Understanding the Power of Pull-based Streaming Protocol: Can We Do Better?

Meng ZHANG, *Qian ZHANG, Lifeng SUN* and Shiqiang YANG

In *IEEE Journal on Selected Areas in Communications, special issue on Advances in Peer-to-Peer Streaming Systems, Vol. 25, No. 8, December 2007*

Presented by Rabin Karki

Background

Evolution of Internet streaming technology

Naïve Unicast Approach

IP Multicast (1987)

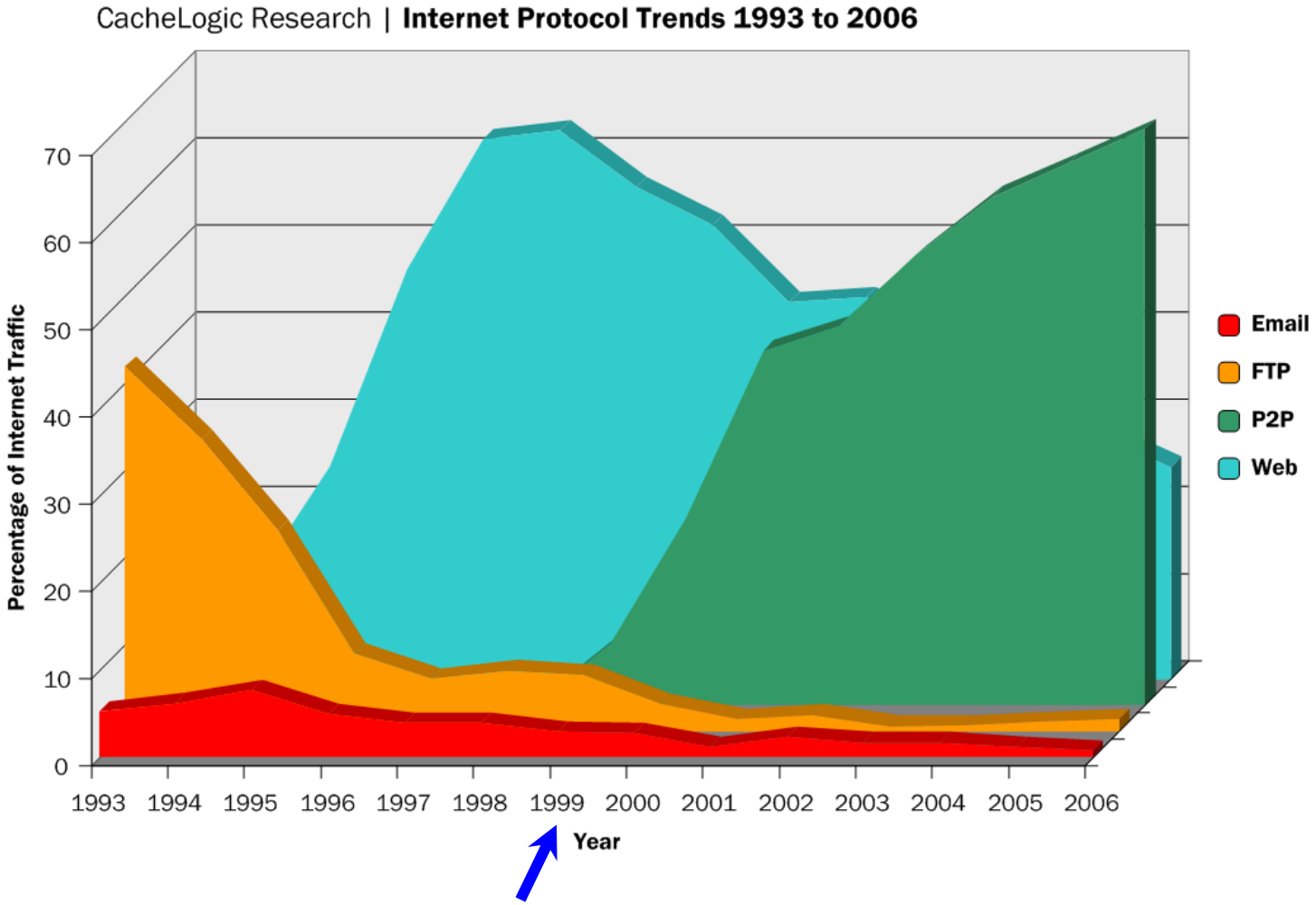
Content Distribution Networks (1998)

Application Layer Multicast (1999)

P2P Internet Video Streaming/Broadcast (2003)

Background

P2P Traffic really matters



1999: *Napster*, first widely used p2p-application

Background

P2P is more than just file download

- 1999: Napster
- 2000: Gnutella, eDonkey
- 2001: Kazaa
- 2002: eMule, BitTorrent
- 2003: Skype
- 2004: Coolstreaming, PPLive, GridMedia
- 2005: TVKoo, TVAnts, PPStream, SopCast…
- 2007: Joost, Babelgum, Xunlei kankan

Application Types:
File download, Telephony, Streaming, Gaming

Introduction

- Many P2P systems employ pull-based streaming protocol.
  - Each node independently selects neighbors to form unstructured overlay network.
  - Every node periodically notifies its neighbors about the packets it has.
  - Neighboring nodes request for packets using those notifications.
- Why pull-based?
  - Simple yet robust.

Introduction contd…

- Issue: maximizing throughput of the P2P overlay
- Traditional ‘tree-based’ protocol have poor utilization.
- Other works have been done but assume that the bandwidth capacity is known in advance.

Pull-based streaming

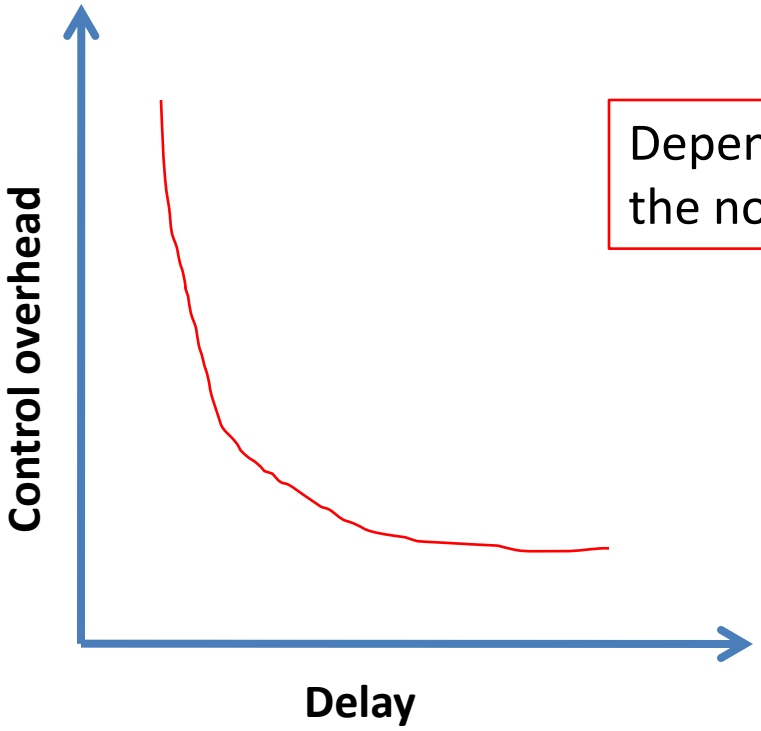
- A nice characteristic of pull-based protocol has not been paid enough attention
  - The simplest pull-based protocol is nearly optimal
  - in terms of bandwidth utilization and system throughput.
  - with appropriate protocol design and parameter settings.

Pull-based streaming

- A nice characteristic of pull-based protocol has not been paid enough attention
  - The simplest pull-based protocol is nearly optimal.
  - in terms of bandwidth utilization and system throughput.
  - with appropriate protocol design and parameter settings.
  - without any intelligent scheduling and proactive bandwidth measurement.

Pull-based streaming

- The near-optimality is achieved at the cost of tradeoff between control overhead and delay.



Depends on how frequently the notifications are sent.

- To break the tradeoff, authors propose pull-push hybrid system.
  - Considers pull-based protocol as highly efficient bandwidth-aware multicast routing approach.
  - And push down the packets along the tree formed by pull-based protocol.
- The hybrid system achieves near optimal throughput with lower delay, smaller overhead – with less server bandwidth.

Before we begin…

Notations

- τ = request interval at which the node asks its neighbors for packets.
- W = request window size, in secs.
- B = buffer size (>W, usually 1 min).
- T_w = waiting timeout ($rtt+\tau+t_w$).
- t_w = additional waiting time to prevent duplicate requests.
- r = packetized streaming rate (including IP, UDP, RTP headers) (309 kbps in the paper).
- R = raw streaming rate (300 kbps in the paper).
- l = packet size (1290 B, including 40 B header).
- u_i = upload capacity of sender i, in kbps.
- b_i = actual bandwidth consumed of sender i, in kbps.

