CS514: Intermediate Course in Computer Systems

Lecture 17: October 24, 2003

Reliable Multicast

Reliable multicast is a key component

CS514

- It is a core element of pub/sub architectures
  - Even when not requiring ordering guarantees
  - Pub/sub is a nice paradigm, but ultimately it is about multicast
- It is a core element of the group communications systems we looked at
  - Every data message is multicast
- So lets spend some time looking at multicast issues

First, what is multicast?

CS514

- One-to-many (1-M) or many-to-many (M-M) communications
  - But so are cache-based CDNs, so…
- *Pushed* 1-M or M-M communications
  - Paradigm is like pub/sub: Receivers *join* (or *subscribe)* to a multicast group, senders send (or *publish*) to the multicast group
- Often it is real-time and "simultaneous", but this is not actually central to our definition

What is reliable multicast?

CS514

- *Pushed* 1-M or M-M communications where all members eventually receive every message with high probability
  - TIB uses the word "*guaranteed*" when the sender gets acknowledged
  - Even then, though, reception is not 100% (i.e. partitions can cause eventual delivery failure)
- This is the definition we will work with

What makes reliable multicast hard?

CS514

- In a word, IP multicast makes reliable multicast hard!!!

A little IP multicast history…

CS514

- Early 80's, people started playing with IP multicast over a single LAN
  - David Cheriton, Stanford, V distributed file system
- This had very nice properties… efficient use of media, simple, …
- Decided to extend this to small networks of routers
- And decided to model it after IPv4
  - Connectionless, unreliable
- And even decided to use the IPv4 header
  - I'm not sure why…

A little IP multicast history…

CS514

- The TCP/IP guys were enamored with the end-to-end paradigm
  - Which at first only said that you have to do things at the end
  - But later came to mean you should never do things in the middle
- After all, reliable unicast streams (TCP) over an unreliable middle (IP) worked great!
  - Well, eventually, more-or-less
- So, why not the same thing for reliable multicast?

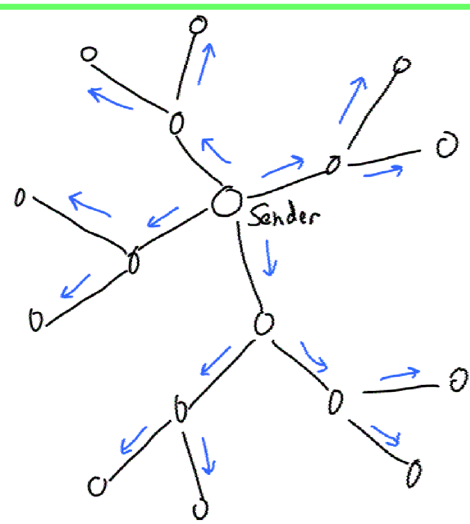
What makes reliable IP multicast hard?

CS514

- Three things:
  1. Dealing with the “implosion” of ACKs or NAKs
  2. Avoiding receiver overrun
  3. Avoiding network congestion
- Note that TCP deals with the last two only through constant feedback
  - (and, for congestion avoidance, much difficulty)

