chunk i from back **then**
10:       **if** $(t(i) = 0)$ **then**
11:         $rem1 = c(j) - c(1) + e(1)$, $rem2 = rem1$
12:       **else**
13:         $rem2 = c(j) - c(t(i))$, $rem1 = rem2 + e(t(i)) + a(i)$
14:       **end if**
15:       **if** $(rem1 < X_n(i))$ **then**
16:         **if** $(X(i) > 0)$ **then** $X_n(i) = X(i)$ **else** $i = i - 1$
17:       **else**
18:         **if** $(rem2 < X_n(i))$ and $rem1 \geq X_n(i))$ **then**
19:           $e(t(i)) = e(t(i)) + rem1 - X_n$
20:         **end if**
21:         $X(i) = X_n(i)$, $I_n \leftarrow I_n \cup i$
22:       **end if**
23:     **end if**
24:     $fetched = min(B(j), X_n(i))$, $B(j) = B(j) - fetched$
25:     $X_n(i) = X_n(i) - fetched$
26:     **if** $(X_n(i) > 0)$ **then** $bf(j) = bf(j) + L$
27:     **if** $(X_n(i) = 0)$ **then** $i = i - 1$
28:     **if** $(B(j) = 0)$ **then** $j = j - 1$
29:   **else**
30:     $j = j - 1$
31:   **end if**
32: **end while**

Running the backward scan at the nth layer (Algorithm 2) finds the maximum number of chunks that can be fetched up to the nth layer quality given the decisions of the previous layers. Then, running the forward scan (Algorithm 3) simulates fetching chunks in sequence as early as possible, so the start time of downloading chunk i (the lower deadline $t(i)$) is found. Lower and Upper ($t(i)$, $deadline(i)$) deadlines will be used to find the next layer decisions (as explained below).

Backward Algorithm for Base Layer: Given the bandwidth prediction, chunk deadlines, and the buffer size, the algorithm simulates fetching the chunks at base layer quality starting from the last towards the first chunk. The deadline of the last chunk is the starting time slot of the backward algorithm scan. The goal is to have chunks fetched closer to their deadlines. For every chunk i, the backward algorithm checks the bandwidth and the buffer; if there is enough bandwidth and the buffer is not full, then chunk i is selected to be fetched (line 18-22). The algorithm keeps checking this

Algorithm 3 Forward Algorithm

1: **Input:** $B, X, C, deadline, Bm, bf, I_n$
2: **Output:** $t(i)$: first time slot chunk i can be fetched (lower deadline of chunk i), $a(i)$, decision of fetched amount of chunk i at its lower deadline time slot $t(i)$, $e(j)$, remaining bandwidth at time j after all non skipped chunk are fetched according to the decided layer size.
3: $j = 1$, $k = 1$
4: **while** $j \leq deadline(C)$ and $k \leq max(I_0)$ (last chunk to fetch) **do**
5: $\quad i = I(k)$
6: $\quad$ **if** $i = 0$ **then** $k = k + 1$
7: $\quad$ **if** $j \leq deadline(i)$ **then**
8: $\quad\quad$ **if** $(bf(j) = B_m)$ **then** $j = j + 1$
9: $\quad\quad fetched = min(B(j), X(i))$
10: $\quad\quad$ **if** j is the first time chunk i is fetched **then**
11: $\quad\quad\quad t(i) = j,$
12: $\quad\quad\quad a(i) = fetched$
13: $\quad\quad$ **end if**
14: $\quad\quad B(j) = B(j) - fetched$
15: $\quad\quad e(j) = B(j)$, $X(i) = X(i) - fetched$
16: $\quad\quad$ **if** $X(i) > 0$ **then** $bf(j) = bf(j) + L$
17: $\quad\quad$ **if** $X(i) = 0$ **then** $k = k + 1$
18: $\quad\quad$ **if** $B(j) = 0$ **then** $j = j + 1$
19: $\quad$ **else**
20: $\quad\quad k = k + 1$
21: $\quad$ **end if**
22: **end while**

feasibility to select chunks to be fetched. If a chunk i' is not selected to be fetched, one of the following two scenarios could have happened. The first scenario is the violation of the **buffer** capacity, where selecting the chunk to be fetched would violate the playback buffer constraint. The second is the **bandwidth** constraint violation where the remaining available bandwidth is not enough for fetching a chunk. This scenario also means that the chunk could not be fetched by its deadline, so it can also be called deadline violation.

For buffer capacity violation, we first note that, there could be a chunk $i'' > i'$ in which if it is skipped, chunk i' can still be fetched. However, the backward algorithm decides to skip downloading chunk i' (line 8). We note that since there is a buffer capacity violation, one of the chunks must be skipped. The reason of choosing to skip chunk i' rather than a one with higher index is that i' is the closest to its deadline. Therefore, i' is not better candidate to the next layer than any of the later ones. In the second case of deadline/bandwidth violation, the backward algorithm decides to skip chunks up to i' since there is not enough bandwidth. As before, since equal number of chunks need to be skipped anyway, skipping the earlier ones is better because it helps in increasing the potential of getting higher layers of the later chunks.

Forward Algorithm for Base Layer: The forward algorithm takes the chunk size decisions from the Backward step which provides the base layer size decision of every chunk i which is either 0 or the BL size. Then, the forward algorithm simulates fetching the chunks in sequence starting from the first one. Chunks are fetched as early as possible with the deadline, buffer, and the bandwidth constraints being considered. The chunks that were not decided to be fetched by the Backward Algorithm are skipped (any chunk $i \notin I_0$, line 6). The forward algorithm provides the the earliest time slot when chunk i can be fetched ($t(i)$, line 10). This time is used as a **lower deadline** on the time allowed to fetch chunk i when the backward algorithm is run for the next layer. Therefore, the backward size decisions of base layer of earlier chunks can not be violated when the backward algorithm is re-run for deciding the first enhancement layer sizes ($E1$ decisions). Moreover, it provides the portion that can be fetched of chunk i at its lower deadline $t(i)$ ($a(i)$, line 11) and the remaining bandwidth at every time slot j after all non skipped chunk are fetched ($e(j)$, line 12).

Modifications for Higher Layers: The same backward and forward steps are used for each layer given the backward-forward decisions of the previous one on the chunk sizes and lower deadlines. The key difference when the algorithm is run for the enhancement layer decisions as compared to that for the base layer is that the higher layer of the chunk is skipped if the previous layer is not decided to be fetched. When running the backward algorithm for $E1$ decisions, for every chunk i, we consider the bandwidth starting from the lower deadline of that chunk $t(i)$, so previous layer decisions (base layer decisions) of early chunks can't be violated. The same procedure is used to give higher layer decisions when all of the lower layer decisions have already been made. An example to illustrate the algorithm is given in Appendix J in the Supplementary Material.

