P2P streaming

Thanks to Y. Chen, R. Karki, M. Monod, M. Zhang, D. Frey for some of their slides
Application-level multicast for high-bandwidth content

Content streaming (today)
- Multiple Trees (Splitstream)
- Mesh
- Gossip
A source produces multimedia content

Regular TV: everything HD

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

VS

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

BBC iStats (April 2010)
Streaming Basics

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

Demand = Supply
Intuitive solution

- “Centralized” solution

Participants are pure consumer
Let’s be smarter

• “Decentralized” solution

Participants collaborate 
...most of them!
Evaluation Metrics

• Stream lag

  – Time difference between creation at the source and delivery to the clients’ player
  – Also:
    • delay penalty (delay wrt IP multicast)
    • hop count

• Stream quality

  – Maximum 1% jitter means at least 99% of the groups are complete = 99%-playback
    • Incomplete groups does not mean “blank”
  – Also: delivery-ratio or continuity index
Streaming 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/Gossip**
- $s_3$ optimal
- Connected is not enough
- Peer selection, Packet scheduling

Janvier 2013
SplitStream approach

Content divided in *stripes*
Each stripe is distributed on an independent tree

- Load balancing
  - Internal nodes in one tree are leaves in others
- Reliability
  - Failure of a node leads to unavailability of $x$ stripes if parents are independent and using appropriate coding protocols

[SOSP 2003 « *SplitStream: High-Bandwidth Multicast in Cooperative Environment* »]
**SplitStream**

- Construction of one tree/group per data stripe
- Each stripe identifier starts with a different digit (independence up to 16 stripes)
SplitStream

• Robust and efficient protocol for large-scale content streaming
  – Forest of independent trees / unique tree
  – Spare-capacity tree for maintenance
  – Decentralized and scalable management relying on Scribe and Pastry
  – Robust in dynamic environments
Existing approaches

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

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

$s_3$ optimal
Connected is not enough
Peer selection, Packet scheduling

Janvier 2013
DoNet

• Data driven protocol
  – Availability of data guides the flow directions, not a specific overlay structure

• Overlay with semi-static structure high dynamic nodes
  – Cope with dynamics to some extent

• Related to P2P on-demand streaming but different
  – Live media streaming with semi-synchronized nodes
Key Design Issues of DoNet

- how the partnerships are formed?
- how the data availability information are encoded and exchanged?
- how the video data are supplied and retrieved among partners?
A generic system diagram for a DONet node.
DoNet Roles

• Membership manager: maintain a partial view of other overlay nodes (SCAMP [GKL IEEE TC 2003])
• Partnership manager: establishes and maintains the partnership with other nodes
• Scheduler: schedules the transmission of video data
• a DONet node can be either a receiver or a supplier, or both, except the origin node
Membership Management

• Membership cache (mCache) contains a partial list of the unique identifiers for nodes

• Scalable Gossip Membership protocol (SCAMP): periodically distribute membership messages
  – Achieve a log(N) degree automatically
  – Indirection
    • New node → origin node → deputy node: redirect to obtain list; improve randomness
SCAlable Membership Protocol

- **Partial membership**: each peer has a partial knowledge of the system
- **Self-parameterized**: adequate parameters for global connectivity, $O(\log(n))$
- **Self-organizing, fully decentralized protocol**: the out-degree converges on average to $(c+1) \log(N)$
Join algorithm

Join request forwarded

$P = \frac{1}{\text{sizeof view}}$

$(1 - P)$

Parameter $c$

Join request to a random member

new

contact
Joining algorithm

Local view

Janvier 2013
Average case analysis

$D(n)$: average degree ($n$ peers).