IP multicast doesn't deal well with feedback

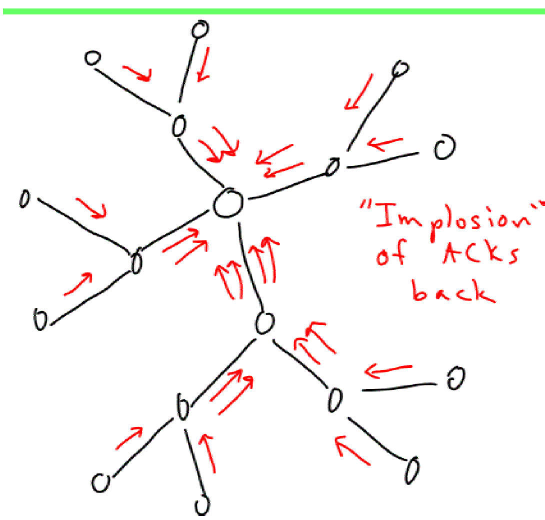
CS514



- Easy enough to transmit packets
  - Each router does only a little work

Implosion of ACKs will kill you

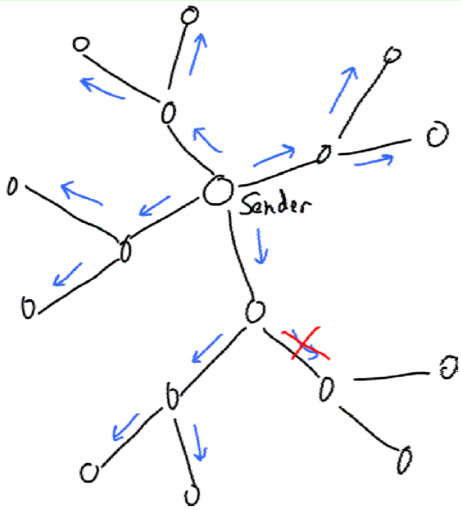
CS514



- Same goes for implosion of receive windows or congestion notifications

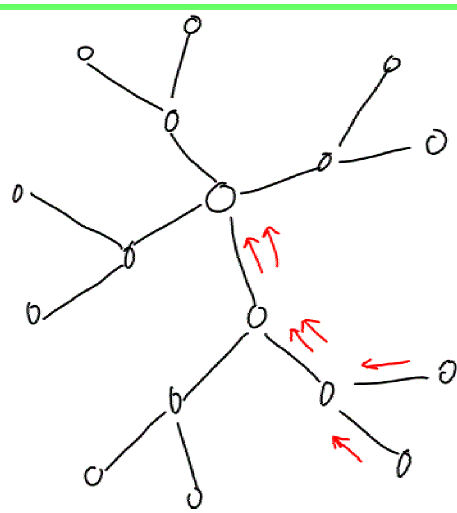
Can try NAKs instead, but…

CS514



That can kill you too

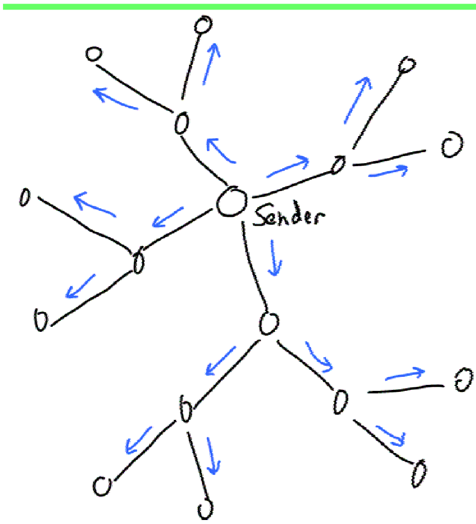
CS514



- If packet loss near source
- And lots of receivers
- Then lots of NAKs…

And the retransmit is inefficient too…

CS514



- Retransmit goes to all nodes

Dealing with the implosion

CS514

- It certainly is possible to aggregate feedback messages uptree, but…
- There will usually be some nodes that slow everything down
  - Say 1000 receivers, chances are high that at any time, one or more will exhibit high drop rate, congestion, or small receive window

Dealing with the implosion

CS514

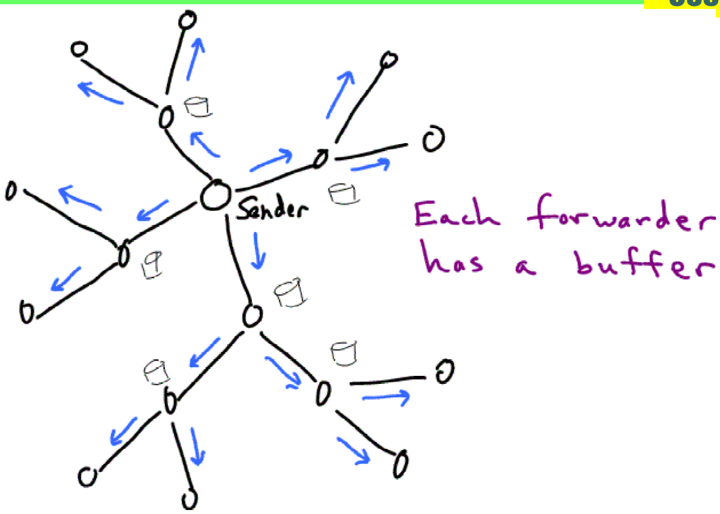
- Fundamentally, the simultaneity of IP multicast generates a “weakest link” effect
  - In small, well engineered environments, this can be avoided to an extent
- Ultimately, you need a strategy of dropping the slow guys
  - I.e., you place a floor on your send rate, and anyone who can’t keep up should drop out

Ok, so what are the alternatives?

