An Overview of Internet Multimedia Networking (Part III)

Prof. Reza Rejaie

Computer & Information Science

University of Oregon

http://www.cs.uoregon.edu/~reza

Winter 2003

Multimedia Proxy Services

- Prefix Caching
- Loss Recovery
- Smoothing & Video staging

Prefix Caching

- Caching only the initial part of each stream at a proxy
- Goal: to reduce startup latency
- Assumption: if server-proxy BW is known a priori, optimal length of prefix can be calculated

Proxy-assisted Loss Recovery

- Goal: to reduce overhead of per-client retx
- When loss occurs between server and proxy
  - Server only retx lost packets to proxy
  - proxy retx losses to all clients
- When loss occurs between proxy and clients
  - Proxy can easily retx to interested clients
- Very useful specially in multicast distr.
  - A client with a higher bw can be in charge of retx to a group nearby clients

Smoothing & Video Staging

- Smoothing: use proxy as a big buffer to smooth out variations of either stream or network bandwidth
- Video Staging: if available server-proxy bw is known, proxy can cache higher bw portion of a video that can not be streamed from the server

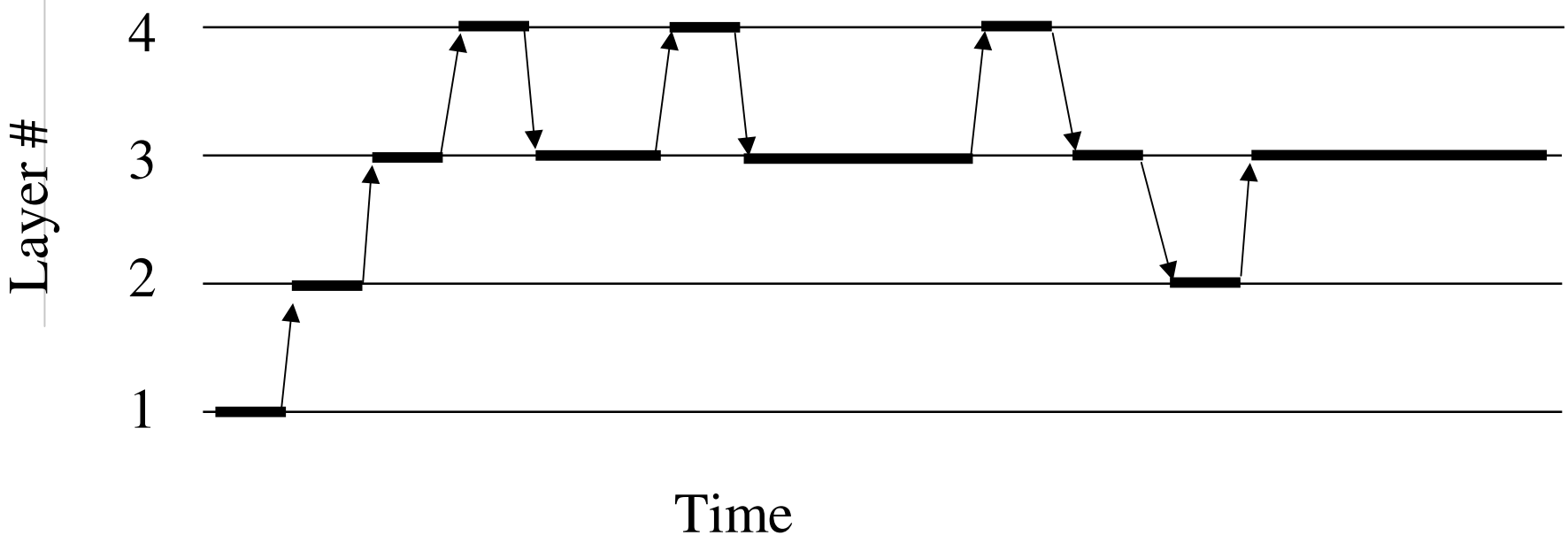
Multicast-based Dist. Tech.

- Receiver-driven Layered Multicast
  - RLM
- Source-driven Layered Multicast
  - Thin Stream
- Skyscraper

RLM: Basic Idea

- Server sends layered encoded stream where each layer is sent to a separate multicast group
- Receivers can control delivered quality by joining proper number of multicast groups. i.e. controls no of delivered layers
  - Goal: to accommodate receiver bw heterogeneity!!
- Basic idea:
  - When no cong. => receiver adds a new layer
  - When cong occurs => receiver drops top layer

RLM: example



Issues for adding layers

- When?
  - Sustained performance with little to no loss.
- Hysteresis problems
  - When available bandwidth is between the rates of two different layers.
  - Probing by new members.
- Join latency
  - Takes a while for multicast joins to occur

Issues for dropping layers.

- When?
  - Usually as a reaction to congestion.
- Co-located receiver must drop the top layer at about the same time!
- New members probing for appropriate layer will cause “false” congestion.
- Leave latency
  - Similar to the problem with join latency.

Join Experiments

- Receivers use "join-experiments" to determine if a layer should be added.
  - A join timer is associated with each layer
    - Initially, timers are fairly short.
  - If timer goes off and no congestion is experienced, then next level joined.
  - If congestion is experienced, then level is dropped and join timer is exponentially backed off.
- Receiver learns from past experiences

RLM Problems

- Scaling issues.
  - Too many participants attempting join-experiments leads to chronic congestion.
- Possible fix?
  - Coordinate join experiments with others.
  - Shared learning
  - Still problematic
    - Want to limit shared learning to topologically close neighbors.
    - Complexity of control mechanisms increases.

