CS514: Intermediate Course in Computer Systems

Lecture 35: April 16, 2003

Replication at Higher Data Rates

File replication and update rate

CS514

- Last time we looked at peer-to-peer file replication
  - Files don't change much, if at all
  - Goal was to find a copy fairly easily, for large-scale sharing
- But what if we want to replicate something that changes faster?

When might we need to replicate at very high speeds?

CS514

- Media playback devices
- Online conferencing and collaboration
- Remote control of scientific experiments
- Distributing military status updates to people with maps or doing tactical decision-making

Various models

CS514

- *Multicast and reliable multicast.*
  - Goal is to send data from a (typically) small number of senders to a potentially large set of receivers
- *Digital fountains.*
  - One sender running on the public Internet. Uses multiple side-by-side channels and erasure coding to spread the load
- *Publish-subscribe, content routing systems*
  - Embed the mechanism beneath a friendly interface. But implementations depart because we lack the clean group structure

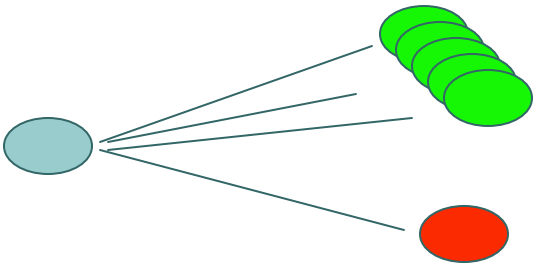
Multicast

CS514

- Issue is the tension between reliability and performance
  - To make it reliable we need a feedback and retransmission mechanism
  - And this can congest the sender
  - Often see overheads that grow roughly as $O(n^2)$!

Virtual Synchrony Scaling Issue

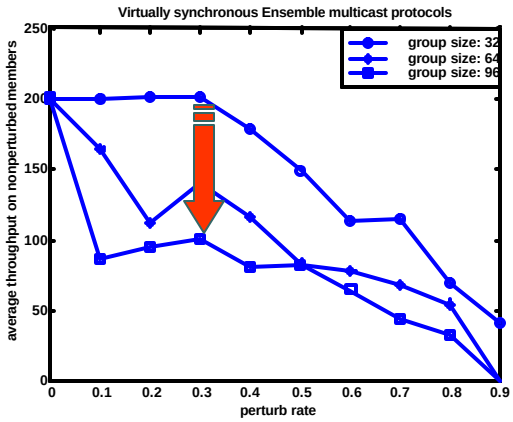
CS514



Group has many healthy members. But one is "perturbed" (slowed down) just a little bit….

Virtual Synchrony scaling issue

CS514



What causes this scaling problem?

CS514

- With "slow" failure detection…
  - Sender needs to wait for acks from slowest member
  - As group grows in size, sender's average buffering need grows both due to increased average latency and growth in delayed acks
  - Causes sender to back off

What causes this scaling problem?

CS514

- With fast failure detection…
  - Group is now far more likely to kick a slow member out
  - But it will just rejoin
  - So we pay for two membership changes, two flushes, etc

Why $O(n^2)$?

CS514

- Frequency of "problems" is roughly proportional to number of group members
- The network overhead caused by the problems is also roughly linear
  - Either in terms of delay, which triggers a round of acks
  - Or because of flush
- Overall impact is roughly n^2

What if we use a weaker reliability model?

CS514

- Berkeley's SRM and AT&T RMTP are best known solutions
- RMTP organizes system hierarchically
  - This is a bit complex to explain
  - Becomes fragile with large scale but we don't have time to discuss details
- SRM is easier to understand

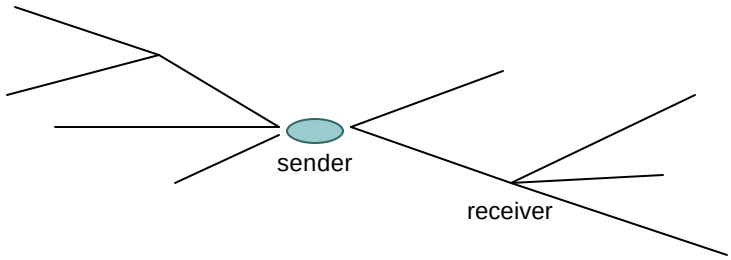
How SRM works

CS514

- Organize the multicast system as a tree
- Sender doesn't receive any ACKs
  - A pure NACK model
- Use IP multicast for many things
  - To send data
  - To request retransmissions
  - To send the retransmissions
  - TTL is useful to limit "scope" of messages