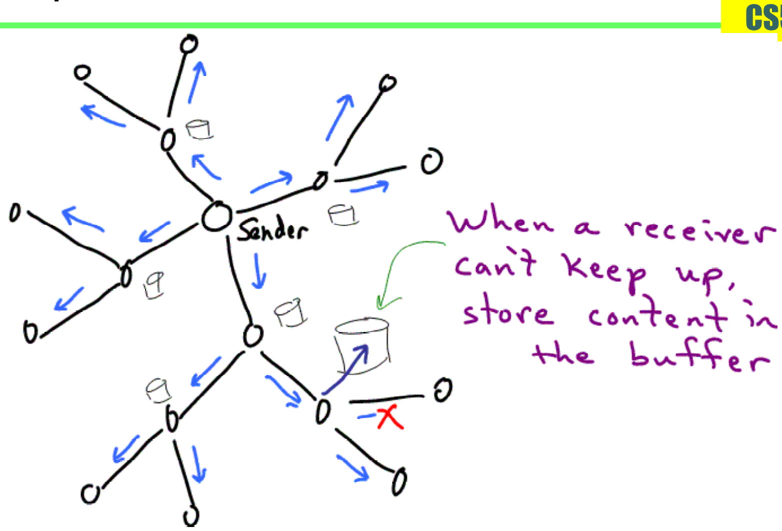
CS514

- The simultaneity effect must be broken…receivers must be decoupled from each other
- Two ways:
  1. Buffering in the forwarders (or other receivers!)
  2. Erasure (a.k.a. forward error correction) coding
- The latter actually works with IP multicast, so there is hope!

Buffering in forwarders

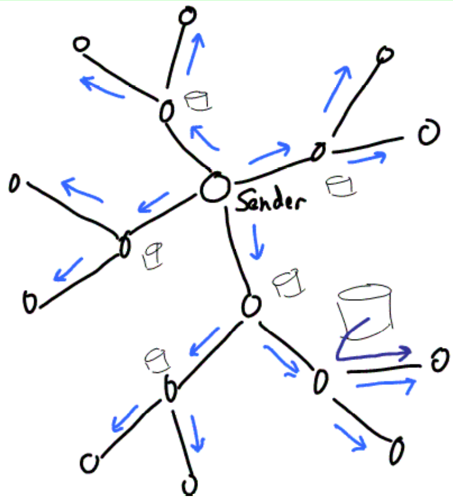


Buffering in forwarders



Buffering in forwarders

CS514



Later, when the problem goes away, feed the receiver from the buffer

Implies that the average receiver rate is good enough

Erasure codes

CS514

- Mainly for multicasting files (not live streams)
- File with M blocks is encoded as N blocks (N > M)
- If any M+K blocks are received, then file can be reconstructed

- Sender cycles through N blocks over and over
- Slower or more lossy receivers simply listen longer
- Also, receivers can start listening at different times

What we'll look at more closely:

CS514

- SRM (Scalable Reliable Multicast)
- PGM (algorithm formerly known as Pretty Good Multicast)
- pbcast (Ken's gossip-supported multicast)
- Digital Fountain (erasure code style)
- Overlay Multicast

SRM (Scalable Reliable Multicast)

CS514

- Developed in the true IP multicast, E2E model spirit
- In other words, IP multicast completely stateless, end hosts do all the work
- Recall IP multicast model:
  - Any host can send to the group
    - (Even if not a receiver, though SRM doesn't use this fact)
  - Also, IP multicast packets have a "scoping" mechanism" (using IP's TTL field)
    - Larger TTL, packet goes further, but not precisely defined as one hop per TTL value

SRM basic idea

CS514

- Packets have per-sender sequence number
- Receivers can tell when a packet was missed when they receive a later packet
  - Or when they receive a periodic "session message"
- Receivers multicast a "repair request" for missing packets
  - With limited scope, so that not all other members see it

