Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.892 Fall 1999 **Reliable Multicast** November 22, 1999

Overview. Reliable multicast problem and approaches. SRM. Replier-based scheme for local recovery.

1 The problem

- Problem: transmit data reliably from one or more sources to multiple receivers over an unreliable multicast delivery channel (e.g., IP multicast) in an efficient manner.
- Two issues:
  1. What is *reliable*?
  2. What is *efficient*?
- *Reliable:* The unicast (TCP) definition is easy. But multicast semantics are harder promarily because "ordering" has many possible meanings. Many reliable multicast protocols are *selectively reliable*, which means that each receiver can (independently) decide whether it wants a particular missing block or not.
- *Efficiency:* Key problems
  1. *Scalability.* Q: What does this mean? A: avoid message implosion. Avoid "exposure" to nodes that aren't affected by loss.
  2. *Congestion control:* Problem hard because of heterogeneity.
  3. *Generality:* Is there a general protocol (toolkit) possible?
- Area of active research. Several different protocols have been proposed for different contexts, work on congestion control and some work on developing general toolkits. The IRTF reliable multicast research group (RMRG) of the IRTF (linked from class Web page) has more information.

2 Taxonomy of approaches

- Token ring schemes: form a logical ring using some formation protocol and pass token around; piggyback ACKs on token; easy to provide total/causal ordering and release messages.

  Examples: TRP, RMP, Totem

  Q: disadvantages? A: scaling; failures, token loss,...
- Sender-based: each receiver ACKs every packet; doesn't scale well. Examples: TCP generalization, XTP.

- Receiver-based: Each receiver NAKs missing data; randomized damping and suppression; session messages; persistent data model.

  Q: Disadvantages. A: doesn't scale well to large numbers of uncorrelated loss neighborhoods; persistent data model.

- Tree-based: Arrange receivers into a logical tree; send ACKs up tree; sender sees few ACKs if tree is built well.

  Q: disadvantages: how to build tree; if receivers come and go, what are ACK semantics?

3 SRM: Scalable Reliable Multicast

- ALF model: application in control of reliability and ordering.
- Motivated by wb, a whiteboard application.
- Receiver-based protocol.
- Contributions:
  1. ALF
  2. Distributed and randomized approach to solving message implosion.
  3. *Multicast everything!* Highly robust, session actually very hard to kill!
  4. Real app (wb) on the MBone.
- ALF: key problem here is *naming data.* Explain in detail using wb and webcast example. No linear sequence space of data.
- Two types of losses: "regular" and "tail".
- How to recover from regular losses? Send Repair Request (RREQ) with missing name, if app wants it.
- Timer backoff $= (C_1 + C_2 r)d$, where r is $U(0, 1)$.
- C_1 term useful to avoid duplicates in a chain topology. C_2 term useful in a star. Most networks can be viewed as being composed of these.
- Same algorithm for Repair Response (RRESP). *Anyone in session can respond.* Data is *persistent.*
- "Tail losses" (losses at the end) more frequent than in protocol like TCP. This is because each ADU is separately named, and each can suffer a loss at the tail.
- Recovered using periodic session announcements. Soft-state announcements.
- Elegant protocol, real app, initial solution to data naming, nice scaling in many cases.
- But has scaling problem when you have a number of uncorrelated loss neighborhoods.

4 Replier-based scheme

- Use minimal router support to do "local" recovery.
- Observation: Any time a loss occurs, it affects a *subtree* of the multicast distribution tree. Q: why?
- So, in the ideal case, the router just below the loss link asks the one just above for the missing data, and then multicasts it to the loss subtree.
- Problem with this ideal view of the world is that you end up putting lots of cruft into the routers.
- Challenge: what's the minimal set of things routers can do to approximate this ideal situation?
- Note: this is still an open question. Papadopoulos' replier-based scheme is one approach. There are other possibilities.
- Overview
  1. Each router picks a replier link from contenders. Receivers willing to be repliers periodically multicast control messages saying so.
  2. Repair request arriving on any other link is sent down replier link (this is the *turning point*. Request on replier link sent up the tree.
  3. Responses are unicast to requesting turning point. From there, they are *subcast* to all receivers behind the requesting link.
- There are some simple details that need to be worked out!
  1. How to add a replier to each router.
  2. How to send/forward requests?
  3. How to send replies?
  4. How to subcast replies?
  5. How to handle changing memberships and replier failures?
  6. How to handle source-based trees?
  7. And how to analyze resulting protocol?!

4.1 Details

- Replier selection: each router selects one of its downward links to be the replier link. I.e., replier chosen from one of its children subtrees. Router only keeps track of link, not particular receiver who may be the replier. Multiple choices dealt with using a metric (loss rate, distance,...).
- Request forwarding: A request is forwarded onto the replier link, unless it came from the replier link, in which case it is forwarded upward.

  Q: So how many requests "escape" subtree for a given loss? A: at most one! This is a key point to remember about this scheme.

- *Turning point:* If you get a request from a non-replier link, turn it toward replier link. This router is the turning point for the request.

  The turning point puts its address and requesing link id into the turned request. Routers between turning point and replier forward packet unchanged.

- Sending replies: If you're a replier and have the requested data, you can reply. But you want to make sure that it only gets to the people who've requested it, not everybody. How do you do this?

  Unicast it to turning point. That's why we put its address in turned request. Turning point gets reply, decapsulates it and gets inner multicast header. It directs this down loss subtree. How does it know which subtree? Well, that's why it had the corresponding requesting link id in turned request!

- *Dynamic memberships:* Routers only have soft state. A leaving replier sends a *replier gone* message. Replier link information also times out.

  They claim that this scheme handles dynamic memberships very well. However, there seem to be some examples where it wouldn't do that well. On the other hand, considering their elegant, minimalist approach with as little router state as possible, I think it does a good job.

- DVMRP has source-based trees, so to make sure that the subcast propagates down the correct tree, they spoof the address of the source!

- *Analysis:* not as complete as it should be (at least in this version of the paper, I don't know if there's something after this). I think that an analysis of this protocol (and its improvements) is tractable in more generality than they did in this paper.