Live Streaming with Gossip

Maxime Monod

June 30, 2010

EPFL
École Polytechnique Fédérale de Lausanne
Live streaming

A source produces **multimedia content**

Regular TV: everything **HD**

- ESPNHD
- TNT
- tbs in HD
- COMCannon
- Comcast SportsNet
- HD TV
- USA HD
- SciFi HD
- WGN America
- Speed HD
- FX HD
- Animal Planet HD
- CNBC HD
- A&E HD
- History HD
- Discovery Channel
- Fox News Channel
- Lifetime HD
- Syfy HD
- TLC HD

**VS**

IP TV, Web TV, P2P TV, ...

- BBC
- iPlayer

192K requests/day
78K users/day
244K simultaneous users (incl. VoD)

BBC iStats (April 2010)
Live streaming

• Stream rate $s$ [kbps]
• $n$ viewers want to receive $s$

Demand = Supply
Natural solution

• “Centralized” solution

Participants are pure consumer
Context of this thesis

• “Decentralized” solution

Participants collaborate
...most of them!
Environment

- Large-scale

- Constrained bandwidth
  - Asymmetric (e.g., ADSL)
Existing approaches

**Single tree**

- $s_1$ is constrained by design
- Disconnection
- Build/maintain tree

**Multiple trees**

- Upload of nodes: multiple of $s_2/z$
- Partial disconnection
- Build/maintain $z$ trees

**Mesh**

- $s_3$ optimal
- Connected is not enough
- Peer selection, Packet scheduling
Beyond mesh: Gossip

• Gossip-based dissemination

Node picks fanout partners at random
Beyond mesh: Gossip

- Gossip-based dissemination
Beyond mesh: Gossip

- Gossip-based dissemination
Beyond mesh: Gossip

- Gossip-based dissemination
Beyond mesh: Gossip

• Gossip-based dissemination
Beyond mesh: Gossip

- Gossip-based dissemination
Beyond mesh: Gossip

- Gossip-based dissemination
Beyond mesh: Gossip

- Gossip-based dissemination
Beyond mesh: Gossip

• Gossip-based dissemination
Beyond mesh: Gossip

• Gossip-based dissemination
Beyond mesh: Gossip

- Gossip-based dissemination

Great for small updates (e.g., databases)
Duplicates are a problem for large content...
Gossip for live streaming

1. Gossip content location
   - Propose chunk ids
     - to fanout partners
     - every gossip period

1. Request (chunk ids)

2. Serve chunks (payload)

Fanout modulates contribution of nodes
Mesh vs Gossip

Peering degree $= |\text{view}| = 4$

BitTorrent default is 50 ($e^{50} = 5.18 \cdot 10^{21}$)

Gossip, $f = 2$
Gossip – Theory

1. \( \text{fanout} = \ln(n) + c \)

\[ P[\text{connected graph}] \text{ goes to } \exp(-\exp(-c)) \]

2. Holds as long as the fanout is \( \ln(n) + c \) on average
### Experimental Setup

<table>
<thead>
<tr>
<th></th>
<th><strong>Grid’5000</strong></th>
<th><strong>PlanetLab</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Environment</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>200 (40*5)</td>
<td>230-300</td>
</tr>
<tr>
<td>BW cap</td>
<td>Token bucket (200KB)</td>
<td>Throttling</td>
</tr>
<tr>
<td>Transport layer</td>
<td>UDP + losses (1-5%)</td>
<td>UDP</td>
</tr>
<tr>
<td>Stream rate $s$</td>
<td>680 kbps</td>
<td>551 kbps</td>
</tr>
<tr>
<td>FEC</td>
<td>5%</td>
<td>10%</td>
</tr>
<tr>
<td>Stream (incl. FEC)</td>
<td>714 kbps</td>
<td>600 kbps</td>
</tr>
<tr>
<td>$T_g$ (gossip period)</td>
<td>200 ms</td>
<td>200-500 ms</td>
</tr>
<tr>
<td>fanout ($f$)</td>
<td>8</td>
<td>7-8</td>
</tr>
<tr>
<td>source’s fanout</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Retransmission</td>
<td>ARQ/Claim</td>
<td>ARQ</td>
</tr>
<tr>
<td>Membership</td>
<td>RPS (Cyclon) and full membership</td>
<td></td>
</tr>
</tbody>
</table>