How SRM works

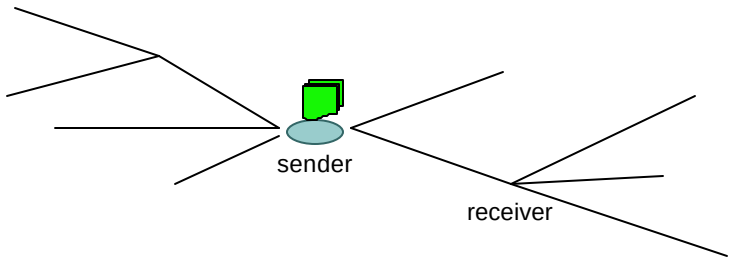
CS514



"Application level framing": The system runs at the application layer.
In fact, even retransmission requests and data are handled in the app.

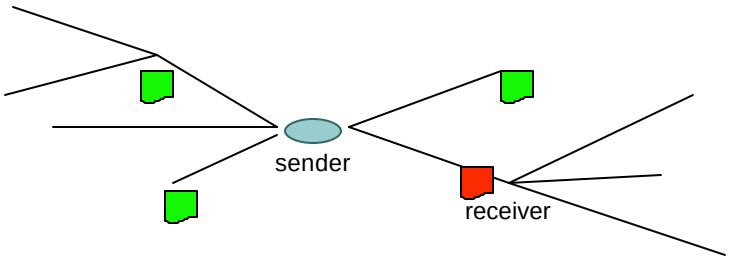
How SRM works

CS514



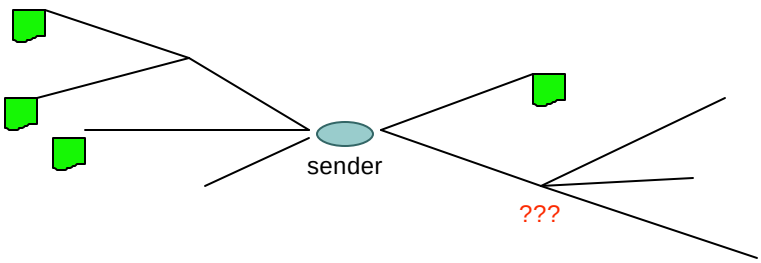
How SRM works

CS514



How SRM works

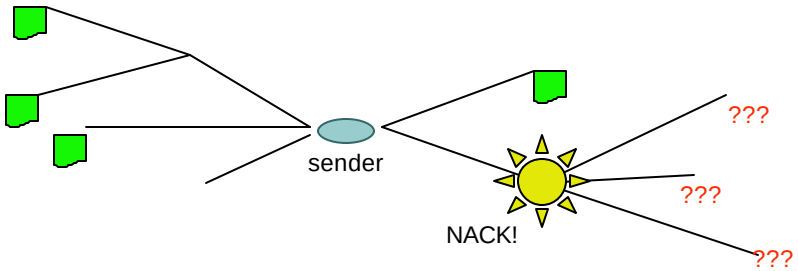
CS514



Loss is detected when receivers notice the subsequent message or timeout. In SRM they send data continuously to keep “busy”.

How SRM works

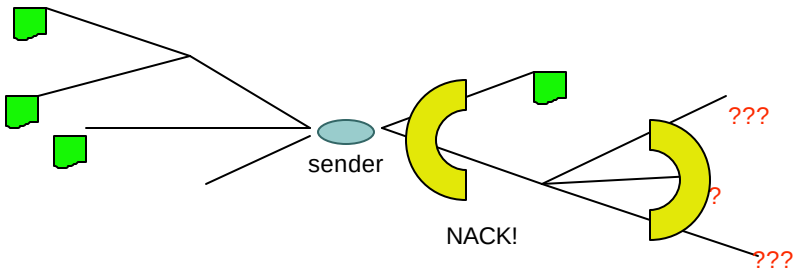
CS514



Retransmission request is a multicast. This "suppresses" retransmit req. by downstream receivers. Similar to ethernet collision handling scheme

