This Week

A Tale of Two Failures

Multicast and QoS: the lost decade

History

- Multicast and QoS dominated research literature in the 90’s
- Both failed in their attempt to become pervasively available
  - Both now available in enterprises, but not in public Internet
- Both now scorned as research topics

Irony

- The biggest critics of QoS were the multicast partisans
  - And the QoS advocates envied the hipness of mcast…
- They complained about QoS being unscalable
  - Among other complaints….
- Irony #1: multicast is no more scalable than QoS
- Irony #2: scaling did not cause either of their downfalls
- Many now think economics was the problem
  - Revenue model did not fit delivery model

Lectures

- Today: multicast
  - Focus on multicast as a state of mind, not on details
- Wednesday: QoS
  - More “why” than “what”

Agenda

- Preliminaries
- Multicast routing
- Using multicast
- Reliable multicast
- Multicast’s philosophical legacy
### Motivation

- Often want to send data to many machines at once
  - Video distribution (TV broadcast)
  - Teleconferences, etc.
  - News updates

- Using unicast to reach each individual is hard and wasteful
  - Sender state: ~O(n) and highly dynamic
  - Total load: ~O(nd) where d is net diameter
  - Hotspot load: load ~O(n) on host and first link

- Multicast:
  - Sender state: O(1), total load O(d log n), hotspot load O(1)

### Multicast Service Model

- Send to logical group address
  - Location-independent

- Delivery limited by specified scope
  - Can reach "nearby" members

- Best effort delivery

### Open Membership Model

- Anyone, anywhere, can join

- Dynamic membership
  - join and leave at will

- Anyone can send at any time
  - Even nonmembers

### Division of Responsibilities

- Host's responsibility to register interest with networks
  - IGMP

- Network's responsibility to deliver packets to host
  - Multicast routing protocol

- Left unspecified:
  - Address assignment (random, MASC, etc.)
  - Application-to-group mapping (session directory, etc.)

### Other Design Choices?

### Target Environment

- LANs connected in arbitrary topology
- LANs support local multicast
- Host network cards filter multicast traffic
Multicast Routing Algorithms

Routing Performance Goals

- Roughly equivalent to unicast best-effort service in terms of drops/delays
  - Efficient tree
  - No complicated forwarding machinery, etc.

- Low join/leave latency

Two Basic Routing Approaches

- Source-based trees: (e.g., DVMRP, PIM-DM)
  - A tree from each source to group
  - State: O(G*S)
  - Good for dense groups (all routers involved)

- Shared trees: (e.g., CBT, PIM-SM)
  - A single tree for group, shared by sources
  - State: O(G)
  - Better for sparse groups (only routers on path involved)

DVMRP

- Developed as a sequence of protocols:
  - Reverse Path Flooding (RPF)
  - Reverse Path Broadcast (RFB)
  - Truncated Reverse Path Broadcasting (TRPB)
  - Reverse Path Multicast (RPM)

- General Philosophy: multicast = pruned broadcast
  - Don’t construct new tree, merely prune old one

- Observation:
  - unicast routing state tells router shortest path to S
  - Reversing direction sends packets from S without forming loops

Basic Forwarding Rule

- Routing state:
  - To reach S, send along link L

- Flooding Rule:
  - If a packet from S is received along link L, forward on all other links

- This works fine for symmetric links
  - Ignore asymmetry today

- This works fine for point-to-point links
  - Can result in multiple packets sent on LANs

Example

- Flooding can cause a given packet to be sent multiple times over the same link

  - duplicate packet

  a

  b

  c

  d

  e

  f

  g

  h

  i

  j

  k

  l

  m

  n

  o

  p

  q

  r

  s

  t

  u

  v

  w

  x

  y

  z
Broadcasting Extension

- For each link, and each source S, define parent and child
  - Parent: shortest path to S (ties broken arbitrarily)
  - All other routers on link are children
- Broadcasting rule: only parent forwards packet to L
- Problem fixed
- But this is still broadcast, not multicast!