Complexity Analysis: The initialization clearly sums the variables over time, and is at most O(C) complexity. At each layer, a backward and a forward algorithm are performed. Both the algorithms have a while loop, and within that, each step is O(1). Thus, the complexity is dependent on the number of times this loop happens. For the backward algorithm, each loop decreases either i or j and thus the number of times the while loop runs is at most $C + deadline(C) + 1$. Similarly, the forward algorithm while loop runs at most $C + deadline(C) + 1$ times. In order to decrease the complexities, cumulative bandwidth for every time slot t, $r(t)$ is used to avoid summing over the bandwidth in the backward and the forward loops.

Adaptation to ABR Streaming: We note that the proposed algorithm selects quality levels for every chunk and can also be used for ABR streaming. For a given set of available ABR rates, the difference between the rates for the coded chunk at quality level $n + 1$ and quality level n can be treated as the nth layer SVC rate for all n.

D. Optimality of the Proposed Algorithm

In this subsection, we prove the optimality of Layered Bin-Packing Adaptive Algorithm in solving the optimization problem (2-9). We first note that it is enough to prove that the algorithm is the best among any in-order scheduling algorithm (that fetches chunks in order based on the deadlines). This is because for any other feasible fetching algorithm, we can convert it to an in-order fetching algorithm with the same bandwidth utilizations for each chunk. Getting in-order helps the buffer and other constraints. Thus, we can obtain the same objective and can satisfy the constraints. The following Lemma states that given the lower and upper deadlines (($t(i)$) and $deadline(i)$) of every chunk i, the $(n-1)th$ layer quality decision, running the backward algorithm for the nth layer maximizes the number of chunks that can have their nth layer fetched.

Lemma 1: Given size decisions up to $(n-1)$th layer, and lower and upper deadlines ($t(i)$, and $deadline(i)$) for every chunk i, the backward algorithm achieves the minimum number of the nth layer skips as compared to any feasible algorithm which fetches the same layers to every chunk up to the layer $n-1$.

Proof: Proof is provided in Appendix F in the Supplementary Material. □

The above lemma shows that backward algorithm minimizes the nth layer skips given the lower and upper deadlines of every chunk. However, it does not tell us if that lower deadline is optimal or not. The following proposition shows that for any quality decisions, the forward algorithm finds the optimal lower deadline on the fetching time of any chunk.

Proposition 1: if $t_f(i)$ is the earliest time to start fetching chunk i using the forward algorithm (lower deadline), and $t_x(i)$ is the earliest time to fetch it using any other in sequence fetching algorithm, then the following holds true.

$$t_f(i) \leq t_x(i).$$

The above proposition states that $t_f(i)$ is the lower deadline of chunk i, so chunk i can't be fetched earlier without violating size decisions of the lower layers of earlier chunks. Therefore, at any layer n, we are allowed to increase the chunk size of chunk i as far as we can fully fetch it within the period between its lower and upper deadlines. If increasing its size to the n-th layer quality level requires us to start fetching it before its lower deadline, then we should not consider fetching the n-th layer of this chunk. Fetching the n-th layer of this chunk in this case will affect the lower layer decisions and will cause dropping lower layers of some earlier chunks. Since, our objective prioritizes lower layers over higher layers ($0 < \gamma < 1$ and (1)), lower deadline must not be violated. As a simple extension of Lemma 1, we can consider any $\beta \geq 1$.

Lemma 2: Given optimal solution of layer sizes up to the $(n-1)$th layer, and lower and upper deadlines ($t(i)$, and $deadline(i)$) of every chunk i. If $\mathcal{Z}_n^* = (Z_{n,i}^* \forall n, i\ )$ is the n-th layer solution that is found by running the backward algorithm for the nth layer for the nth layer sizes, and $\mathcal{Z}_n' = (Z_{n,i}' \forall n, i\ )$ is a feasible solution that is found by running any other algorithm, then the following holds for any $\beta \geq 1$.

$$\sum_{i=1}^{C} \beta^i Z_{n,i}' \leq \sum_{i=1}^{C} \beta^i Z_{n,i}^* \tag{10}$$

Proof: Proof is provided in the Appendix G in the Supplementary Material. □

We note that Lemma 1 is a corollary of Lemma 2, which can be obtained when $\beta = 1$.

Using Lemma. 1, Proposition. 1, and Lemma. 2, we are ready to show the optimality of Layered Bin Packing Adaptive Algorithm in solving problem (2-9), and this is stated in the following theorem.

Theorem 1: Up to a given enhancement layer $M, M \geq 0$, if $Z_{m,i}^*$ is the size of every layer $m \leq M$ of chunk i that is found by running Layered Bin Packing Adaptive Algorithm, and $Z_{m,i}'$ is the size that is found by running any other feasible algorithm, then the following holds for any $0 < \gamma < 1$, satisfies (1), and $\beta \leq 1$.

$$\sum_{m=0}^{M} \gamma^m \sum_{i=1}^{C} \beta^i Z_{m,i}' \leq \sum_{m=0}^{M} \gamma^m \sum_{i=1}^{C} \beta^i Z_{m,i}^*. \tag{11}$$

In other words, Layred Bin Packing Adaptive Algorithm achieves the optimal solution of the optimization problem (2-9) when $0 < \gamma < 1$, satisfy (1), and $\beta \geq 1$.

Proof: Proof is provided in the Appendix H in the Supplementary Material. □

Algorithm 4 Online Layered Bin Packing Adaptive Algorithm

1: **Input:** $Y_n, deadline(i), s, B_m, C, B(j), W$: the prediction window size, α: the decision reconsideration period.
2: **Output:** $X(i) \forall i$: The maximum size in which chunk i can be fetched, I_n: set contains the indices of the chunks that can be fetched up to layer n quality.
3: **Initialization:**
4: same as Algorithm 1, offline version plus the following:
5: $sc = 1$, the index of the chunk to start with.
6: $ec = 1$, the index of the last chunk to consider.
7: $st = 1$, the current time slot.
8: **Every α seconds do:**
9: collect user position and speed.
10: predict the bandwidth for W seconds ahead.
11: ec =The index of the first chunk has its deadline $\geq st + W$
12: **For each layer,** $n = 0, \cdots, N$
13: $[X, I_n] = backwardAlgo(B, X, X_n, sc, ec, L, deadline, B_m, bf, t, c, a, e)$
14: $[t, a, e] = forwardAlgo(B, X, sc, ec, deadline, Bm, bf, I_n)$
15: sc =last fetched chunk+1
16: st =current time slot

E. Online Algorithm: Dealing With Short and Inaccurate BW Prediction

We face two issues in reality. First, the bandwidth information for the distant future may not always be available. Second, even for the near future, the estimated bandwidth may have errors. To address both of these challenges, we design an online algorithm (Algorithm 4). The algorithm works as follows. Every α seconds, we predict the bandwidth for W seconds ahead (lines 9-10). Typically α is much smaller than W ($\alpha \ll W$). We find the last chunk to consider in this run of the algorithm (line 11). The online algorithm thus computes the scheduling decision only for the chunks corresponding to the next W seconds ahead. We re-compute the quality decisions periodically (every α seconds) in order to adjust to any changes in the prediction. We can also run the computation after the download of every chunk (or layer) due to the low complexity of our algorithm.

