Federated Adaptive Bitrate Live Streaming over Locality Sensitive Playback Coalitions

Abhijit Mondal
abhimondal@iitkgp.ac.in
Department of CSE, IIT Kharagpur
Kharagpur, WB, India

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in
Department of CSE, IIT Kharagpur
Kharagpur, WB, India

ABSTRACT

Live video broadcasts to particular communities or targeted audiences many-at-a-times indulge a cluster of localities from where the end-users are interested in participating. In this paper, we leverage this idea to develop a system called *Federated Live Streaming over DASH* (FLiDASH) which forms end-users coalitions based on the locality of the network connectivity (like a set of end-users behind a common core-network service gateway). In FLiDASH, the coalition members stream the live data collectively based on a federated adaptive bitrate streaming mechanism where the coalition as a whole decides the optimal bitrate for video streaming and distributes the download overhead among its members. We have thoroughly evaluated FLiDASH in an emulated setup, and observe a 40% improvement in the live streaming QoE with a 20% reduction in the network traffic usage.

CCS CONCEPTS

• **Networks** → **Network design principles**; *Peer-to-peer networks*; • **Information systems** → **Multimedia streaming**.

KEYWORDS

Live video, collaborative streaming, adaptive video quality

ACM Reference Format:
Abhijit Mondal and Sandip Chakraborty. 2020. Federated Adaptive Bitrate Live Streaming over Locality Sensitive Playback Coalitions. In *Workshop on Network Application Integration/CoDesign (NAI'20),August 14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3405672.3405806

1 INTRODUCTION

During the last decade, social video streaming for targeted audiences have seen a huge boom with applications like Twitch.tv, Periscope, Meerkat along with the traditional YouTube & Facebook live and similar other personalized live streaming services [19]. Live broadcasts over such platforms have increased many-folds during the recent COVID-19 pandemic due to over-the-top (OTT) services like online live broadcast of classroom lectures to the students[1]. Many existing studies indicate that live streaming of popular events, such as a live cricket or football match, creates multiple traffic bottlenecks in the network, particularly at the Internet gateways of private organizational networks or Internet Service Providers (ISP) [21]. Consequently, a question arises – how can we prevent traffic bottlenecks in the Internet while allowing high definition video streaming to millions of users?

In this paper, we consider a class of live but non-interactive streaming applications, where the video is broadcast to a set of targeted audiences over social streaming applications. Social streaming applications many-a-times form communities which are localized, forming one or more geographical clusters [19]. We utilize this localized community formations among live streaming viewers to construct one or more playback coalitions, as shown in Figure 1. The coalition members share a common network gateway (such as an organizational local network gateway or the service gateway for a cellular core network) to connect to the Internet, however, there are direct high-speed local connections among the coalition members (like LAN connections or cellular device-to-device connections). It can be noted that such a coalition can be formed based on the principles of *Application-Layer Traffic Optimization* (ALTO), where an ALTO server can provide the locality information of video players without requiring any explicit network or device firmware change. The coalition members collectively download the video from the content provider based on an adaptive bitrate streaming (ABR) strategy, such as dynamic adaptive streaming over HTTP (DASH). The clients in a coalition collectively decide the adaptive playback rate and share data-download loads among themselves maintaining the playback synchronization.

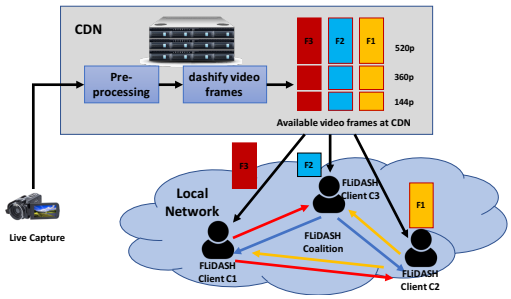


Figure 1: Overview of FLiDASH: The clients under a local network create a coalition, every members of the coalition share the total download load.

[1]https://www.nokia.com/blog/network-traffic-insights-time-covid-19-march-23-29-update/ (Accessed: March 25, 2021)

NAI'20, August 14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-8044-7/20/08…$15.00
https://doi.org/10.1145/3405672.3405806

Challenges: However, developing such a system has multiple challenges. First, the coalition needs to be designed in a way such that downloading data directly from the content provider is costlier than sharing the data over the local network. Second, there should be a proper distribution of segment-wise data-download scheduling among the coalition members such that playback synchronization is not violated. A proper playback synchronization ensures that every player in the coalition should acquire the video segment s_i, either downloaded by itself or fetched from another coalition player through the direct local link, by the time it completes playing the previous video segment s_{i-1}; otherwise, there might be a rebuffering delay affecting the quality of experience (QoE). Third, the Internet bandwidth of individual players may vary over time; therefore, the coalition as a whole should schedule the video segment downloads among its members as well as decide the bitrate of every video segment based on the ABR principle.

