CS 268: Multicast Transport

Kevin Lai

April 24, 2001

The Goal

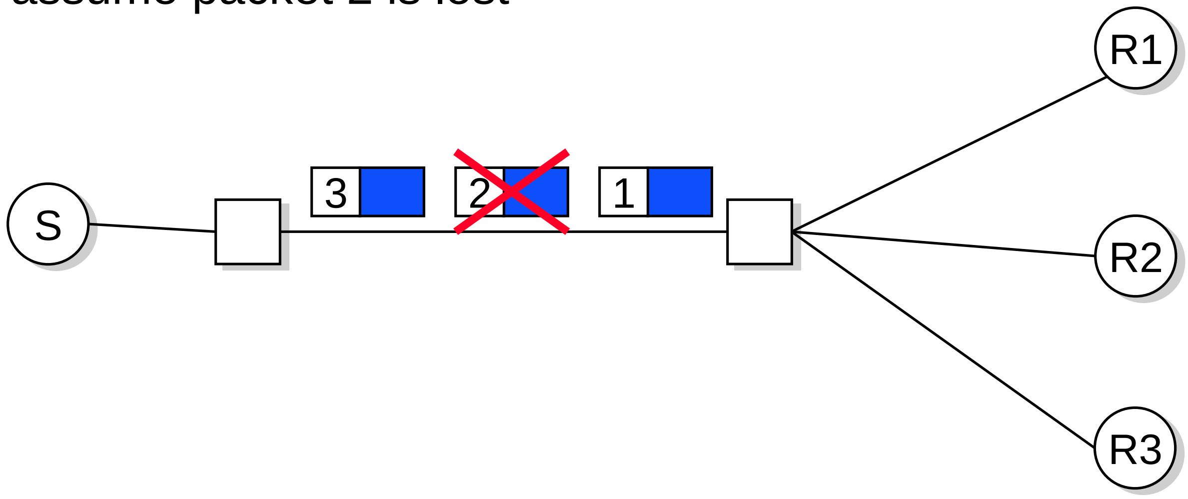
- Transport protocol for multicast
  - Reliability
    - apps: file distribution, non-interactive streaming
  - Low delay
    - apps: conferencing, distributed gaming
  - Congestion control for multicast flows
    - critical for all applications

Reliability: The Problems

- Assume reliability through retransmission
  - even with FEC, may still have to deal with retransmission (why?)
- Sender can not keep state about each receiver
  - e.g., what receivers have received, RTT
  - number of receivers unknown and possibly very large
- Sender can not retransmit every lost packet
  - even if only one receiver misses packet, sender must retransmit, lowering throughput
- Estimating path properties is difficult
  - must estimate RTT to set retransmit timers
  - unicast algorithms (e.g., TCP) don't generalize to trees
- N(ACK) implosion
  - described next