Moreover, to handle inaccurate bandwidth estimation, we set lower buffer threshold (B_{min}), so if the buffer is running lower than this threshold, we reduce the layer decision by 1 (except if a chunk is already at base layer quality) (lines 15-16). In the real chunk download, if we are within a certain threshold from the deadline of the current chunk and it is not yet fully downloaded, we stop fetching the remaining of the chunk as far as the base layer is fetched and we play it at the quality fetched so far.

F. No-Skip Based Streaming Algorithm

In No-Skip streaming (*i.e.,* watching a pre-recorded video), when the deadline of a chunk cannot be met, rather than skipping it, the player will stall the video and continue downloading the chunk. The objective here is to maximize the weighted sum of the layer sizes while minimizing the stall duration (the rebuffering time). The objective function is slightly different from equation. (2) since we do not allow to skip the base layers. However, we still allow for skipping the higher layers. For the constraints, all constraints are the same as skip based optimization problem except that we introduce constraint (13) to enforce the $Z_0(i)$ for every chunk i to be equal to the BL size (Y_0). We define the total stall (re-buffering) duration from the start till the play-time of chunk i as $d(i)$. Therefore, the deadline of any chunk i is $(i-1)L+s+d(i)$. The No-Skip formulation can thus be written as:

$$\textbf{Maximize:} \sum_{n=1}^{N}\gamma^n\sum_{i=1}^{C}\beta^i Z_{n,i} - \lambda d(C) \tag{12}$$

$$\text{subject to,} \quad \sum_{j=1}^{(i-1)L+s+d(i)} z_0(i,j) = Y_0 \ \ \forall i \tag{13}$$

$$\sum_{j=1}^{(i-1)L+s+d(i)} z_n(i,j) = Z_{n,i}, \quad \forall i, n>0 \tag{14}$$

$$Z_{n,i} \leq \frac{Y_n}{Y_{n-1}} Z_{n-1,i}, \quad \forall i, n>0 \tag{15}$$

$$\sum_{n=0}^{N}\sum_{i=1}^{C} z_n(i,j) \leq B(j) \quad \forall j \tag{16}$$

$$\sum_{n=0}^{N} \sum_{i,(i-1)L+s+d(i)>t} \mathbf{I}\left(\sum_{j=1}^{t}\left(z_n(i,j)\right)>0\right)$$

$$L \leq B_m \quad \forall t \tag{17}$$

$$z_n(i,j) \geq 0 \quad \forall i \tag{18}$$

$$z_n(i,j) = 0 \quad \forall i, j > (i-1)L+s+d(i) \tag{19}$$

$$d(i+1) \geq d(i) \geq 0 \quad \forall i = 1,\cdots,C-1 \tag{20}$$

$$Z_{n,i} \in \mathcal{Z}_n \quad \forall i, n \tag{21}$$

$$\text{Variables: } z_n(i,j), Z_{n,i}, d(i) \quad \forall i = 1,\cdots,C,$$
$$1 \leq j \leq (C-1)L+s+d(C), \quad n = 0,\cdots,N$$

This formulation converts multi-objective optimization problem with the stall duration and weighted quality as the two parameters into a single objective using a tradeoff parameter λ. λ is chosen such that avoidance of one stall is preferred as compared to fetching all the layers of all chunks since users tend to care more about not running into rebuffering over better quality. Specifically, λ satisfies the following equation.

$$\lambda > \sum_{n=0}^{N}\gamma^n Y_n \sum_{i=1}^{C}\beta^i \tag{22}$$

With this assumption, we can solve the optimization problem optimally with a slight modification to the algorithm proposed for the skip based streaming version. The proposed algorithm for the No-Skip version is referred to by "*No-Skip Layered Bin Packing Adaptive Algorithm*" (No-Skip LBP, Algorithm 5 in Appendix I in the Supplementary Material). There are a few key differences in this algorithm as compared to the skip version, and we explained them below.

One difference as compared to the skip version is that the first step is to determine the minimum stall time since that is the first priority. In order to do this, we simulate fetching chunks in order at BL quality (Base layer forward algorithm, Algorithm 6 in Appendix I in the Supplementary Material). We first let $d(1) = \cdots = d(C) = 0$. We start to fetch chunks in order. If chunk i can be fetched within its deadline $((i-1)L+s+d(i))$, we move to the next chunk (line 20-21). If chunk i cannot be fetched by its deadline, we continue fetching it till it is completely fetched, and the additional time spent in fetching this chunk is added to $d(k)$ for every $k \geq i$ since there has to be an additional stall in order to fetch these chunks (line 22-24). Using this, we obtain the total stall and the deadline of the last chunk ($d(C)$, and $deadline(C)$) The stall duration of the last chunk (chunk C) gives the total stall duration for the algorithm.

The other difference is in running the backward algorithm for the base layer decisions (see base layer backward algorithm, Algorithm 7 in Appendix I in the Supplementary Material). The key difference in running the backward algorithm for the base layer with compare to the skip version is that there must be no BL skips. With the backward algorithm, we will work on moving stalls as early as possible. We run the base layer backward algorithm starting at time slot $j = deadline(C) = (C-1)L+s+d(C)$. The scenario of deadline violation cannot happen due to the procedure of forward step before this. Thus, the possibility of buffer constraint violation must be managed. If we reach a chunk in which there is a buffer constraint violation, we decrement its deadline by 1 and check if the violations can be removed. This decrement can be continued until the buffer constraint violation is avoided (lines 11, 28-29). This provides the deadlines of the different chunks such that stall duration is at its minimum and stalls are brought to the earliest possible time, so we get minimum number of stalls and optimal stall pattern. When stalls are brought to their earliest possible, all chunks can have more time to get their higher layers without violating any of the constraints. Therefore, we have higher chance of getting higher layers of later chunks. Forward algorithm (Algorithm 3) is run after that to simulate fetching chunks in order and provide lower deadlines of chunks for the E1 backward run. For enhancement layer decisions, the backward-forward scan is run as in the skip version case since skips are allowed for the enhancement layers. The main algorithm that calls the forward and backward scans in the sequence we described is "*No-Skip Layered Bin Packing Adaptive Algorithm*" (Algorithm 5). An illustrative example of the algorithm is described in Appendix J in the Supplementary Material.