One subscription adds $D(n)+1$ oriented edges

$$(n+1) \ D(n+1) = n \ D(n) + D(n)+1$$

The solution of the recurrence is

$$D(N) = D(1) + 1/2 + 1/3 + \ldots + 1/N$$

$$\approx \log(N)$$
Reliability

Proportion of nodes reached by the multicast

Percentage of node failures

0% 10% 20% 30% 40% 50% 60% 70%

0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Full membership
SCAMP
Resulting Overlay

log(n) degree
Buffer Map

• a video stream is divided into segments (1s video)
• BufferMap represents the segment availability (~120 segments bitmap)
• a sliding window of 120-segment
Buffer Map Representation and Exchange
Buffer Map Representation and Exchange

1 2 4
   ↓
1 2

2
   ↓
1 2 3
   ↓
2 3

3
   ↓
1 2 3
   ↓
2 3

4
   ↓
1 2

5
   ↓
2 3

I have 1,2, 4
I have 1,2, 3

I have 1,2
I have 2,3

root
Buffer Map Representation and Exchange
Scheduling

- Given a BufferMap and a list of partners
- Simple round-robin scheduler for homogenous and static network
- Given a BufferMap and a list of partners
- Two constraints:
  - playback deadline for each segment
  - heterogeneous streaming bandwidth from the partners

*Parallel machine scheduling, NP-hard!*
Scheduling Algorithm

- Compute the number of potential suppliers for each segment (partners containing their buffer)
- Determine the suppliers starting for segments with one supplier (to meet the constraint deadlines)
- Among multiple suppliers: pick partner with the highest bandwidth and enough available time is selected
- Real-time protocol
Failure Recovery and Partnership Refinement

- Departure can be detected after an idle time or BM exchange
- An affected node can quickly react through re-scheduling using the BM information of the remaining partners (the probability of concurrent departures is small)
- Each node periodically establishes new partnership
  - maintain a stable number of partners
  - each partner has a score: average number of packets received
  - reject the ones with the lowest score
  - $M = 4$ leads to 95% of nodes reached in 6 hops for a DONet of 500 nodes
Control overhead: stable setting
Continuity index, stable setting

Number of segments that arrive before or on playback time/ total number
Stream rate, stable setting
Continuity index, dynamic setting

![Bar chart showing continuity index for different ON/OFF periods and node counts. The x-axis represents ON/OFF Period T (s) with values 50, 100, 200, 400, and 800. The y-axis represents Continuity index ranging from 0.8 to 1.0. Different colors represent different node counts: 10 nodes (blue), 50 nodes (dark blue), 100 nodes (turquoise), 150 nodes (yellow), and 200 nodes (red).]
Control overhead, dynamic setting
Comparison with Tree-based Overlay

• Single Tree
• 3 children per node, except source with 4
  – Yields same degree as M=4
• Some children moved down one level until bw constraint satisfied
• Tree repair grafting nodes to upstream neighbor when parent fails
Comparison with Tree-based Overlay
Comparison with Tree-based Overlay

Huge impact of some nodes in a tree
Comparison: continuity index
A Practical DONet Implement

- Broadcast live sports programs (450 - 755Kbps RealVideo/Windows Media format)

**TABLE I**

**USER IP DISTRIBUTION OF COOLSTREAMING V.0.9.**

(Approximation)

<table>
<thead>
<tr>
<th>Time</th>
<th>Total</th>
<th>CN</th>
<th>HK</th>
<th>US</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 17</td>
<td>1500</td>
<td>300</td>
<td>400</td>
<td>250</td>
<td>50</td>
</tr>
<tr>
<td>June 22</td>
<td>2400</td>
<td>400</td>
<td>1000</td>
<td>900</td>
<td>100</td>
</tr>
<tr>
<td>June 25</td>
<td>3000</td>
<td>600</td>
<td>1300</td>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>June 27</td>
<td>4000</td>
<td>1000</td>
<td>1500</td>
<td>1400</td>
<td>100</td>
</tr>
</tbody>
</table>

Janvier 2013
Summary- DONet

- One of the first successful implementation of mesh-based streaming system
- Issues in a very dynamic environment
Existing approaches

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

**Multiple trees**
- $s_1$ is constrained by design
- 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

Janvier 2013
iGridMedia

• Pull-based protocols are effective
  – Select neighbors from unstructured overlay
  – Periodically notify neighbors of available packets
  – Neighboring nodes request packets