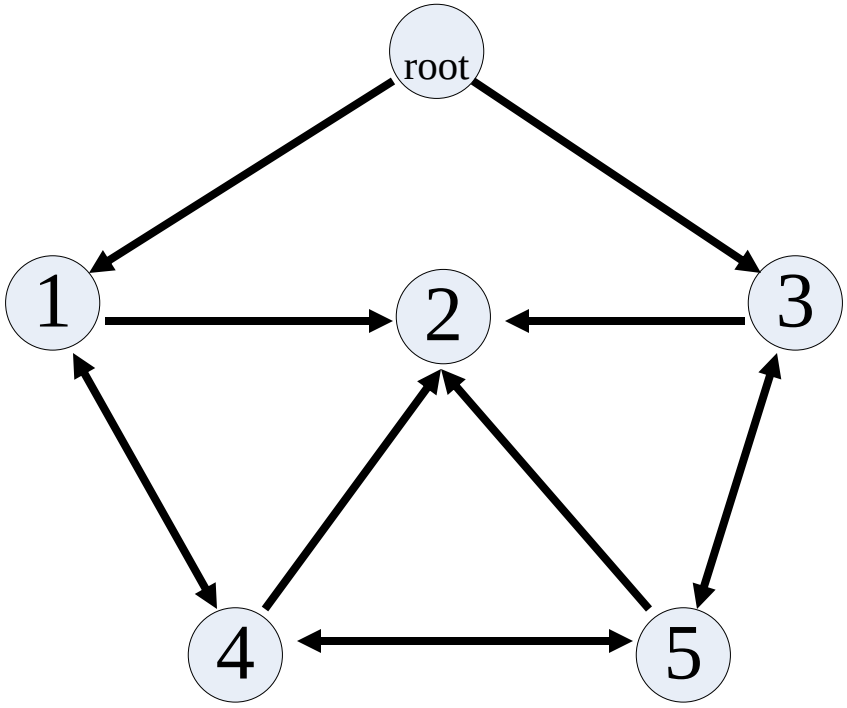
Pull-based method: Protocol

Overlay construction

- Nodes first contact Rendezvous Point (RP) to join a streaming session.
- Then each node randomly finds some other nodes to form an unstructured overlay network.

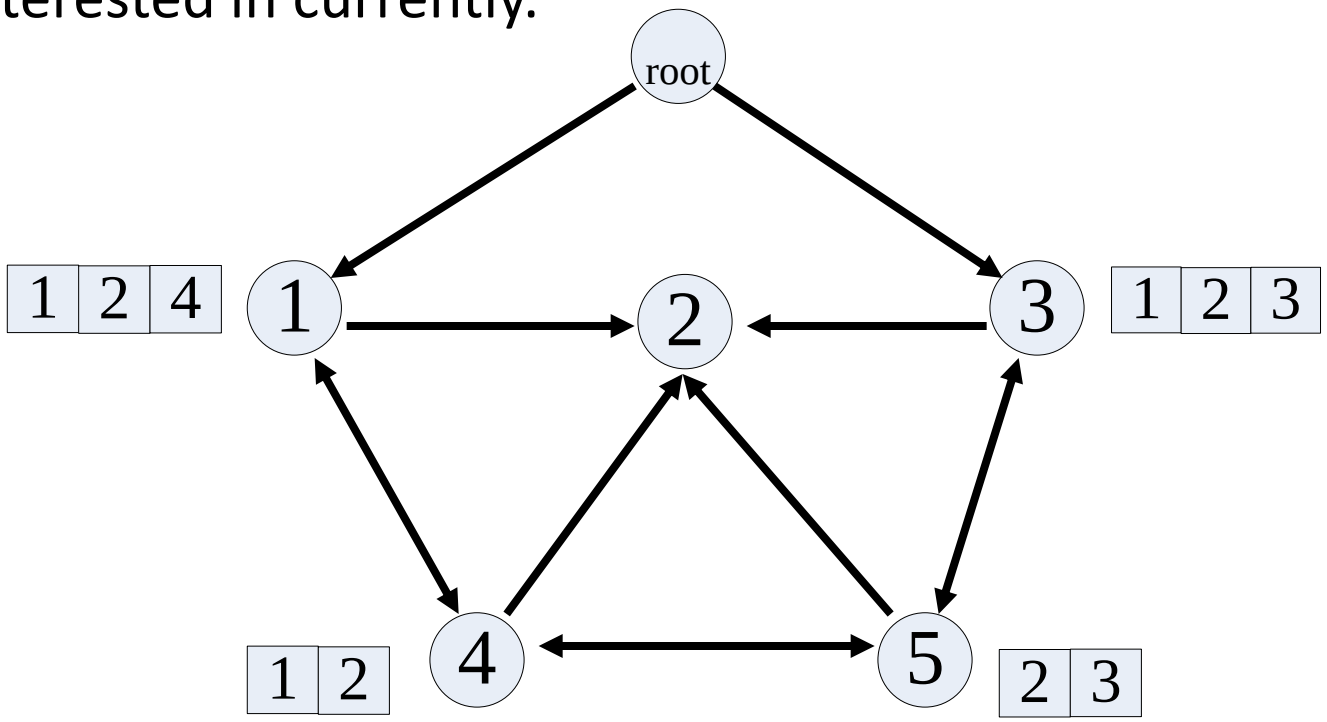
Pull-based method: Protocol

All the nodes self-organize into a random graph.



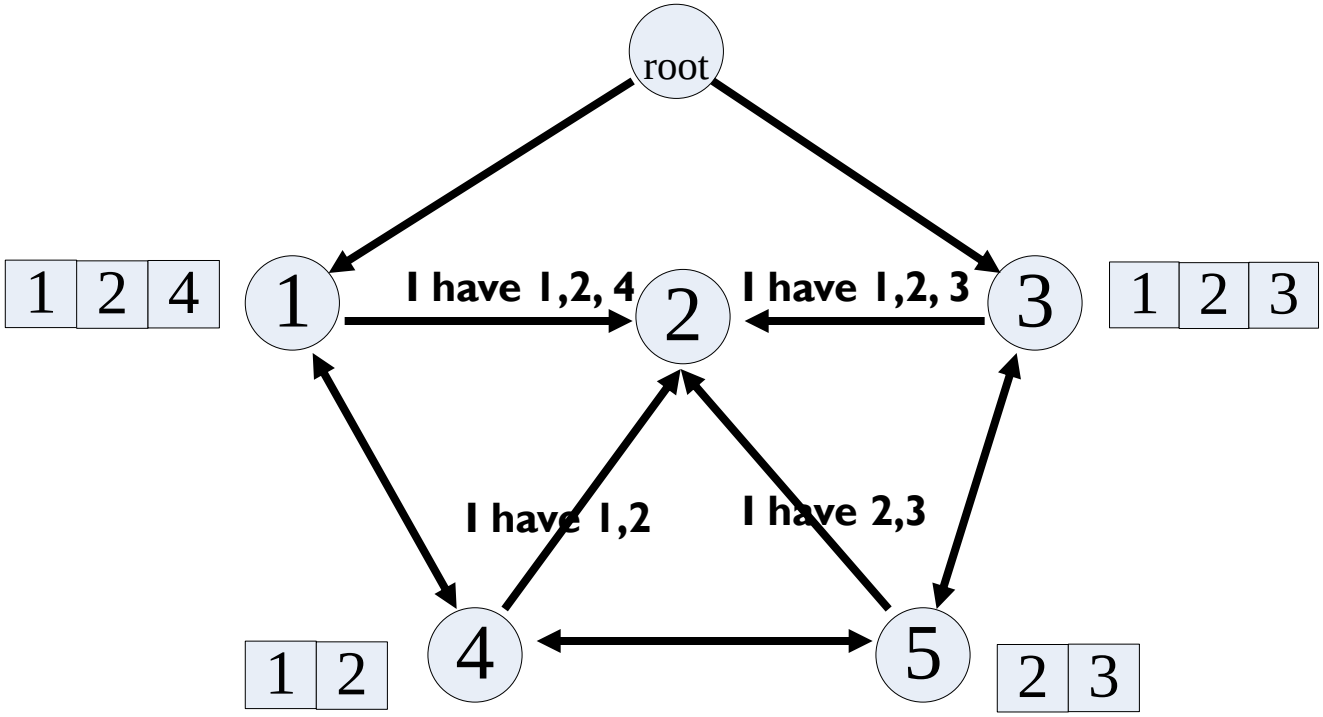
Pull-based method: Protocol

- Video stream is divided into fixed length packets called *streaming packets* marked by *sequence numbers*.
- Each node has a sliding window containing all the packets it is interested in currently.