Lemma 3: If $d^*(C)$ is the total stall duration that is found by No-Skip base layer forward algorithm and $d'(C)$ is the total stall duration that is found by running any other feasible algorithm, then the following holds true:

$$d'(C) \geq d^*(C)$$

TABLE I
SVC ENCODING BITRATES USED IN OUR EVALUATION

playback layer	BL	EL1	EL2	EL3
nominal Cumulative rate (Mbps)	0.6	0.99	1.5	2.075

In other words, the No-Skip base layer forward algorithm achieves the minimum stall duration.

Proof: Proof is provided in Appendix K in the Supplementary Material. □

From Lemma 3, we note that No-Skip forward algorithm would finish playing all chunks at their earliest time. Since all the chunks are obtained at the base layer quality and there is a minimum number of stalls, we note that the objective function is optimized for any $\beta \geq 1$ when only base layer is considered. When running base layer backward algorithm, the deadlines of the chunks are shifted to the last possibilities which gives the maximum flexibility of obtaining higher layers of chunks before their deadlines.

Having shown the result for the base layer and having determined the deadline for the last chunk, the rest of the algorithm is similar to the skip version where only the weighted quality need to be considered (the stall time is already found). Thus, the optimality result as described in the following Theorem holds, where the proof follows the same lines as described for the skip version theorem.

Theorem 2: If $z^*_{m,i}$ is the feasible size of every layer $m \leq M$ of chunk i that is found by running No-Skip Layered Bin Packing Adaptive Algorithm, and $z'_{m,i}$ is a feasible size that is found by any other feasible algorithm for the same stall duration, then the following holds for $0 < \gamma < 1$, (1), $\beta \geq 1$, and (22):

$$\sum_{m=0}^{M} \gamma^m \sum_{i=1}^{C} \beta^i Z'_{m,i} \leq \sum_{m=0}^{M} \gamma^m \sum_{i=1}^{C} \beta^i Z^*_{m,i}$$

In other words, No-Skip Layered Bin Packing Adaptive Algorithm achieves the optimal solution of the optimization problem (12)-(21).

Proof: Proof is provided in Appendix L in the Supplementary Material. □

The No-Skip scheme faces the same challenges described in §IV-E: short bandwidth prediction in the distant future and inaccurate bandwidth prediction, and they are handled the same way described in section §IV-E.

V. EVALUATION

In this section, we evaluate our algorithms (LBP) using both simulation and emulation. Simulation allows us to explore a wide spectrum of the parameter space. We then implemented a TCP/IP-based emulation testbed to compare its performance with simulation and to measure the runtime overhead in §V-D.

A. Simulation Parameters

Simulation Setup: To make our simulation realistic, we choose the SVC encoding rates of an SVC encoded video "Big Buck Bunny", which is published in [22]. It consists of 299 chunks (14315 frames), and the chunk duration is 2 seconds (48 frames and the frame rate of this video is 24fps). The video is SVC encoded into one base layer and three enhancement layers. Table I shows the cumulative nominal rates of each of the layers. The exact rate of every chunk might be different since the video is VBR encoded. 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$_{(i-1)}$.

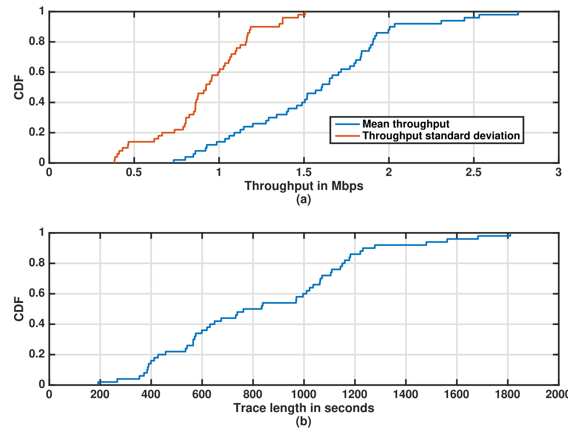

Fig. 4. Statistics of the bandwidth traces: (a) mean and standard deviation of each trace's throughput, and (b) trace length, across the 50 traces.

For all schemes (both the baseline approaches and our algorithms), we assume a playback buffer of 10 seconds ($B_m = 10s$) for the skip version and 2 minutes for the No-Skip version, and a startup delay of 5 seconds. We will systematically study the impact of different algorithm parameters, including prediction accuracy, prediction window size, and playback buffer size in Appendix M in the Supplementary Material. Finally, for all the variants of our algorithms with short prediction ($W \leq 20s$), we choose the lower buffer threshold to be half of the maximum buffer occupancy ($B_{min} = B_m/2$). When the buffer is less than B_{min}, we drop the highest layer that was decided to be fetched (unless the decision is fetching only the base layer). We still run the optimization problem, collect the layer size decisions, but we decrement the number of layers by 1 if enhancement layers are decided to be fetched. This helps being optimistic when the buffer is running low since the algorithm with short prediction have limited knowledge of the bandwidth ahead. All reported results are based on the 50 diverse bandwidth traces described next.

Bandwidth Traces: For bandwidth traces, we used the dataset in [31], which consists of continuous 1-second measurement of video streaming throughput of a moving device in Telenor's 3G/HSDPA mobile network in Norway. The dataset contains 86 bandwidth profiles (traces) for different transportation types including bus, car, train, metro, tram, and ferry. We exclude traces with either very high or low bandwidth since in both cases the streaming strategies are trivial (fetching all layers and only base layers, respectively). We then ended up having 50 traces whose key statistics are plotted in Fig. 4. Overall the traces are highly diverse, with lengths varying from 3 to 30 minutes. We note that since the "Big Buck Bunny" is 598s. The video is re-started for long traces and cut at the end of the trace for short traces.

The average throughput across the traces varies from 0.7Mbps to 2.7 Mbps, with the median being 1.6 Mbps. In each trace, the instantaneous throughput is also highly variable, with the average standard deviation across traces being 0.9 Mbps.