• Nearly optimal
  – Bandwidth utilization
  – Throughput

• Without intelligent scheduling and bw measurement: merge of P2P streaming techniques with a guaranteed delay
Tradeoff

- Control overhead
- Depends on how frequently the notifications are sent.
- Delay

Janvier 2013
Overlay Construction

• Contact rendezvous point
• Randomly find set of partners (15)
  – RPS can be used
• Build (static) random graph
Pull-Based Method

• Same as DoNet
guaranteed live streaming service. This is the first work to use P2P technology to support delay-sensitive applications. Meanwhile, there is a great demand for streaming services to support interactive applications using P2P technology. However, an unstructured multiple tree protocol to degrade the transmission delay and to better adapt to the peer dynamics. For overlay construction, to join a P2P streaming session, presenters need to upload their stream to the deployed dedicated servers. Presenters who want to cast their live show usually use DSL or cable to access the Internet and hence have low upload bandwidth. We assume this is because it is very difficult for the industry. We believe this is because it is very difficult to pass its deadline, it will be requested through the rescue mechanism.

### Architecture

![Architecture Diagram]

- **Channel 1**
  - Presenter/Publisher
  - Rescue Connection
  - Pushing
  - Viewers

- **Channel 2**
  - Presenter/Publisher
  - Rescue Connection
  - Pushing
  - Viewers

**Servers**

- **Rescue Connection**
- **Pushing**

To get the packet's playback deadline, all nodes synchronize with the server when they start to join any channel. Then each node randomly finds some other 15 nodes as its neighbors to keep connections with so that a rich-connected unstructured overlay is built. For the streaming packet loss, we evenly partition the stream into 16 sub streams. For any packet, if the sequence numbers are congruent to the same value modulo 16, the server always relays the most fresh packets to the peers and sends back the late packets requested from the peers. When a neighbor quits or upload bandwidth on each participating peer, we call it a 1-time-server-push. When a neighbor quits or upload bandwidth on each participating peer, we call it a 1-time-server-push.
Scheduling

• 1-time-server-push : the server pushes one packet to any peer
  – Peer having the most contribution is picked

• An absent packet about to pass the deadline: requested from the server through the rescue connection

• Playback deadline: synchronization with the server
Performance with increasing BW

![Graph showing performance with increasing BW](image)
Request Interval

![Graph showing the relationship between request interval and average delivery ratio for different capacity supply ratios.](image)
Request Window

![Graph showing the relationship between Request window size (sec) and Average Delivery Ratio for different request intervals. The graph includes lines for request intervals of 200ms, 600ms, 1000ms, 1400ms, and 2000ms, each represented by distinct markers and line styles.]
Delay – Stream Lag
iGridMedia

• Pull-push hybrid system
  – Pull-based protocol as bandwidth aware multicast routing
  – Push down packets along trees identified by pull-based protocol
• Maintain throughput performance with lower delay
Existing approaches

$s_1$ 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

Janvier 2013
Mesh vs Gossip

View: $\geq \text{fanout}$

Gossip, $f = 2$

Janvier 2013
Gossip and Live Streaming

• Gossip: load-balancing and robustness
• Proactive solution (as opposed to Mesh)
• Drawback:
  – Redundancy is an issue with high-bandwidth content
Testing Gossip for Live Streaming

<table>
<thead>
<tr>
<th>Environment</th>
<th>Grid’5000</th>
<th>PlanetLab</th>
</tr>
</thead>
<tbody>
<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>

Janvier 2013
Gossip – Theory

1. \( \text{fanout} = \ln(n) + c \)
   
   \( P[\text{connected graph}] \) goes to \( \exp(-\exp(-c)) \)

2. Holds as long as the fanout is \( \ln(n) + c \) on average
Gossip in Practice

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

Janvier 2013
Stretching Gossip

Fanout

Proactiveness

The larger the better?