Contributions: Owing to the above challenges, we develop a coalition-based adaptive live streaming called *Federated Live Streaming over DASH* (FLiDASH) where the streaming clients or players form a dynamic coalition based on the network quality parameters and collectively stream a live video. We use the playback buffer statistics at individual streaming clients to develop a distributed mechanism for coalition formation with the help from a proximity server (which can be an ALTO server). The members of a coalition use a low-overhead gossip-based protocol for playback synchronization and takes following two decisions – (1) scheduling the downloads of video segments among the coalition members based on their individual instantaneous network condition and the overall fairness criteria, and (2) bitrate of each video segments to optimize the overall QoE of the coalition. We use the following QoE objectives while making the above decisions – (a) improve the overall video quality level, (b) improve the playback smoothness by reducing the quality fluctuations, (c) reduce rebuffering, and (d) improve fairness among the coalition members in terms of the downloaded data share. We have implemented FLiDASH over an emulated environment and have thoroughly compared its performance with various other baselines. We observe that FLiDASH improves the overall QoE with less traffic overhead at the backbone network.

2 RELATED WORK

There are few works in the literature like *DASH over Information Centric Networking* (ICN) [9] and *Multicast ABR* [1], which use adaptive bitrate in a collaborative setup. All the segments of a DASH video contain a unique name through the URL. ICN provides a way to get the contents based on a name, and thus the DASH video segments can be routed according to the name. In case of a ICN, if the video players are behind a common ICN gateway, the gateway can cache the DASH video segments based on the unique name, and thus, can reduce duplicate delivery of contents. However, ICN needs significant changes in the network architecture including its routing policies, and thus, is not readily deployable on the existing networks. In 2016, Cablelabs introduced multicast ABR (M-ABR) [1] to provide the same video content to a large group of viewers while ensuring low network overhead via IP multicast. In M-ABR, all the ISPs maintain an M-ABR device that gets connected to the original video server. The M-ABR device receives the data from the video content server and then uses IP multicast to forward the content to the end-users. However, in this architecture, the content provider needs to push a middlebox (M-ABR server) to the ISP. Further, there needs to be a software change at the client devices. There exist few works in the literature to take care of the ABR selection over a collaborative setup, however, in these systems, the video players need to coordinate with an Internet middlebox that works as a central controller, such as a tracker [4], a software controller [7] or the cloud [15, 20]. This limits the scalability as all the video players need to coordinate with the controller for each ABR decisions which are taken for every video segment requests. In contrast to such existing approaches, FLiDASH works without the support from any such Internet middleboxes.

3 SYSTEM ARCHITECTURE

FLiDASH has three components as shown in Figure 2.

(1) **Streaming Server:** The streaming server is a content delivery network (CDN) server which encodes the live videos and hosts the video segments. We consider a scenario of a slightly delayed broadcast of the live streaming contents [5], where a playout delay (in the order of a few seconds) is introduced between the content recording and the content broadcast, within which the content server can process the recorded video segments to encode them in multiple DASH-supported bitrates. As a consequence, at any instance of time, a few number of DASH-encoded video segments are available at the content server for downloading them in parallel. It can be noted that such a playout delay is common in many commercial live streaming systems [23].

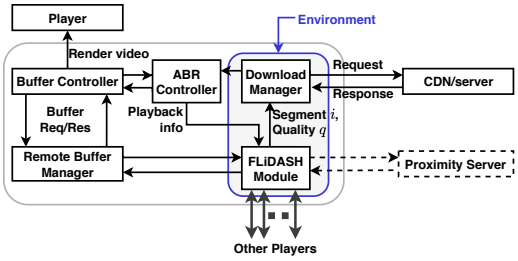


Figure 2: Architectural Components for FLiDASH

(2) **Proximity Server:** It keeps the locality information about the video players who stream the same live video content. A streaming server or a player can act as the Proximity server. However, if the network have an ALTO service, the coalition formation becomes easier [2]. ALTO mainly provides two basic pieces of information – a cost map and a network map. These two pieces of information help the player to understand the distance and delay from another player. Without the ALTO service, FLiDASH players have to ping or traceroute to understand the distance and delay, which is comparatively more expensive.