Bandwidth Prediction: We consider two different techniques for bandwidth prediction. First is a harmonic mean based prediction in which the harmonic mean of the bandwidth of the last 5 seconds is used as a predictor of the bandwidth for the next 20 seconds. We refer to our algorithm with harmonic mean based prediction by HM. Second, we assume crowd sourced prediction, and a combination of prediction window size with prediction error percentages. Longer prediction window comes with the cost of higher prediction error. For example we use $(10, 25\%)$ to refer to the prediction window (W) of 10 seconds and the prediction error pe of 25%. In our simulation, the predicted bandwidth is computed by multiplying the actual value in the bandwidth trace (the ground truth) by $1 + e$ where e is uniformly drawn from $[-pe, pe]$ (based on our findings in Appendix B in the Supplementary Material, the prediction error tends to have a mean of 0 in the long run). For skip version (real time streaming), we evaluated our algorithm in case of $(10, 25\%)$ and $(20, 50\%)$ since chunks beyond 20 seconds ahead might not be available yet. However, for the No-Skip version (non-real time streaming), we considered $(20, 50\%)$ and $(100, 60\%)$. We also include the offline scheme *i.e.,* $(\infty, 0)$, for comparison. It corresponds to the performance upper bound for an online algorithm, which is given by our offline algorithm.

B. Skip Based Streaming

We compare our skip-based streaming algorithm (§IV-C) with three baseline algorithms with different aggressiveness levels. Baseline 1 is a conservative algorithm performing "horizontal scan" by first trying to fetch the base layer of all chunks up to the full buffer. If there is spare bandwidth and the playout buffer is not full, the algorithm will fetch the first enhancement layer of buffered chunks that can be received before their playback deadline. If the bandwidth still permits, the algorithm will fetch the second enhancement layer in the same manner. Baseline 2 instead aggressively performs "vertical scan", it fetches all layers of the next chunk before fetching the future chunks. Baseline 3 is a hybrid approach combining Baseline 1 and 2. It first (vertically) fetches all layers of the next chunk and if there is still available bandwidth, it subsequently (horizontally) fetches the base layer of all later chunks before proceeding to their higher layers.

We compare the above three baseline approaches with three representative configurations of our proposed online LBP algorithm. They are referred to as HM (harmonic mean based prediction), $(10, 25\%)$, and $(20, 50\%)$. Moreover, we include our offline algorithm which has a perfect bandwidth prediction for the whole period of the video.

The results are shown in the three subplots of Fig. 5. Fig. 5-a plots the breakdown of the highest fetched layers of each chunk ("S" refers to skipped chunks). For example, for Baseline 1, 26.5% of chunks are fetched only at the base layer quality (shown in light blue). The average playback rate (across all 50 traces) for each scheme is also marked in the plot. As shown, our schemes significantly outperform the three baseline algorithms by fetching more chunks at higher layers with fewer skips. Even when the prediction window is as short as 10 seconds, our scheme incurs negligible skips compared to Baseline 2 and 3, and yields an average playback bitrate that is $\sim$25% higher than Baseline 1. As the prediction window increases (*i.e.,* $W = 20s$ and $pe = 50\%$), the layer distribution becomes very close to the offline scheme.

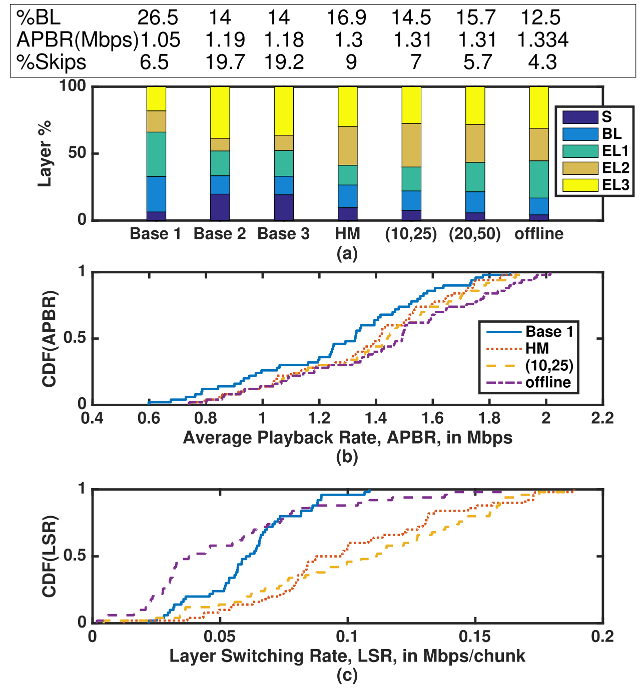

Fig. 5. Skip based streaming results for different schemes: (a) layer distribution, (b) average playback rate, and (c) layer switching rate.

Fig. 5-b plots the CDF of the average playback rate of all the schemes across all traces. As shown, even with a prediction window of as short as 10 seconds, our online scheme achieves playback rates that is the closest to those achieved by the offline scheme across the 50 traces. One more interesting observation from Fig. 5-b is that both variants of our algorithm (HM, and (10,25%)) outperform Baseline 1 in terms of average playback rate in every bandwidth trace. Also note that although Baseline 2 and 3 achieve higher playback rates than Baseline 1, they suffer from a large number of skips as shown in Fig. 5-a.

Fig. 5-c plots for each algorithm the distribution of the layer switching rates (LSR), which is defined as $\frac{1}{C*L}\sum_{i=2}^{C}|X(i) - X(i-1)|$ where C is the number of chunks, L is the chunk duration, and $X(i)$ is the size of chunk i (up to its fetched layer). Intuitively, LSR quantifies the frequency of the playback rate change, and ideally should be minimized. Baseline 1, behaves very conservatively by first fetching the base layer for all chunks up to full buffer. Therefore it has lower layer switching rates at the cost of lower playback rates. Our algorithms instead achieve reasonably low layer switching rates while being able to stream at the highest possible rate with no skips.

We note that larger prediction windows can lead to better decisions even if the prediction has higher error. As long as the bandwidth prediction is unbiased, we see that higher prediction errors can be tolerated. Appendix B in the Supplementary Material shows that crowdsourcing-based prediction is an unbiased predictor of the future bandwidth. Moreover, more results about the effect of the prediction error on the proposed algorithm are described in Appendix M in the Supplementary Material. Further, we show that the computational overhead of the proposed approach is low, as described in Appendix N in the Supplementary Material.