**Gossip**
Evaluation Metrics

• **Stream lag**
  
  – Time difference between creation at the source and delivery to the clients’ player

• **Stream quality**
  
  – Maximum 1% jitter means at least 99% of the groups are complete
    
    • Incomplete groups does not mean “blank”
Gossip – Practice

PlanetLab nodes have:
- Large bandwidths
- Small delays

PlanetLab (230)
\( s = 600 \text{ kbps} \)
\( f = 7 \)
Live Streaming with Gossip

- **Constrained environment**
  - Observations
  - Gossip++

- **Heterogeneous environment**
  - HEAP

- **Presence of freeriders**
  - LiFT
Stretching Gossip

How often should a node change its fanout partners?

The larger the better?

... in a constrained environment
Optimal fanout

Percentage of nodes viewing the stream with less than 1% jitter

**Increasing fanout**
- **Theory**
  - More robust
  - Faster dissemination
- **Practice**
  - Heavily requested nodes exceed their bandwidth

**Offline viewing**
- 20s lag
- 10s lag

PlanetLab (230)
700 kbps cap
$s = 600$ kbps
Optimal proactiveness

Different dissemination tree for each chunk:
- Ultimate way of splitting the stream

PlanetLab (230)
700 kbps cap
\( s = 600 \text{ kbps} \)
\( f = 7 \)
Contributions

Constrained environment
- Observations
- Gossip++

Heterogeneous environment
- HEAP

Presence of freeriders
- LiFT
Gossip++

Observations:
- Fanout has an optimal value/range
- Change partners every gossip period
  - Ultimate way of splitting the stream

How to receive a chunk that is not even proposed?

How to exploit the many duplicates?
Gossip++: Codec & Claim

Group G (k+c)

G contains k chunks

decode(G)

recode(G)

Inject reconstructed chunks

to player

serve(c_{i+4})
Gossip++: Codec & Claim
Gossip++ with freeriders

Nodes viewing a clear stream (10s stream lag)

Percentage of freeriders

Percentage of nodes (CDF)

Bandwidth usage of non-freeriding nodes

ARQ
Claim
ARQ (BW)
Claim (BW)

Grid'5000 (200)
1000 kbps cap
s = 714 kbps (5% FEC)
## Contributions

<table>
<thead>
<tr>
<th>Constrained environment</th>
<th>Observations</th>
<th>Gossip++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterogeneous environment</td>
<td>HEAP</td>
<td></td>
</tr>
<tr>
<td>Presence of freeriders</td>
<td>LiFT</td>
<td></td>
</tr>
</tbody>
</table>
Gossip is load-balancing...

- Proposals arrive randomly
  - Nodes pull from first proposal

- Highly-dynamic

Node q will serve $f$ nodes whp

Node q will serve $f$ nodes wlp
... but the world is heterogeneous!

3 classes (691kbps avg):
- 512kbps: 85%
- 3Mbps: 5%
- 1Mbps: 10%

Percentage of nodes receiving at least 99% of the stream:
- Standard gossip – 691 kbps
- Standard gossip – flat 691 kbps
- No cap

Stream lag (s)
How to cope with heterogeneity?

• **Goal:** contribute according to capability

• Propose more = serve more
  – Increase fanout...
    ... and decrease it too!

• Such that
  – average fanout \( f_{avg} \) ≥ initial fanout = \( \ln(n) + c \)
Heterogeneous Gossip - HEAP

- $q$ and $r$ with bandwidths $b_q > b_r$
  - $q$ should upload $\frac{b_q}{b_r}$ times as much as $r$

- Who should increase/decrease its contribution?
  ... and by how much?