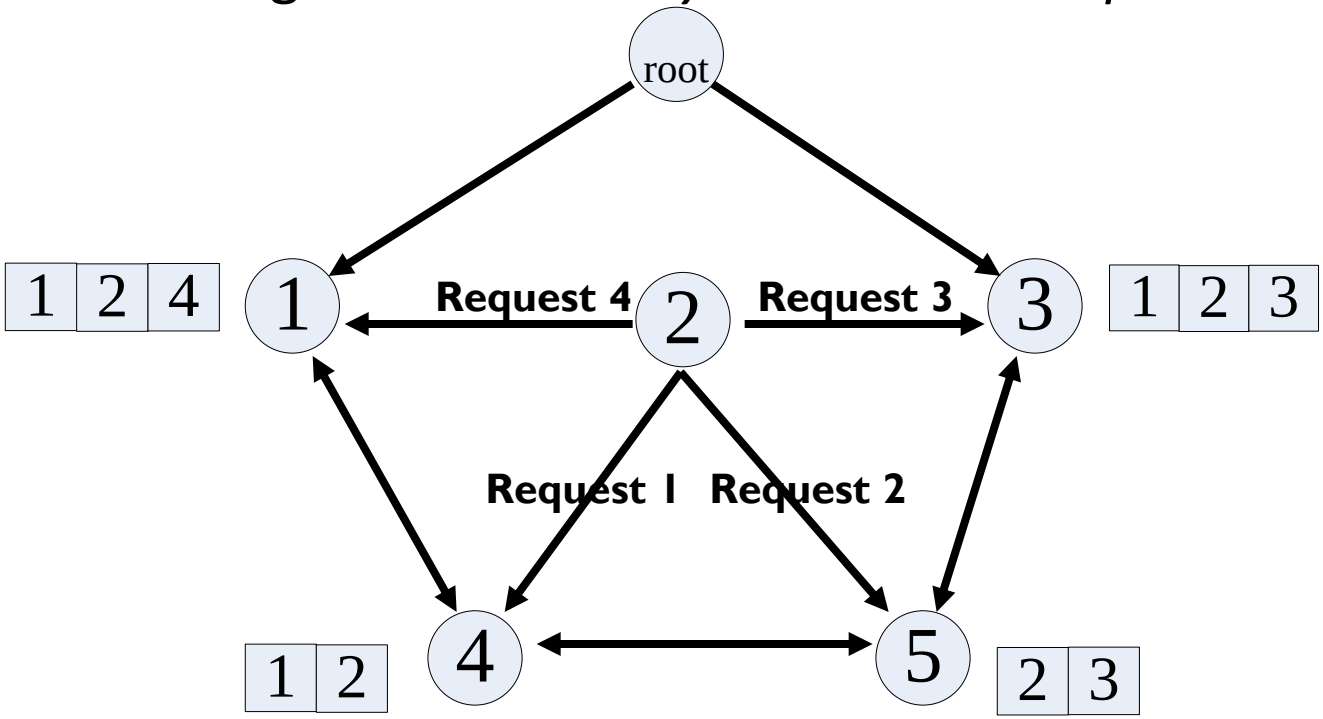
Pull-based method: Protocol

- Each node periodically sends *buffer map packets* to notify all its neighbors about the packets it has in its buffer.



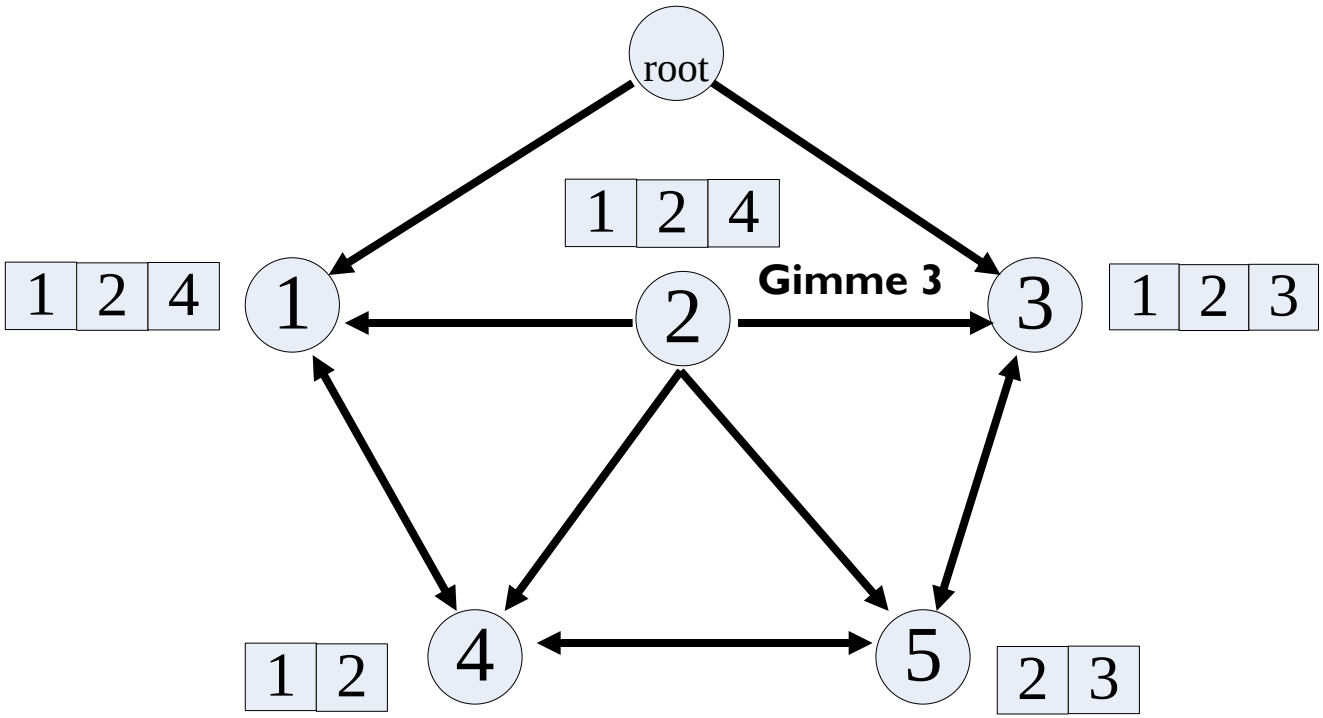
Pull-based method: Protocol

- Now the head of the *request window* of Node 2 becomes 4, and it asks for packets in its *request window* from its neighbors.
- If multiple nodes have the same packet, it will be requested from one of its neighbors *randomly with the same probability*.



Pull-based method: Protocol

- If the packet doesn't arrive within T_w of the request and it is still in the request window, it will be requested again.



Pull-based method: Protocol

Using UDP for *streaming packets* transmission

- Can use relatively small timeout to conclude if a packet is dropped.
- If TCP is used, packets spend a lot of time in sending buffer if sending rate is higher than the upload rate.
- The sender will send all the requested packets within τ at CBR.
- Dropped packets won't be retransmitted until a new request for the same packet is received.
- So, if a packet arrives, it will do so within $rtt + \tau$.
  - To account for jitter, additional t_w is added to T_w.

Pull-based method: Evaluation by simulation – Setup

- Simulators run on the simulator written by *Meng ZHANG* (first author).
- On two 4-CPU 8 GB machines.
- No queue management in routers.
- Default number of nodes = 10,000.
- Default neighbor count = 15.
- Default request window = 20 secs.
- Upload capacity of source node = 2Mbps.
- Nodes with 3 types of connections: upload/download – 1mbps/3mbps, 384kbps/1.5mbps and 128kbps/768kbps

Pull-based method: Evaluation by simulation – Metrics used

- *Capacity supply ratio* = $(\sum_{i=1}^{n} u_i) / R*(n-1)$

  (n = total number of nodes)

- So necessary condition to support P2P streaming system is *capacity supply ratio ≥ 1*.
- Maximum throughput is achieved when every node in a system of *capacity supply ratio = 1,* can get the entire stream.

Capacity Supply Ratio	Fractions			Capacity Supply Ratio	Fractions		
	1M	384k	128k		1M	384k	128k
1.02	0.1	0.346	0.554	1.2	0.15	0.39	0.46
1.05	0.1	0.38	0.52	1.25	0.16	0.405	0.435
1.1	0.1	0.45	0.45	1.3	0.18	0.4	0.42
1.15	0.1	0.5	0.4	1.4	0.2	0.45	0.35

Table I: Different capacity supply ratios by adjusting the fraction of different peer types

Pull-based method: Evaluation by simulation – Metrics used

- *Average deliverable rate and average delivery ratio: deliverable rate* of a node is the streaming rate received (excluding duplicate packets and control packets, including streaming packet headers). The packets have to arrive before playback deadline to be counted in. *delivery ratio* of a node is its *deliverable rate/packetized streaming rate* sent out from the source node.
- Coming up… the derivations of these two metrics.