C. No-Skip Based Streaming

We now evaluate the no skip based algorithm. We compare it with three state-of-the-art algorithms: buffer-based algorithm (BBA) proposed by Netflix [10], Naive port of Microsoft's Smooth Streaming algorithm for SVC [9], and a state-of-the-art slope-based SVC streaming approach [23]. To ensure apple-to-apple comparisons, we adopt the same parameter configuration (2-minute buffer size and 1-second chunk size) and apply the algorithms to all our 50 traces. Before describing the results, we first provide an overview of the three algorithms we compare our approach with.

Netflix Buffer-Based Approach (BBA [10]) adjusts the streaming quality based on the playout buffer occupancy. Specifically, it is configured with lower and upper buffer 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 buffer-rate relationship is determined by a pre-defined step function. We use 40 and 80 seconds as the lower and upper thresholds. The quality levels are specified in terms of the SVC layers (*e.g.,* "the highest quality" means up to the highest layer).

Naive Port of Microsoft Smooth Streaming for SVC [9] (NMS) employs a combination of buffer and instantaneous bandwidth estimation for rate adaptation. NMS is similar to BBA in that it also leverages the buffer occupancy level to determine the strategy. The difference, however, is that it also employs the instantaneous bandwidth estimation (as opposed to the long-term network quality prediction we use) to guide rate adaptation. As a result, for example, it can fetch high-layer chunks without waiting for the buffer level reaching the threshold as is the case for BBA.

Slope-Based SVC Streaming [23] takes the advantage of SVC over AVC. It can download the base layer of a new chunk or increase the quality of a previously downloaded (but not yet played) chunk by downloading its enhancement layers. This is achieved by defining a slope function: the steeper the slope, the more backfilling will be chosen over prefetching. Following the original paper's recommendations, we empirically choose 2 slope levels (SB1: -7%, and SB2: -40%). We verified that these two settings provide good results compared to other slope configurations (*e.g.,* going steeper than SB1 causes longer stall duration and going flatter than SB2 makes the playback rate lower).

The results are shown in four subplots in Fig. 6. Fig. 6-a plots the layer breakdown. The average playback rate and the total rebuffering time (across all 50 traces) for each scheme are also marked. As shown, in terms of rebuffering time, our online schemes with crowd sourced bandwidth prediction achieve the lowest stall duration even when the prediction window is as short as 20 seconds ahead. On other hand, NMS performs poorly in terms of avoiding stalls since It runs into almost an hour of stalls (53 minutes). Moreover, all variants of our online algorithm including HM significantly outperform other algorithms in fetching higher layers. For example, (20,50%) fetches only 16% of the chunks at BL quality which is 57%, 70%, 62%, and 58% fewer then BBA0, SB1, SB2, and NMS respectively. Also, as the prediction window increases, the layer distribution becomes closer to the offline scheme, with the shortest stall duration incurred. Fig. 6-b and Fig. 6-c plot for each algorithm the distribution of the (per trace) average playback rate and the stall duration

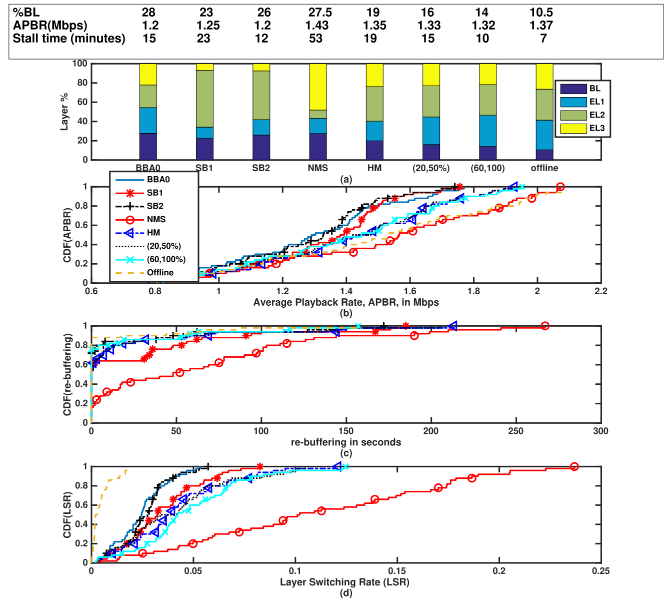

Fig. 6. No-Skip based streaming results for different schemes: (a) layer distribution, (b) average playback rate, (c) total rebuffering time, and (d) layer switching rate.

across all traces. The results are consistent with our findings from Fig. 6-a: our scheme achieves high playback rate that is the closest to the very optimistic algorithms (*e.g.,* NMS) while incurring stalls that are as infrequent as the very conservative algorithms (*e.g.,* SB3 and BBA). Thus, it is clearly shown that our algorithm is maintaining a good trade-off between minimizing the stall duration and maximizing the average playback rate. Fig. 6-d plots for each algorithm the distribution of the layer switching rates (LSR, defined in §V-B). Similar to the skip based scenario, our schemes achieve much lower LSR compared to the aggressive approach (*e.g.,* NMS). The LSR can further be reduced but at the cost of reduced playback rate.

To conclude this section, we would like to point out the key points behind achieving better performance for our algorithm as compared to the baselines. First, incorporating chunk deadlines, bandwidth prediction, and buffer constraint into the optimization problem yields a better decision per chunk. Moreover, favoring the later chunks helps the algorithm avoid being overly optimistic now at the cost of running into skips later on. Finally, re-considering the decisions after the download of every chunk with the new updated bandwidth prediction helps make the algorithm self-adaptive and more dynamically adjustable to the network changes. The low complexity of the algorithm allows for re-running the algorithm and changing decisions on the fly.

D. Emulation Over TCP/IP Network

To complement our simulation results, we have built an emulation testbed using C++ (about 1000 LoC) on Linux. The testbed consists of a client and a server. All streaming logics described in §IV are implemented on the client side, which fetches synthetic chunks from the server over a persistent TCP connection. We deploy our emulation testbed between a commodity laptop and a server inter-connected using high-speed Ethernet (1Gbps link and 1ms RTT). We use `Dummynet` [32] on the client side to replay a bandwidth profile by dynamically changing the available bandwidth every one second. We also use the Linux `tc` tool to inject additional latency between the client and server.

