ShadowVoD: Performance Evaluation as a Capability in Production P2P-CDN Hybrid VoD Networks

Hanzi Mao, Chen Tian, Jingdong Sun, Junhua Yan, Weimin Wu, and Benxiong Huang
Huazhong University of Science and Technology
Email: {hanzimao, tianchen, sunjd, junhuayan, wuwm, huangbx} @ hust.edu.cn

Abstract—Video-on-Demand (VoD) services have achieved great success recently. Most such streaming systems are P2P-CDN hybrid systems. To ensure reliable performance, the most efficient way is to subject those VoD streaming networks to large-scale, realistic performance evaluations. Our previous ShadowStream system is a production Internet live streaming network with performance evaluation as a built-in capability. In this paper, we extend the same idea into the VoD services. There exists significant difference between live and VoD, hence ShadowStream cannot be directly used in VoD context. Firstly, clients in P2P-VoD service are not synchronized in viewing progress; secondly, in VoD there exists interactive operations (e.g., pause and drag); thirdly, the different playpoints of users also bring difficulty to replacing departed real clients. In this paper, we solve all above mentioned challenges. We implement ShadowVoD and demonstrate its benefits through extensive evaluations.

I. INTRODUCTION

In recent years, Video-on-demand (VoD) service has become the most popular Internet streaming application. With a VoD service, users can view previously recorded video content at a different time [1]. It is believed that the VoD traffic will triple by 2015, which equivalents to 3 billion DVDs per month [2]. Most such streaming systems are P2P-CDN hybrid systems; it becomes more and more important to guarantee that VoD service networks can provide reliable performance.

To ensure reliable performance, the most efficient way is to subject those VoD streaming networks to large-scale, realistic performance evaluations. Traditional testing technologies, such as lab/testbed testing [7], [8], fail to obtain reliable testing results due to the lack of both scale and realistic features. For example, it is hard for a testbed to capture the heterogeneous network features such as PowerBoost [9] in cable networks, large hidden buffers in access networks [10], and shared bottlenecks at ISP peering or enterprise ingress/egress links [12]. It is clear that only in large-scale, realistic environment, the capacity of VoD networks to handle increasingly complex Internet and large number of users can be fully tested.

In ShadowStream [13], we introduces live testing, where performance evaluation is a built-in capability in production Internet live streaming networks: a production streaming network itself can be a unique testing system with both scale and realism. Two major challenges are solved. The first is protection: since real viewers are used in testing, live testing needs to protect the real viewers from possible performance failures of experimental systems; the second is orchestration, where desired experimental scenarios (e.g., flash-crowd) can be orchestrated by adding real viewers.

In this paper, we extend the live testing capacity to VoD systems, which is called ShadowVoD. The new live testing system built in production Internet VoD streaming networks, with the major objective of obtaining accurate testing results, should also guarantee both protection and orchestration.

There exists different playback and interaction pattern between live and VoD systems; as a result, the testing system designed for live streaming can not be directly applied to VoD networks. To be more specific, we shall address three key challenges that are not presented in live streaming.

Firstly, clients in P2P-VoD service are not synchronized in viewing progress as those in live streaming systems [3]. In live, the newest video piece is constrained by the current time; viewers’ playpoints (currently viewing video piece) are close to each other, and can not be lag from real time too much. While in VoD, due to the different start time of clients, the playpoints are different; also, due to the content prefetch characteristic of VoD, it is necessary for clients to store previously watched video pieces to help each other. Therefore, the content play buffer (which is the key data structure of a streaming client software), should be redesigned for VoD.

Secondly, another major difference between VoD and live streaming is that in VoD there are interactive operations. In live, viewers’ interactions are limited; they are only permitted to start or stop watching a streaming. Differently, in VoD users can pause and drag at any time they want. We should guarantee that real viewers’ quality of experience is not affected after the live testing design is implemented. In addition, evaluations based on real viewers as they naturally pause and drag, may not always match the expected patterns of developers. It is more desirable that testing clients’ behavior scenarios of pause and drag can be created.

Thirdly, the different playpoints of users also bring difficulty to replacing departed real clients. A distributed orchestration is introduced in ShadowStream where each client locally chooses to become a replacement candidate or not based on the possibility distributed by the orchestrator. However, it is more advisable to select specific users as substitutes of early departed clients in VoD as we should take their different playpoints into consideration.