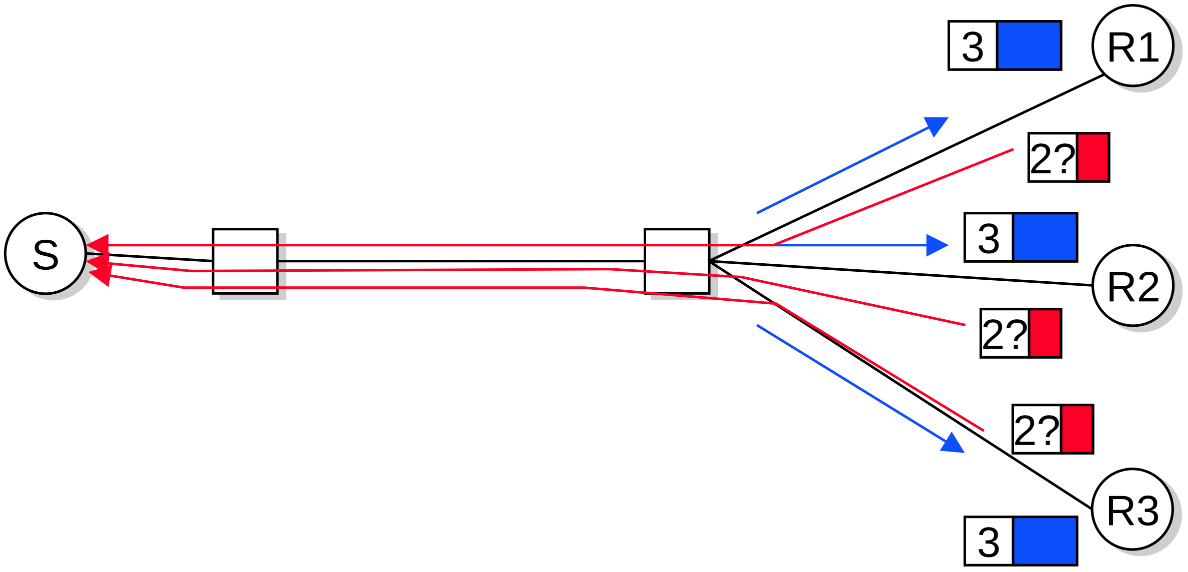
(N)ACK Implosion

- (Positive) acknowledgements
  - ack every n received packets
  - what happens for multicast?
- Negative acknowledgements
  - only ack when data is lost
  - assume packet 2 is lost



NACK Implosion

- When a packet is lost all receivers in the sub-tree originated at the link where the packet is lost send NACKs



Application Layer Framing (ALF)

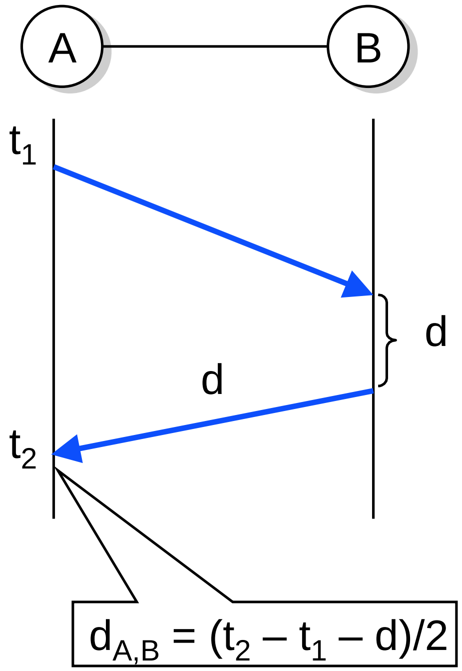
- [Clark and Tennenhouse 90]
- Application should define Application Data Unit (ADU) to lower layers
  - ADU is unit of error recovery
    - app can recover from whole ADU loss
    - app treats partial ADU loss/corruption as whole loss
  - App names ADUs
  - App can process ADUs out of order
  - Small ADUs (e.g., a packet): low delay, keep app busy
  - Large ADUs (e.g., a file): more efficient use of bw and cycles
  - Lower layers can minimize delay by passing ADUs to apps out of order

Scalable Reliable Multicast (SRM) [Floyd et al ’95]

- Receivers use timers to send NACKS and retransmissions
  - randomized
    - prevent implosion
  - uses latency estimates
    - short timer $\rightarrow$ cause duplicates when there is reordering
    - long timer $\rightarrow$ causes excess delay
- Any node retransmits
  - sender can use its bandwidth more efficiently
  - overall group throughput is higher
- Duplicate NACK/retransmission suppression

Inter-node Latency Estimation

- Every node estimates latency to every other node
  - uses session reports (< 5% of bandwidth)
    - assume symmetric latency
    - what happens when group becomes very large?



Repair Request Timer Randomization

- Chosen from the uniform distribution on

$$2^i[C_1 d_{S,A}, (C_1 + C_2) d_{S,A}]$$

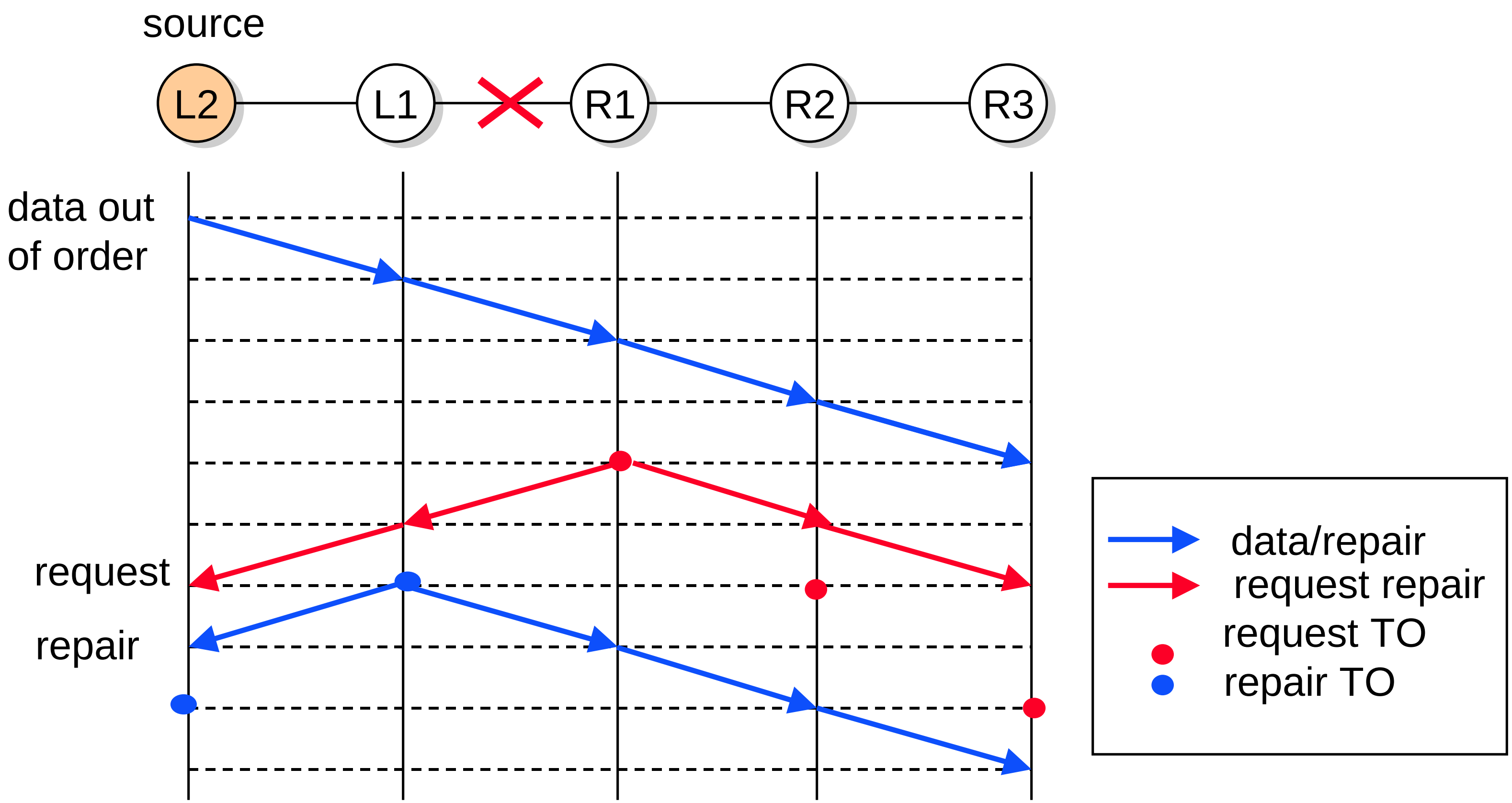
  - A – node that lost the packet
  - S – source
  - C_1, C_2 – algorithm parameters
  - $d_{S,A}$ – latency between S and A
  - i – iteration of repair request tries seen
- Algorithm
  - Detect loss $\rightarrow$ set timer
  - Receive request for same data $\rightarrow$ cancel timer, set new timer, possibly with new iteration
  - Timer expires $\rightarrow$ send repair request

Timer Randomization

- Repair timer similar
  - every node that receives repair request sets repair timer
  - latency estimate is between node and node requesting repair
- Timer properties
  - Minimize probability of duplicate packets
    - reduce likelihood of implosion (duplicates still possible)
      - poor timer, randomized granularity
      - high latency between nodes
  - Reduce delay to repair
    - nodes with low latency to sender will send repair request more quickly
    - nodes with low latency to requester will send repair more quickly
    - when is this sub-optimal?