(3) **Player:** A video player renders the video at the end-user side as well as creates and maintains the coalition. Each player consists of three modules – i) *Playback*, ii) *Environment*, and iii) *Adaptive Bit Rate* (ABR). A *Playback* module keeps track of the playback buffer and the playback time. The *Environment* module forms a coalition with the nearby players and shares segments among other players in

the coalition, following a distributed policy enforcement principle. Every time it receives a segment, the playback module asks the *ABR* module for the next quality level and the sleep-time before it can start downloading. The sleep-time dictates the frequency of attempting a direct download from the CDN.

3.1 Streaming Coalition Formation

To form a coalition, FLiDASH first uses the expected playback quality for a player based on its view-port size and resolution (for example, playing a video on a laptop requires higher quality level than playing it over a smartphone, to ensure similar QoE for the user), along with the average network bandwidth. To estimate the average network bandwidth, a player P_i waits for some time to download $\mathcal{T}$ number of video segments directly from the streaming server with a server-client based ABR algorithm [11, 18, 22] and measures the throughput τ_i based on the amount of data downloaded. Let Q_G and Q_{P_i} be the average playback quality of a coalition G_p and a player P_i. The player P_i tries to find a coalition G_p in the vicinity (based on the information received from the *Proximity Server*) such that (a) P_i is within the same network of the existing players in G_p (all the players are behind a common core network gateway), (b) $Q_G \approx Q_{P_i}$, (c) $\forall_{P_k \in G_p} delay(P_i, P_k) < t_d$ (using the cost map and network map from the ALTO server) where t_d is a threshold on the permissible delay between two coalition members[2], and (d) the current playback time of the player P_i has a maximum one segment delay from the current playback time of the coalition (this condition ensures playback synchronization during coalition formation). If no such coalition exists in the vicinity, the player continues as a standalone player until another player joins to form a coalition.

3.2 Segment Download and Distribution

The Internet bandwidth between a player and the content server may change over time, and this change may be different for different FLiDASH players. Therefore, the coalition needs to take two collective decisions – (i) which player would download the next video segment, and (ii) what would be the download quality of the next video segment depending on the collective QoE of the coalition. FLiDASH selects an owner for every video segment s_i (O_{s_i}), which download the video segment, as well as takes two decisions – (a) the bitrate of the downloaded video segment s_i, and (b) the owner of the next video segment s_{i+1} ($O_{s_{i+1}}$). Once O_{s_i} elects $O_{s_{i+1}}$, this information is broadcast among the coalition members through gossip protocol. The details follow.

3.2.1 Owner Selection for the Next Segment. In FLiDASH, the owner of each segment is selected based on the following objectives – (i) maximize parallel downloads of video segments from the content server, (ii) max-min fair download load allocation among the coalition members, (iii) maximize the playback bitrate of the coalition, (iv) maintain playback synchronization among coalition members.

As we mentioned earlier that the playout delay at the content server ensures the availability of a few number of video segments encoded in different supported bitrates, the coalition members download them in parallel. The owner selection mechanism distributes the segments among the coalition members based on their instantaneous Internet bandwidth. For example, if a player P_1 has twice the Internet bandwidth than another player P_2, then player P_2 should download one video segment from the CDN, while the player P_2 should download two video segments of the same quality level within the same instance of time. Therefore, a total of three video segments can be downloaded collectively by the two players in parallel; they can share the remaining segments with each other for the video playback.

Based on the above principal, O_{s_i} uses Eq. (1) to find $O_{s_{i+1}}$. Here, $\mathcal{I}_x$ is the duration (idle time) from the last download for O_{s_x}. $\mathcal{D}q_x$ and $\mathcal{D}l_x$ are the download queue length (i.e. number of pending segments to be download) at O_{s_x} and the download status[3] of the ongoing download, respectively. $\mathcal{I}_{max}$ and $\mathcal{D}q_{max}$ are maximum possible idle time (which is $groupsize \times segment\_duration$) and maximum possible download queue length (same as play buffer length in term of number of segments).

$$O_{s_x} = \underset{x \in G_p}{\operatorname{argmax}} \left(\mathcal{I}_x/\mathcal{I}_{max} - \mathcal{D}q_x/\mathcal{D}q_{max} - \mathcal{D}l_x - \mathcal{M}_x \right) \quad (1)$$