Pull-based method: Evaluation by simulation – Metrics used

- Latest 10 second samples of a node (received by it and sent to it by the source) are called *sampled deliverable rate* and *sampled delivery* ratio, respectively.
- Average of those two, of all the nodes at a particular time, are called *average sampled* – *deliverable rate and delivery ratio* – *of that session,* respectively.
- When system reaches steady state, average of all the average sampled deliverable rate and delivery ratio of the session throughout the whole session are called *average deliverable rate* and average delivery ratio.
- Those two are the terms we set out to define in the first place and will be used later in evaluations.

Pull-based method: Evaluation by simulation – Metrics used

- *Average upload rate*: the *upload rate* of a node is the rate at which it successfully uploads packets(including *streaming and control packets*). The *average upload rate* of the whole session is calculated as before.

Pull-based method: Evaluation by simulation – Metrics used

- *Average upload rate*: the *upload rate* of a node is the rate at which it successfully uploads packets(including *streaming and control packets*). The *average upload rate* of the whole session is calculated as before.
- *Packet arrival delay*: difference between time at which it arrives at the node and at which it was sent out from the source.

Pull-based method: Evaluation by simulation – Metrics used

- *Average upload rate*: the *upload rate* of a node is the rate at which it successfully uploads packets(including *streaming and control packets*). The *average upload rate* of the whole session is calculated as before.
- *Packet arrival delay*: difference between time at which it arrives at the node and at which it was sent out from the source.
- *Control packet rate*: rate at which *control packets* (buffer map, request, member table, connection setup, heart-beat packets etc.) are sent.

Pull-based method: Evaluation by simulation – Metrics used

- *α-playback delay*: minimum buffered time when *delivery ratio* reaches α is *α-playback time*. *α-playback delay* is the delay between time at which the packet is sent out from the server and α-playback time of that packet. (α = 0.99)
- *Packet arrival delay*: difference between time at which it arrives at the node and at which it was sent out from the source.
- *Control packet rate*: rate at which *control packets* (buffer map, request, member table, connection setup, heart-beat packets etc.) are sent.

Pull-based method: Evaluation by simulation – Simulation results

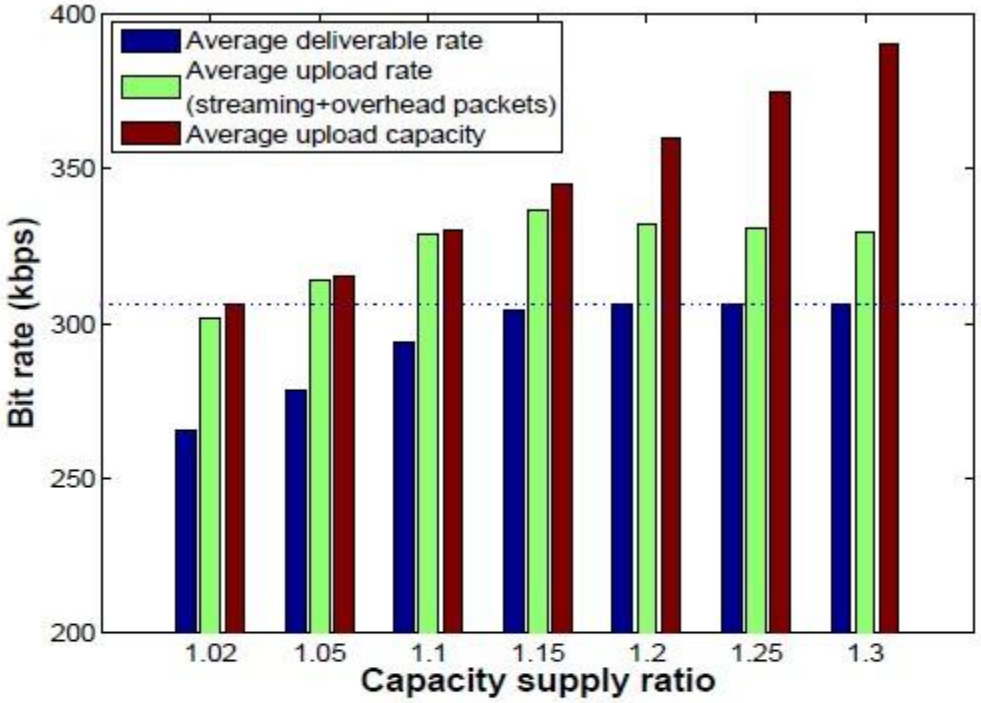


Fig. 1

AVERAGE DELIVERABLE RATE, AVERAGE UPLOAD RATE AND AVERAGE UPLOAD CAPACITY IN PULL-BASED PROTOCOL

Packetized streaming rate

When capacity supply ratio ≥ 1.15, average deliverable rate reaches nearly best deliverable rate.

When capacity supply ratio < 1.15, average upload rate is very close to the average upload capacity, which means the capacity is almost fully utilized.

Pull-based method: Evaluation by simulation – Simulation results

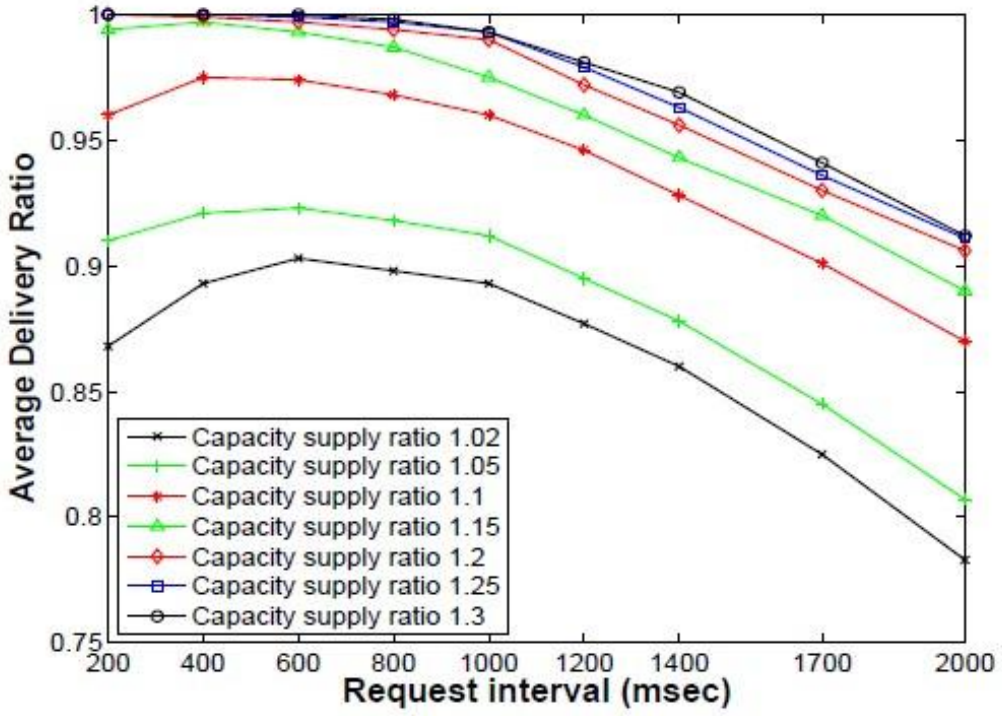


Fig. 2

AVERAGE DELIVERY RATIO WITH RESPECT TO DIFFERENT REQUEST INTERVALS AND CAPACITY SUPPLY RATIOS IN PULL-BASED PROTOCOL

When capacity supply ratio ≥ 1.15 and request interval = [200, 800], average delivery ratio is close to 1. And smaller request interval gives the better delivery ratio. (*because we can afford to irritate source nodes by pinging as often as we can*)

When capacity supply ratio < 1.15, delivery ratio gets worse when the request interval is very small (<400 ms)