We solve all above mentioned challenges in this paper. We implement ShadowVoD and demonstrate its benefits through extensive evaluations. The rest of the paper is organized as follows: In Section II we present how ShadowStream achieves the PCE scheme in live. Section III contains the client design and implementation in ShadowVoD. Behavior emulation of
II. BACKGROUND

A. Streaming Basics

A key capability to ensure streaming networks providing reliable performance is to subject them into large-scale, realistic performance evaluations. ShadowStream is a novel Internet live streaming system that integrates performance evaluation as an intrinsic capability. It is a hybrid P2P-CDN piece based streaming system, where a live streaming client downloads and uploads streaming data in units of pieces. In ShadowStream, a self-complete set of algorithms to download and upload pieces is called a streaming machine or a machine for short.

The machine’s play buffer keeps track of pieces that are already downloaded as well as the pieces that the machine needs to download. Figure 1 (a) is an example illustrating the play buffer status at time \( t = 100 \) of a client \( i \). The index of a piece is the time that the piece was produced at the source; a shaded piece is one that has been downloaded. The most piece of play buffer is the most recent piece produced by the source. We refer to this piece as the sourcepoint. Another important piece at an instance of time is the next piece to be delivered to the media player at the client. We refer to it as the playpoint. For the example in Figure 1 (a), the playpoint is 90. The playpoint and sourcepoint advance in time. Figure 1 (b) shows the advancement of the play buffer from Figure 1 (a). We see that at the next time \( t = 101 \), the playpoint becomes 91, and the sourcepoint becomes 101. If the client is not able to download any piece during \( t = 100 \) to \( t = 101 \), then the machine fails to download piece 91 in time and we say that the piece is missing.

B. ShadowStream

ShadowStream needs to address two major challenges. The first is protection: since real viewers are used, live testing needs to protect the real viewers quality of experience from the performance failures of experimental systems. The second is orchestration: live testing needs to orchestrate desired experimental scenarios from production viewers, without disturbing their quality of experiences.

In the design of ShadowStream, three streaming machine run concurrently. The current production streaming version is named production; the streaming machine which downloads pieces directly from a dedicated CDN resources is called rCDN; the streaming machine called experiment contains the experimental algorithms of which we want to make performance evaluations.

ShadowStream applies a novel Experiment → Validation → Repair scheme. The key idea is to handle each downloading task of a piece in a temporal sequential pattern. To reveal the true performance of experiment, ShadowStream assigns the task of downloading a piece first to experiment alone. If experiment cannot download it by its playpoint, rCDN takes over the responsibility and try to “repair” by downloading from CDN. If the repair of rCDN fails, production shall try to download it as a final protection. We evaluate the performance of the testing VoD streaming network by the piece missing ratio of experiment. Fig.2 shows how this scheme works.

A streaming hypervisor is introduced to inform streaming machines of pieces which they should download respectively. Conceptually the total downloading range spanning from real playpoint to sourcepoint is divided into three parts by production, rCDN and experiment sequentially. Each part is referred as the task window of corresponding streaming machine. The playpoint and sourcepoint advance in time along with the task windows and the piece missed by the right task window becomes the downloading task of the left one.

To ensure that the failure of experiment is not visible to the real viewer during our evaluation, we need that the playpoint of experiment is not the actual viewer visible playpoint. Hence, we say that the playpoint of an experimental streaming machine is a virtual playpoint, and the playpoint of a production streaming machine is a real playpoint.

A ShadowStream client always starts with production alone. Then at certain computed time, corresponding rCDN and experiment are notified to join the test to create desired testing client behavior scenarios. Their state before joining the test can be considered as missing every piece, the virtual “arrival” and “departure” of rCDN and experiment are simple state transition.

III. CLIENT DESIGN AND IMPLEMENTATION

We start with the design and implementation of ShadowVoD client and focus on the change of the play buffer and task window management. The novel PCE scheme is still implemented in ShadowVoD with a mechanism of flexible window length of rCDN introduced.
In ShadowVoD, clients download and upload streaming data in units of pieces. Each piece is assumed to contain 1-second of streaming data. Here we introduce the play buffer to record the status of pieces about whether they have been downloaded or still need to be downloaded.