How often should a node change its fanout partners?
Optimal proactiveness

PlanetLab (230)  
700 kbps cap  
s = 600 kbps  
\( f = 7 \)

<table>
<thead>
<tr>
<th>Percentage of nodes viewing the stream with less than 1% jitter</th>
<th>Change partner every X gossip period(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>offline viewing</td>
<td></td>
</tr>
<tr>
<td>20s lag</td>
<td></td>
</tr>
<tr>
<td>10s lag</td>
<td></td>
</tr>
</tbody>
</table>

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

Janvier 2013  
62
Gossip-based content dissemination

• n: nodes
• e: the content to disseminate (stream of packets)
• 3 phase Push-request-Push algorithm
  – [propose] to \( f \) nodes
  – [request] from 1 node
  – [serve] to the requester
Phase 1: propose

Periodically: Push ids (infect and die)

Gossip (event ids)
commPartners := selectNodes(getFanout())
for all p in commPartners do
    send(p)[Propose, event ids]
}
Phase 2: request events

upon receive[propose, eProposed] do
  wantedEvents := {}
  for all e.id in eProposed do
    if (e.id not in eRequested) then
      wantedEvents := wantedEvents U e.id
  eRequested := eRequested U wantedEvents
reply[request, wantedEvents]
Phase 3: push payload

upon receive[request, wantedEvents] do
    askedEvents := {}
    for all e.id in wantedEvents do
        if (e.id not in eRequested) then
            askedEvents := askedEvents U event(e.id)
        reply[serve, askedEvents]

upon receive[serve, events] do
    for all e in events do
        if (e not in eDelivered) then
            eToPropose := eToPropose U e.id
            eDelivered := eDelivered U event(e.id)
        reply[serve, askedEvents]
Streaming with gossip

• Proposals arrive randomly
  – Nodes pull from first proposal

• Highly-dynamic

Node q will serve $f$ nodes whp

Node q will serve $f$ nodes wlp
Works very well...

- In homogeneous or non constrained scenarios
- Not so well if the algorithm stabilizes and low capacity nodes saturate their bandwidth (and high capacity nodes are under-utilized)
The world is heterogeneous

3 classes (691kbps avg):

512kbps 85%
3Mbps 5%
1Mbps 10%

Percentage of nodes receiving at least 99% of the stream

Janvier 2013
How to cope with heterogeneity?

• **Goal**: contribute according to capability

• Propose more = serve more
  – Increase fanout...
    ... and decrease it too!
• $q$ 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 \( f_{\text{init}} = \ln(n) + c \)
HEAP

• 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

- Standard gossip – flat 691kbps
- HEAP – 691kbps
- Standard gossip – 691kbps
- No cap

Janvier 2013
Quality improvement

• Stream lag of 10s
Stream lag

- For those who can have a jitter-free stream

Average stream lag to obtain a jitter-free stream

<table>
<thead>
<tr>
<th>Speed (kbps)</th>
<th>Stream lag (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
</tbody>
</table>

Janvier 2013
Proportional contribution
Average bandwidth usage by bandwidth class

Janvier 2013
20% nodes crashing

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

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

Janvier 2013
Streaming 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/Gossip**
- $s_3$ optimal
- Connected is not enough
- Peer selection, Packet scheduling

Janvier 2013
Summary

• Multiple Trees
  – Effective but hard to split bw perfectly
• Mesh
  – Easier to build but efficiency – delay tradeoff
  – Packet scheduling can improve performance
• Gossip
  – Improves over mesh by making it dynamic
• Pull-Push
  – Use mesh to identify trees

Several solutions, but no unifying vision yet
Some references

- Meng Zhang; Qian Zhang; Lifeng Sun; Shiqiang Yang; "Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?", Selected Areas in Communications, IEEE Journal on , vol.25, no.9, pp.1678-1694, December 2007
- [ HEAP] Davide Frey; Rachid Guerraoui; Anne-Marie Kermarrec; Maxime Monod; Koldehoefer Boris; Mogensen Martin; Vivien Quéma. Heterogeneous Gossip. Middleware 2009, Dec 2009, Urbana-Champaign, IL, United States.
- Davide Frey; Rachid Guerraoui; Anne-Marie Kermarrec; Maxime Monod; Vivien Quéma. Stretching Gossip with Live Streaming. DSN 2009, Jun 2009, Estoril, Portugal.