Eq. (1) selects O_{s_x} who is idle for the longest time (by looking at the $\mathcal{I}_x$) among all the players in the coalition. In case all the players are busy downloading video segments, the equation considers the download load ($\mathcal{D}q_x$ and $\mathcal{D}l_x$) on every player. To make sure that the furthest segment should be allocated to the slowest player (which is important for playback synchronization), we calculate the **deadline miss penalty** $\mathcal{M}_x$. Every direct download from the CDN server is associated with a deadline. $\mathcal{M}_x$ is a function that returns zero if the deadline has not been missed by player x during the last download operation; otherwise, it calculates a penalty based on the deadline miss duration and the current time. So, the player with the lowest load and the highest bandwidth is selected as the owner of the next segment.

Forceful self-download: FLiDASH expects that corresponding owners would be able to download the segment before its playback time; otherwise, the coalition players may experience a stall time to rebuffer the video, which severely degrades their QoE. However, we cannot avoid this situation completely as the bitrate selection decision is based on the observation of the historical throughput and the instantaneous measure of the available bandwidth; whereas the owner may experience a sudden bandwidth drop during the actual download of the target segment. To avoid this situation, we put a deadline for each segment. If a owner fails to download the segment before the deadline (other players in the coalition fail to fetch the segment from the download buffer of the owner), other players in the coalition who are free at that time (within the sleep-time duration) trigger a forceful self-download of the segment at a low quality, to avoid rebuffering due to a loss in playback synchronization.

3.2.2 Bitrate Selection. FLiDASH uses a dynamic tuning approach for bitrate selection based on collective influence of the coalition members. To select the bitrate for segment s_i, we use Algorithm 1. This algorithm executes just before a player starts downloading a segment. In the algorithm, Θ is the measured throughput share of

[2] In all of our experiments, we used t_d =8 ms, which has been decided empirically based on the impact of this parameter on the performance of the system.

[3] If the segment length is m bytes, and n bytes has been downloaded as of now, them $\mathcal{D}l_x = \frac{m-n}{m}$

Algorithm 1: findCurrentQuality()

```
if d_t ≤ 0 then
    m*_n = 0
    Return
m*_n ← m*_{n-1}
Θ ← min(Θ_w, Θ_last, Θ_h)
m' ← argmin_{m∈Q} { | d_t − Cl_{n,m} / Θ | }
if m' > m*_n then
    if m*_{n-1} = m*_{n-2} and m*_{n-2} = m*_{n-3} then
        m*_n ← m*_{n-1} + 1
else
    m*_n = ⌈ (m' + m*_n) / 2 ⌉
```

the current player in comparison to the total bandwidth available for the coalition. The measured throughput has three components – the weighted throughput (Θ_w) which changes its value slowly over the time, Θ_{last} which is the throughput measured during the last finished download by the current player, and Θ_h is the harmonic mean of the throughput observed till now. The weighted throughput Θ_w is measured as $\Theta_{w_i} = 0.8 \times \Theta_{w_{i-1}} + 0.2 \times \Theta_{last}$. We use Θ as the minimum of Θ_w, Θ_{last}, and Θ_h, because it is the worst throughput the player observed till now. As we use Θ to predict the time require to download a segment, it gives us a worst time bound. d_t is the time left to download the segment s_i so that no player in the coalition stalls. $Cl_{i,j}$ is the content length of the i^{th} segment at the quality level j. m_i^* is the selected quality level for the i^{th} segment. Algorithm 1 allows the coalition to increase the video quality when there is a steady network. So, we do not put any restriction in the upper limit of the bitrate even though it involves slight risk of additional stalls and bitrate fluctuations. We analyze the individual QoE parameters in the evaluation, as discussed in the next section.

4 EVALUATION

We compare FLiDASH performance with following baselines – BOLA [18], MPC [22], Pensieve [11] which are client-server based ABR streaming mechanism, and a distributed hash table (DHT) based peer-assisted live streaming system [17]. We do not compare the performance of FLiDASH with the performance of middlebox-based adaptive live streaming platforms, such as [4, 7, 15, 20], as the architectures of the two environments are completely different. [17] uses a distributed architecture for overlay formation and streaming data scheduling, so it is close to our proposed architecture. The source code of the implementation is available at https://github.com/abhimp/FLiDASH (accessed: March 25, 2021).