Pull-based method: Evaluation by simulation – Simulation results

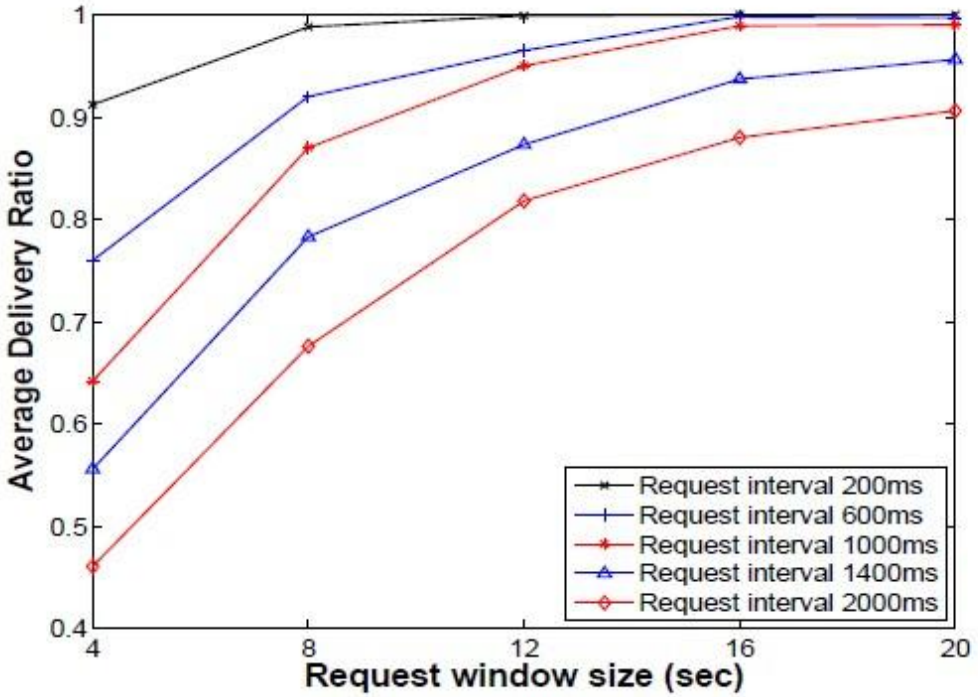


Fig. 3

DELIVERY RATIO WITH RESPECT TO DIFFERENT REQUEST INTERVAL AND REQUEST WINDOW SIZE IN PULL-BASED PROTOCOL

Delivery ratio ≈ 1, when request window size reaches 20 sec and request interval < 1 sec.

For the same window size, smaller request interval gives better delivery ratio .

Capacity supply ratio is fixed at 1.2

Pull-based method: Evaluation by simulation – Simulation results

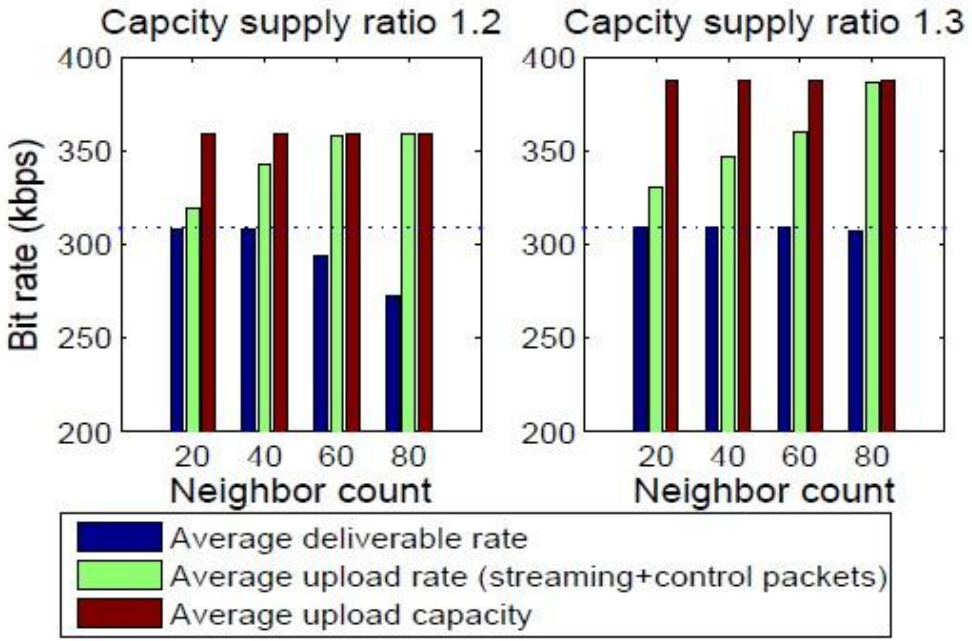


Fig. 4

DELIVERY RATIO WITH RESPECT TO NEIGHBOR COUNT AND CAPACITY SUPPLY RATIO. CAPACITY SUPPLY RATIO IS 1.2 AND REQUEST INTERVAL IS 500MS.

When capacity supply ratio = 1.2 and neighbor count exceeds 60, average deliverable rate drops below packetized streaming rate, but the upload capacity is almost fully utilized.

But when capacity supply ratio = 1.3, deliverable rate remains optimal even when neighbor count reaches 80.

Pull-based method: Evaluation by simulation – Simulation results

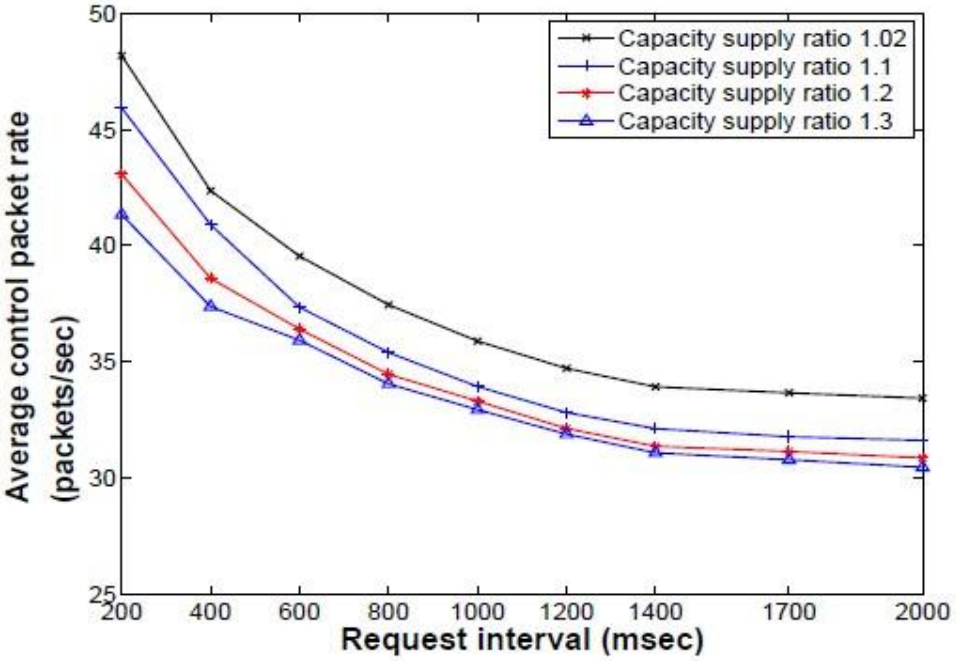


Fig. 5

AVERAGE CONTROL PACKET RATE WITH RESPECT TO DIFFERENT REQUEST INTERVALS AND CAPACITY SUPPLY RATIOS IN PULL-BASED PROTOCOL

This graph is self-explanatory.

Pull-based method: Evaluation by simulation – Simulation results

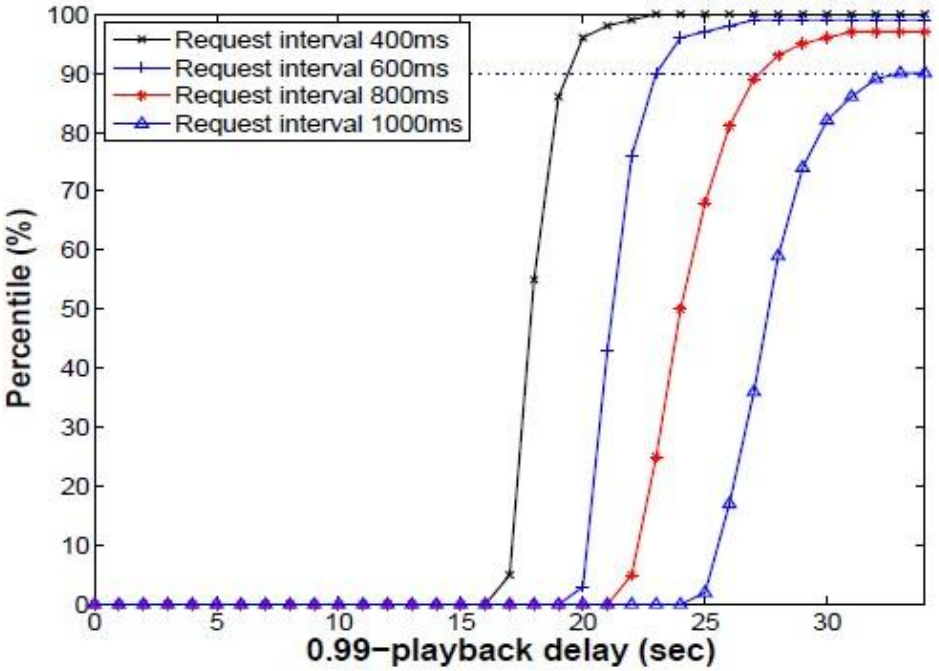


Fig. 6

CDF OF 0.99-PLAYBACK DELAY AMONG ALL PEERS UNDER DIFFERENT REQUEST INTERVAL (CAPACITY RATIO 1.2) IN PULL-BASED PROTOCOL

Almost all the peers have playback delay over 16 secs. Smaller request intervals seem to decrease the delay, but they produce more control packets, as we saw before.

