6.829 Fall 2002

Lecture 22: Scalable Reliable Multicast

December 3 2002

- 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.
- 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 two things.
- 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.