4.1 Emulation Environment

We have developed an emulation platform similar to Pensieve emulator [11] to analyze the performance of FLiDASH, however, extending it for multi-player environment as discussed next. In the emulation platform, all the players in the system have access to the system clock, which is an event-driven clock handled by the emulator core. The emulator uses a reference network to define the connectivity across the networked nodes; where every node of the network runs a streaming player. The reference network provides the network map only. For our experiments, we assume that the inter-node link capacity is 100mbps, and the latency varies from 4ms to 64ms. We maintain a global playback time defined by the live streaming server, and the streaming server generates DASH segments of a live video based on the global playback time, maintaining the video frame recording timestamps. We use Mahimahi [14] network traces to emulate the network condition of the links between the streaming server and the players, where we randomly assign different Mahimahi traces to every player in a network. The emulated player can use any existing ABR implementation without any modification as long as it takes the player-state as the input and gives the next segment quality as the output. In the emulated environment, we use Eq. (2) to compute the transmission time T_{ij} between two players P_i and P_j in a coalition. Here $clen$ is the data length of the video segment, d_{ij} is delay between the two nodes, buf is the sender buffer and z is the noise factor which is uniformly distributed between θ_1 to θ_2. In our implementation, we consider the value of θ_1 and θ_2 as 0.95 and 1.05, respectively.

$$T_{ij} = clen \times \frac{buf}{2 \times d_{ij}} \times z \qquad \theta_1 <= z <= \theta_2 \tag{2}$$

4.2 Experimental Setup

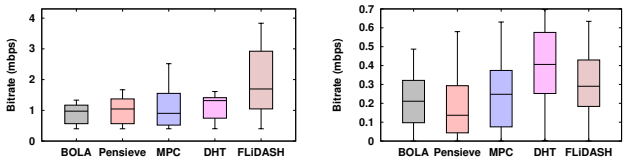
We run our emulation with a broad set of autonomous system data available from SNAP database [10] as reference networks. We have executed the systems over 710 reference networks, with 100 to 1000 nodes per network. The link-speed for every node is set based on the Mahimahi network traces. As mentioned earlier, every node runs a streaming client. To train the model for learning based adaptive streaming like Pensieve [11], we use 58 DASH-encoded videos with a total duration of 45 hours. We have created Mahimahi compatible traces from the following publicly available dataset – a broadband trace from FCC [3] and the 3G/HSDPA mobile dataset collected in Norway [16]. We modified these datasets as described in [11] to make it Mahimahi compatible.

For experimentation, we use the QoE definition as given in [11]. According to this definition, we consider three QoE components – (i) average quality level, (ii) average jump in the quality level (smoothness of the video playback) and (iii) re-buffering time (stall time). Let Q_n denote the quality level for video segment n and $\mathcal{T}_n$ be the re-buffering time. Considering that there are N number of segments in a reference video, the average QoE is defined as follows. Here, α, β and γ are weight factors, whose values have been considered as 1, 1 and 4.3, respectively, similar to Pensieve [11].

$$QoE = \frac{\alpha}{N}\sum_{n=1}^{N}\mathcal{F}(Q_n) - \frac{\beta}{N-1}\sum_{n=2}^{N}|\mathcal{F}(Q_n) - \mathcal{F}(Q_{n-1})| - \frac{\gamma}{N}\sum_{n=1}^{N}\mathcal{T}_n \tag{3}$$

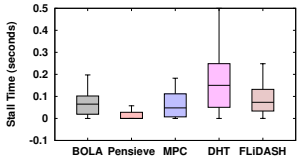
4.3 QoE Analysis

We first observe the individual QoE components for FLiDASH in comparison with other baselines. Figure 3a compares the average playback bitrate for various streaming applications. In Figure 3b, we show the variation in average playback bitrates, which indicate the lack of smoothness of the video rendering. We observe that the performance of BOLA in terms of average playback bitrate is very low; BOLA is very conservative about the bitrate, whereas it is much concerned about the re-buffering time. Pensieve and MPC improve the video quality compared to BOLA by utilizing a learning-based approach. DHT uses the knowledge of existing

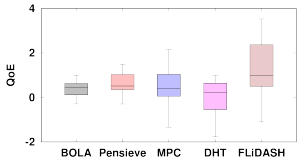


(a) Average Bitrate (More is better)

(b) Bitrate Variation (Less is better)