SRM basic idea

CS514

- Packets have per-sender sequence number
- Receivers can tell when a packet was missed when they receive a later packet
  - Or when they receive a periodic "session message"
- Receivers multicast a "repair request" for missing packets
  - But *randomly timed*, so that not all other members with missing packet send a repair request
  - And with *limited TTL scope*, so that not all other members see it

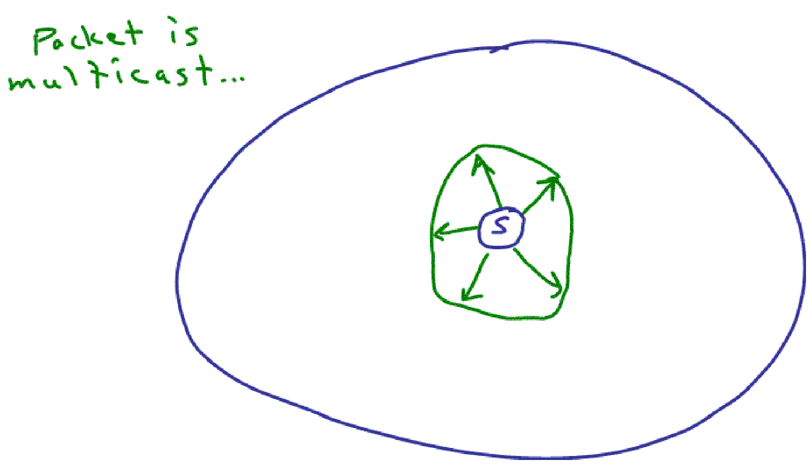
SRM basic idea

CS514

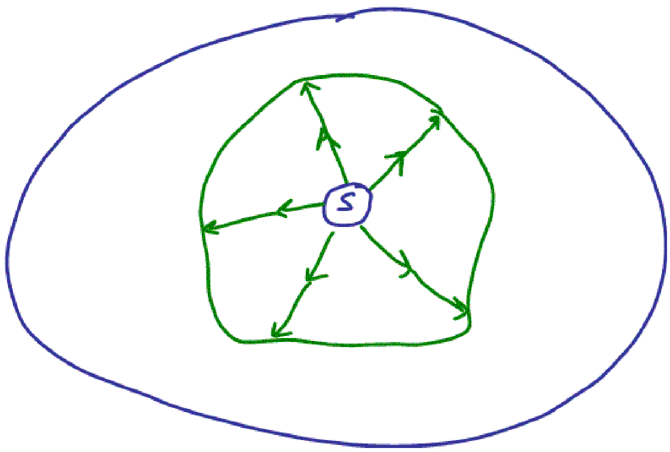
- Upon receiving a repair request, if the member has the packet, it multicasts the repair packet
  - Also randomly timed and with limited TTL scope
- If receiver with missing packet doesn't hear a repair after a while, it retransmits repair request with larger TTL
- Etc.

