This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2019.2915355, IEEE Transactions on Vehicular Technology

SmartStreamer: Preference-Aware Multipath Video Streaming Over MPTCP

Anis Elgabli and Vaneet Aggarwal

***Abstract*—Simultaneous access to multiple interfaces (e.g., WiFi and cellular networks) can significantly improve the users' quality of experience (QoE) in video streaming. However, some interfaces could be more expensive to use and less energy efficient. Therefore, in this paper, we propose a preference-aware multipath video streaming algorithm over HTTP using multipath TCP (MPTCP). First, we formulate the quality decisions of the video chunks and the chunk's download policy subject to the chunk's deadlines, the available bandwidth of the different paths, and the link preferences as a non-convex optimization problem. The objective is to optimize a novel QoE metric that maintains a tradeoff between maximizing the quality of every video's chunk and ensuring quality fairness among all chunks for the minimum re-buffering (stall) duration, and without violating the link preference constraint. Second, we develop a polynomial time complexity algorithm to solve the proposed optimization problem, and provide guarantees for the proposed algorithm. We further propose a sliding window based online algorithm where several challenges including short bandwidth prediction with prediction errors are addressed. Extensive emulated experiments with real bandwidth traces of public datasets reveal the robustness of our scheme and demonstrate its significant performance improvement compared to the state-of-the-art multi-path streaming algorithms.**

***Index Terms*—Video streaming, multi-path, multi-path TCP, stall duration, non convex optimization.**

I. Introduction

VIDEO streaming is one of the major sources of traffic in mobile networks. Currently, video content accounts for 50% of the cellular traffic and it is expected to account for around 75% of the mobile data traffic by the year of 2023 [1]. While its popularity is on the rise, its quality of experiences (QoE) is still often far from satisfactory. One solution to improve the user's QoE, especially in the mobile scenario, is to leverage multiple network interfaces of a single device and aggregate their available bandwidths in order to boost the quality of the video. It is common that today's client hosts are equipped with multiple network interfaces. For example, mobile devices (*e.g.,* Apple iOS 7 [2]) inherently support WiFi and cellular networks at the same time. To better describe a use case, assume a user is riding a train or bus in which a local WiFi is provided. Since, there could be many users sharing the WiFi network, a user could use its LTE link to help stream the video at a higher quality and to reduce stalls. However, in multi-path video streaming, some of the links are in general more preferable as compared to the others. For instance, the users may not wish to use too much of cellular link since it is, in many cases, more expensive (limited plans) and less energy efficient (far from the base station) as compared to the WifI link. Therefore, the scenario where some of the links are less preferable than others need to be considered. In this paper, we propose an efficient video streaming algorithms to improve users' QoE using multiple paths and with considering user's link preference.

An intuitive approach to enable multi-path is to replace the conventional transport (*i.e.,* TCP) with Multi-path TCP (MPTCP [3]), which is the de-facto multi-path solution allowing applications to transparently use multiple paths. Specifically, MPTCP opens multiple sub-flows (usually one over each path), distributes the data onto the sub-flows at the sender, and reassembles data from each path at the receiver. The key advantage of MPTCP is that it allows applications to use multiple paths without changing the existing socket programming interface. Thus, in this paper, we will consider an approach for fetching video chunks on multiple paths with the use of MPTCP. However, traditional MPTCP does not prefer one link over the other, which is the focus of this paper.

In order for a client to be able to stream different portions of the video at different quality levels according to the current situation of the network and/or buffer occupancy, the video itself needs to be divided into chunks and available at different quality levels. The predominant video encoding scheme of today is "Advanced Video Coding" (AVC, e.g MPEG4-AVC). In AVC, the video is divided into chunks, and each video's chunk is stored into L *independent* encoding versions. When fetching a chunk, the player's adaptation mechanism, Adaptive Bit Rate (ABR) streaming logic such as MPEG-DASH (Dynamic Adaptation Streaming over HTTP) [4], needs to select one out of the L versions based on its estimation of the network condition and the buffer capacity. In this paper, we propose a streaming algorithm which decides up to which quality should every chunk be fetched, and how much every link should be downloading for every chunk such that chunk's deadlines, and link preference constraints are respected. We note that different byte ranges of a

Manuscript received May 19, 2018; revised January 27, 2019; accepted May 3, 2019.

This work was supported by the U.S. National Science Foundation under Grants CCF-1527486 and CNS-1618335. The review of this paper was coordinated by Prof. Y. Cheng. *(Corresponding author: Vaneet Aggarwal.)*

A. Elgabli was with School of Electrical Computer Engineering, Purdue University, West Lafayette, IN 47907 USA. He is now with the Center of Wireless Communications, University of Oulu, Oulu 90014, Finland (e-mail: aelgabli@purdue.edu).

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

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

Digital Object Identifier 10.1109/TVT.2019.2915355

chunk can be fetched using different links. This flexibility helps to provide additional improvement on QoE compared to a single link streaming scenario.

In this paper, we formulate the preference-aware adaptive multipath streaming as an optimization problem for perfectly predicted bandwidths. The objective is to is to optimize a QoE metric that minimizes the video's stall duration as the first priority and maintains a tradeoff between maximizing the quality of every chunk and ensuring quality's fairness among all video's chunks, *i.e.,* minimizing quality switching rate, while respecting the link preference constraints. However, since in practice, the future bandwidth cannot be perfectly predicted, but can be estimated for a short time ahead using a bandwidth prediction technique such as crowd-sourced method [5], [6], or harmonic mean of the bandwidth achieved for the last few seconds [7], we propose an online sliding window based algorithm that solves the proposed optimization problem every α seconds to make a decision for the next W chunks. We evaluate our algorithms using real bandwidth traces of public datasets collected from commercial networks. The evaluation demonstrates that our approach is robust to prediction errors, and works well with a short prediction window. We compare our algorithms against a number of adaptive streaming strategies for multi-path streaming. The results demonstrate that our algorithm outperforms them in term of avoiding the stall duration and achieving a higher average playback quality without violating link preference constraint.

Our Contributions: The main contributions of the paper are as follows.

- We formulate the multi-path video streaming over HTTP and MPTCP with perfect bandwidth prediction as an optimization problem, whose objective is to optimize a novel QoE metric that maintains a tradeoff between maximizing the quality of every video's chunk and ensuring quality fairness among all chunks for the minimum re-buffering (stall) duration, and without violating the link preference constraint.
- We propose an online sliding window based algorithm for the practical scenario in which the bandwidth can only be predicted for sometime ahead with prediction errors. The online algorithm solves the proposed optimization problem every α seconds to make quality and downloading policy decisions for the next W chunks using the predicted bandwidth.
- The proposed problem is a non-convex discrete optimization problem. However, we develop an efficient algorithm that solves this specific problem in polynomial time. Guarantees for the proposed algorithm are provided.
- We evaluate our algorithms using real videos, and bandwidth traces from public datasets collected from commercial networks. The evaluation demonstrates that our approach is robust to prediction errors, and works well with a short prediction window, where we estimate the bandwidth using harmonic mean of the bandwidth values of the past few seconds. We compared our algorithms to a number of multipath adaptive streaming strategies. The results demonstrate that our algorithm outperforms them by improving the key QoE metrics such as the playback quality, the number of layer switches, lower preference link usage, and the stall duration.