(c) Rebuffering Time (Less is better)



(d) Overall QoE (More is better)

Figure 3: Performance in terms of Various QoE Components and Overall QoE

players in the network and forms a peer-to-peer architecture for collectively download the videos. So, it improves the average video quality compared to client-server based ABR. However, FLiDASH clusters the players based on their network conditions and render the videos keeping the coalition members in sync. In Figure 3a, it is clear that there are clusters of players who play a video in almost equal quality levels. Although we observe that the average variation in the video quality is high for FLiDASH compared to other baselines. FLiDASH, by default, plays the video in a high bitrate compared to other baselines; therefore, even a single quality-level fluctuation significantly impacts the overall QoE. Further, a forceful self-download contributes to the bitrate variation. Therefore, we observe higher bitrate variation in FLiDASH compared to other baselines.

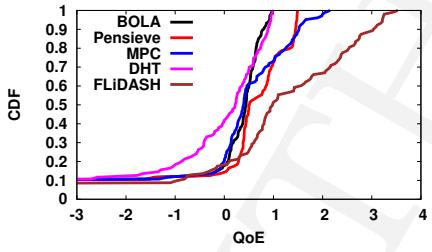
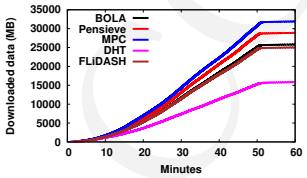


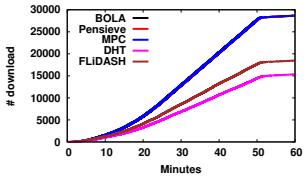
Figure 4: Overall QoE distributions: FLiDASH outperforms other baselines for 80% of the scenarios

Next, Figure 3c compares the total rebuffering time among various baselines. We observe that the re-buffering time is very high for DHT because it needs more time to search for a video segment from the network before it can fetch it directly from the streaming server. The re-buffering time for FLiDASH is moderated, although it includes the skip time during the synchronization among the members of the coalition. BOLA incurs almost no re-buffering time; whereas Pensieve and MPC suffer from noticeable re-buffering time. As the re-buffering time is a significant contributor in the overall QoE measurement (Eq. (3)), the overall QoE for various baselines, as shown in Figure 3d, indicates that FLiDASH outperforms other baselines in term of maximum achievable QoE. Among the various scenarios simulated over our developed platform, more than 50% of the cases, FLiDASH incurs a high QoE (value between 2–4). Figure 4 shows the distribution of the overall QoE for FLiDASH in comparison with other baselines. We observe that FLiDASH outperforms other baselines for 80% of the time.

4.4 Direct Traffic from the CDN



(a) Data downloaded from server



(b) Segment downloads from server

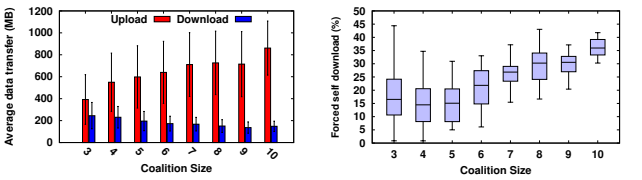
Figure 5: Direct traffic from the CDN: The direct download load from the server is more than the DHT-based approach bus less than other baselines

One of the major objectives of FLiDASH is to reduce the direct traffic from the CDN server when multiple co-located players play the same live video. In Figure 5, we plot the direct traffic from the content server by different baselines in terms of (a) the total bytes downloaded, and (b) video segments directly downloaded from the content server. We observe that the DHT based system downloads minimum data from the CDN. In FLiDASH, players are bounded to receive data from its own coalition only, while in case of DHT, a player can share segments with as many players as possible in the complete network. The standalone players need to download all the segments directly from the server. As a consequence, we observe a performance trade-off here between a complete peer-to-peer based approach and the coalition-based approach – FLiDASH significantly improves the QoE performance while having little increase in the CDN traffic overhead. In a nutshell, the proposed approach makes a balance between the QoE and the network traffic overhead during live video streaming.