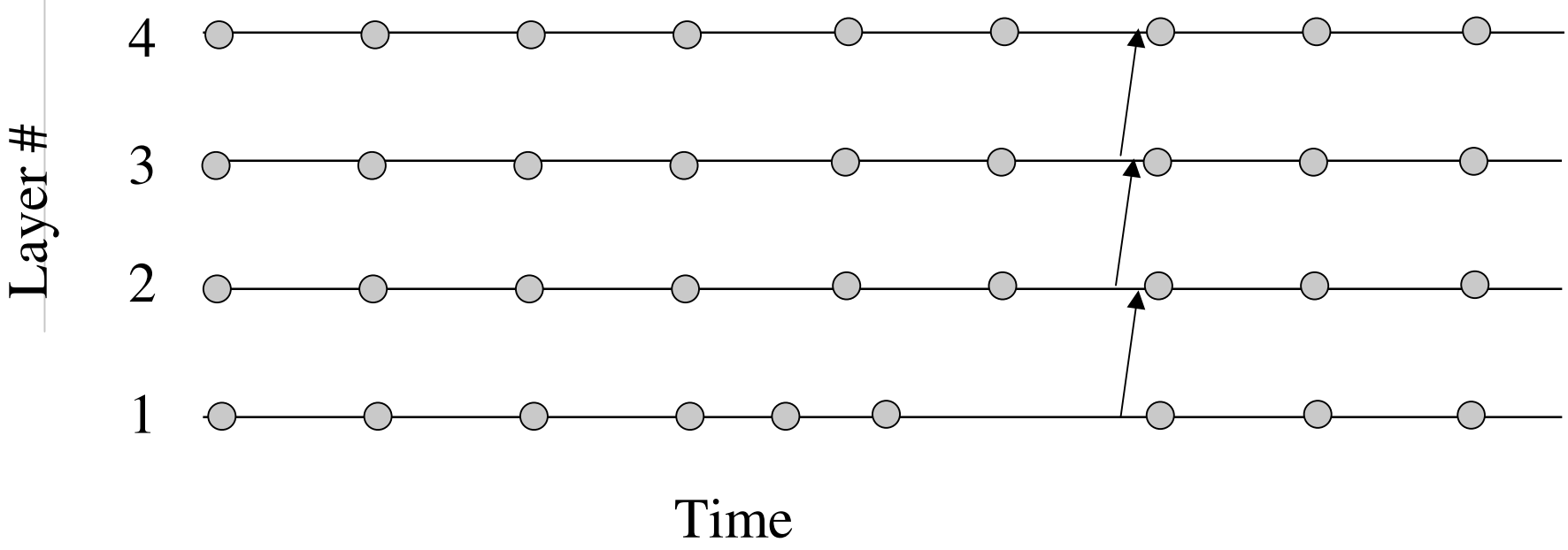
Thin Streams

- Split layers into many thin layers. Each layer is sent to a separate multicast session
- Server explicitly synchronizes join experiments.
  - Clock signal in the base stream.
- Receivers compare expected throughput vs. achieved throughput instead of loss to determine proper no of layers.
- Works better than RLM, but requires longer convergence time. Why?

Source driven experiments.

- Source periodically bunches together packets to "simulate" next higher rate.
  - Explicit signal allows receiver to differentiate between this and random network effects.
  - Average throughput over longer timescales remains the same.
  - Receivers that "survive", join next level.
- Periodic quality interruptions.

Source driven probing.



SkyScraper

- Assumptions:
  - Error control is performed through FEC or concealment (but not retx)
  - CC (and thus QA) is NOT an issue
    - Stream is CBR or smoothed and delivered with a certain rate
  - Clients can receive data at a certain rate
- Goal: to deliver a stream to a *large* number of async. clients with *minimum startup latency* for each client

Basic Idea

- Divide a stream into consecutive segments with increasing length
- Repeatedly multicast a given segment on one channel
- Receiver simultaneously listens to *n* channels to get *n* segments until the entire stream (or file) is received

Mcast-based dist.

Example

No segments/str=6, no of listening channel/client=2,
Ch. Tx rate = str playout rate

Design Issues I

- Length of the first segment determines startup latency
- Segments should be delivered before their playout time
- Fundamental tradeoffs
  - Startup latency
    - Longer latency => smaller no of tx channels
  - Number of tx channels, i.e. server bw
    - degree of segment size increase => client buf size

Design Iusses II

- Fundamental Tradeoff (cont'd)
  - Client buffer requirement
    - Larger buffer => larger no of rx channels OR ability to receive larger segments
  - Ratio of (channel delivery rate/playout rate)
    - Determines min number of listening channels
  - Number of listening channels for clients
    - Determined by client available bw

Remaining topics

- Multimedia traffic measurement
  - Thomas will present on Tue 3/4
  - Review the following paper and submit your reviews
    - Characterizing User Access to Videos on the WWW, MMCN00
- Peer to peer streaming
  - Manoj will present on Thu 3/6
  - Review the following paper and submit your reviews
    - Distributed video streaming over the Internet, MMCN02