SRM Example

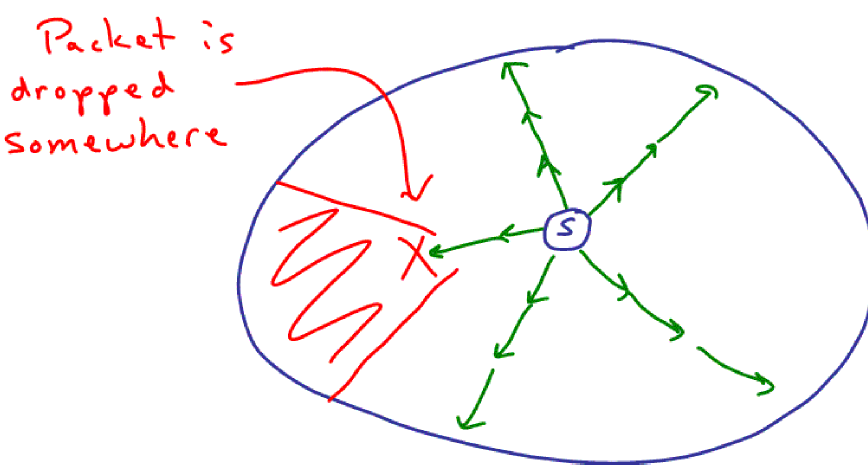
CS514



SRM Example

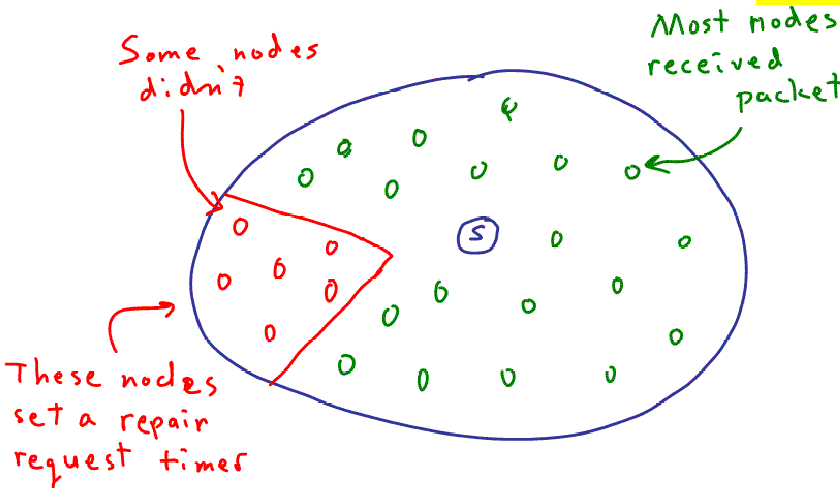


SRM Example



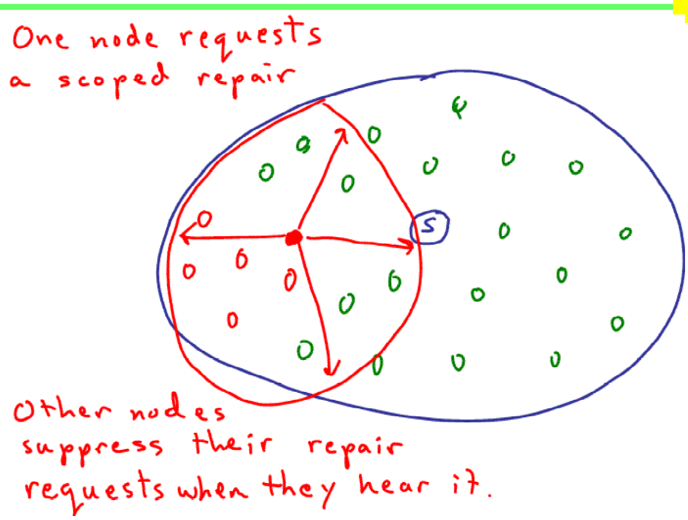
SRM Example

CS514



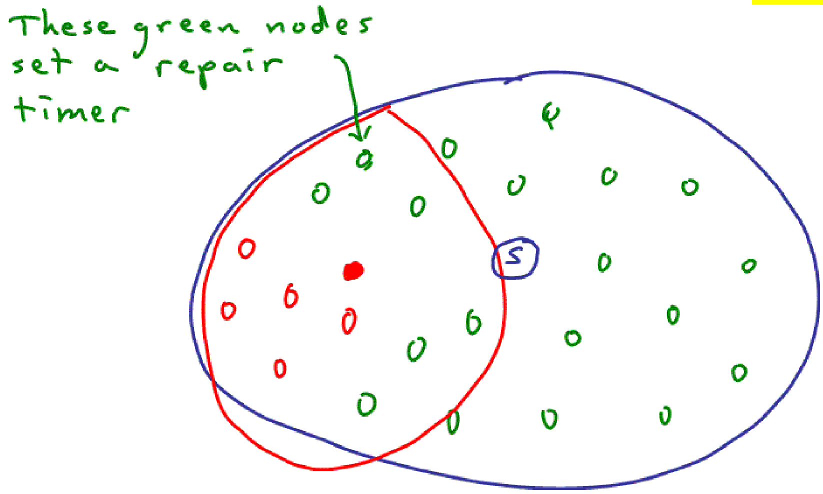
SRM Example

CS514



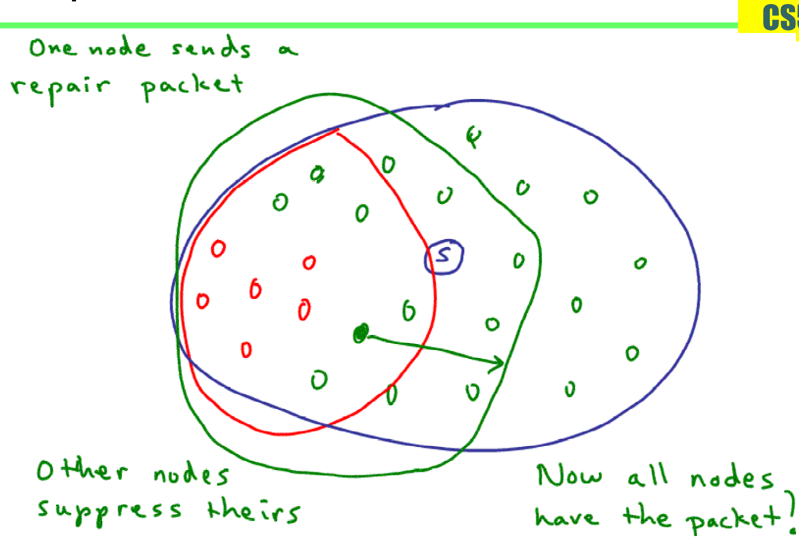
SRM Example

CS514



SRM Example

CS514



SRM timers

CS514

- Set to a value proportional to distance from sender
  - The closer to the sender, the smaller the value
- This way, nodes nearer to the sender tend to respond first
- True for both nodes requesting repairs, and node providing repairs
- Ideal: One repair request, one repair!

SRM excitement

CS514

- Initially there was lots of excitement about SRM
  - And, early results looked promising
- But . . .

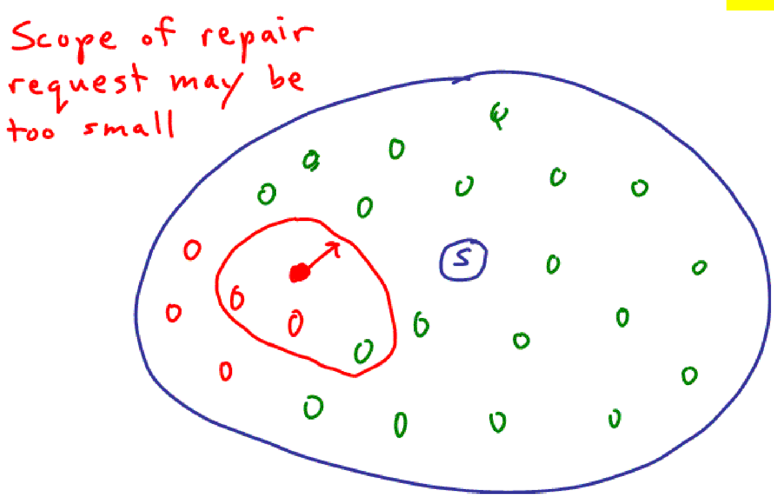
Turns out it was hard to make SRM scale

CS514

- Tension between size of scope and value of timers
  - Exacerbated by vague definition of TTL
- Increase in dropped packets with size of multicast group
- Congested links tended to cause dropped repair requests and repairs
  - Causing yet more repair requests, which caused still more congestion, etc.

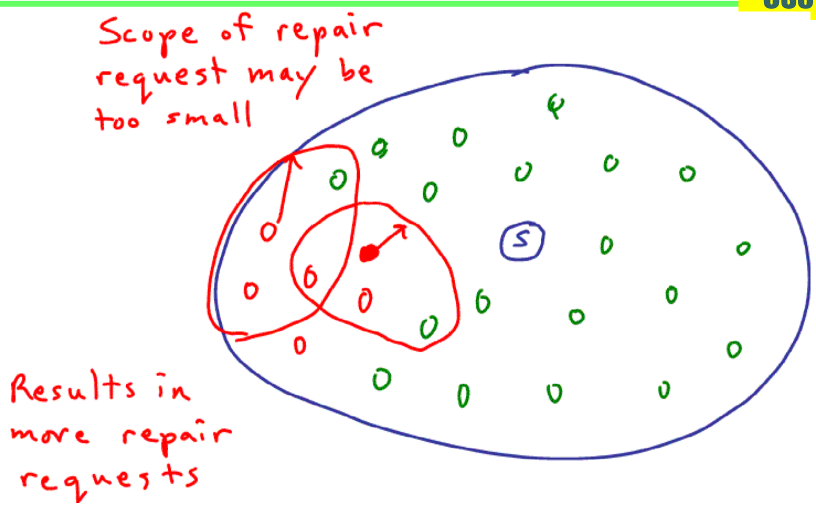
SRM difficulties

CS514



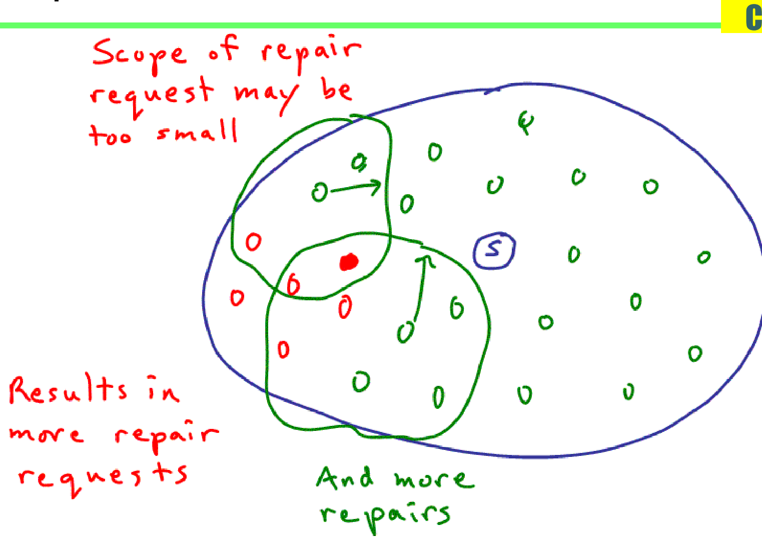
SRM difficulties

CS514



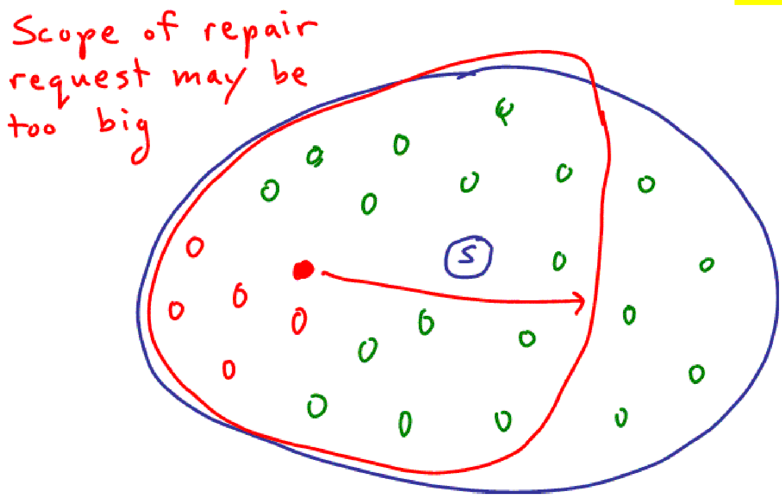
SRM difficulties

CS514



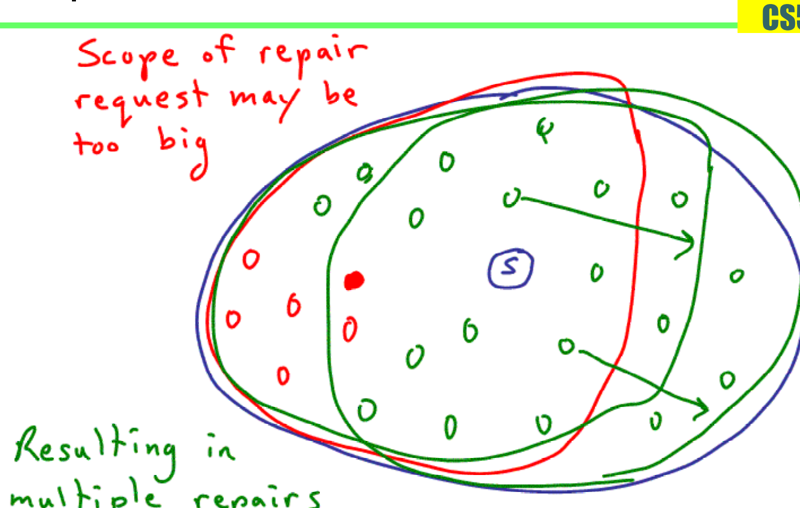
SRM difficulties

CS514



SRM difficulties

CS514



PGM

CS514

- Originally “Pretty Good Multicast”
  - From cisco
- But they were sued by the PGP (pretty good privacy) folks
- So changed to “Pragmatic General Multicast”

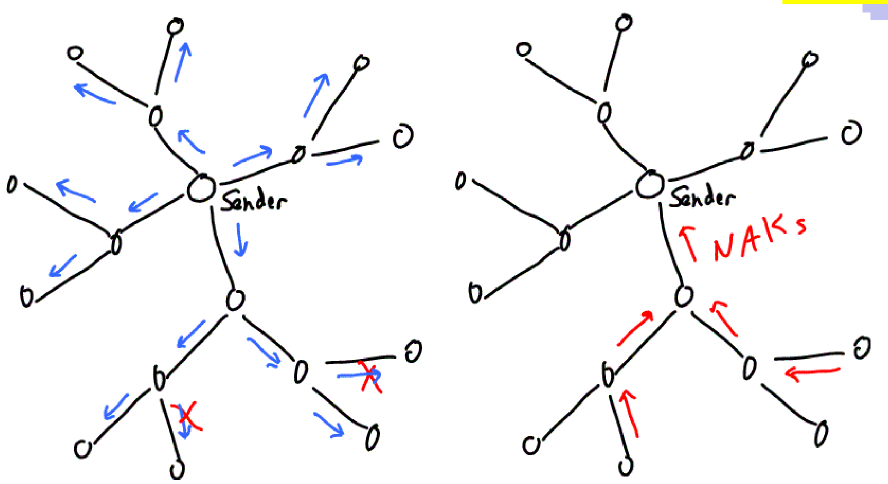
Router support for reliability

CS514

- Not surprising that it was driven by Cisco
- Idea is that routers would have “transport layer” intelligence
- NAKs travel uptree through routers towards source
- Routers remember NAKs, and transmit resends only on interfaces that received NAKs
- Later, routers could even store packets, retransmit from local store

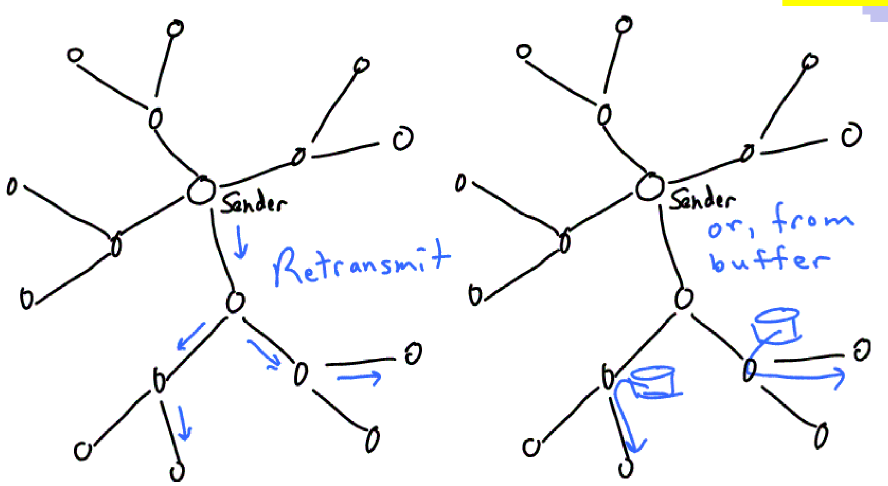
PGM example

CS514



PGM example

CS514



PGM never really took off

CS514

- Hard to say why, but…
- Turned out to be pretty complex
  - Hosts had to be modified
  - Had to work with mix of PGM and non-PGM routers…lots of tricky corner cases
- Didn't really decouple receivers
  - Still "weakest link" problem

PGM never really took off

CS514

- Possibly more to the point, PGM was not really general
- Different reliable multicasts have different needs
  - Guarantees, prioritization, even ordering
  - PGM didn't really do this
- Ultimately, it made more sense to build reliability into middleware hosts (like pub/sub), and really customize it to application needs