4.5 Impact of Coalition Size

We next analyze the performance of FLiDASH in the context of various design choices. First, we check the impact of coalition size on the performance of FLiDASH[4]. Figure 6a shows the amount of data downloaded at each client from the streaming server (CDN) and the total amount of upload data from each client indicating the local network traffic that has been used to distribute the downloaded segments among the coalition members. We observe that a large coalition size reduces the download data share among the streaming clients, as the total video data gets distributed among the coalition members. However, we further observe an increase in the

[4]In all the previous experiments, we have fixed the maximum coalition size to 4 players



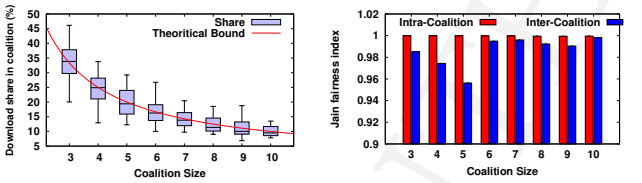
(a) Amount of Data Upload and Download from Individual Players

(b) Percentage of forceful self-download due to deadline missess

Figure 6: Effect of Coalition Size – Large coalitions reduce the CDN load but increase intra-network data consumption and the forceful self-download due to deadline misses

upload data share, as each client needs to distribute the individually downloaded video segments to all other clients. However, it can be noted that the download data actually contributes to the network (CDN) load, whereas the upload data is over the local network only.

Figure 6b indicates *the percentage of forceful self-download* with different coalition sizes. We observe that with increasing coalition size, the percentage of forceful self-download drops till coalition size 5, and then it increases. Ensuring playback synchronization is difficult when the coalition size is either too small or too large. With a very small coalition size, the data download overheads for individual clients increase, and thus, a small variation in the network bandwidth may result in a deadline miss. On the other hand, with a large coalition size, the variation in the instantaneous network bandwidth among the coalition members is more, and therefore the clients having less instantaneous network bandwidth may experience a deadline miss, resulting in a forceful self-download. Therefore, we see that there exists an optimal coalition size (5 in our setup) which indeed reduces the forceful self-download thus improves the overall QoE by reducing the bitrate variation.



(a) Download share of a Coalition Member: The average download share of individual players meet the theoretical fair share bound ($y = \frac{1}{x}$)

(b) Jain fairness index on QoE acheived among players: Intra-coalition QoE fairness is always good, inter-coalition fairness depends on coalition size

Figure 7: Effect of Coalition Size – Large coalitions reduce the individual download share; the intra-coalition QoE fairness index is always good

Next, we analyze how the coalition size impacts the total download share among coalition members, indicating the load-fairness of the system. Figure 7a indicates that a large coalition reduces individual download shares among coalition members. In terms of load-fairness, we observe that large coalitions provide better fairness. We also plot the $y = \frac{1}{x}$ curve which shows the theoretical fair share among the coalition members. It is conforming for us to see that the average load-fairness fits the theoretical fair share.

To quantify the QoE fairness among the coalition members, we measure the Jain fairness index [6] on the measured QoE of the individual players in two categories – (i) *intra-coalition QoE fairness* (fairness among the individual members of a coalition) and (ii) *inter-coalition QoE fairness* (fairness among different coalition members). Figure 7b shows that the intra-coalition QoE fairness index is always close to 1, which indicates good fairness among the coalition members in terms of the QoE experienced by individual clients of a coalition. However, the inter-coalition QoE fairness initially gets reduced with increasing coalition size up to 5. As different coalitions can play the video in different average bitrates, which has a major contribution to the overall QoE, we initially see this drop in the inter-coalition QoE fairness. Interestingly, inter-coalition QoE fairness increases with a large coalition size, as large coalitions result in less number of coalitions, hence less bitrate variation among different coalitions.

5 DISCUSSION AND CONCLUSION

In this paper, we develop a middlebox-free collaborative adaptive live streaming system which improves the overall QoE while reducing the network traffic. The architecture utilizes a federated platform where the streaming players form coalitions and schedule video downloads in a distributed way without the help of any Internet middleboxes. The proposed architecture has been implemented and evaluated over a thorough emulation platform, and we observe significant improves in the overall QoE with a reduction in the load at the live streaming server.

A real-world implementation of FLiDASH requires the support of a proximity server and installation of the playback client in the end-user's device. It can be noted that the currently available commercial client applications (like YouTube, Twitch clients) need to have FLiDASH support, although no changes are required at the content-server or ISP side. Further, the system does not use any Internet-middlebox for coalition formation. We can also address the issues faced by existing internet middleboxes, like when the clients are behind a NAT, by using a *Session Traversal Utilities for NAT* (STUN) [13] or a *Traversal Using Relays around NAT* (TURN) [12] server. In this aspect, FLiDASH provides a more deployment-ready solution compared to ICN-DASH [9] or Multicast ABR [1].