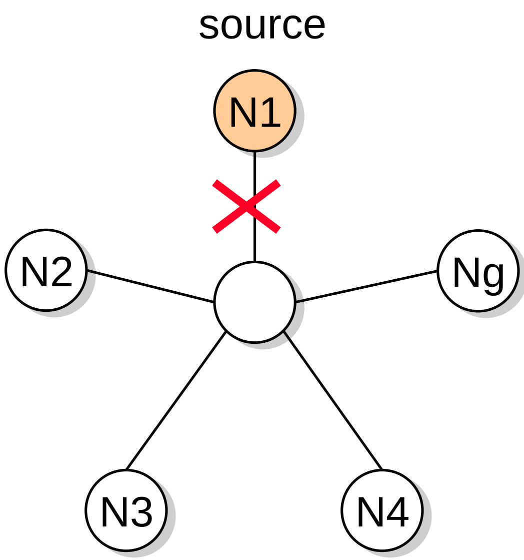
Chain Topology

- $C_1 = D_1 = 1, C_2 = D_2 = 0$
- All link distances are 1



Star Topology

- $C_1 = D_1 = 0$,
- Tradeoff between (1) number of requests and (2) time to receive the repair
- $C_2 <= 1$
  - E(# of requests) = $g - 1$
- $C_2 > 1$
  - E(# of requests) = $1 + (g-2)/C_2$
  - E(time until first timer expires) = $2C_2/g$
- $C_2 = \sqrt{g}$
  - E(# of requests) = $\sqrt{g}$
  - E(time until first timer expires) = $\sqrt{g}$



Bounded Degree Tree

- Use both
  - Deterministic suppression (chain topology)
  - Probabilistic suppression (star topology)
- Large C_2/C_1 → fewer duplicate requests, but larger repair time
- Large C_1 → fewer duplicate requests
- Small C_1 → smaller repair time

Adaptive Timers

- C and D parameters depends on topology and congestion → choose adaptively
- After sending a request:
  - Decrease start of request timer interval
- Before each new request timer is set:
  - If requests sent in previous rounds, and any dup requests were from further away:
    - Decrease request timer interval
  - Else if average dup requests high:
    - Increase request timer interval
  - Else if average dup requests low and average request delay too high:
    - Decrease request timer interval

Local Recovery

- Some groups are very large with low loss correlation between nodes
  - Multicasting requests and repairs to entire group wastes bandwidth
- Separate recovery multicast groups
  - e.g. hash sequence number to multicast group address
  - only nodes experiencing loss join group
  - recovery delay sensitive to join latency
- TTL-based scoping
  - send request/repair with a limited TTL
  - how to set TTL to get to a host that can retransmit
  - how to make sure retransmission reaches every host that heard request

Multicast Congestion Control Problem

- Unicast congestion control:
  - send at rate not exceeding smallest fair share of all links along a path
- Multicast congestion control:
  - send at minimum of unicast fair shares across all receivers
    - problem: what if receivers have very different bandwidths?
  - segregate receivers into multicast groups according to current available bandwidth

Issues

- What rate for each group?
- How many groups?
- How to join and leave groups?

Assumptions

- a video application
  - can easily make size/quality tradeoff in encoding of application data (i.e., a 10Kb video frame has less quality than a 20Kb frame)
  - separate encodings can be combined to provide better quality
    - e.g., combine 5Kb + 10Kb + 20Kb frames to provide greater quality than just 20Kb frames
- 6 layers
- 32×2^i kb/s for the *i*th layer

Example of Size/Quality Tradeoff

784 bytes

1208 bytes

1900 bytes

3457 bytes

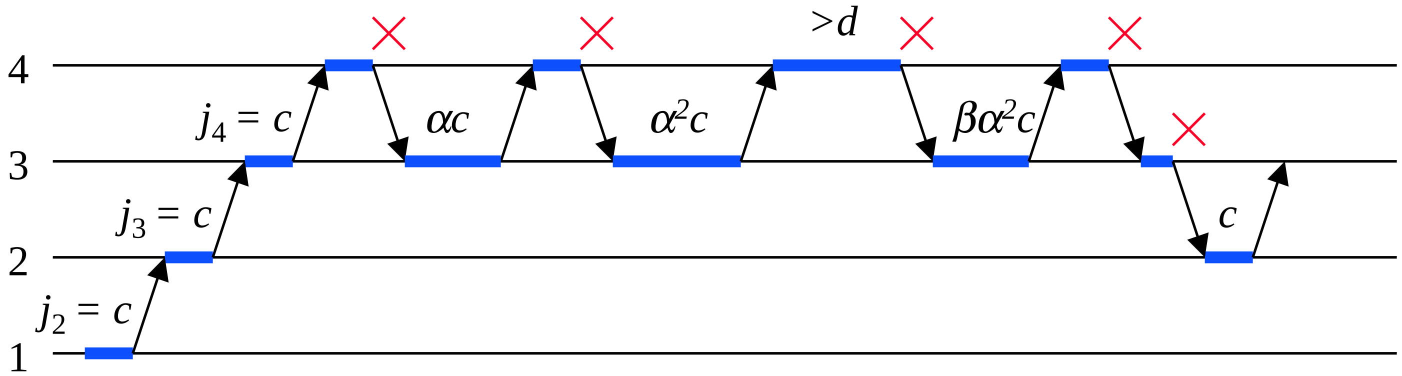
5372 bytes

30274 bytes

Basic Algorithm

- join a new layer when there is no congestion
  - joining may cause congestion
  - join infrequently when the join is likely to fail
- drop largest layer when there is congestion
  - congestion detected through drops
  - could use explicit feedback, delay
- how frequently to attempt join?
- how to scale up to large groups?

Join Timer



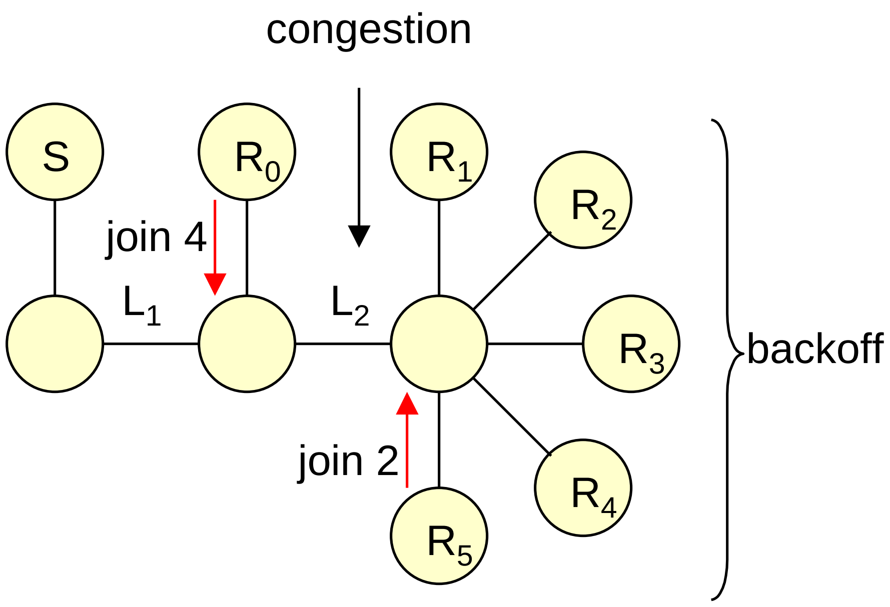
- Set 2 timers for each layer
  - use randomization to prevent synchronization
  - join timer expires $\rightarrow$ join next larger layer
  - detect congestion $\rightarrow$ drop layer, increase join timer, update detection timer with time since last layer add
  - detection timer expires $\rightarrow$ decrease join timer for this layer
- Layers have exponentially increasing size $\rightarrow$ multiplicative increase/decrease (?)
- All parameters adapt to network conditions

Scaling Problems

- Independent joins do not scale
  - frequency of joins increase with group size $\rightarrow$ congestion collapse (why?)
  - joins interfere with each other $\rightarrow$ unfairness
- Could reduce join rate
  - convergence for large groups will be slow

Scaling Solution

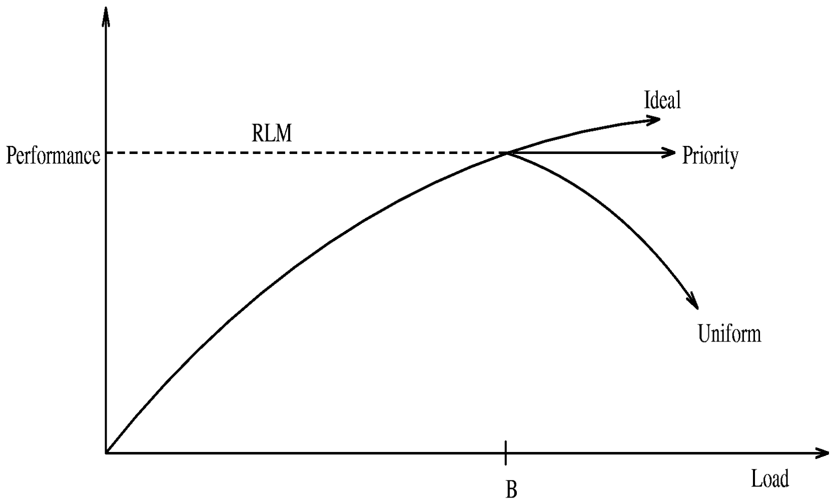
- Multicast join announcement
- node initiates join iff current join is for higher layer
- congestion $\rightarrow$ backs off its own timer to join that layer
  - shares bottleneck with joiner
- no congestion $\rightarrow$ joins new layer iff it was original joiner
  - does not share bottleneck with joiner
- convergence could still be slow (why?)



Simulation Results

- Higher network latency $\rightarrow$ less stability
  - congestion control is control problem
  - control theory predicts that higher latency causes less stability
- No cross traffic
- Scales up to 100 nodes

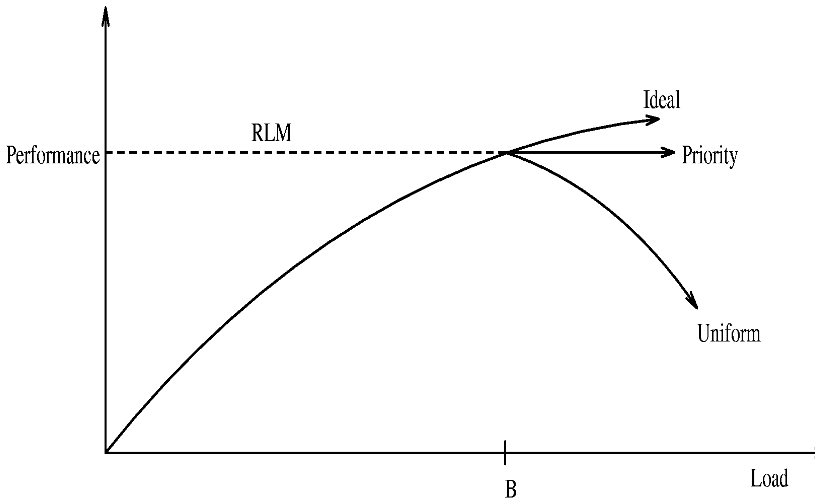
Priority-drop and Uniform-drop



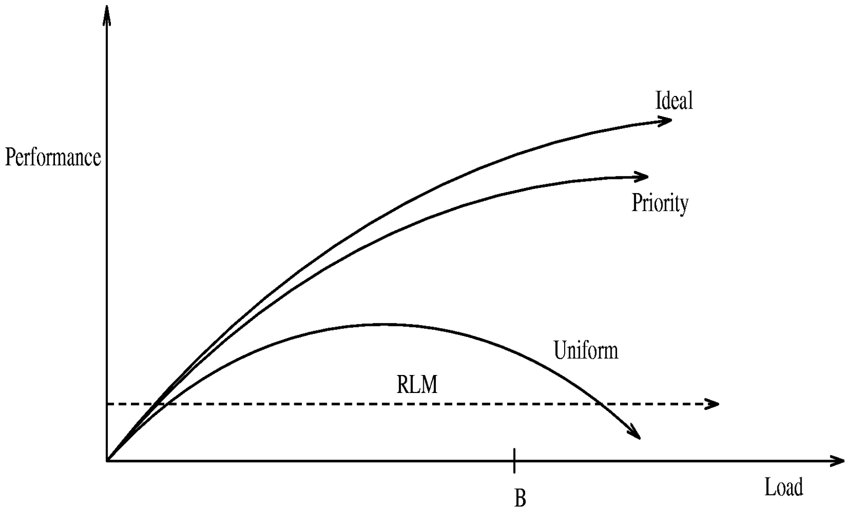
[McCanne, Jacobson 1996]

- Uniform drop
  - drop packets randomly from all layers
- Priority drop
  - drop packets from higher layers first
- Sending rate <= bottleneck
  - no loss, no difference in performance
- Sending rate > bottleneck
  - important, low layer packets may be dropped $\rightarrow$ uniform drop performance decreases
- Convex utility curve $\rightarrow$ users encouraged to remain at maximum

Later Work Contradicts



[McCanne, Jacobson 1996]



[Bajaj, Breslau, Shenker 1998]

- Burstiness of traffic results in better performance for priority drop
  - 50-100% better performance
  - measured in throughput, not delay
- Neither has good incentive properties
  - n flows, P(drop own packet) = 1/n, P(drop other packet) = (n-1)/n
  - need Fair Queueing for good incentive properties

Discussion

- Could this lead to congestion collapse?
- Do SRM/RLM actually scale to millions of nodes?
  - Session announcements of SRM
- Does RLM generalize to reliable data transfer?
  - What if layers are independent?
  - What about sending the file multiple times?
- Is end-to-end reliability the way to go?
  - What about hop-by-hop reliability?

Summary

- Multicast transport is a difficult problem
- One can significantly improve performance by targeting a specific application
  - e.g., bulk data transfer or video
- Depend on Multicast routing working

Resilient Multicast: STORM [Rex et al '97]

- Targeted applications: continuous-media applications
  - E.g., video and audio distribution
- Resilience
  - Receivers don't need 100% of data
  - Packets must arrive in time for repairs
  - Data is continuous, large volume
  - Old data is discarded

Design Implications

- Recovery must be fast
  - SRM not appropriate (why?)
- Protocol overhead should be small
- No ACK collection or group management

Solution

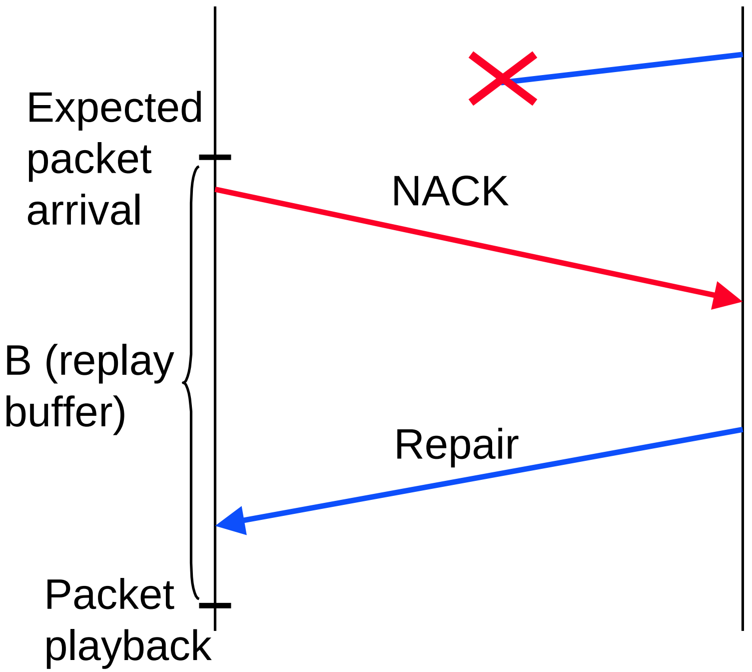
- Build an application recovery structure
  - Directed acyclic graph that span the set of receiver
    - Does not include routers!
  - Typically, a receiver has multiple parents
  - Structure is built and maintained dsitributedly
- Properties
  - Responsive to changing conditions
  - Achieve faster recovery
  - Reduced overhead

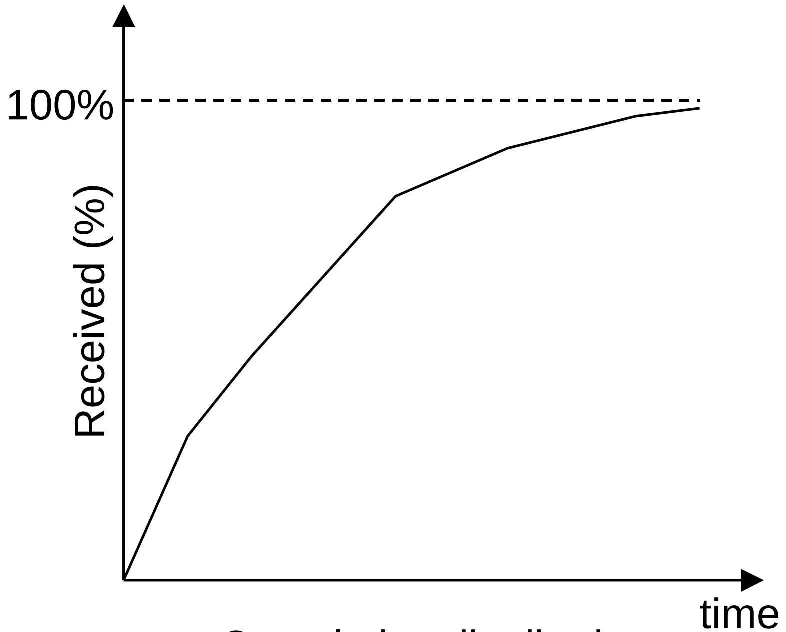
Details

- Use multicast (expanding ring search) to find parents
- When there is a gap in sequence number send a NACK
  - Note: unlike SRM in which requests and repairs are multicast, with STORM NACKs and repairs are unicast
- Each node maintain
  - List of parent nodes
  - A quality estimator for each parent node
  - A delay histogram for all packets received
  - A list of timers for NACKs sent to the parent
  - A list of NACKs note served yet
  - Note: excepting the list of NACKs shared by parent-child all other info is local

Choosing a Parent

- What is a good parent?
  - Can send repairs in time
  - Has a low loss correlation with the receiver





Cumulative distribution
of received packets

Choosing a Parent

- Source stamps each packet to local time
- t_a – adjusted arrival time, where
  - t_a = packet stamp – packet arrival time
- Each node compute loss rate as a function of t_a:

$$L(t) = 1 - \frac{number\ of\ \ packets\ such\ that\ t_a \leq t}{total\ umber\ of\ \ packets\ \exp ected}$$

- Choose parent that maximizes the number of received packets by time $t_a + B$

Loop Prevention

- Each receiver is assigned a level
- Parent’s level < child’s level
- Level proportional to the distance from source
  - Use RTT + a random number to avoid to many nodes on the same level

Adaptation

- Receivers evaluate parents continually
- Choose a new parent when one of current parents doesn't perform well
- Observations:
  - Changing parents is easy, as parents don't keep track of children
  - Preventing loops is easy, because the way the levels are assigned
  - Thus, no need to maintain consistent state such as child-parent relationship