The rest of the paper is organized as follows. Section II discusses the related work. Section III describes the problem formulation for multi-path streaming. Further, a polynomial run time algorithm is provided for this non convex problem in Section IV. The algorithm is extended to the case of short bandwidth prediction with prediction error (online algorithm) in Section IV-C. Section V presents the trace-driven evaluation results with a comparison to the different baselines. Section VI concludes the paper.

II. Related Work

Video streaming has received a lot of attention from both the academia and industry in the past decade. There are ABR and adaptive SVC adaptation algorithms. Some of the widely used ABR streaming techniques include MPEG-DASH [4], Apple's HLS [8], Microsoft's Smooth Streaming [9], and Adobe's HDS [10]. In recent studies, various approaches for making ABR streaming decisions have been investigated, for example, by using optimization theory [11]–[15], Markov Decision Process [16], machine learning [17], client buffer information [18], and data-driven techniques [19]–[21].

The knowledge of the future network conditions can play an important role in Internet video streaming. A prior study [22] investigated the performance gap between state-of-the-art streaming approaches and the approach with accurate bandwidth prediction. The results indicate that prediction brings additional performance boost for ABR streaming, and thus motivates our study. The bandwidth have been shown to be predictable for some time ahead using a crowd-sourced method to obtain historical data [5], [6], or harmonic mean of the past bandwidth [7].

Adaptive streaming strategies have been proposed for multi-path channels in [7] where a heuristic based on prediction, MSPlayer, was proposed for streaming video chunks using WiFi and LTE. In their proposed heuristic, alternate chunks are downloaded using WiFi and LTE connections respectively. However, MSPlayer does not assume video streaming over MPTCP in which orthogonal links appear as one link to the application layer. Moreover, MSPlayer does not provide any mechanism to prefer one link over the other. Recently, the authors of [23] proposed algorithms for using multi-path TCP video streaming where the primary objective is to reduce the usage of LTE as well as minimizing the video stall duration. However, their proposed approach does not explicitly consider that LTE link should be minimized when making quality decision of each chunk. In fact, the proposed scheme in [23] does not make quality decision of chunks. It uses a rate adaptation algorithm, like BBA [18] or Festive [24] to make the quality decisions based on the buffer state or the bandwidth prediction of the WiFi link, and then it uses MPTCP and tries to fetch the chunks at the decided quality with the minimum possible usage of the LTE link. Hence, this approach operates at two stages: quality decision and then chunk download. Therefore, the proposed scheme separates the

problem of finding the quality decision of chunks from the problem of LTE usage minimization. Thus, both objectives (quality decisions and LTE usage minimization) are not optimized jointly which leads to sub-optimal solution. In contrast, this paper proposes algorithm that consider preference of one link over the other explicitly in the optimization problem, optimize jointly over the two objectives, and use the approach of [23] as a comparison in the evaluations.

III. Problem Formulation

In this section, we describe our problem formulation considering two links (*e.g.,* WiFi and cellular networks) in the exposition, but the formulation and proposed algorithms can be easily extended to more links. We assume a video that is played with an initial start-up, (*i.e.,* buffering) delay s and there is a playback deadline for each of the chunks where chunk i needs to be downloaded by time $deadline(i)$. If a chunk is not received by its deadline, a stall will take a place. *i.e.,* the video will pause until the chunk is fully downloaded since the buffer is running empty. Our objective is to find the fetching policy, *i.e.,* what every link should download for every chunk, such that the stall duration is minimized as the first priority and average playback bitrate is maximized while maintaining minimum quality switching between chunks as the next priority. Moreover, link preference, and bandwidth constraints should not be violated. For now, we will assume a perfect bandwidth prediction for the whole period of the video and an infinite buffer size at the client (offline problem). We will relax these assumptions when we describe our online algorithm in Section IV-C.

Let's assume a video that is divided into C chunks (segments), where every chunk is of length L seconds is encoded at $(N+1)$ quality levels with rates $r_0, r_1, \ldots, r_N$. Let X_n be the size of a chunk when it is encoded at the n-th quality level, $X_n = L * r_n$. Let Y_n be the size difference between the n and $(n-1)$-th quality levels, so $Y_n = X_n - X_{n-1}, n \geq 1$. However, $Y_0 = X_0$. Moreover, let's assume that $Z_{n,i}$ is what can be fetched out of Y_n. $Z_{n,i} \in \{0, Y_n\}$. In words, Y_n is either totally fetched, so $Z_{n,i} = Y_n$ and the chunk is a candidate to the n-th quality level or it is totally skipped ($Z_{n,i} = 0$). We will call $Z_{n,i}$: the decision variable of the n-th quality level of chunk i. Since no chunk is totally skipped, we have $Z_{0,i} = Y_0$. every chunk i must be downloaded by time $s + (i-1)L$ in order to avoid any stalls. If a chunk cannot be downloaded by its deadline, a stall (*i.e.,* rebuffering) will occur and the total stall (re-buffering) duration from the start until the play-time of chunk i is $d(i)$. Therefore, the final deadline of chunk i is $deadline(i) = (i-1)L + s + d(i)$.

Let $X(i) = \sum_{n=0}^{N} Z_{n,i}$ be the final decision of the size of chunk i (quality decision of chunk i), and let $z_n^{(k)}(i,j)$ be the amount that needs to be downloaded from $Z_{n,i}$ over the link k (e.g., WiFi link) at time slot j. Moreover, let $Z_{n,i}^{(k)}$ be the total size that needs to be downloaded for the level n of chunk i by link k. *i.e.,* $Z_{n,i}^{(k)} = \sum_{j=1}^{(i-1)L+s+d(i)} z_n^{(k)}(i,j)$. We will refer to $Z_{n,i}^{(k)}$ by the decision variable of quality level n, chunk i, and link k. Let $B^{(k)}(j)$ be the available bandwidth over the link $k \in \{1,2\}$ at time j. Further, let B_m be the playback buffer size in time units (*i.e.,* the playback 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. Therefore, we ensure that there is enough space in the buffer before starting to download chunk i.

We assume that link 1 can be used as much as its bandwidth allows. However, we assume that link 2 can only help in fetching up to the quality level $n_2 \leq N$ if link 1 can't meet the deadline. For example, when $n_2 = 0$, link 2 can be used only to avoid stalls if link 1 fails to do so. We now formulate an optimization problem that optimizes the following metrics in their order *(i)* minimizes the stall duration, *(ii)* maximizes the average playback rate of the video, *(ii)* minimizes the use of link 2. However, the average playback rate maximization is constrained to prefer pushing more chunks to the n-th quality level over any other choice that pushes the quality of some chunks to levels $>n$ with the cost of dropping the quality of some other chunks to levels $<n$. This constraint on average playback maximization will minimize 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 stalls cause more quality-of-experience (QoE) degradation compared to playing back at a lower quality [11].