We believe that FLiDASH has the potential for deploying a highly-scalable architecture for mass-scale live streaming of videos while incurring low overhead to the backbone network. Indeed, such a system can be very useful for various developing countries where live streaming of video contents has significantly increased in the recent years, although the network backbone infrastructure is yet to be matured [8].

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers and the shepherd of our paper, Dr. Kai Gao, Sichuan University, China, for providing insightful comments and suggestions to improve the quality of this paper.

REFERENCES

[1] 2016. IP Multicast Adaptive Bit Rate Architecture Technical Report. https://specification-search.cablelabs.com/ip-multicast-adaptive-bit-rate-architecture-technical-report. Accessed: 2020-06-26.

[2] Richard Alimi, R Penno, Y Yang, S Kiesel, S Previdi, W Roome, S Shalunov, and R Woundy. 2014. Application-layer traffic optimization (ALTO) protocol. *RFC 7285* (2014).

[3] Federal Communications Commission. 2018. Raw Data - Measuring Broadband America - Eighth Report. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-eighth. [accessed March 25, 2021].

[4] Andrea Detti, Bruno Ricci, and Nicola Blefari-Melazzi. 2016. Tracker-assisted rate adaptation for MPEG DASH live streaming. In *IEEE INFOCOM*. 1–9.

[5] Rafael Huysegems, Jeroen Van Der Hooft, Tom Bostoen, Patrice Rondao Alface, Stefano Petrangeli, Tim Wauters, and Filip De Turck. 2015. HTTP/2-based methods to improve the live experience of adaptive streaming. In *ACM MM*. 541–550.

[6] Raj Jain, Arjan Durresi, and Gojko Babic. 1999. Throughput fairness index: An explanation. In *ATM Forum contribution*, Vol. 99.

[7] Ahmed Khalid, Ahmed H Zahran, and Cormac J Sreenan. 2019. An SDN-based device-aware live video service for inter-domain adaptive bitrate streaming. In *ACM MMSys*. 121–132.

[8] Diego Kiedanski, Mateo Nogueira, and Eduardo Grampín. 2019. Youtube traffic from the perspective of a developing country: the case of Uruguay. In *IEEE INFOCOM Workshops*. 714–719.

[9] S. Lederer, C. Mueller, B. Rainer, C. Timmerer, and H. Hellwagner. 2013. An experimental analysis of Dynamic Adaptive Streaming over HTTP in Content Centric Networks. In *IEEE ICME*. 1–6.

[10] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *ACM SIGKDD*. 177–187.

[11] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM*.

[12] Philip Matthews, Jonathan Rosenberg, and Rohan Mahy. 2010. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766. https://doi.org/10.17487/RFC5766

[13] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. 2008. Session Traversal Utilities for NAT (STUN). RFC 5389. https://doi.org/10.17487/RFC5389

[14] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX ATC*. 417–429.

[15] Amir H Payberah, Hanna Kavalionak, Vimalkumar Kumaresan, Alberto Montresor, and Seif Haridi. 2012. Clive: Cloud-assisted p2p live streaming. In *IEEE P2P*. 79–90.

[16] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *ACM MMSys*. 114–118.

[17] Haiying Shen, Ze Li, and Jin Li. 2013. A DHT-aided chunk-driven overlay for scalable and efficient peer-to-peer live streaming. *IEEE TPDS* 24, 11 (2013), 2125–2137.

[18] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *ACM MMSys*. 123–137.

[19] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y Zhao. 2016. Anatomy of a personalized livestreaming system. In *ACM IMC*. 485–498.

[20] Feng Wang, Jiangchuan Liu, Minghua Chen, and Haiyang Wang. 2014. Migration towards cloud-assisted live media streaming. *IEEE/ACM Transactions on networking* 24, 1 (2014), 272–282.

[21] Huan Yan, Tzu-Heng Lin, Chuhan Gao, Yong Li, and Depeng Jin. 2018. On the understanding of video streaming viewing behaviors across different content providers. *IEEE TNSM* 15, 1 (2018), 444–457.

[22] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *ACM SIGCOMM*. New York, New York, USA.

[23] Thomas Zinner, Stefan Geissler, Fabian Helmschrott, and Valentin Burger. 2017. Comparison of the initial delay for video playout start for different HTTP-based transport protocols. In *IEEE IM*. 1027–1030.