Different from live streaming, where the newest piece is constrained by the current time and can thus be defined as the sourcepoint, in VoD all movie data is already there, and there does not exist the sourcepoint anymore. Instead we define the right most piece of the play buffer as the bufferpoint. Both the play buffers of production and experiment are divided into two parts: the left one is called back buffer while the right one is referred as front buffer. The real playpoint and virtual playpoint serve as medians to partition the two streaming machines respectively. We should notice that the piece missing ratio is collected at the playpoint instead of the left boundary of back buffer. Fig.3 illustrates the play buffer of the new PCE layout in ShowdowVoD.

**B. PCE Design**

We still adopt the PCE scheme in ShadowVoD. When downloading a piece, the hypervisor will assign this task following a sequential experiment-rCDN-production scheme. The downloading task would be assigned to next streaming machine if the previous one fails, thus testing users’ quality of experience is guaranteed .

Besides, we propose a new mechanism that the window length of rCDN is flexible to achieve clients’ pause behavior emulation. Assuming that we already know the frequency of actual users’s pause for a certain video; we define the average of it as $X_0$. And we desire to make evaluations of experiment based on this video for the average frequency of pause as $X_1$.

- **Situation a:** If $X_1 = X_0$, the behavior of the task windows of the three streaming machines should be consistent with the behavior of users, which means once a user pauses the movie, the task windows stop moving forward while the downloading of pieces can continue.

- **Situation b:** If $X_1 > X_0$, we have to simulate additional users’ behavior of pause for experiment and guarantee the real viewers’ quality of experience at the same time. The additional pause should behave like actual users’ behavior including distributions of pause position and pause duration [15].

- **Situation c:** If $X_1 < X_0$, at certain times, when users choose to pause, we should let the task window of experiment continues to move forward while rCDN and production pause along with users.

The basic idea to achieve this mechanism is this: make the window length of rCDN flexible. The right boundary of rCDN (ie. The left boundary of experiment) could be freely shifted.

For the additional frequency ($X_1 - X_0$) in situation b, we simulate users’ behavior of pause. After getting the message from the hypervisor, the experiment buffer position freezes. To shield real users from being affected, production and rCDN move forward as usual. The trick here is that the hypervisor would dynamically advance the left boundary of rCDN but freeze the right boundary. The length of rCDN would be dynamically reduced as presented in Fig.4 (b) compared with the initial task window length of all three streaming machines shown in Fig.4 (a).

For situation c, we choose to let experiment move forward as usual while the production and rCDN buffer positions freeze. The hypervisor would dynamically advance the right boundary of rCDN but freeze the left boundary accordingly. The length of rCDN would get enlarged. Fig.4 (c) describes such situation.

The freely shifted right boundary of rCDN makes it possible to emulate users’ pause behavior based on frequency different from what real users choose to do. Algorithm details and other issues will be discussed in Section IV.

**C. Task Window Management**

In ShadowVoD, task window management is implemented by a simple streaming hypervisor to notify a streaming machine about pieces which it should download.

Here, we use two new functions x.getLength() and x.getPlayTime(). The x.getLength() function is used to get the length of the downloading task window of each streaming machine. Specially, for production and experiment the return values are the length of their front buffers. The x.getPlayTime() function is used to define the left boundaries of the downloading task windows. For production, it is the real playpoint, but for rCDN and experiment, they are actually virtual playpoints. We make this change because there does not exist sourcepoint in VoD anymore. And we can always confirm the playpoint of production easily, the playpoints of the other two streaming machines can be identified accordingly. The start time and returned values of getPlayTime() are listed in TABLE I. The subscripts l.front and l.back represent the length of front buffer and back buffer respectively.

<table>
<thead>
<tr>
<th>machine</th>
<th>start time</th>
<th>getPlayTime()</th>
</tr>
</thead>
<tbody>
<tr>
<td>production</td>
<td>Viewer arrival</td>
<td>playpoint</td>
</tr>
<tr>
<td>rCDN</td>
<td>Enter testing</td>
<td>playpoint + $p_{l.front}$</td>
</tr>
<tr>
<td>experiment</td>
<td>Enter testing</td>
<td>playpoint + $p_{l.front}$ + length of rCDN</td>
</tr>
</tbody>
</table>