In order to account for these objectives and respect the priority order, *i.e.,* link 2 has to be used less, and it can't fetch beyond the level n_2, we maximize a weighted sum of $Z_{n,i}^{(k)}, \forall n, i, k$, where the decision variables of the lower levels and the more preferable links are given higher weights. We introduce weights λ_n^k where n is the quality level index and k is the link index. The weights need to satisfy the following conditions.

$$\lambda_a^{(2)} Y_a > C \cdot \left(\sum_{n=a+1}^{N} \lambda_n^{(1)} Y_n \right), n \leq n_2 \tag{1}$$

$$\lambda_a^{(1)} Y_a > C \cdot \left(\sum_{n=a+1}^{N} \lambda_n^{(1)} Y_n \right), \forall n \tag{2}$$

$$\lambda_n^{(1)} > \lambda_n^{(2)}, \forall n \tag{3}$$

Equation 1 implies that, for any quality level a, the levels higher than a using link 1 have lower utility than a chunk at level a fetched using link 2. Since link 1 has higher weight than link 2 (due to (3)), the use of λ helps in giving higher priority to fetch more chunks at the nth quality level over fetching some at higher quality at the cost of dropping the quality of other chunks to below the nth quality level. For $n > n_2$, link 2 is not used and thus (2) implies the same as (1) for higher layers. (3) will obey the priority order of the links, and it will not use a less preferable link to fetch a bytes that can be fetched by the more preferable link. Moreover, we give the highest weight μ to avoid stalls, so $\mu \gg \max(\lambda_n^k)$. The overall optimization problem can

be formulated as follows:

$$\textbf{Maximize}: \sum_{i=1}^{C}\sum_{n=0}^{N}\left(\lambda_n^{(1)}Z_{n,i}^{(1)}+\lambda_n^{(2)}Z_{n,i}^{(2)}\right) - \mu d(C) \quad (4)$$

subject to

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

$$Z_{0,i}^{(1)} + Z_{0,i}^{(2)} = Y_0\ \forall i \quad (6)$$

$$Z_{n,i}^{(1)} + Z_{n,i}^{(2)} = Z_{n,i}\ \forall i, n>0 \quad (7)$$

$$Z_{n,i} \le \frac{Y_n}{Y_{n-1}} Z_{n-1,i}\ \forall i,n \quad (8)$$

$$Z_{n,i}^{(2)} = 0\ \forall i,j,n>n_2 \quad (9)$$

$$\sum_{n=0}^{N}\sum_{i=i'}^{C} z_n^k(i,j) \le B^k(j)\ \forall j, k\in\{1,2\} \quad (10)$$

$$z_n^k(i,j) \ge 0\ \forall i,j,k\in\{1,2\} \quad (11)$$

$$z_n^k(i,j) = 0\ \forall i, j>deadline(i), k\in\{1,2\} \quad (12)$$

$$Z_{n,i}\in\{0,Y_{n,i}\}\ \forall i, n>0 \quad (13)$$

$$d(i)\ge 0, d(i+1)\ge d(i), deadline(i) = s+(i-1)L+d(i)\ \forall i \quad (14)$$

Variables: $z_n^k(i,j), Z_{n,i}^{(k)}, Z_{n,i}, d(i)\ \ \forall i=1,\dots,C,\ j=1,\dots,deadline(C),\ n=0,\dots,N$

Constraint (6) ensures that every chunk is fetched at least at the lowest quality, i.e., there are no skips. Constraint (7) ensures that for a chunk i, whatever is fetched over all times for any quality level higher than the lowest is equal to the decision variable of that quality level. Constraint (8) enforces a chunk to not be considered for the n-th quality level that if it is not a candidate to $(n-1)$-th level. Constraint (9) ensures that link 2 is not used to fetch beyond the quality level n_2. (10) imposes the available bandwidth constraint at each time slot j and link k. Constraint (11) imposes the non-negativity of chunk download. Constraint (12) enforces the deadline of a chunk, so a chunk can't be fetched after its deadline. Constraint (13) enforces the decision variables $Z_{n,i}$ to belong to the discrete set $\{0, Y_n\}$. Finally, Constraint (14) states that the stall duration before any chunk is non-negative and, it defines the deadline of chunk i.

The problem (4)–(14) is a non-convex since the feasible set (the set of the decision variables) is discrete. However, In next section, we propose a polynomial complexity algorithm and we show that the proposed algorithm achieves the optimal solution to the problem (4)–(14).

IV. SmartStreamer Algorithm

In this section, we describe the proposed algorithm. To develop the algorithm, we consider the offline algorithm, *i.e.,* the bandwidth is perfectly known for the whole period of the video. We first assume the case in which both links can be used without any preference. We refer to this algorithm as "Offline No-Pref SmartStreamer". Formally, we consider the conditions when $n_2 = N$, and the weights satisfy the condition defined by Eq. (1) and the following condition:

$$\lambda_n^{(1)} = \lambda_n^{(2)} = \lambda_n\ \forall n. \quad (15)$$

This algorithm is, then extended to the general case in which one of the link can be used only to help fetching up to the n-th quality level. We refer to this algorithm as "Offline Pref SmartStreamer". Guarantees are provided for both the algorithms, including the optimality for No-Pref version. Finally, we propose an online algorithm (Online Pref/No-Pref SmartStreamer) in which more practical assumptions are considered such as short bandwidth prediction with prediction error and finite buffer size.

A. Offline No-Pref SmartStreamer Algorithm

In No-Pref SmartStreamer, we assume that both links can be used equally likely without any preferences. The No-Pref SmartStreamer algorithm (Algorithm 1) initially calculates the cumulative bandwidth of every second j and link k ($R^{(k)}(j)$) (Line 5). The second step of the algorithm is to perform a forward scan that simulates fetching the chunk at the 0-th quality level. The forward scan determines the minimum stall time such that all chunks are fetched at least at the 0-th quality level since that is the first priority. Therefore, the objective of the forward scan of the 0-th quality level is to check if all chunks can be fetched at least at the 0-th quality level with the current startup delay. The maximum number of chunks that can be fetched at least at the 0-th quality level before the deadline of the ith chunk is: $V_{0,i} = \lfloor \frac{\sum_{k=1}^{K} R^k(deadline(i)}{r_0} \rfloor$(line 6). Hence, at the deadline of any chunk i, if $V_{0,i} < i$, the algorithm keeps incrementing the deadline of every chunk $\ge i$ by 1 until the condition $V_{0,i} = i$ is satisfied, then the algorithm resumes the forward scan (Line 10-13). At the end, the algorithm sets the final deadline of every chunk i to be: $deadline(i) = (i-1)L + s + d(C)$, and this deadline can't be violated. Therefore, the 0-th quality level forward scan achieves the minimum stall duration and brings all stalls to the very beginning since that comes before the deadline of all chunks. Hence, it offers bandwidth to more chunks and can help in increasing the quality of more chunks to the next quality levels.

In the third step, the algorithm performs backward scan that simulates fetching chunks in the reverse order and starts from the deadline of the last chunk (lines 17-31). The backward algorithm (Algorithm 2) simulates fetching every chunk i starting from its deadline and use all links with their remaining bandwidth after all chunks $i+1$ to C are fetched. For example if the deadline of the i-th chunk is the j-th time slot, the algorithm will use all the remaining bandwidth in every link at the j-th time slot to fetch the chunk. However, if that bandwidth is not enough to fully download the chunk, the algorithm moves to the $(j-1)$-th time slot and check if the remaining portion of the chunk can be fully downloaded in this time slot using the bandwidth of all links. The algorithm proceeds the same way with every time slot until

chunk i is fully downloaded, then it proceeds to the chunk $i-1$, start from its deadline and do the same way.

