Adding Functionality to Multicast

- Receiver driven Layered Multicast (RLM)
  - How to adapt transmission rates to match heterogeneous link bandwidths
- Scalable Reliable Multicast (SRM)
  - How do you recover from losses
- Unicast – Sender does all the hard work (remember sliding window)
- Multicast
  - Cannot have sender adapt effectively (and scale)

**RLM: Problem**

- Want streaming video on the internet
- With multicast routing, this is becoming possible
  - Best effort is ok, we can tolerate some losses and retransmissions not useful anyway
- But, what rate should we stream the video at?
  - Path “capacity” different for different receivers
  - What determines capacity?
Rate Determination and Adaptation

• One possibility: Use a fixed rate
  – Lowest common denominator?
  – Send as high as the best link?

• Allow sender to adapt to congestion
  – Solves congestion, but converges to lowest common denominator

• Solution – Adapt to congestion in a heterogeneous way
  – Sender transmits at highest rate
  – Degrade flows as they go down more constrained links (how?)
  – Everyone receives the best possible quality for them

Determining Capacity

• Who decides what the path capacity is, and how?

• Receiver-driven approach
  – Receiver determines the path bottleneck capacity
    • Using drop rates, for example
  – Receiver decides how the flow gets degraded
    • By choosing the layer it should listen to
  – No changes to routers beyond IP multicast

How it works

• Sender sends in multiple layers, each in a separate multicast group

• Receiver listens to as many layers as its bandwidth can take
  – This is its “level of subscription”
  – Routers implicitly configured to add or drop a layer based on
    receivers joining/leaving the group
  – Multicast tree automatically extended when needed

• Adaptation
  – Congestion – drop a layer
  – How to go back up if there is available capacity?

• We will only skim the implementation details

Key Idea: Layered Compression

• A layered approach to compressing video data
  – Receiving at a low layer, provides bad quality (but viewable video)
  – Successive layers refine the stream
  – Sender transmits all layers
  – Degradaion is possible by dropping to the layer matching the
    available bandwidth

• Alternative is “simulcast”: send different streams concurrently at
  different quality levels
  – Is this better or worse?
Shared Learning

- Receivers multicast experiment notifications to the group for the layer
  - Receivers interested in layer watch for congestion
    - One experiment serves many receivers
  - Receivers decide based on failed experiments only
    - An experiment can succeed for some receivers but fail for others
  - Should the learning multicast be group wide??

Measuring Capacity

- Receivers probe link capacity by doing “join experiments”
  - Join the group for the next layer; back off if congestion
  - Increase if no congestion
  - Repeat periodically to adapt to changes in the network

- Need to control the frequency of join-experiments
  - Solution used: a join-timer with exponential backoff

Discussion

- Many parameters/magic numbers involved. Not clear how to determine what the best set is, or whether there is one set that fits all
- Relies on cooperate of all sources (one bad source can congest all)
- Is shared learning an optimization or a critical component
  - With enough sources, there will always be one probing a higher layer

Scalability

- What happens if receivers do join-experiments independently?
- Ideally, join experiment rate is independent of group size
  - But we would like the adaptation rate not to suffer
- Solution: shared learning
  - Receivers of a layer learn from experiments to join that layer
  - How?
Scalable Reliable Multicast

- Problem: IP is best effort
  - How do we add reliability to a multicast implementation...
  - preferably efficiently :-)
  - Is this needed by any applications?
- A complex problem domain requiring creating solutions
  - Many open issues remain
  - Many other approaches suggested (RBP, LBRM, RMTP, XTP to name a few)

Motivating Application – White Board

- Group Members
  - Named using a globally unique persistent source ID
  - Many to many model (member can be sender or receiver)
- Whiteboard has pages
  - Members create pages/draw on them
  - Pages have a globally unique persistent page ID (source ID + local page ID)
- Operations are timestamped and sequenced relative to sender (but independent of other senders)

Preventing “NACK Explosion”

- Use randomization
  - Receivers wait a random time before NACK or repair
  - Wait time function of distance from source
  - Hosts closer to failure more likely to time out
- Requests suppress other receivers with losses
  - Exponentially back off
- Upon receiving a repair request/NACK, a member
  - waits a random time then multicasts repair
  - responses supress other members that want to reply

SRM Approach

- Uses naming convention similar to wb
- Sources and pages have global long lived ids

SRM Approach

- Sender multicasts data to the group
  - Data may be lost in the network
- Receiver driven approach to recover (notice a trend?)
  - Receivers detect loss (missing sequence in stream)
  - Receiver requests retransmission – multicast to group
  - Any group member can respond (multicast again)
  - One retransmission can repair many losses

- Any concerns?
Suppression Alternatives

- Deterministic: rely upon member distances from sources
  - Members closest to failure should request/repair first
  - Deterministically suppress requests and repairs
  - Losses far from source repaired much faster

- Probabilistic suppression
  - Members detect loss at the same time, randomly set timers

- Timer tradeoffs?
- Strategy success varied with the topology
  - Deterministic worked well with dense networks
  - Adaptive timer setting explored, based on past behavior

Discussion

- Will such schemes (RLM or SRM) ever be deployed?
  - Scalability issues

- Still needs an underlying multicast protocol

- NACK/repair is a nightmare
  - Suppression helps, but it still could explode
  - Too much suppression and excessive delays possible

- Use redundancy to increase reliability? Digital Fountain paper

Application Level Multicast Paper

- Not going to wait for Multicast to be deployed

- Multicast at the application level
  - Less efficient (recall Mbone)
    - Is it any better than multiple unicast?
  - But much more easily deployable

Problems/Discussion

- Like RLM’s shared learning, recovery through group multicast
  - Major problem – requests/repairs multicast to whole group
  - Does not scale well, even with suppression mechanisms

- Hierarchically constrain recovery loss to neighborhoods
  - Set of members that are experiencing same loss profile
  - In a local loss, number of members experiencing the loss is much smaller than total members

- How to determine neighborhood?
  - Administratively? (e.g., within same AS)
  - Create separate multicast groups?
  - Can it be derived?
Application Level Multicast

- Useful to compare to multiple unicast
- Less stress
- Distributes the responsibility/load
- Several application level multicast proposals already in existence
  - Contribution of this paper; a hierarchical tree approach to organizing the multicast tree, with some nice properties

Joining the Tree

- Start from top level RP and work your way downwards to find the closest cluster at each level
  - While this is going on, temporarily attach node to the cluster at the level it is currently exploring
  - reduces effective join delay
- Will not discuss node promotion, tree maintenance, and failure resiliency

Hierarchical Aggregated Tree

- Aggregates and Localizes communication
- Geographically sensitive

Figure 2: Hierarchical arrangement of hosts in NICE. The layers are logical entities overlaid on the same underlying physical network.
Next Topic: End-to-End (Transport) Protocols

• IP and the network layer provide host-to-host connectivity across a scalable heterogeneous network

• Unfortunately, IP is a best-effort network; it can
  – Drop Messages
  – Reorder messages
  – Duplicate messages
  – Delay messages a long time
  – Limit size of messages

• How do these features compare with the requirements of applications?
  – End-to-End Protocols provide better service models to applications
  – Recall the End-to-End argument, these guys should do all the work!
End-to-end Services

• Ideally: transport protocol worries about the end-to-end service provided to the application; it does not care about the communication path

• What are common end-to-end services of interest?
  – Allow multiple processes on a host (is this possible with IP?)
  – Guarantee message delivery
  – Guarantee ordered delivery
  – No duplicates
  – Arbitrary size messages
  – Flow control (don’t overflow receiver)
  – Congestion control (don’t overflow network) – is this a service?
  – other? (QoS, Encryption, Synchronization…)

• Are these needed by all applications?

Discussion

• Why is end-to-end operation different from link-level communication?
  – At the link level, 2 ends of the link communicate with each other
  – End-to-end, 2 ends of the connection communicate with each other