**IV. DISTRIBUTED BEHAVIOR EMULATION**

A major difference between live and VoD is that users can choose to pause or drag at any time they want in VoD scenario. The performance evaluations of VoD networks should be deliberately designed to consider such interactive operations.
is proposed to emulate the pause behavior as presented, where \( x \) is defined as the pause duration and \( c.\text{reduce}(t) \) will reduce the length of \( e.\text{pause}(t) \) and \( c.\text{reduce}(t) \) rCDN to pause along with the user based on which we want is flexible also brings a key issue. In \( f(x) \) represents the probability accordingly. The mathematical simulated pause of \( \int_0^\infty \) may affect real view-

c.\text{enlarge}() by an expected production \( \mu > \bar{X} \) and the average of it is defined

\[ \mu = \frac{\sum_{i=1}^{n} X_i}{n} \]

\( p.\text{pause}(), c.\text{pause}() \) and \( e.\text{pause}() \) are called by the hyperv-
visor in situation a. For situation b, \( e.\text{pause}(t) \) and \( c.\text{reduce}(t) \) are called. For situation c, \( p.\text{pause}() \) and \( c.\text{enlarge}() \) are called. The pseudo-code for the distributed algorithm is shown in Fig. 5.

2) Protection Issue: The idea that the window length of rCDN is flexible also brings a key issue. In situation b, the decrease of the window length of rCDN may affect real view-

ers’ quality of experience because their time for downloading missing pieces from CDN is reduced. For situation c, if the window length enlarges too much that the right boundary of experiment is close to the end of the movie, this user is forced to quit testing.

To address this issue, we implement a simple distributed control mechanism. Every certain time, like \( 5s \), the hypervisor will call the function \( c.\text{getLength}() \) to get the length of the task window of rCDN. Once it reduces or enlarges beyond the range we defined, the user is notified to have an early departure.

To reduce the possibility of early departures, we compute the recommended value for the initial window length of rCDN. Given the probability distribution function of pause duration as \( f(x) \), where \( x \) is defined as the pause duration and \( f(x) \) represents the probability accordingly. The mathematical expectation of \( x \) can be represented as \( E(x) = \int_0^\infty x f(x) dx \).

If \( X_1 > X_0 \), the additional \( (X_1 - X_0) \) simulated pause of experiment will reduce the length of rCDN by an expected value of \( (X_1 - X_0) \int_0^\infty x f(x) dx \). This is the minimum value we recommend for the window length of rCDN.

B. Drag Emulation

Here we still assume that we already know the frequency that actual users decide to drag and the average of it is defined as \( Y_0 \). We desire to make evaluations based on this video for the average frequency of drag as \( Y_1 \).

For drag emulation, we focus on the situation of \( Y_0 = Y_1 \). In this situation, the behavior of the task windows of the three streaming machines should be consistent with the behavior of users. The new playpoint of \( \text{production} \) after drag should be the playpoint chosen by the real user. Challenge here is that if we handle real users’ drag behavior with simplistic notifications from hypervisor about the playpoint change of all three streaming machines, \( \text{production} \) and rCDN will download pieces to fulfill their empty task windows. Such downloading behavior will cause unnecessary interference to experiment as the downloading capability of the network is limited and the accuracy of testing will get impaired.

The approach proposed here is this: when the drag behavior of a user is notified, the hypervisor will inform the \( \text{production} \) of its new playpoint and the client can continue viewing the video after the drag delay. The rCDN and experiment are controlled to depart the testing locally. They will rejoin the testing channel after the buffer time \( \Delta t \) which is notified by the hypervisor. Two observations should be guaranteed when they rejoin the test: (1) pieces at the range for experiment should be empty; and (2) pieces for \( \text{production} \) and rCDN should be full. \( \text{Production} \) and rCDN can fulfill their task windows directly from the dedicated CDN during the buffer time \( \Delta t \).

Thus the total testing time for a certain client \( i \) changes from \((t_{\text{leave},i} - t_{\text{start},i})\) to \((t_{\text{leave},i} - \sum_{j=1}^{m} \Delta t_j - t_{\text{start},j})\), where \( t_{\text{leave},i} \) and \( t_{\text{start},j} \) represent the departure time and start time of client \( i \) respectively and \( \Delta t \) means the defined buffer time for client \( i \). The formula to compute the piece missing ratio of experiment should be changed accordingly.

When the drag frequency of actual users \( Y_0 \) is not equal to
the frequency $Y_1$ based on which we want to make evaluations. We desire to arrange current real clients to depart the testing and select suitable substitutes to continue. However there are many issues unsolved now and details will be discussed in further research.

C. Replace Early Departed Clients

Early departure may occur due to two different reasons in VoD. One is viewer-initiated operations and the other is the command of the hypervisor because the length of the task window of rCDN exceeds the maximum threshold we defined.