Higher Quality Levels modification: The algorithm proceeds by performing backward scan per level in order. The bandwidth is now modified to be the remaining bandwidth after excluding whatever has been reserved to fetch decision sizes of lower levels (Line 7 in Algorithm 2). The chunks that can't be fully downloaded at the n-th quality level during the backward scan are not considered as candidates to the n-th quality level. Also note that nth level for a chunk is not considered if it is not a candidate to the $n-1$th quality level (Line 19 in Algorithm 1). Finally, during the actual fetching of the chunks, each link fetches the the chunks/portions of chunks that are assigned to it in order.

Complexity Analysis: The algorithm sequentially decides each level. For each level n, the algorithm first finds $R^{(k)}(j)$, the cumulative bandwidth of every link k and time slot j, which is linear with respect to every link. Thus, it has a run-time complexity $O(deadline(C))$. Then, it performs forward scan that has a run-time complexity $O(deadline(C))$. Finally, a backward scan per chunk is performed. Since the complexity of backward algorithm is linear in C, the run-time complexity for a level is $O(C^2)$. Thus, the overall run-time complexity is $O(N \cdot C^2)$.

Optimality of No-Pref SmartStreamer: We will now prove that the proposed No-Pref SmartStreamer algorithm achieves the optimal solution for the link assignment and the quality decision for the problem specified by Equations (4)–(13) when $n_2 = N$, and λs satisfy (1) and (15). We note that given two strategies with the same number of chunks fetched at each quality level from 0 to $n-1$, among all remaining levels (layers n to N), the strategy that pushes more chunks to the $(n+1)$-th quality level achieves higher objective. This follows from the constraint for the weights, given in Equation (1) since the contribution to the objective of one chunk at lower quality level is higher than all the chunks at the higher quality levels.

We note that it is enough to show the optimality of the algorithm as compared to any in-order algorithm, where the chunks are fetched in order. This is because any feasible algorithm can be converted to an in-order algorithm achieving the same objective. The following result shows that the proposed algorithm minimizes the number of skips for a given quality level among all the other algorithms that fetch the same quality levels up to the immediate next lower level.

Lemma 1: Given size decisions up to $n-1$th level ($Z_{0,i}, \ldots, Z_{n-1,i}$, for all i), remaining bandwidth, and $deadline(i)$ for every chunk i, the proposed algorithm obtains the maximum number of chunks at level n (or achieves the minimum number of nth level skips) as compared to any feasible algorithm which has the same quality level decisions of every chunk up to the quality level $n-1$.

Proof: Proof is provided in Appendix A-A. ■

The algorithm starts from the last chunk, fetches chunks in reverse order, and skip chunks only if the remaining bandwidth is not enough to fetch a full chunk. Therefore, for any skip of the backward algorithm, there must be a skip for any feasible algorithm. Otherwise, the bandwidth constraint is violated. The next result demonstrates that the proposed algorithm reserves the maximum possible bandwidth before the deadline of every chunk that is a candidate to the next layer.

Algorithm 1: Offline No-Pref smartStreamer Algorithm.

1: **Input:** $\mathbf{Y} = \{Y_n \forall n\}$, L, s, C, $\mathbf{B}_k = \{B_k(j) \forall j\}$, $k = 1, 2$.
2: **Output:** $I_{n,i}^{(k)}$, $n = 0, \ldots, N$: the size that need to be fetched for the n-th layer of chunk i by link k.
3: $deadline(i) = (i-1)L + s, \forall i$, $d(i) = 0, \forall i$
4: **for** each layer $n = 0, \ldots, N$ **do**
5: $R^{(k)}(j) = \sum_{j'=1}^{j} B_k(j')$, $j = 1, \ldots .deadline(C)$, $k = 1, 2$
6: $V_{n,i} = \lfloor \frac{\sum_{k=1}^{K} R^k(deadline(i)}{r_n} \rfloor, \forall i$
7: **if** $n = 0$ **then**
8: **for** $i = 1 : C$ **do**
9: $d(i) = d(i-1)$, $deadline(i) = (i-1)L + s + d(i)$
10: **while** $(V_{n,i} < i)$ **do**
11: $d(i) = d(i) + 1$, $deadline(i) = (i-1)L + s + d(i)$
12: $V_{n,i} = \lfloor \frac{\sum_{k=1}^{K} R^k(deadline(i)}{r_n} \rfloor$
13: **end while**
14: **end for**
15: $deadline(i) = (i-1)L + s + d(C), \forall i$
16: **end if**
17: **for** $i = C : -1 : 1$ **do**
18: $j = deadline(i)$, $R = R^{(1)}(j) + R^{(2)}(j)$
19: **if** ($n = 0$ or $I_{n-1,i}^{(1)} > 0$ or $I_{n-1,i}^{(2)} > 0$) **then**
20: **if** $R \geq Y_n$ **then**
21: $\mathbf{BB}_1 = \mathbf{B}_1$, $\mathbf{BB}_2 = \mathbf{B}_2$, $RR^{(1)} = R^{(1)}$, $RR^{(2)} = R^{(2)}$
22: $[r^{(k)}, \mathbf{B}_k, k \in \{1, 2\}] =$
23: Backward$(i, j, RR^{(k)}, \mathbf{BB}_k, Y_n, j, k \in \{1, 2\})$
24: $I_{n,i}^{(k)} = R^{(k)}(deadline(i)) - RR^{(k)}(deadline(i))$, $k \in \{1, 2\}$
25: $\mathbf{B}_1 = \mathbf{BB}_1$, $R^{(1)} = RR^{(1)}$, $R^{(2)} = RR^{(2)}$, $\mathbf{B}_2 = \mathbf{BB}_2$, $Y_n = 0$
26: **else**
27: skip the n-th layer of all the remaining chunks
28: Break
29: **end if**
30: **end if**
31: **end for**
32: **end for**

Lemma 2: Among all algorithms with the same number of nth level skips, the proposed algorithm reserves the largest possible bandwidth for considering the $n+1$th level of every candidate chunk. In other words, the proposed algorithm maximizes the resources to the higher levels among all algorithms that have same decisions up to the current quality level.

Proof: Proof is provided in Appendix A-B. ■

Intuitively, the proposed algorithm brings all the n-th level skips as early as possible. Therefore, since the earlier chunks are closer to their deadlines, not considering them for the next quality level offers more bandwidth to the later ones. Hence,

Algorithm 2: Backward Algorithm.

1: **Input:** $i, j, B_1, B_2, R^{(1)}, R^{(2)}, Y_n$
2: **Output:** r is the residual cumulative bandwidth at different times after fetching chunk i, B is the residual bandwidth after accounting for fetching of chunk i.
3: **Initialization:**
4: $j_1 = j_2 = j$
5: **while** $(Y_n > 0)$ **do**
6: $fetched1 = min(B_1(j_1), Y_n)$, $Y_n(i) = Y_n(i) - fetched1$
7: $B_1(j_1) = B_1(j_1) - fetched1$, $R^{(1)}(j_1) = R^{(1)}(j_1) - fetched1$
8: **if** $(B(j_1) = 0)$ **then** $j_1 = j_1 - 1$
9: $fetched2 = min(B_2(j_2), Y_n)$, $Y_n(i) = Y_n(i) - fetched2$
10: $B_2(j_2) = B_2(j_2) - fetched2$, $R^{(2)}(j_2) = R^{(2)}(j_2) - fetched2$
11: **if** $(B(j_2) = 0)$ **then** $j_2 = j_2 - 1$
12: **end while**

skipping earlier chunks in the n-th quality level minimizes the number of $(n+1)$-th level skips, thus offering higher number of candidate chunks to the $(n+1)$-th quality level than any other feasible algorithm. Using Lemma 1 and Lemma 2, the following theorem shows the optimality of No-Pref SmartStream algorithm in solving problem ((4)–(14)).