TABLE II
LTE BANDWIDTH TRACES

Trace No.	1	2	3	4	5	6
Average rate (Mbps)	5.05	6.95	5.9	6.14	5.3	6.8
Standard deviation (Mbps)	4.3	6.65	4.7	5.25	3.84	7.02

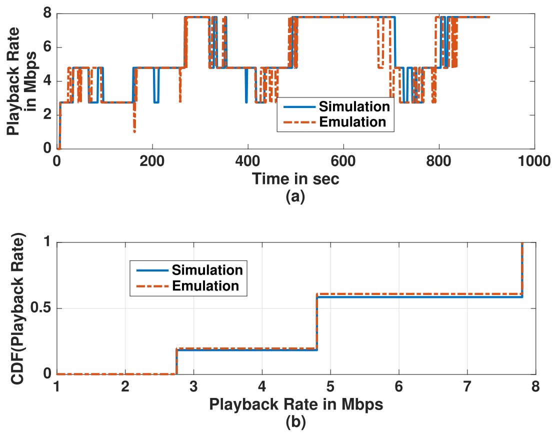

Fig. 7. Emulation vs simulation: (a) playback bitrate over time, (b) chunk quality distribution.

We next run the emulation experiment using six bandwidth traces, each of length 15-minutes. These traces were collected on an LTE network on different drive routes (as described in Appendix B in the Supplementary Material). Table II shows the statistics of the bandwidth traces, and since the bandwidth of the traces are high, we used the following cumulative SVC rates, $1.5Mbps$ (BL), $2.75Mbps$ (EL_1), $4.8Mbps$ (EL_2), $7.8Mbps$ (EL_3) [22]. We configure the end-to-end RTT to be 60ms, which roughly corresponds to the last-mile latency in today's LTE networks. Meanwhile, we run the same bandwidth traces under identical settings using the simulation approach. Since all traces confirm similar behavior, we explain the results of one bandwidth trace, so we can have both the quality CDF and the playback quality over time.

Fig. 7-a compares the simulation and emulation results in terms of the qualities of fetched chunks, and Fig. 7-b compares the chunk quality distribution. As shown, the simulation and emulation results well cross-validate each other. Their slight difference in Fig. 7-a is mainly caused by the TCP behavior (*e.g.,* slow start after idle) that may underutilize the available bandwidth.

VI. CONCLUSIONS AND FUTURE WORK

We formulated the SVC rate adaptation problem as a non-convex optimization problem that has an objective of minimizing the skip/stall duration as the first priority, maximize the average playback as the second priority, and minimize the quality switching rate as the last priority. We develop LBP (Layered Bin Packing Adaptive Algorithm), a low complexity algorithm that is shown to solve the problem optimally in polynomial time. Therefore, offline LBP algorithm that uses perfect prediction of the bandwidth for the whole period of the video provides a theoretic upper bound. Moreover, an online LBP that is based on sliding window and solves the optimization problem for few chunks ahead was proposed for the more practical scenarios in which the bandwidth is predicted for short time ahead and has prediction errors. The results indicate that LBP is robust to prediction errors, and works well with short prediction windows. It outperforms existing streaming approaches by improving key QoE metrics. Finally, LBP incurs low runtime overhead due to its linear complexity.

Extending the results to consider streaming over multiple paths with link preferences is an interesting problem, and is being considered by Elgabli *et al.* [33], [34] and Elgabli and Aggarwal [35].

REFERENCES

[1] *Ericsson Mobility Report on the Pulse of the Networked Society*, Ericsson, Stockholm, Sweden, Jun. 2016.
[2] *H.264/MPEG-4 AVC*. Accessed: Jun. 12, 2018. [Online]. Available: http://handle.itu.int/11.1002/1000/6312
[3] *MPEG-DASH*. Accessed: Jun. 12, 2018. [Online]. Available: https://www.iso.org/standard/57623.html
[4] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.
[5] M. Mathew, "Overview of temporal scalability with scalable video coding (SVC)," Texas Instrum., Appl. Rep. SPRABG3, Dallas, TX, USA, 2010, pp. 1–7. [Online]. Available: http://www.ti.com/lit/an/sprabg3/sprabg3.pdf
[6] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015, pp. 325–338.
[7] Y.-C. Chen, D. Towsley, and R. Khalili, "MSPlayer: Multi-source and multi-path LeverAged YoutubER," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, 2014, pp. 263–270.
[8] K. Miller, A.-K. Al-Tamimi, and A. Wolisz, "QoE-based low-delay live streaming using throughput predictions," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 13, no. 1, 2017, Art. no. 4.
[9] J. Famaey *et al.*, "On the merits of SVC-based HTTP adaptive streaming," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2013, pp. 419–426.
[10] 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, pp. 187–198.
[11] *Apple HTTP Live Streaming*. Accessed: Jun. 12, 2018. [Online]. Available: https://developer.apple.com/streaming/
[12] *Microsoft Smooth Streaming*. Accessed: Jun. 12, 2018. [Online]. Available: https://www.iis.net/downloads/microsoft/smooth-streaming
[13] *Adobe HTTP Dynamic Streaming*. Accessed: Jun. 12, 2018. [Online]. Available: https://www.adobe.com/devnet/hds.html
[14] K. Miller, D. Bethanabhotla, G. Caire, and A. Wolisz, "A control-theoretic approach to adaptive video streaming in dense wireless networks," *IEEE Trans. Multimedia*, vol. 17, no. 8, pp. 1309–1322, Aug. 2015.
[15] D. Jarnikov and T. Özçelebi, "Client intelligence for adaptive streaming solutions," *Signal Process., Image Commun.*, vol. 26, no. 7, pp. 378–389, 2011.
[16] M. Claeys, S. Latré, J. Famaey, and F. De Turck, "Design and evaluation of a self-learning HTTP adaptive video streaming client," *IEEE Commun. Lett.*, vol. 18, no. 4, pp. 716–719, Apr. 2014.
[17] X. Liu *et al.*, "A case for a coordinated Internet video control plane," in *Proc. ACM SIGCOMM*, 2012, pp. 359–370.
[18] A. Ganjam *et al.*, "C3: Internet-scale control plane for video quality optimization," in *Proc. NSDI*, 2015, pp. 131–144.
[19] Y. Sun *et al.*, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. ACM SIGCOMM*, 2016, pp. 272–285.
[20] R. Rejaie, M. Handley, and D. Estrin, "Layered quality adaptation for Internet video streaming," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 12, pp. 2530–2543, Dec. 2000.
[21] Y. Sanchez *et al.*, "Efficient HTTP-based streaming using scalable video coding," *Signal Process., Image Commun.*, vol. 27, no. 4, pp. 329–342, 2012.
[22] 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, pp. 213–218.
[23] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala, "Quality selection for dynamic adaptive streaming over HTTP with scalable video coding," in *Proc. ACM MMSys*, 2012, pp. 149–154. [Online]. Available: http://doi.acm.org/10.1145/2155555.2155580

[24] C. Sieber, T. Hoßfeld, T. Zinner, P. Tran-Gia, and C. Timmerer, "Implementation and user-centric comparison of a novel adaptation logic for DASH with SVC," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2013, pp. 1318–1323.