Capacity supply ratio is fixed at 1.2

Pull-based method: Evaluation by simulation – Simulation results

- All the simulations so far were done on static environment (i.e. users do not join or leave during the session).
- When using traces from dynamic environment (real-deployed P2P streaming system: *GridMedia*), authors found results were pretty similar to what we saw in the simulations.

Pull-based method: Evaluation on PlanetLab

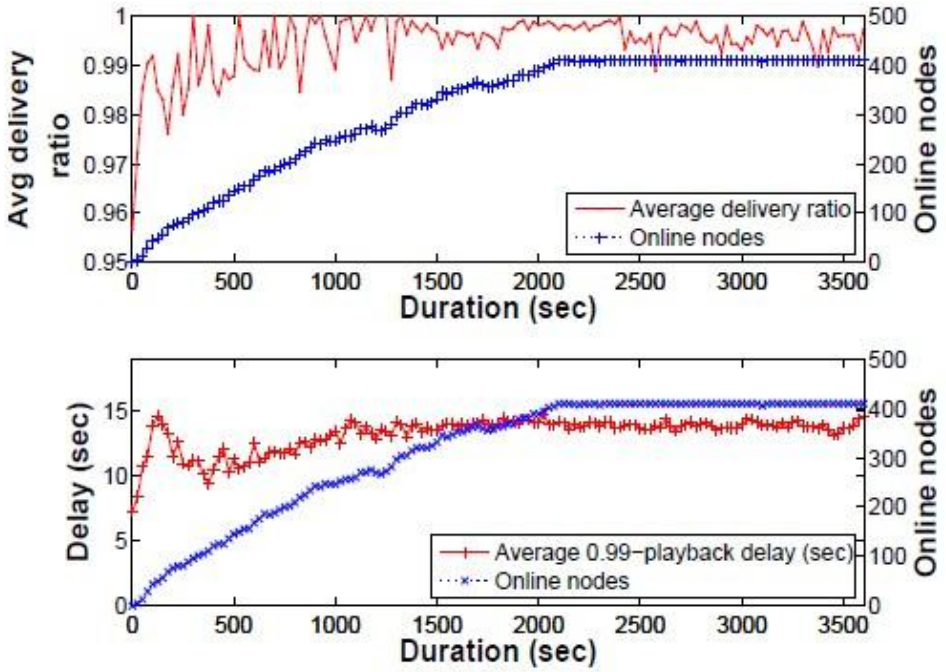


Fig. 9

PLANETLAB EXPERIMENT WITH 409 NODES. AVERAGE DELIVERY RATIO AND AVERAGE PLAYBACK DELAY DRIVEN BY REAL P2P STREAMING SYSTEM USER TRACE

- **Number of nodes = 409.**
- **Each node can have up to 15 neighbors.**
- **Request interval = 500 ms.**
- **Source node upload capacity = 2Mbps.**

- **Delivery ratio is above 0.99 most of the time.**
- **Playback delay is around 13 secs (*smaller than that in simulation*).**

Capacity supply ratio is fixed at 1.2

Pull-push hybrid method

- We saw that pull-based protocol is nearly optimal in terms of capacity utilization.
- Push-pull hybrid method tries to improve the pull-based protocol by pushing the streaming packets down the tree formed by the pull technique.

Pull-push hybrid method: Protocol

- Overlay construction is done as before.

