Multicast Packet Ordering

- Easy to order unicast packets => seq #s
- Easy to order multicast packets from a single source => seq #s
- What if multiple sources?
  - Packets can arrive in different order at different receivers
  - Is this bad?
  - If so, what can we do to fix it?
Multicast Ordering Example

Example: Email Groups
Multicast Total Ordering

- All packets are delivered in same order everywhere
- Single seq # for all packets to group
  - every source sends packets to arbiter
  - arbiter assigns sequence #
  - if arbiter fails, elect new one
  - receivers don’t process packets out of order

Multicast Causal Ordering

- Total ordering inefficient for subcasts
- Instead, causal ordering
  - packets are never delivered before packets that could have “caused” them
    - receiver must have gotten all the packets source has seen
  - packets that originate concurrently can be delivered in any order
Implementing Causal Ordering

- Packets carry per-host sequence #
  - increment on each send
- Each host maintains a “version vector”
  - max seq #’s seen (in order) from each host
  - put version vector in each outgoing packet
- At receiver, delay packet until host vector > packet vector, for all sources

Causal Ordering Example
Multicast Congestion Control

- What if receivers have very different bandwidths?
- Send at max?
- Send at min?
- Send at avg?

Layered Multimedia

- Transmit signal at multiple granularities
  - 56Kb/s - voice only
  - 1Mb/s - choppy video
  - 100Mb/s - high quality video
- Layers can be
  - independent (redundant)
  - dependent (progressive refinement)
Receiver-Driven Layered Multicast

- Each layer a separate group
  - receiver subscribes to max group that will get through with minimal drops
- Dynamically adapt to available capacity
  - use packet losses as congestion signal
- Assume no special router support
  - packets dropped independently of layer

How does receiver know which layers to add?

- System dynamically adapts to available capacity
  - Use packet drops as congestion signal
  - No drops => try subscribing to higher layer
  - Drops => unsubscribe to layer
- Alternative: ask the user
**RLM Join**

- Periodically, receivers try subscribing to higher layer
- If enough capacity, no congestion, no drops
  - => keep layer (& try next layer)
- If not enough capacity, congestion, drops
  - => drop layer (& increase time to next retry)
- Coordination between receivers
  - use random delay and broadcast that join is in progress, so that others don’t try at the same time
  - shared learning -> if neighbor join fails, wait longer to try yourself

---

**RLM Join Example**

- R1 joins layer 1, joins layer 2
- R1 joins layer 3
- R2 joins layer 1, join layer 2
- R3 joins layer 2
- R4 joins layer 1, fails at layer 2
- R2, R3 join layer 1, join layer 2
- R2, R3 join layer 3
- R4 joins layer 1, fails at layer 2
Drop Policies for Layered Multicast

- Priority
  - prioritize low bandwidth layers
  - drop packets for higher layers
  - ex: everyone still gets audio, even if video degrades
  - requires router support for priorities

- Uniform (e.g., drop tail, RED)
  - packets arriving at congested router are dropped regardless of their layer

- Which is better?

Intuition vs. Practice

- Intuition: priorities should be better
  - priority drops are less wasteful; always get something useful through

- However, with RLM, uniform has
  - better incentives to well-behaved users
    - if oversend, performance rapidly degrades
  - clearer congestion signal
    - allows shared learning
Multicast Summary

- Multicast needed for efficiency, group coordination
- Need to revisit all aspects of networking
  - Routing
  - Administration
  - Reliable delivery
  - Ordered delivery
  - Congestion control

Quality of Service

- What kinds of service do different applications need?
  - Web is built on top of “best-effort” service
  - Other applications may need more
    - Internet telephone service (voice over IP)
    - Streaming audio/video
    - Real-time games
    - Remote controlled robotic surgery
- What mechanisms do we need to support these more demanding applications?
  - As with multicast, will need network to do more
IP Best Effort Service

- Our network model so far:
  - IP at routers: a shared, first come first serve (drop tail) queue
  - TCP at hosts: probes for available bandwidth, causing loss
- Router/host behavior determines the kind of service applications will receive
  - TCP causes loss, along with variable delay, variable bandwidth

An Audio Example

- Playback is a real-time service
  - audio must be received by a deadline to be useful
- Real-time applications need assurances from the network
  - What assurances does playback require?
Network Support for Playback

- **Bandwidth**
  - There must be enough on average
  - But we can tolerate to short term fluctuations

- **Delay**
  - Ideally it would be fixed
  - But we can tolerate some variation (jitter)

- **Loss**
  - Ideally there would be none
  - But we can tolerate some losses

---

**Example: Delay and Jitter**

![Graph showing packets percentage over delay (milliseconds)]
Tolerating Jitter with Buffering

- Insert variable delay before playout to give time for late samples to arrive

---

Taxonomy of Applications
Specifying Bandwidth Needs

- Problem: Many applications have variable demands

- Same average bandwidth, but very different needs over time
  - how do we describe bandwidth to the network?

Token Buckets

- Simple model
  - reflects both average, variability over time

- Use tokens to send bits
- Avg bandwidth is R bps
- Maximum burst is B bits