For both situations, the early departed clients will piggy-back a small state snapshot in the disconnection message which includes parameters such as the scheduled arrival time of the client and their current playpoint.

It is the orchestrator’s duty to select a substitute from the users who have not joined testing yet. In ShadowVoD, the playpoint of substitute should be close to the playpoint which is included in the disconnection message sent by the early departed client. Here we adopt a centralized orchestration which means the substitutes are selected by the orchestrator.

Every certain time, users who have not joined testing should inform the orchestrator of their playpoints. The orchestrator builds a data sheet to record this. Upon detecting an early-departed client, the orchestrator will notify the user whose playpoint is closest to the playpoint of early-departed client to join testing.

V. EVALUATIONS

In this section, we evaluate the practicability of ShadowVoD on a complete VoD streaming system we implement. We focus on the testification that, by collecting the piece missing ratio of experiment in ShadowVoD, we can accurately evaluate the performance of VoD networks. We also attest that the distributed algorithm to emulate pause behavior proposed in Section IV achieves the desired pause-behavior scenarios.

In the test, we arrange for 100 clients to join the testing at 5-second interval. They continue playing for 100 seconds after all clients have joined. In this paper we focus on the evaluation of experiment accuracy as the protection issue has been proved by ShadowStream. We remove the CDN capacity limitation which means all pieces missed by the experiment can be repaired by the dedicated CDN and viewers’ quality of experience can be guaranteed.

A bug is injected to the experiment on purpose. We divide the index of each piece by 20; if the remainder is equal to 1, the corresponding piece will not be downloaded or uploaded. We set the configuration like this to prove that even simple parameter changes may incur serious performance impact and fully evaluations are necessary to guarantee reliable performance of VoD networks.

A. PCE design with pause and drag behavior omitted

Firstly, we collect the piece missing ratio of the buggy version by running the experiment alone. The result is shown in the second column of Table III. We should notice that when the experiment is run alone, its behavior is totally the same with actual production. Then, we use our PCE design. Results shown in the third column of Table III come up to our expectations that the measured piece missing ratio of experiment is accurate compared with the result when the buggy version is run alone at a negligible error. The piece missing ratio at $p_{play}$ is 0% because CDN repairs all the pieces missed by the experiment.

<table>
<thead>
<tr>
<th>Buggy</th>
<th>PCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{play}$ Missed</td>
<td>5.51%</td>
</tr>
<tr>
<td>$p_{play}$ Missed</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The cumulative distribution functions of the piece missing ratio of all 100 clients with and without ShadowVoD are shown in Fig.6. We can see that by collecting the piece missing ratio of experiment in ShadowVoD, the performance of the VoD networks can be evaluated quite accurately.

![Fig. 6. Piece missing ratio of 100 clients](image)

B. PCE design with pause emulation

Next, we take the pause behavior of clients into consideration and testify that after introducing the methodology of flexible window length of rCDN, experiment accuracy can still be guaranteed. We orchestrate that actual clients choose to pause at the rate of one time every 100 seconds. Thus the production and rCDN of the first joined client shall pause 6 times during its 600-second play time. We still consider the three situations presented in Section III and the experiment is orchestrated to pause 6 times in situation $a$, 8 times in situation $b$.

TABLE II. API FUNCTIONS BETWEEN STREAMING MACHINES AND HYPERVISOR.

<table>
<thead>
<tr>
<th>Call</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.pause()</td>
<td>$H \rightarrow M$</td>
<td>Hypervisor notifies the streaming machine to stop its task window from moving forward while the downloading continues.</td>
</tr>
<tr>
<td>c.reduce(t)</td>
<td>$H \rightarrow M$</td>
<td>Hypervisor notifies the rCDN to advance its left boundary and freeze the right boundary at the same time. The downloading continues.</td>
</tr>
<tr>
<td>c.enlarge()</td>
<td>$H \rightarrow M$</td>
<td>Hypervisor notifies the rCDN to advance its right boundary and freeze the left boundary at the same time. The downloading continues.</td>
</tr>
<tr>
<td>x.restore()</td>
<td>$H \rightarrow M$</td>
<td>Hypervisor notifies the streaming machine to move forward and the length of its task window is fixed.</td>
</tr>
</tbody>
</table>

1. The parameter $t$ is used to notify $M$ how long it should keep execution of the function before restoring to its former state.

b. 4 times in situation c. The pause times of all later joined clients obeys the same rate with the first one.

To simplify the problem, we assume that the pause duration of actual viewers is a discrete random variable which obeys uniform distribution denoted by \( T \sim U(5, 15) \) in terms of seconds. The additional two pauses we simulate will reduce the length of rCDN in the range between 10 second and 30 seconds. We fix the length of rCDN window as 35 seconds to avoid early departures. We should notice that the window length of rCDN in actual deployment only need to be set to the recommended value according to the formula given in Section IV as the orchestrator can always select suitable substitutes for early departed clients.

As we adopt the distributed algorithm shown in Fig.5, in situation c each client independently control the simulated experiment utilizing the probability \( \mu = \frac{X_n - X_1}{X_n} \). Once the pause behavior of a client is notified, the hypervisor has the possibility of \( \mu \) to inform the experiment to continue moving forward. In our evaluations, the probability \( \mu = \frac{1}{2} \) and the simulated pause times \( X_1 \) is expected to be 4. The arithmetic average value of pause times of all 100 clients in our evaluations is 3.97.

To testify the simulated experiment can evaluate the performance of the VoD networks accurately, we run the buggy version alone with pause frequency followed by 4, 6, 8 for the first joined client. The pause times of later joined clients are orchestrated at the same rate.

TABLE IV. EXPERIMENT ACCURACY WITH ELASTIC TASK WINDOW

<table>
<thead>
<tr>
<th>( \epsilon_{\text{play}} ) Missed</th>
<th>Buggy</th>
<th>PCE</th>
<th>Buggy</th>
<th>PCE</th>
<th>Buggy</th>
<th>PCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Pause Times</td>
<td>5.58%</td>
<td>5.33%</td>
<td>5.43%</td>
<td>5.38%</td>
<td>5.64%</td>
<td>5.27%</td>
</tr>
<tr>
<td>6 Pause Times</td>
<td>5.37%</td>
<td>5.27%</td>
<td>5.43%</td>
<td>5.38%</td>
<td>5.64%</td>
<td>5.27%</td>
</tr>
<tr>
<td>8 Pause Times</td>
<td>5.37%</td>
<td>5.27%</td>
<td>5.43%</td>
<td>5.38%</td>
<td>5.64%</td>
<td>5.27%</td>
</tr>
</tbody>
</table>

TABLE IV shows the comparison of piece missing ratio between those collected when the buggy version of experiment is run alone and when our ShadowVoD is implemented. We can see that in all three situations, the experiment accuracy can always be guaranteed.

C. PCE design with drag emulation

In our evaluations, 50 clients are orchestrated to drag the progress bar 20 seconds forward every 50 seconds. The production will have 5-second drag delay before resuming moving again. Firstly, we run the buggy version of experiment alone. Pieces lost during the drag delay are omitted when computing the piece missing ratio. The result is shown in the second column of TABLE V.

Then we use our PCE design. We set the buffer time \( \Delta t \) as 5-second which is equal to the drag delay. This means once the drag behavior of a client is notified, the experiment and production will locally quit the testing and rejoin again after 5 seconds. Then the behavior of experiment in ShadowVoD is consistent with actual production when computing the piece missing ratio. The piece miss ratio at \( \epsilon_{\text{play}} \) is 5.56 % and it evaluate the performance of the VoD networks accurately with a negligible error.

VI. CONCLUSIONS

We implemented the PCE scheme in VoD networks to provide large-scale, realism performance evaluations. To create desirable testing client behavior scenarios of pause and drag for behavior emulation, a novel methodology of elastic window length of rCDN and a rejoin mechanism of certain streaming machines are achieved. We proposed a centralized mechanism to select suitable substitutes for early-departed clients. We demonstrated the benefits of ShadowVoD through extensive evaluations. The results showed that our ShadowVoD can always achieve experiment accuracy.

In the future, we will work on the further research of drag emulation especially situations where actual frequency is not equal to the simulated frequency of which we want to make evaluations.

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their valuable comments. This work is partially supported by “National Natural Science Foundation of China (No. 61202107, No.61100220, No. 61202303),” by “National High Technology Research and Development Program of China (863 Program No. 2014AA01A702),” by “Natural Science Foundation of Hubei Province (No. 2014CFB1007),” by “National Key Technology Research and Development Program of China (No. 2012BAH46F03),” and by the “Fundamental Research Funds for the Central Universities”.

REFERENCES