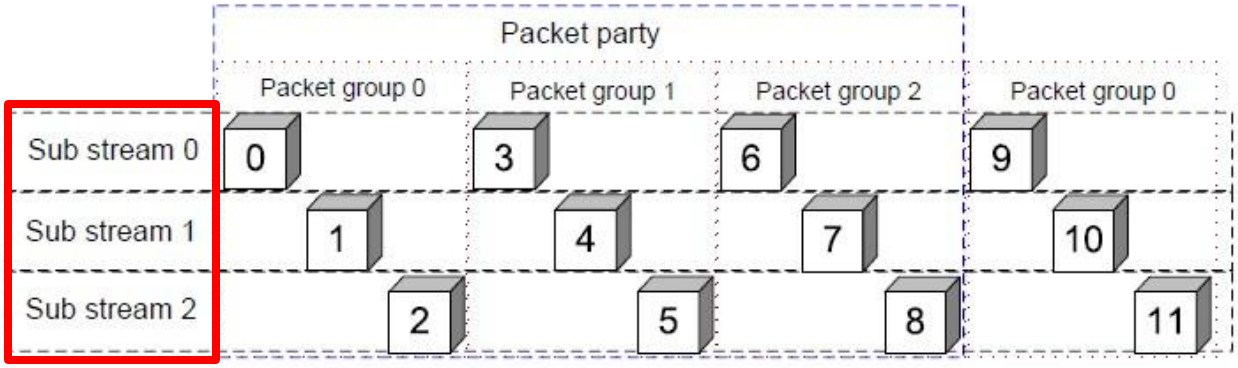


Fig. 18

AN EXAMPLE THAT HAS 3 SUB STREAMS. EVERY PACKET GROUP HAS 3 PACKETS, AND 3 PACKET GROUPS MAKE UP A PACKET PARTY

1. Partition stream evenly into *n* sub streams.

Pull-push hybrid method: Protocol

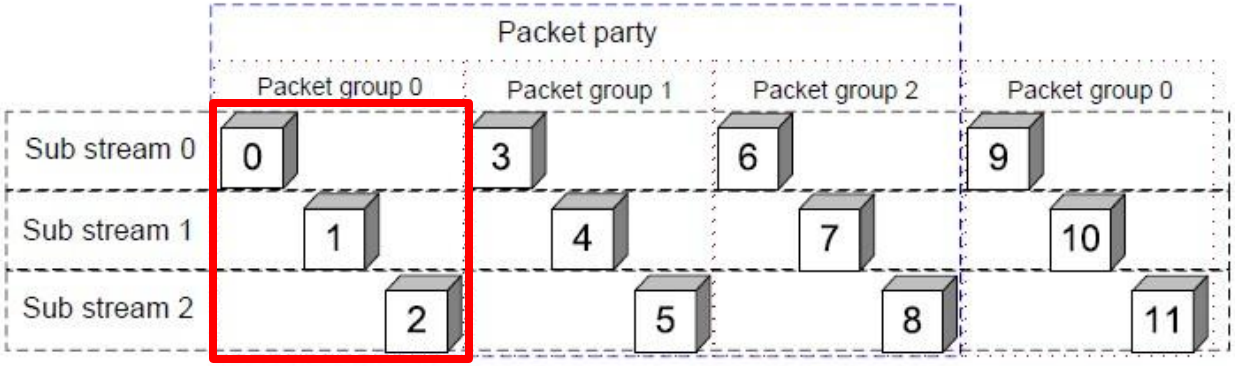


Fig. 18

AN EXAMPLE THAT HAS 3 SUB STREAMS. EVERY PACKET GROUP HAS 3 PACKETS, AND 3 PACKET GROUPS MAKE UP A PACKET PARTY

1. Partition stream evenly into *n* sub streams.

2. Group every continuous *n* packets into a *packet group.*

Pull-push hybrid method: Protocol

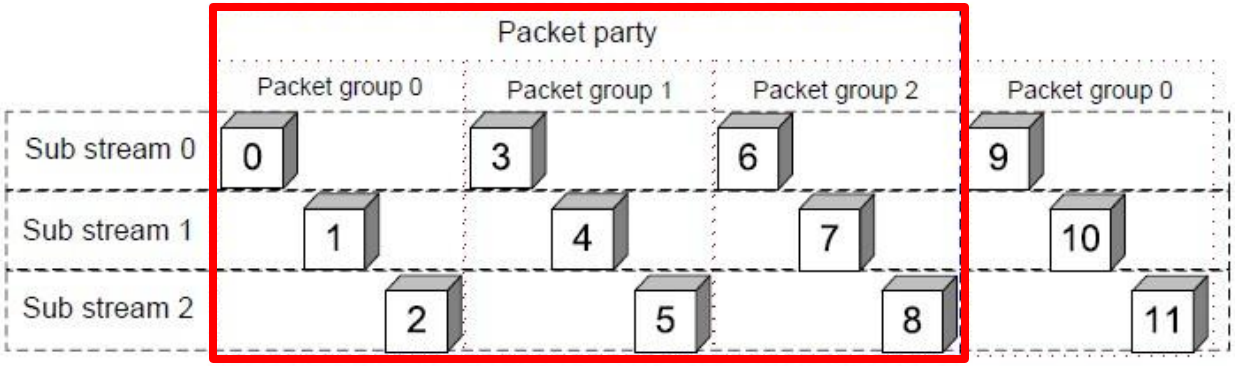


Fig. 18

AN EXAMPLE THAT HAS 3 SUB STREAMS. EVERY PACKET GROUP HAS 3 PACKETS, AND 3 PACKET GROUPS MAKE UP A PACKET PARTY

1. Partition stream evenly into *n* sub streams.
2. Group every continuous *n* packets into a *packet group.*

3. Cluster every continuous *g* packet groups into a *packet party.*

Pull-push hybrid method: Protocol

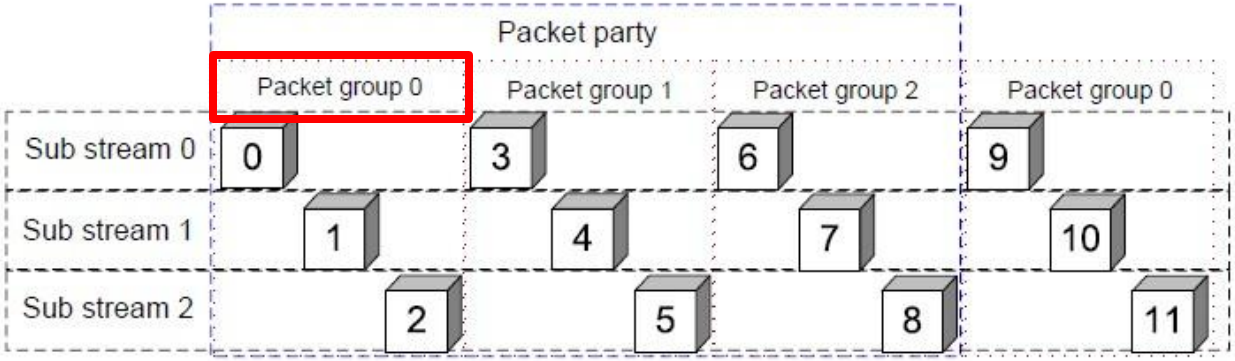


Fig. 18

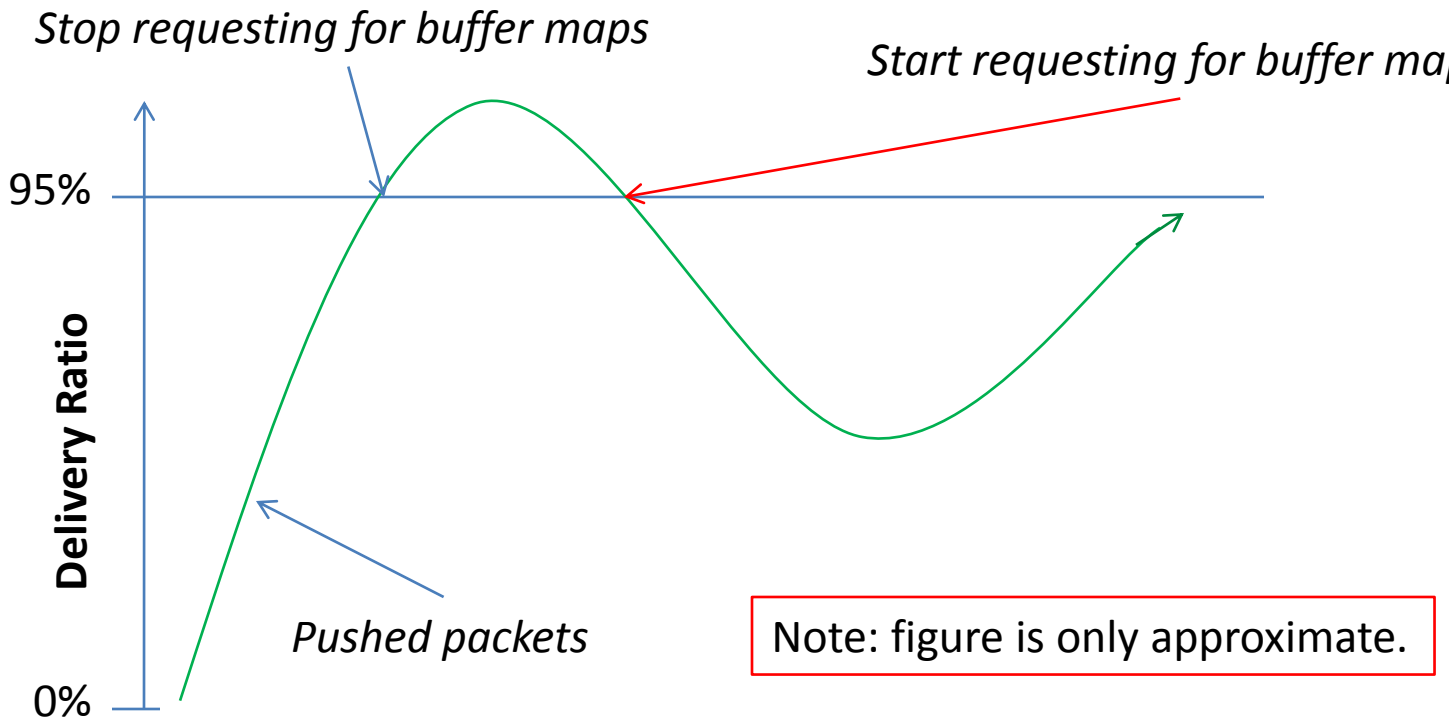
AN EXAMPLE THAT HAS 3 SUB STREAMS. EVERY PACKET GROUP HAS 3 PACKETS, AND 3 PACKET GROUPS MAKE UP A PACKET PARTY

1. Partition stream evenly into *n* sub streams.
2. Group every continuous *n* packets into a *packet group.*
3. Cluster every continuous *g* packet groups into a *packet party.*
4. Each packet group in a party is numbered from 0 to *g-1,* so *packet group no = floor(s/n) mod g.*

Pull-push hybrid method: Protocol

- When a peer joins, it asks neighbors to send the *buffer maps* periodically (unlike pull-based method, *buffer maps* are requested explicitly).
- Then it pulls the required packet according to the *buffer maps*.
- Once a packet in *packet group 0 of one packet party* is requested successfully, peer will send a *sub stream subscription* to let it push the rest of the packets in the same sub stream.

Pull-push hybrid method: Protocol



- When over 95% packets are pushed, the node will stop requesting for buffer maps.
- When delivery ratio drops below 95%, start requesting again.
- Pushed but lost packets are “pulled” after a timeout.

Pull-push hybrid method : Evaluation by simulation – Metrics

Following additional metrics are used in pull push hybrid method evaluation:

- *Redundancy and redundancy packet rate*: *Redundancy* = duplicate streaming packets/total traffic. *Redundancy packet rate* is rate of duplicated streaming packets.
- *Push fraction*: pushed streaming packets/total streaming packets