[25] X. K. Zou *et al.*, "Can accurate predictions improve video streaming in cellular networks?" in *Proc. ACM HotMobile*, 2015, pp. 57–62.

[26] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video streaming using a location-based bandwidth-lookup service for bitrate planning," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 8, no. 3, pp. 24:1–24:19, 2012.

[27] J. Hao, R. Zimmermann, and H. Ma, "GTube: Geo-predictive video streaming over HTTP in mobile environments," in *Proc. ACM MMSys*, 2014, pp. 259–270.

[28] B. Wang and F. Ren, "Towards forward-looking online bitrate adaptation for DASH," in *Proc. ACM Multimedia Conf.*, 2017, pp. 1122–1129.

[29] X. K. Zou *et al.*, "Can accurate predictions improve video streaming in cellular networks?" in *Proc. ACM 16th Int. Workshop Mobile Comput. Syst. Appl.*, 2015, pp. 57–62.

[30] G. L. Nemhauser, M. W. P. Savelsbergh, and G. C. Sigismondi, *Constraint Classification for Mixed Integer Programming Formulations* (Memorandum COSOR). Eindhoven, The Netherlands: Technische Universiteit Eindhoven, 1991. [Online]. Available: https://research.tue.nl/en/publications/constraint-classification-for-mixed-integer-programming-formulati

[31] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: Analysis and applications," in *Proc. 4th ACM Multimedia Syst. Conf.*, 2013, pp. 114–118.

[32] *The Dummynet Project*. Accessed: Jun. 12, 2018. [Online]. Available: http://info.iet.unipi.it/luigi/dummynet/

[33] A. Elgabli, V. Aggarwal, and K. Liu, "Low complexity algorithm for multi-path video streaming," in *Proc. Int. Conf. Signal Process. Commun. (SPCOM)*, Jul. 2018.

[34] A. Elgabli, K. Liu, and V. Aggarwal. (Jan. 2018). "Optimized preference-aware multi-path video streaming with scalable video coding." [Online]. Available: https://arxiv.org/abs/1801.01980

[35] A. Elgabli and V. Aggarwal, "GroupCast: Preference-aware cooperative video streaming with scalable video coding," in *Proc. IEEE Infocom Workshop*, 2018.

[36] C. Müller, D. Renzi, S. Lederer, S. Battista, and C. Timmerer, "Using scalable video coding for dynamic adaptive streaming over HTTP in mobile environments," in *Proc. IEEE 20th Eur. Signal Process. Conf. (EUSIPCO)*, Aug. 2012, pp. 2208–2212

Anis Elgabli (S'17) received the B.S. degree in electrical and electronic engineering from the University of Tripoli, Tripoli, Libya, in 2004, the M.Eng. degree in electrical and computer engineering from UKM, Kajang, Malaysia, in 2006, and the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, IN, USA, in 2015 and 2018, respectively. His research interest is in applying optimization techniques in networking and communication systems. He was a recipient of the 2018 Infocom Workshop HotPOST Best Paper Award.

Vaneet Aggarwal (S'08–M'11–SM'15) received the B.Tech. degree from the Indian Institute of Technology Kanpur, Kanpur, India, in 2005, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, USA, in 2007 and 2010, respectively, all in electrical engineering.

He was a Senior Member of the Technical Staff Research with AT&T Labs Research, Bedminster, NJ, USA, from 2010 to 2014, and an Adjunct Assistant with Columbia University, New York, NY, USA, from 2013 to 2014. He is currently an Assistant Professor with Purdue University, West Lafayette, IN, USA. He is also a VAJRA Adjunct Professor with IISc Bangalore. His current research interests are in communications and networking, video streaming, cloud computing, and machine learning.

Dr. Aggarwal received Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, the AT&T Vice President Excellence Award in 2012, the AT&T Key Contributor Award in 2013, the AT&T Senior Vice President Excellence Award in 2014, the 2017 Jack Neubauer Memorial Award recognizing the Best Systems Paper published in the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and the 2018 Infocom Workshop HotPOST Best Paper Award. He is on the Editorial Board of the IEEE TRANSACTIONS ON COMMUNICATIONS and the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING.

Shuai Hao received the B.Comp. degree (Hons.) and the M.S. degree from the National University of Singapore, Singapore, in 2005 and 2006, respectively, and the Ph.D. degree from the University of Southern California, Los Angeles, CA, USA, in 2014, all in computer science.

He was a Research Staff with the National University of Singapore from 2006 to 2008. He was an Annenberg Fellow from 2008 to 2012 and a Research Assistant from 2012 to 2014 with the University of Southern California. In 2014, he joined AT&T Labs Research, Bedminster, NJ, USA, where he is currently a Senior Inventive Scientist. His research interests include networked systems, mobile computing, sensor network and IoT, and video streaming.

Dr. Hao has published in the ACM MobiSys, the CoNEXT, the ICSE, and the IEEE INFOCOM. His work on reducing web latency was published in the ACM SIGCOMM 2013. He received the Applied Networking Research Prize from the Internet Research Task Force at the IETF 91 Meeting in 2014.

Feng Qian received the bachelor's degree from Shanghai Jiao Tong University, and the Ph.D. degree from the University of Michigan. He is currently an Assistant Professor with the Computer Science Department, Indiana University Bloomington. His research interests cover the broad areas of mobile systems, VR/AR, computer networking, and system security.

Subhabrata Sen (M'01–SM'14–F'16) received the Ph.D. degree in computer science from the University of Massachusetts, Amherst, MA, USA, in 2001. Since 2001, he has been with AT&T Labs Research, where he is currently a Lead Scientist. He has authored or coauthored 96 peer-reviewed research articles. He holds 62 awarded patents. His research interests include Internet technologies and applications, IP network management, application and network performance, video streaming, cross-layer interactions and optimizations for cellular networks, network measurements, and traffic analysis.

Dr. Sen was a recipient of the AT&T Science and Technology Medal, the AT&T Labs President Excellence Award, and the AT&T CTO Innovation Award. He was a co-inventor of the widely used open-source Application Resource Optimizer Tool for analyzing cellular friendliness of mobile application designs that earned top industry honors, including the American Business Awards Tech Innovation of the Year Gold Stevie Award in 2013. He has served as an Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING.