ShadowStream: Performance Evaluation as a Capability in Production Internet Live Streaming Networks

Chen Tian
Richard Alimi
Yang Richard Yang
David Zhang
Aug. 16, 2012
Live Streaming is a Major Internet App
Updated Xbox and now streaming Netflix has bad picture quality?

I updated my Xbox and downloaded the new Netflix app and it works and everything, but the picture quality is horrible on everything I stream. Before the update I would get full bars with HD, after the update I only get one bar and the picture quality is really fuzzy. Is there anything I can do to improve the picture quality or at least is anyone else having the same problem?

Flash Player continually freezes and crashes my computer
2011-12-13 下午1:47

I am posting a new discussion page at the request of staff. The details posted are at the request of staff.

This problem started immediately after I updated my flash player in October and it extends through multiple updates of Adobe Flash Player.

Opening certain web pages that have flash applications on them completely freezes the browser and then the computer within a second. Within 30 seconds, the entire computer physically crashes with a very brief blue error screen before the computer shuts itself off. Opening two web pages that use Adobe Flash Player is also a near guarantee to crash the computer. Note that this includes all advertisements that use Flash Player, which means that opening any two web pages at once mostly guarantees that Flash Player crashes the computer.

I have diligently updated everything to fix this problem that I can find, and nothing helped. Drivers, browsers, plug-ins, applications, everything. None of the posted solutions I have found on the discussion forums have been beneficial or relevant.

Lacking sufficient evaluation before release
Don’t We Already Have ...

- Testbeds
  - Emulab
  - PlanetLab
  - ...

- Testing Channels
  - Gradually rolling out

They are not enough!
We focus on hybrid live streaming systems: CDN + P2P
We focus on hybrid live streaming systems: CDN + P2P
### Testbed: Misleading Results at Small Scale

#### Piece Missing Ratio

<table>
<thead>
<tr>
<th></th>
<th>Small-Scale</th>
<th>Large-Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production Default</td>
<td>3.5%</td>
<td>0.7%</td>
</tr>
<tr>
<td>With Connection Limit</td>
<td>3.7%</td>
<td>64.8%</td>
</tr>
</tbody>
</table>

Live streaming performance can be highly non-linear.
### Testbed: Misleading Results due to Missing Features

<table>
<thead>
<tr>
<th></th>
<th>LAN Style (Same BW)</th>
<th>ADSL Style (Same BW)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Piece Missing Ratio</strong></td>
<td>1.5%</td>
<td>7.3%</td>
</tr>
<tr>
<td><strong># Timed-out Requests</strong></td>
<td>1404.25</td>
<td>2548.25</td>
</tr>
<tr>
<td><strong># Received Duplicate Packets</strong></td>
<td>0</td>
<td>633</td>
</tr>
<tr>
<td><strong># Received Outdated Packets</strong></td>
<td>5.65</td>
<td>154.20</td>
</tr>
</tbody>
</table>

Realistic features can have large performance impacts.
Testing Channel: Lacking QoE Protection
Testing Channel: Lacking Orchestration

What we want is ...  What we have is ...

![Chart showing a comparison between expected and provided values over time. The chart tracks the number of peers over time (in seconds). The expected values are represented by a green line, and the provided values are represented by a red line.]
ShadowStream Design Goal

Use production network for testing with

- Protection of real user QoE
- Transparent orchestration of testing conditions
Roadmap

Motivation

Protection Design

Orchestration Design

Evaluations

Conclusions and Future Work
Protection: Basic Scheme

Note: R denotes Repair, E denotes Experiment
Example Illustration: E Success

tasks

virtual playpoint

validation

success

failure

real viewer
Example Illustration: E Success

virtual playpoint

real viewer

tasks

E

91

validation

success

failure

R

Yale LANS
Example Illustration: E Success

Diagram showing a process with tasks leading to validation. If the validation is successful, it proceeds to the success label '91'. If validation fails, it goes to the failure label '91'. The process can also loop back from the real viewer.
Example Illustration: E Success
Example Illustration: E Fail

- **virtual playpoint**
- **tasks**
- **success**
- **validation**
- **failure**
- **real viewer**
Example Illustration: E Fail
Example Illustration: E Fail

tasks

E

validation

failure

R

91

real viewer

virtual playpoint

success
Choice 1: dedicated CDN resources (R=rCDN)

- **Benefit:** simple
- **Limitations**
  - requires resource reservation,
    - e.g., 100,000 clients x 1 Mbps = 100 Gbps
  - may not work well when there is network bottleneck
How to Repair?

- Choice 2: production machine (R=production)
  
  - Benefit 1: Larger resource pool
  
  - Benefit 2: Fine-tuned algorithms
  
  - Benefit 3: A unified approach to protection & orchestration (later)
R = Production: Resource Competition

Repair and Experiment compete on client upload bandwidth

Competition leads to underestimation on Experiment performance
R= Production: Misleading Result

\[ y = m(\theta) \]

\[ x + y = \theta_0 \]

repair demand

missing ratio

accurate result

misleading result

\[ \theta_L \]

\[ \theta^* \]

\[ \theta_R \]

\[ \theta_0 \]

\[ x = \theta \]
Putting Together: PCE

$\text{tasks}$

$\text{E}$

$\text{P}$

failure $5\%$  

success $95\%$

$100\%$
Putting Together: PCE

Diagram showing the flow of tasks with 100% starting at the top, splitting into failure (5%) and success (95%). Failure leads to C with 0%, and success leads to E. Further splitting with failure leading to P with a question mark.
Putting Together: PCE

tasks

E

validation

success

failure

R

P

virtual playpoint

real viewer

Yale LANS
Implementing PCE

Requirements

• Streaming machine transparent of testing state
• Streaming machines are isolated from each other
Implementing PCE: base observation

- A simple **partitioned sliding window** to partition downloading tasks among PCE automatically.
Implementing PCE: base observation

- A simple partitioned sliding window to partition downloading tasks among PCE automatically
Implementing PCE: base observation

- A simple **partitioned sliding window** to partition downloading tasks among PCE automatically.
Client Components

Streaming Hypervisor

Task Window Management
- API Calls
  - getMaxLag()
  - getSourceTime()

Data Distribution Control
- API Calls
  - writePiece()
  - checkPiece()
  - deliverPiece()

Network Resource Control
- API Calls
  - sendMessage()
  - recvMessage()
Roadmap

- Motivation
- Protection Design
- **Orchestration Design**
- Evaluations
- Conclusions and Future Work
Orchestration Challenges

- How to start an Experiment streaming machine
  - Transparent to real viewers
- How to control the arrival/departure of each Experiment machine in a scalable way
Viewer Enters Channel
Transparent Orchestration Idea

Experiment Enters Testing

real playpoint

virtual playpoint

R

E
Transparent Orchestration Idea

Experiment Leaves Testing

real playpoint

\[ R \quad E \]
Distributed Activation of Testing

- Orchestrator distributes parameters to clients
- Each client \textit{independently} generates its arrival time according to the same distribution function \( F(t) \)
- Together they achieve global arrival pattern
  - Cox and Lewis Theorem

\begin{verbatim}
Orchestrator:
01. Let \( M \) be the total number of available clients
02. Let \( p = \frac{\Lambda(t_{\text{exp}})}{M} \)
03. Send \( t_{\text{start}}, t_{\text{exp}}, \lambda(t), \) and \( p \) to each client

Client \( i \), upon receiving \( t_{\text{start}}, t_{\text{exp}}, \lambda(t), \) and \( p \):
04. \textbf{if } \text{random()} > p \text{ then return}
05. Draw waiting time \( w_i \) according to \( F(t) = \frac{\Lambda(t)}{\Lambda(t_{\text{exp}})} \)
06. Compute arrival time: \( a_{e,i} = t_{\text{start}} + w_i \)
\end{verbatim}
Orchestrator Components

- Specification and Triggering
  - Independent Arrival Control
  - Independent Departure Control
  - Experiment Transition
  - Replace Early Departed Clients
Motivation
Protection Design
Orchestration Design
Evaluations
Conclusions and Future Work
Software Implementation

- Compositional Runtime
  - Modular design, including scheduler, dynamic loading of blocks, etc.
  - 3400 lines of code
- Pre-packaged blocks
  - HTTP integration, UDP sockets and debugging
  - 500 lines of code
- Live streaming machine
  - 4200 lines of code
Experimental Opportunities

The graph shows the relationship between the number of clients (in thousands) and the maximum duration (in minutes). It compares two scenarios:

- **HN Satellite** represented by the red line.
- **SH Sports** represented by the green dotted line.

As the number of clients increases, the maximum duration decreases for both scenarios. The HN Satellite scenario shows a more significant decrease compared to SH Sports.
### Protection and Accuracy

#### Piece Missing Ratio

<table>
<thead>
<tr>
<th></th>
<th>Buggy</th>
<th>R=rCDN</th>
<th>R=rCDN w/ bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Playpoint</td>
<td>8.73%</td>
<td>8.72%</td>
<td>8.81%</td>
</tr>
<tr>
<td>Real Playpoint</td>
<td>N/A</td>
<td>0%</td>
<td>5.42%</td>
</tr>
</tbody>
</table>
## Protection and Accuracy

### Piece Missing Ratio

<table>
<thead>
<tr>
<th></th>
<th>PCE bottleneck</th>
<th>PCE w/ higher bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Playpoint</td>
<td>9.13%</td>
<td>8.85%</td>
</tr>
<tr>
<td>Real Playpoint</td>
<td>0.15%</td>
<td>0%</td>
</tr>
</tbody>
</table>
Orchestration: Distributed Activation
Utility on Top: Deterministic Replay

Control non-deterministic inputs

- Event
- Message
- Random seeds

Practical per-client log size

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 clients; 650 seconds</td>
<td>223KB</td>
</tr>
<tr>
<td>300 clients; 1,800 seconds</td>
<td>714KB</td>
</tr>
</tbody>
</table>
Motivation
Protection Design
Orchestration Design
Evaluations
Conclusions and Future Work
Contributions

- Design and implementation of a novel live streaming network that introduces performance evaluation as an intrinsic capability in production networks
  - Scalable (PCE) protection of QoE despite large-scale Experiment failures
  - Transparent orchestration for flexible testing
**Future Work**

- Large-scale deployment and evaluation

- Apply the Shadow (Experiment->Validation->Repair) scheme to other applications

- Extend the Shadow (Experiment->Validation->Repair) scheme
  - E.g., repair does not mean do the same job as Experiment, as long as it masks visible failures
## Adaptive Rate Streaming Repair

<table>
<thead>
<tr>
<th></th>
<th>Follow</th>
<th>Base</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>1.26x</td>
<td>1.26x</td>
<td>1.26x</td>
</tr>
<tr>
<td>Protected QoE</td>
<td>1.59x</td>
<td>1.42x</td>
<td>1.58x</td>
</tr>
<tr>
<td>Protection Overhead</td>
<td>1.49 Kbps</td>
<td>3.69 Kbps</td>
<td>1.39 Kbps</td>
</tr>
</tbody>
</table>
Thank you!
Questions?
backup