- How to ensure reliability?
  - How to keep $f_{avg}$ constant?
• Total/average contribution is equal in both homogeneous and heterogeneous settings

\[ f_q = f_{\text{init}} \cdot \frac{b_q}{b_{\text{avg}}} \]

...ensuring the average fanout is constant and equal to \( f_{\text{init}} = \ln(n) + c \)
• Get $b_{avg}$ with (gossip) aggregation
  – Advertize own and freshest received capabilities
  – Aggregation follows change in the capabilities

• Get $n$ with (gossip) size estimation
  – Estimation follows change in the system
    • Join/leave
    • Crashes
    • …
Stream lag reduction

Percentage of nodes receiving at least 99% of the stream

No cap
Standard gossip – flat 691kbps
HEAP – 691kbps
Standard gossip – 691kbps
Quality improvement

- Stream lag of 10s

![Graph showing Jitter-free percentage of the stream for different bandwidths and gossip methods]
Stream lag

- For those who can have a jitter-free stream

Average stream lag to obtain a jitter-free stream

- Standard Gossip
- HEAP

512kbps
1Mbps
3Mbps
Proportional contribution

Average bandwidth usage by bandwidth class

- 512kbps
- 1Mbps
- 3Mbps

Standard Gossip
- 99.89% 91.56% 48.44%

Heap
- 94.38% 90.58% 87.56%
20% nodes crashing

Failure of 20% of the nodes at time $t=60s$

- HEAP - 12s lag
- Standard Gossip - 20s lag
- Standard Gossip - 30s lag
Contributions

Constrained environment
- Observations
- Gossip++

Heterogeneous environment
- HEAP

Presence of freeriders
- LiFT
Freeriders

• Selfish participants
  – Maximize benefit
  – Minimize contribution
  – Avoid to be detected
  – May collude

• Attacks on Tit-for-Tat protocols
  – Opportunistic unchoking
    • Can get without giving anything in return
      – (e.g., Large-view exploit, etc)
Freeriding Gossip

• Reduce fanout

• Propose less chunks than received

• Serve less chunks than requested

• Bias partner selection (colluders)
Managers of $p$

Check/update $p$’s score

- blacklist
- blame
- serve
- check actions
LiFT: direct verifications

• Direct check
  – Requested chunks are served

• Cross-checking
  – Served chunks are proposed
Attacking the direct verifications

• Colluder-in-the-middle
LiFT: \textit{a posteriori} verifications

- Statistical check
  - Partners chosen at random

- Verification on both fanin/fanout histories

\[
\begin{align*}
\text{History} & \\
p_1, p_3, p_5 & \\
p_0, p_4, p_7 & \\
p_2, p_3, p_5 & \\
\end{align*}
\]

\[
\begin{align*}
\text{Entropy, } & \chi^2, \\
\text{Kolmogorov-Smirnov} & \\
\end{align*}
\]
LiFT: scores

• Absolute scores
  – Need to be compensated
  – Message loss = wrongful blames
LiFT: evaluation

Cross-checking and blaming overhead

<table>
<thead>
<tr>
<th>$p_{cc}$</th>
<th>0</th>
<th>0.5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>674 kbps</td>
<td>1.07%</td>
<td>4.53%</td>
<td>8.01%</td>
</tr>
<tr>
<td>1082 kbps</td>
<td>0.69%</td>
<td>3.51%</td>
<td>5.04%</td>
</tr>
<tr>
<td>2036 kbps</td>
<td>0.38%</td>
<td>1.69%</td>
<td>2.76%</td>
</tr>
</tbody>
</table>
Summary of results

Live Streaming with Gossip? Yes

**Constrained environment**

**Observations**
- Optimal fanout value/range
- Optimal proactiveness

**Gossip++**
- Codec + Claim
- Tolerance to freeriders

**Heterogeneous environment**

**HEAP**
- Bandwidth measurement
- Fanout adaptation
- Preserved simplicity
- Preserved reliability
- Preserved Efficiency

**Presence of freeriders**

**LiFT**
- Lightweight
- Simple
- Secures asymmetric exchanges
Open problems

Gossip++
- Chunk priorities
- Dissemination tree prediction

HEAP
- Do more with $b_{avg}$
- Different adaptation
  - $p_{acc}$

LiFT
- Punishing vs revoking
- Heterogeneous environment
- Oracle & spies
Thank you

That's all Folks!