Theorem 1: Up to a given quality level $M, M \geq 0$, if $Z^*_{n,i}$ is the decided size of the nth level decision variable of chunk i $(m \leq M)$ that is found by running No-Pref SmartStreamer algorithm, and $Z'_{n,i}$ is the decided size of the nth level decision variable of chunk i that is found by any other feasible algorithm such that all constraints are satisfied, then the following holds when $n_2 = N$ and λ's satisfy (1), (2) and (15).

$$\sum_{n=0}^{M} \lambda_n \sum_{i=1}^{C} Z'_{n,i} \leq \sum_{n=0}^{M} \lambda_n \sum_{i=1}^{C} Z^*_{n,i}. \tag{16}$$

In other words, No-Pref SmartStreamer achieves the optimal solution of the optimization problem (4–14) when $n_2 = N$ and λ's satisfy (1), (2) and (15).

Proof: Proof is provided in Appendix A-C. ■

B. Offline Pref SmartStreamer

In this section, we consider a preference to the use of link 1 as compared to link 2, *i.e.,* $n_2 < N$. This will allow using link 2 to help downloading up to the quality level n_2 while not over-using it to achieve higher quality. Further, among different schemes that obtain same number of chunks at quality levels $\leq n_2$, we wish to minimize the usage of link 2. For $n_2 = 0$, this implies that link 2 is only used to avoid stalls. Further, link 2 is used only if necessary.

The Pref SmartStreamer is described in Algorithm 3. In the first step, the algorithm calls the No-Pref SmartStreamer algorithm for quality levels $n = 0$ to n_2 (line 4). However, the outcome of this call is not final since the preference has not been

Algorithm 3: Offline Pref SmartStreamer Algorithm.

1: **Input:** $\mathbf{Y} = \{Y_n \forall n\}$, L, s, C, U, $\mathbf{B}_k = \{B_k(j) \forall j\}$, $k = 1, 2$.
2: **Output:** $I^{(k)}_{n,i}$, $n = 0, \ldots, N$: the size that need to be fetched for the n-th level decision variable of chunk i by link k.
3: $n_2 =$ max layer index that both links can fetch
4: $[I^{(k)}_{n,i} \forall i, R^{(k)}, \mathbf{B}_k, k \in \{1, 2\}]$ =No-Pref SmartStreamer($\mathbf{Y} = \{Y_n \forall n = 0, \ldots, n_2\}$, L, s, C, $\mathbf{B}_k$)
5: $[I2^{(k)}_{n,i} \forall i, R^{(1)}, \mathbf{B}_1]$ =No-Pref SmartStreamer($I^{(2)}_{n,i} \forall i$, L, s, C, $\mathbf{B}_1$, $\mathbf{0}$)
6: $I^{(1)}_{n,i} = I^{(1)}_{n,i} + I2^{(1)}_{n,i}$, $I^{(2)}_{n,i} = I^{(2)}_{n,i} - I2^{(1)}_{n,i}, \forall i$,
7: $[I^{(k)}_{n,i} \forall i, R^{(k)}, \mathbf{B}_k, k \in \{1, 2\}]$ =No-Pref SmartStreamer($\mathbf{Y} = \{Y_n \forall n = n_2, \ldots, N\}$, L, s, C, $\mathbf{B}_1, \mathbf{0}$)

taken into consideration. Then, the algorithm minimizes the usage of Link 2 such that the decisions up to level n_2 remain the same. In order to do that, the algorithm makes a second call to No-Pref SmartStreamer, but this call considers only link 1 (Bandwidth of link 2 is 0) and chunks or portion of chunks that were initially decided to be fetched by link 2 (line 5). The algorithm changes the assignment of the chunks that were decided to be moved from link 2 to link 1 in the second call of No-Pref SmartStreamer (line 6). Once decisions of quality levels 0 to n_2 are found, the remainder of the algorithm is similar to No-Pref algorithm, but the algorithm uses only link 1 (line 7). In other words $B(j) = B^1(j)$. This is because Link 2 does not help in fetching chunks at qualities that are higher than the n_2 quality level.

Guarantees for Pref SmartStreamer: We now provide some guarantees for the Pref SmartStreamer algorithm, when $n_2 < N$ and λ's satisfy (1)–(3).

Lemma 3: Offline Pref SmartStreamer Algorithm achieves the maximum number of chunks fetched at the quality level n for every $n \leq n_2$ given the decisions of quality levels $< n$.

Proof: Proof is provided in Appendix B-A. ■

We note that among two strategies with the same number of chunks fetched up to n_2 quality level, the proposed problem prefers obtaining content on link 1 as compared to link 2, due to (3). The following result shows that the proposed algorithm minimizes link 2 usage among all algorithms with the same quality decisions.

Lemma 4: Offline Pref SmartStreamer Algorithm has the minimum data usage of link 2 among all algorithms with the same quality decisions. In other words, no other algorithm can achieve less data usage for link 2 with the same number of chunks fetched at every quality level $n \leq n_2$.

Proof: Proof is provided in Appendix B-B. ■

We note that the proposed algorithm for the first n_2 levels achieves the same decision as No-Pref algorithm, while minimizes link 2 usage. After these levels, only link 1 is used. The overall optimality of the problem is tricky since slightly more link 2 can be used instead of link 1 to free up enough bandwidth to get an entire chunk at a higher level $> n_2$ using link 1. Thus,

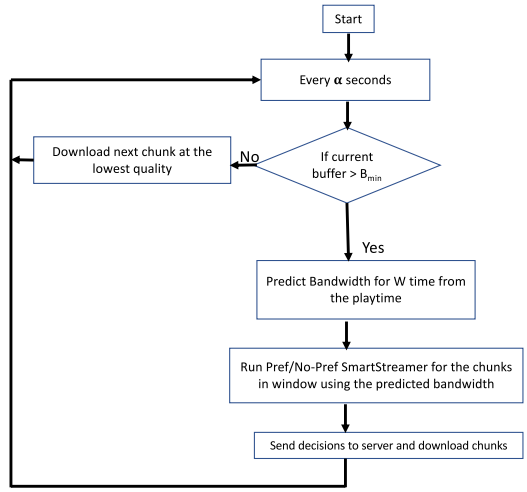

Fig. 1. Illustration of the online Pref/No-Pref SmartStreamer Algorithm.

the gain is an additional higher chunk using link 1, while the loss in the objective is using a δ amount from link 2 instead of link 1 at lower layer ($\leq n_2$). Even though the overall algorithm is not optimal, the results in Lemmas 3 and 4 show interesting properties of the proposed algorithm.

C. Online No-Pref/Pref SmartStreamer: Dealing With Short and Inaccurate BW Prediction

For the algorithms described in Sections IV-A and IV-B, we assumed a perfect bandwidth prediction, and unlimited client buffer capacity. However, practically, the prediction can be performed for a few chunks ahead, and the prediction will not be perfect. Moreover, the client buffer might be limited. In this section, we describe the online smartStreamer algorithm that obtains prediction for a window of size W chunks ahead and makes decisions based on the prediction. Note that the buffer capacity at the user may be limited, and in that case we consider that W is no more than the buffer capacity since the chunks beyond that can't be downloaded. The way we capture the buffer capacity is by assuming that at every re-run of the algorithm (every α seconds), only chunks that are B_{max} ahead of the chunk that is currently being played can be fetched.

There are multiple ways to obtain the prediction, including a crowd-sourced method to obtain historical data [5], [6]. Another approach may be to use a function of the past data rates obtained as a predictor for the future, an example is to compute the harmonic mean of the past β seconds to predict the future bandwidth [7]. The decisions are re-computed for the chunks that have not yet reached their deadlines periodically every α seconds.

As shown in Fig. 1, Online smartStreamer runs every α seconds. At the first step, the algorithm checks if the current buffer occupancy is lower than a pre-decided threshold B_{min}, and if so, the next chunk is requested at the lowest quality level. If the buffer occupancy is higher than B_{min}, the algorithm predicts the bandwidth for next W time using any of the aforementioned schemes. With the predicted bandwidth, the No-Pref(Pref) SmartStreamer algorithm described in Section IV-A (IV-B) is used to find the quality and fetching policy decisions of the chunks that are in the window formed by W chunks from the playtime which have yet not been downloaded. Finally, the decision is sent to the server, and the chunks are fetched according to the algorithm decision. If all W chunks are fetched before the next re-computation time, the current time is set as the re-computation point, and the fetching policy for the next W chunks from the one that is currently being played is computed. Finally, at the start of the download, all links are assigned chunks to download at the lowest quality level since there is no bandwidth prediction available yet.

TABLE I
AVC ENCODING CHUNK SIZES IN MB OF THE ENVIVIO VIDEO USED IN OUR EVALUATION

Quality level	level-0	level-1	level-2	level-3	level-4
Min	0.0433	0.0786	0.1265	0.2213	0.3205
Mean	0.1693	0.2916	0.4795	0.949	1.403
Max	0.2342	0.3855	0.6217	1.286	1.918

V. EVALUATION

A. Setup

To evaluate the performance of the proposed algorithms, we built an emulation testbed using a client laptop and a server machine, both running in a Linux machine with kernel version 2.6.32, 24 CPU cores, and 32MB memory. We emulated MPTCP by two TCP connections between the client and the server over two different TCP ports. The client is equipped with a built-in WiFi interface. We use a dedicated 802.11n WiFi access point running on the 5 GHz frequency band with RTT <1ms and bandwidth >50 Mbps. To introduce bandwidth variations into the emulated experiments, we adopt `Dummynet` [25] running on the client side, thus the incoming rates of the TCP packets from the server on the both links (i.e., 1 or 2) will be throttled according to the bandwidth datasets used. The bandwidths profile of the datasets for both the links are described later in this section.

Video parameters: We used the parameters and chunks of Envivio video from DASH- 264 JavaScript reference client test page which is 260s long, consisting of 65 4s chunks ($L = 4s$). The video is encoded by H.264/MPEG-4 AVC codec into 5 different quality levels, and the nominal bitrate of the levels are as follows: $R = \{$0.338Mbps, 0.583Mbps, 0.959Mbps, 1.898Mbps, 2.806Mbps$\}$. The actual encoding rates of the chunks at any of the 5 different levels are different from the nominal rates since the video is encoded in variable bit rate (VBR). The min, mean, and max chunk size of every quality level in Mega Bytes (MB) are shown in table I (chunk size of level i is equal to $L * R(i)$). The table clearly shows the variability of the chunk sizes of every level. Finally, to emulate the MPTCP, since in the original video files, for every chunk there are 5 files each file contains the same chunk but encoded into one of the 5 different encoding rates, we generated 5 different synthetic files with the same exact sizes of the 5 original files. Then, we further split every file into smaller files each of size that is about 1KB. Therefore, we make a decision of the quality of the chunk and also a decision of what link to use in order to fetch every 1KB of the chunk.

Bandwidth Traces: For bandwidth traces, we used two public datasets, representing the two paths. The first dataset (denoted Dataset1) consists of continuous 1-second measurement of throughput of a moving device in Telenor's mobile network in Norway [26]. The second dataset (denoted Dataset2), is the FCC dataset [27], which consists of more than 1 million sets of throughput measurements. Both these datasets have been post-processed in [28] to give 1000 traces, each of 6-minute length which will be used in this paper for evaluations. Dataset1 has higher average bandwidth than Dataset2, but it also has higher variance. We use Dataset1 as traces for the first link while Dataset2 as traces for the second link.

Algorithm Parameters: For the online version of the proposed algorithm (No-Pref SmartStreaming and Pref SmartStreaming, described in Sections IV-C and IV-B), we set our algorithm's parameters as follows. We choose $W = 10$ chunks, $B_{min} =$ 4 seconds, and $B_{max} = 2$ minutes (60 chunks). We tried different buffer sizes, and the comparisons between different algorithms were qualitatively the same. Moreover, we choose $\alpha =$ 2 seconds. We use the harmonic mean of the last 10 seconds ($\beta = 10$ seconds) to predict the future bandwidth. In other words, the predicted bandwidth for the entire window is set to the value of the harmonic mean of the last 10 seconds (or less in the start, when less data is available). Since there is no prediction at the beginning, the first two chunks are fetched at the 0-th quality level where the first chunk is fetched over the first link, and the second one is fetched over the second link. For the link 1 preference case, we assume $n_2 = 1$. Thus, link 2 is only used to help pushing chunk's qualities to the 1-st quality level. The proposed online algorithms without and with link 1 preference (No-Pref SmartStreaming and Pref SmartStreaming) are compared with the following baseline algorithms.

MP-DASH-BBA: The authors of [23] proposed Multi-path Dash (MP-DASH) which is an Adaptive MPTCP Video Streaming Over Preference-Aware multi-path that takes the chunk quality decision based on BBA algorithm [18]. We use the 30 and 90 seconds as the lower, and upper thresholds on the buffer length for BBA algorithm. For the No-Pref scenario, we do not impose any constraint on link 2 usage.

MP-DASH-Festive: The authors of [23] also considered a Preference-Aware algorithm based on Festive algorithm [24]. We use the default setting of Festive algorithm as described in [24] over the two links combined using multi-path TCP. Moreover, we do not impose any constraints on Link 2 for the No-Pref scenario.

MSPlayer [7]*:* MSPlayer takes quality decisions of the next two chunks based on the bandwidth prediction. Odd chunks are fetched on the first link, while even chunks are fetched on the second link. If the current bandwidth measurement of the slow link is larger than $(1 + \delta)$ times the estimated value, the chunk size is doubled. Similarly, if the current value is less than $(1 - \delta)$ times the estimated value, the chunk size is halved. The size of the chunk which is to be downloaded using the fast link is adjusted based on the throughput ratio. We set δ to its default value (5%) [7]. We compare the proposed algorithms with MSPlayer only in case of No-Pref since there is no known MSPlayer version when one link is more preferable.

off-SmartStreamer and off-Pref-SmartStreamer Algorithms: These are the offline No-Pref SmartStreamer and off- Pref-SmartStreamer as described in Sections IV-A and IV-B where a genie-aided perfect bandwidth prediction is known for the entire video duration. The offline algorithm's results are generated using simulation rather than the real implementation.

B. No-Pref SmartStreaming Algorithm

In this subsection, we will evaluate No-Pref SmartStreaming with a comparison to the baseline approaches described in Section V-A. The results are shown in Fig. 2. The startup delay is chosen to be 5 seconds. off-SmartStreaming represents the offline fetching policy.

Fig. 2-a shows the probability mass function of the number of chunks fetched at the different qualities (0=0-th quality level). The average playback rate among all traces, and the total stall duration among all bandwidth traces is displayed on the top of Fig. 2-a. We first see that SmartStreamer significantly outperforms the baseline algorithms. For instance, about 50%, and 40% of the chunks are fetched at the 4-th quality level in MP-DASH-BBA, and MSPlayer respectively. In contrast, SmartStreaming fetches about 85% of the chunks at the 4-th quality level. Further, MSPlayer runs into longest stall duration (213 seconds, 3.5 minutes of stall duration). On the other hand, MP-DASH-BBA and SmartStreamer achieve the shortest stall duration as compared to the other algorithms (only 8 seconds of stall duration). Thus, SmartStreamer achieves the highest playback rate and at the same time it maintains a stall duration that is as low as the one achieved by the very conservative algorithm. Moreover, SmartStreamer outperforms MP-DASH-Festive both in terms of shorter stall duration and achieving higher average playback rate. SmartStreamer, incorporates bandwidth prediction and the deadlines of the chunks into its optimization based decisions, prioritizes the later chunks, and re-considers the decisions every 2 seconds. These properties help SmartStreamer be adaptive to different bandwidth regimes and variations in the bandwidth profiles.

Fig. 2-b shows the CDF of the average playback rate per bandwidth trace. We note that SmartStreamer achieves the highest average playback rate in every single bandwidth trace as compared to the other baseline algorithms. Fig. 2-c shows the distribution of the quality switching rates (QSR), which is defined as:

$$\frac{1}{M}\sum_{i=2}^{C}|E\{X(i)\} - EX\{(i-1)\}|$$

where C is number of chunks and $E\{X(i)\}$ is the expected playback rate of i-th chunk, so if the i-th chunk is fetched at n-th quality level, $E\{X(i)\}$ will be equal to the nominal cumulative rate of the n-th quality level. Intuitively, QSR quantifies the frequency of the quality change across the chunks and should be low for better quality of experience (QoE). MSPlayer performs poorly in terms of switching rate as compared to the other algorithms since it does not consider the buffer length and doubles or halves the quality based on the predicted bandwidth. We observe that SmartStreamer has slightly higher quality switching rate as compared to MP-DASH-Festive and MP-DASH-BBA. However, our objective as described in Section IV gives higher

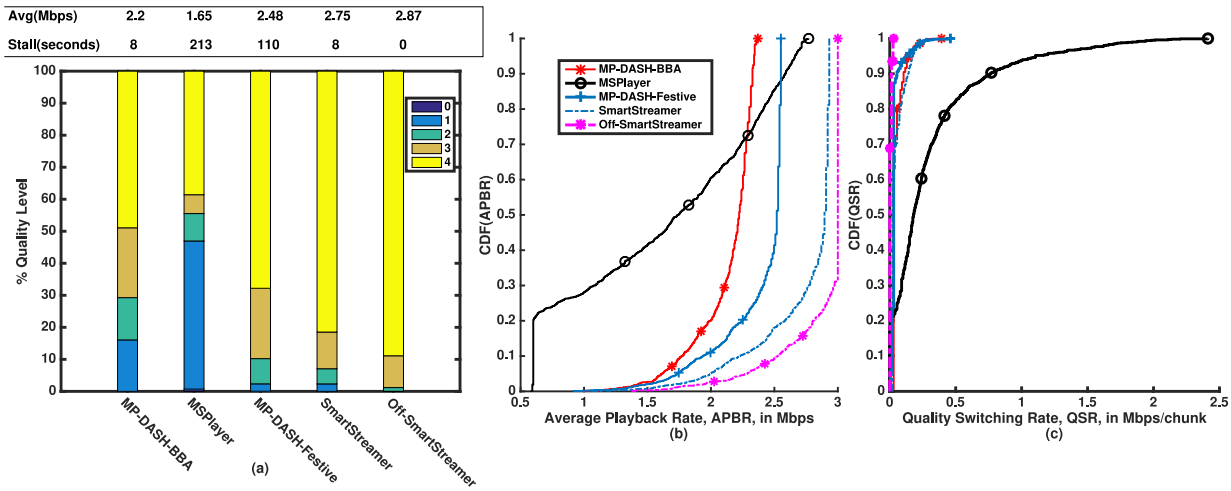

Fig. 2. Streaming without link preference: (a) Quality Level distribution, (b) CDF of Average Playback Rate, and (c) CDF of Quality switching rate.

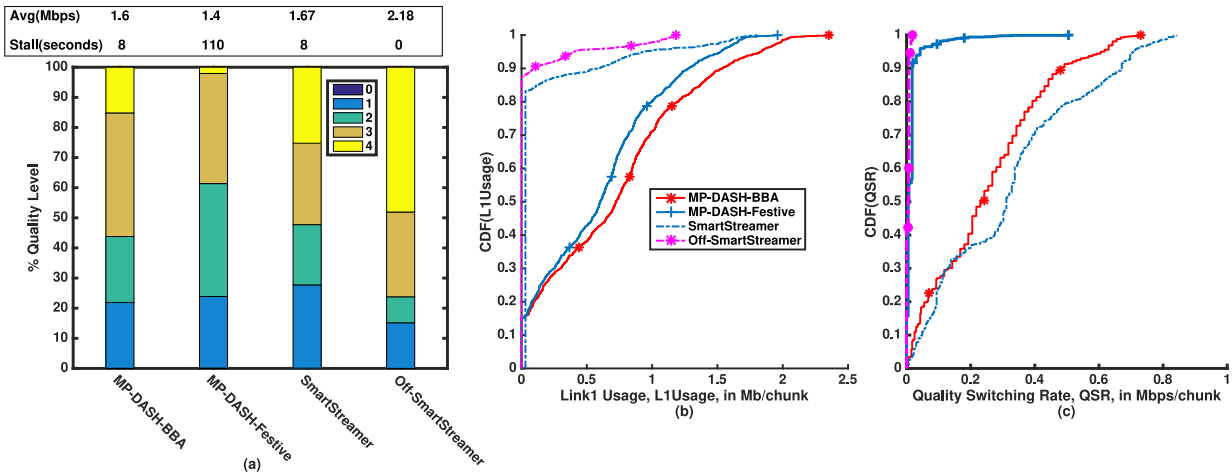

Fig. 3. Streaming with Link Preference: (a) Quality Level distribution, (b) CDF of link 2 usage, and (c) CDF of Quality switching rate.

priority to avoid stalls and to increase quality. Therefore, achieving a less stall duration as compared to MP-DASH-Festive and a higher playback average as compared to MP-DASH-BBA come at the cost of higher quality switching rate.

Finally, we compare the online SmartStreamer with the offline version of it. We note from Fig. 2 that although SmartStreamer uses prediction window of only 10 chunks (short window), and harmonic mean of the last $\beta = 10$ seconds for predicting future bandwidth, it achieves a fetching policy that is very close to the one achieved by the offline algorithm (perfect prediction for entire video duration) with slightly higher quality switching rate and a drop of the quality of a few chunks. The slight increase in the quality switching rate for online schemes is a result of the short prediction and the prediction error. Prioritizing later chunks, re-considering the decisions every 2 seconds, and adjusting to prediction error, all help reducing the gap between the online and the offline algorithm.

C. *Pref SmartStreaming*

In this subsection, we will evaluate the proposed algorithms, Pref SmartStreamer, and compare it to the link preference based baseline approaches. The results are shown in Fig. 3. The startup delay is chosen to be 5 seconds. The algorithms off-Pref SmartStreamer represents the fetching policy when the bandwidth profiles are known non-causally.

Fig. 3-a shows the probability mass function of the number of chunks fetched at the different qualities. We see that Pref SmartStreamer achieves about 20% higher average playback rate than MP-DASH-Festive with shorter stall duration, and about 5% higher average playback rate than MP-DASH-BBA with the same stall duration.

However, when one of the link is preferred, not only is the average playback rate is important, but also is how much of the bandwidth of the less preferable link is used. Thus, in Fig. 3-b, we plot the CDF of the less preferable link usage (link 2, L2Usage). We see that Pref SmartStreamer uses much lower amount of link 2 bandwidth as compared to MP-DASH-BBA and MP-DASH-Festive. In about 80% of the bandwidth traces, Pref SmartStreamer used link 2 to fetch only one chunk, and that chunk is necessary at the beginning to predict the available bandwidth of link 2 (the offline algorithms did not use link 1 for 80% of the traces). However, both MPTCP-BBA and MPTCP-Festive used link 2 to fetch more than 1Mb/chunk

for 50% of the bandwidth traces. Thus, the proposed algorithms give comparable or better average qualities than the baselines with significantly lower utilization of link 2.

VI. Conclusion and Future Work

This paper provides a Preference-Aware adaptive streaming algorithm over MPTCP. The problem of optimizing the user's quality of experience (QoE) subject to the available bandwidth, chunks deadlines and link preferences is formulated as a non-convex discrete optimization problem. A low-complexity algorithm is provided for the problem, and some guarantees for the proposed algorithm are provided. Further, an online algorithm is proposed where several challenges including bandwidth prediction errors are addressed. Extensive emulations with real traces of public dataset reveal the robustness of our schemes and demonstrate their significant performance improvement compared to other state-of-the-art multi-path algorithms.

We note that this work assumed use of MPTCP, while the approach could be extended when MPTCP is not a possible approach [29].

References

[1] "Ericsson mobility report on the pulse of the networked society," Jun. 2016. [Online]. Available: https://www.abc.es/gestordocumental/uploads/internacional/EMR_June_2016_D5%201.pdf. Accessed: Jul. 14, 2017.

[2] "iOS: Multipath TCP support in iOS 7," 2017, [Online]. Available: https://support.apple.com/en-us/HT201373

[3] "Multipath TCP in the linux kernel," 2017. Accessed: Jul. 14, 2017. [Online]. Available: http://www.multipath-tcp.org

[4] "MPEG-DASH," 2017. Accessed: Jul. 14, 2017. [Online]. Available: http://goo.gl/QxtpZ9

[5] 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, 2012, Art. no. 24.

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

[7] Y.-C. Chen, D. Towsley, and R. Khalili, "Msplayer: Multi-source and multi-path video streaming," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2198–2206, Aug. 2016.

[8] "Apple HTTP live streaming," 2017. Accessed: Jul. 14, 2017. [Online]. Available: https://goo.gl/6yYWg

[9] "Microsoft smooth streaming," 2017. Accessed: Jul. 14, 2017. [Online]. Available: http://goo.gl/TQHWL

[10] "Adobe HTTP dynamic streaming," 2017. Accessed: Jul. 14, 2017. [Online]. Available: http://goo.gl/IZWE8d

[11] 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.

[12] 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.

[13] A. Elgabli, V. Aggarwal, S. Hao, F. Qian, and S. Sen, "LBP: Robust rate adaptation algorithm for SVC video streaming," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1633–1645, Aug. 2018.

[14] A. Elgabli and V. Aggarwal, "FastScan: Robust low-complexity rate adaptation algorithm for video streamingover HTTP," *IEEE Trans. Circuits Syst. Video Technol.*, to be published, doi: 10.1109/TCSVT.2019.2914388.

[15] A. Elgabli and V. Aggarwal, "Deadline and buffer constrained knapsack problem," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 5, pp. 1564–1568, May 2019.

[16] D. Jarnikov and T. Özçelebi, "Client intelligence for adaptive streaming solutions," *Signal Process., Image Commun.*, vol. 26, no. 7, pp. 378–389, 2011.

[17] M. Claeys, S. Latré, J. Famaey, and F. D. 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.

[18] 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.

[19] X. Liu *et al.*, "A case for a coordinated internet video control plane," in *Proc. ACM SIGCOMM*, 2012, pp. 359–370.

[20] A. Ganjam *et al.*, "C3: Internet-scale control plane for video quality optimization," in *Proc. Conf. Netw. Syst. Des. Implementation*, 2015, pp. 131–144.

[21] Y. Sun *et al.*, "CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proc. ACM SIGCOMM*, 2016, pp. 272–285.

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

[23] B. Han, F. Qian, L. Ji, V. Gopalakrishnan, and N. Bedminster, "Mp-dash: Adaptive video streaming over preference-aware multipath," in *Proc. 12th Int. Conf. Emerg. Netw. Exp. Technol.*, 2016, pp. 129–143.

[24] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol.*, 2012, pp. 97–108.

[25] "The dummynet project," 2017. Accessed: Jul. 14, 2017. [Online]. Available: http://info.iet.unipi.it/ luigi/dummynet/

[26] 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.

[27] "FCC," 2017. Accessed: Jul. 14, 2017. [Online]. Available: https://www.fcc.gov/general/measuring-broadband-america

[28] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 325–338, 2015.

[29] A. Elgabli, K. Liu, and V. Aggarwal, "Optimized preference-aware multipath video streaming with scalable video coding," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2018.2889039.

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

Vaneet Aggarwal (S'08–M'11–SM'15) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 2005, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 2007 and 2010, respectively. From 2010 to 2014, he was a Senior Member of the Technical Staff Research with AT&T Labs Research, Bedminster, NJ, USA. He is currently an Associate Professor with Purdue University, West Lafayette, IN, USA. His current research interests are in communications and networking, video streaming, cloud computing, and machine learning.

He is currently an Associate Professor with Purdue University, West Lafayette, IN, USA. He was an Adjunct Assistant Professor at Columbia University, NY (2013–2014) and a VAJRA Adjunct Professor at IISc Bangalore (2018–2019). He was the recipient of the Princeton University's Porter Ogden Jacobus Honorific Fellowship in 2009, 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.