Multicast = Pruned Broadcast

- Start with full broadcast (RPB)
- If leaf has no members, prune state
  - Send non-membership report (NMR)
- If all children of a router R prune, then router R sends NMR to parent
- New joins send graft to undo pruning

Problems with Approach

- Starting with broadcast means that all first packets go everywhere
- If group has members on most networks, this is ok
- But if group is sparse, this is lots of wasted traffic
- What about a different approach:
  - Source-specific tree vs shared tree
  - Pruned broadcast vs explicitly constructed tree

Core Based Trees (CBT)

- Ballardie, Francis, and Crowcroft,
  - “Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing”, SIGCOMM 93
- Similar to Deering’s Single-Spanning Tree
- Unicast packet to core, but forwarded to multicast group
- Tree construction by receiver-based “grafts”
  - One tree per group, only nodes on tree involved
- Reduce routing table state from O(S x G) to O(G)

Example

- Group members: M1, M2, M3
- M1 sends data

Disadvantages

- Sub-optimal delay
- Single point of failure (core)
- Small, local groups with non-local core
  - Need good core selection
  - Optimal choice (computing topological center) is NP complete
Why Isn’t Multicast Pervasive?

- Sound technology
- Implemented in most routers
- Used by many enterprises
- But not available on public Internet

Possible Explanation [Holbrook & Cheriton ‘99]

- Violates ISP input-rate-based billing model
  - No incentive for ISPs to enable multicast!
- No indication of group size (needed for billing)
- Hard to implement sender control
  - Any mcast app can be subject to simple DoS attack!!
- Multicast address scarcity
  - Global allocation required
- Awkward interdomain issues with “cores”

Solution: Single-Source Multicast

- Each group has only one source
- Use both source and destination IP fields to define a group
  - Each source can allocate 16 millions “channels”
  - Use RPM algorithm
- Add a counting mechanism
  - Use a recursive CountQuery message
- Use app-level relays to for multiple sources

Discussion

- Does multicast belong in the network layer?
  - Why not implemented by end hosts?
- How important is economic analysis in protocol design?
  - Should the design drive economics, or the other way around?
- Multicast addresses are “flat”
  - Doesn’t that make it hard for routers to scale?
  - Address allocation and aggregation?
- Should everything be multicast?
- What other delivery models are needed?

Using Multicast

- Video Distribution
- Digital Fountain
- Web Caching

A Few Examples
Distributing Video

- Send stream to multicast address at particular rate
- Paths to different receivers have different capacities
- At what rate do you send?
  - At lowest capacity rate?
  - At highest capacity rate?
  - Somewhere in between?
- None of these are good answers.....

Layered Video

- Break video into layers
- Low-rate video: only lowest layer
- High-rate video: all layers combined
- Intermediate video: only some of the layers

Example of Rate/Quality Tradeoff

- Low-rate video: only lowest layer
- High-rate video: all layers combined
- Intermediate video: only some of the layers

Receiver-driven Layered Multicast

- Each layer is sent to a separate multicast group address
- Receivers join as many layers as their capacity can handle
- But how do receivers know the capacity of their path?
  - They can measure loss rate
  - They can experiment with joins

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 highest layer when there is congestion
  - congestion detected through drops
  - could use explicit feedback, delay
- Delicate design decisions:
  - how frequently to attempt join?
  - how to scale up to large groups?

Join Timer

- Timer for each level
- Use randomization to prevent synchronization
- Join timer expires $\rightarrow$ join next larger layer ("join experiment")
- Detect congestion $\rightarrow$ drop layer, increase join timer
- Detection timer (d) expires $\rightarrow$ decrease join timer for this layer
Scaling Problem

- Having receivers join independently does not scale
- # joins increases with group size → constant congestion
- Joins interfere with each other → unfairness
  - Every time I join, I see congestion caused by you
- Just reducing join rate would slow convergence
- How do we coordinate join experiments?

Scaling Solution: Shared Learning

- Multicast join announcement
- Node initiates join if ongoing join is for higher layer
  - Shares bottleneck with joiner
  - As if it had run own experiment
- No congestion → [joins new layer if it was original joiner]
  - does not share bottleneck with joiner
- Convergence could still be slow

RLM Paper

- Simulation validation weak
- Not clear how well it works on large scale in real world
  - Large latencies a big problem
- Makes points about whether or not to use priority dropping
  - Isn’t it obvious that priority dropping would be better?

Priority-drop and Uniform-drop

- 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 → uniform drop performance decreases
- Convex utility curve → users encouraged to remain at maximum

Later Work Contradicts

- Burstiness of traffic results in better performance for priority drop
  - 50-100% better performance
- 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
  - Congestion collapse for large n

Assumptions

- High drop rates lower video quality
- Is this always true?
- New class of codes lead to “digital fountain”
- Quality is function only of received bandwidth, not drop rate
- No need for layering (except to be friendly to network)
### The Cache Consistency Problem

- Client downloads data from server, and copy is kept in some cache(s) along the way
- When other clients ask for same data, if request passes through one of these caches copy is returned
- Benefits: reduced load (server) and latency (client)
- Problem: stale data
  - How does cache know when to get rid of data
  - TTL only a guess, not good enough
  - Need explicit cache invalidation

### Traditional Cache Invalidation

- Server keeps track of all caches with copy of data
- Whenever data is changed, server contacts relevant caches
- Can apply to hierarchy of caches
- What is problem with this?

### Multicast Cache Invalidation

- Construct hierarchy of caches
- Each cache is associated with a multicast group
  - All children of that cache join the group
- Caches send out periodic heartbeats
  - Invalidations are piggy-backed on heartbeats
  - If cache has entry for that file, it then propagates update to its children
- Advantage: caches keep no state about children
  - Delivery efficiency less important than state reduction

### Reliable Multicast

### How to Make Multicast Reliable?

- FEC can help, but isn’t perfect
- Must have retransmissions
- But sender can’t keep state about each receiver
  - Has to be told when someone needs a packet

### SRM Design Approach

- Let receivers detect lost packets
  - By holes in sequence numbers
- They send NACK when loss is detected
- Any node can respond to NACK
- NACK/Response implosion averted through suppression
  - Send NACKs at random times
  - If hear NACK for same data, reset NACK timer
  - If node has data, it resends it, using similar randomized algorithm
**Repair Request Timer Randomization**

- Chosen from the uniform distribution on
  \[ 2^2(C_S^d + C_A^d + 1) \]
- \( A \): node that lost the packet
- \( S \): source
- \( C_S, C_A \): algorithm parameters
- \( d_{SA} \): latency between \( S \) and \( A \)
- \( i \): iteration of repair request tries seen

- Algorithm
  - Detect loss → set timer
  - Receive request for same data → cancel timer, set new timer
  - Timer expires → send repair request

**Suppression**

- Two kinds:
  - Deterministic suppression
  - Randomized suppression

- Subject of extensive but incomplete scaling analysis

**Local Recovery**

- Large groups with low loss correlation
  - 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?

**Application Layer Framing (ALF)**

- Application should define Application Data Unit (ADU)

- ADU is unit of error recovery
  - app can recover from whole ADU loss
  - app treats partial ADU loss/corruption as whole loss

- App can process ADUs out of order

**Multicast's True Legacy**

- Efficient delivery to multiple hosts (initial focus)
  - Addressed by SSM and other simple mechanisms

- Logical addressing (pleasant byproduct)
  - Provides layer of indirection
  - New focus of much architecture research
  - Provided by DHTs and other kinds of name resolution mechanisms

**Benefits of Multicast**