Pull-push hybrid method : Evaluation by simulation – Results

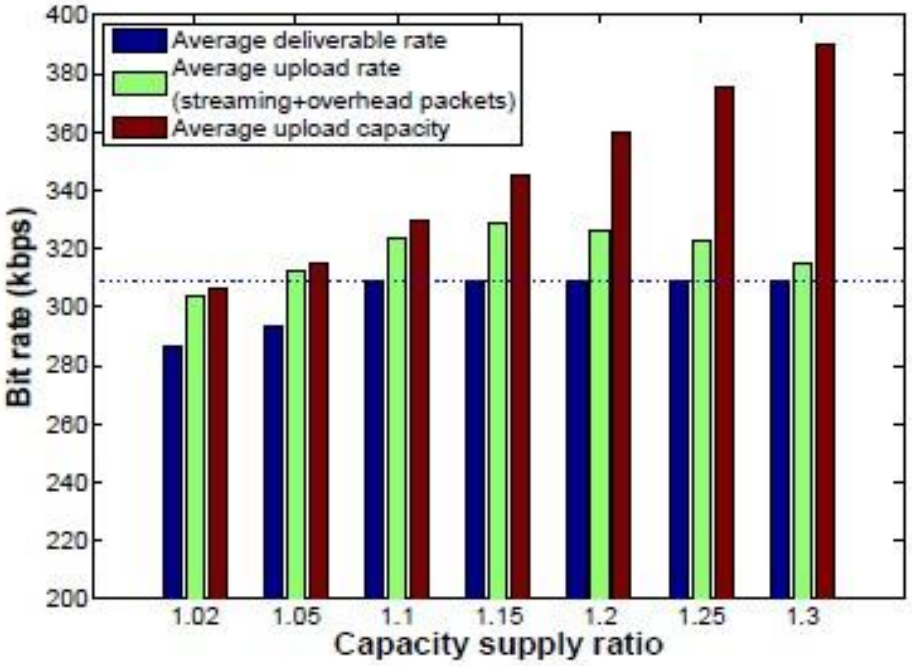


Fig. 19

AVERAGE DELIVERY RATIO WITH RESPECT TO DIFFERENT REQUEST INTERVALS AND CAPACITY SUPPLY RATIOS IN PULL-PUSH HYBRID PROTOCOL

Pull-push is also nearly optimal in terms of bandwidth utilization. Deliverable rate reaches optimum at capacity supply ratio of 1.10, *which was 1.15 in pull-based method.*

Pull-push hybrid method : Evaluation by simulation – Results

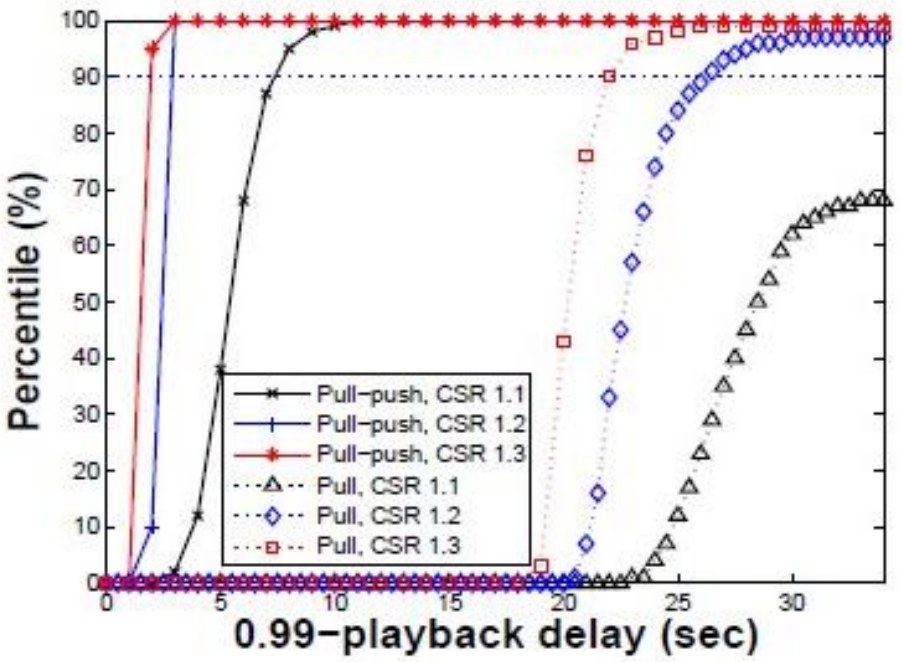


Fig. 20

CDF OF 0.99-PLAYBACK DELAY AMONG ALL PEERS IN PULL-PUSH HYBRID PROTOCOL AND PULL-BASED PROTOCOL

Playback delays are considerably smaller in push-pull method.

Pull-push hybrid method : Evaluation by simulation – Results

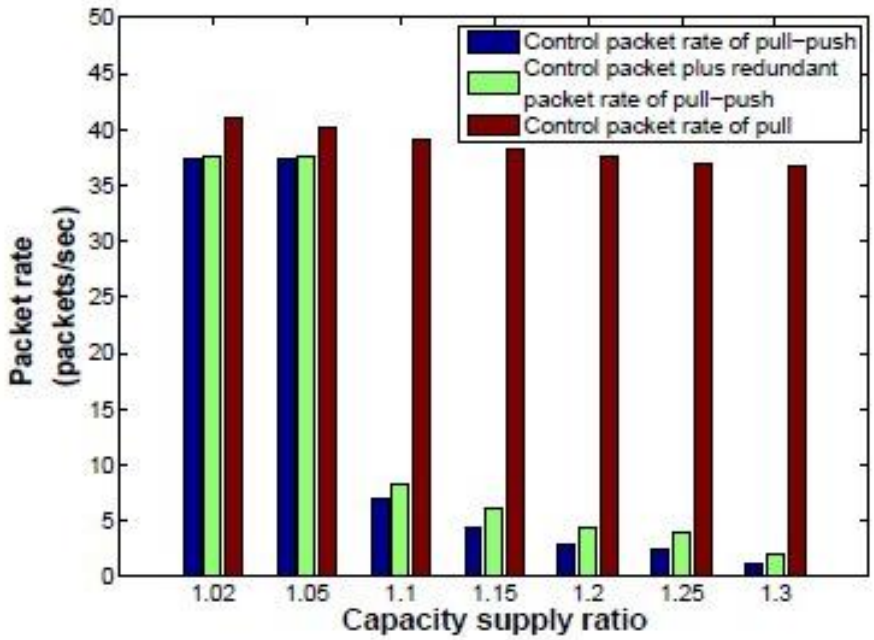


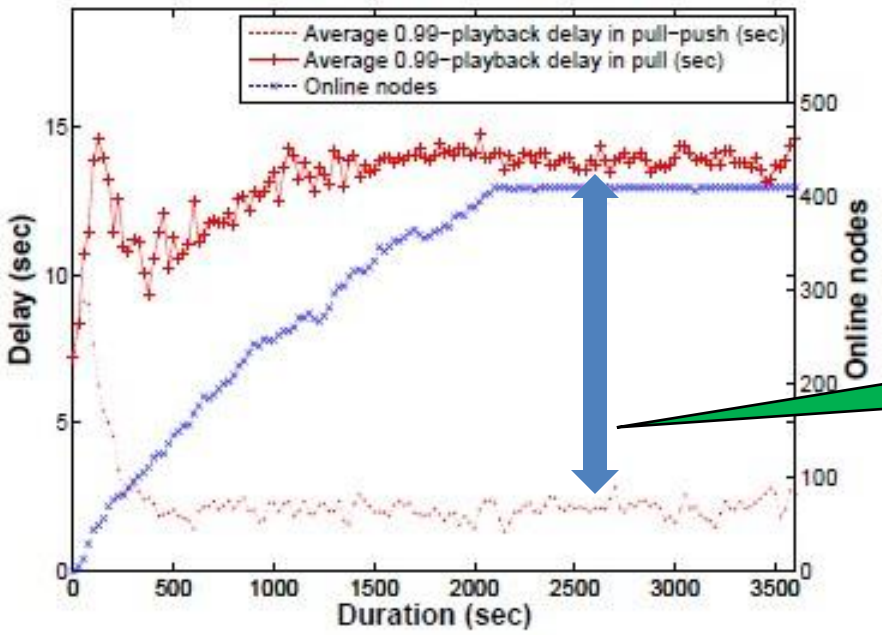
Fig. 21

AVERAGE CONTROL PACKET RATE COMPARISON BETWEEN PULL-BASED AND PULL-PUSH HYBRID PROTOCOL

The overhead of push-pull hybrid method is much smaller than that of pull-based method.

Push-pull hybrid method: Evaluation on PlanetLab

- Configuration is the same as before.



Significant delay reduction!

Fig. 28

PLANETLAB EXPERIMENT WITH 409 NODES. COMPARISON OF AVERAGE PLAYBACK DELAY AND PACKET ARRIVAL DELAY

Push-pull hybrid method: Deployment

- Pull-push hybrid deployed in GridMedia, adopted by CCTV to live broadcast since Jan 2005.
- Supported up to 224,453 concurrent users, which is ~270x more users than client-server based system. (Sounds good!)

Limitations

- Both the protocols are not contribution-aware.
- Generate heavy core-ISP and cross-ISP traffic due to random peer selection.
- Server upload bandwidth required is not low enough for a typical home user to broadcast a video.
- What happens when a node is behind Firewall/NAT?

Personal opinions

+ Detailed experimentations and result sets.

- Poor grammar (*examples: …how to only use…, …when it is finally arrived…, …protocols has…, …it may joins again. etc.)*

Acknowledgements

- Some figures and slides taken from M. Zhang's PhD defense slides.
- Some images taken from Google/Bing searches.

Thank you!