How SRM works

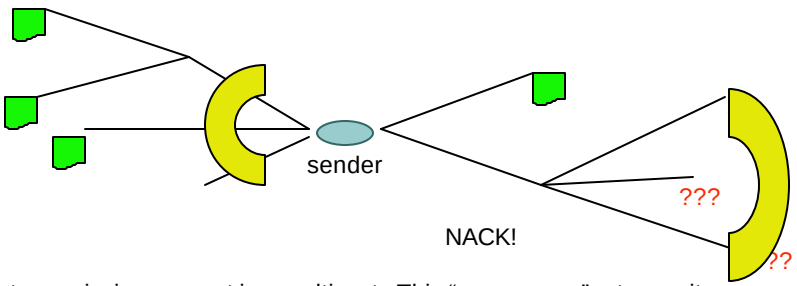
CS514



Retransmission request is a multicast. This "suppresses" retransmit req. by downstream receivers. Similar to ethernet collision handling scheme

How SRM works

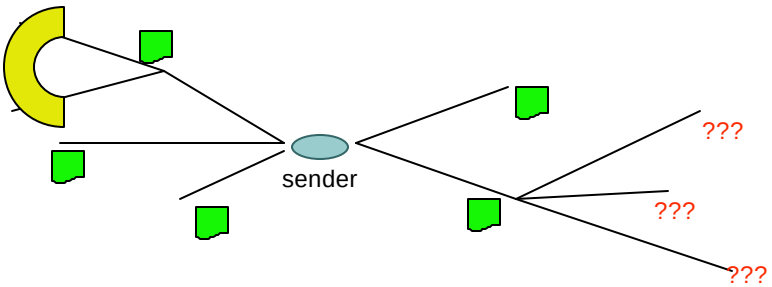
CS514



Retransmission request is a multicast. This "suppresses" retransmit req. by downstream receivers. Similar to ethernet collision handling scheme

How SRM works

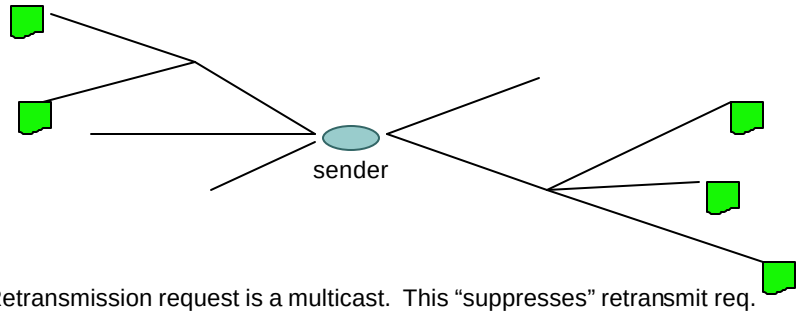
CS514



Retransmission request is a multicast. This "suppresses" retransmit req. by downstream receivers. Similar to ethernet collision handling scheme

How SRM works

CS514



Retransmission request is a multicast. This "suppresses" retransmit req. by downstream receivers. Similar to ethernet collision handling scheme

Scalable Session Messages

CS514

- Idea is to try and limit the propagation of the NACK and retransmission
- But just a heuristic…
  - Studies have shown that it can sometimes work quite well
  - But that in other situations, it flops

SRM summary

CS514

- By multicasting retransmit requests and retransmitted data gains several benefits
  - Developers argued that on average only one node will NACK and only one will retransmit the data
  - Any node can potentially do the retransmission for you (not necessarily the sender)

SRM reality

CS514

- Experiments show that reality is a different story
- In fact, number of requests and retransmissions seems to rise *linearly* with the number of receivers!
  - Problem is that sender is in often in the middle of the tree. SRM designers didn't really think of it this way…
  - Effect is that suppression mechanism is only probabilistically reliable, and rather unlikely to work in a very large network

Even worse

CS514

- Frequency of loss also rises with tree size and number of receivers
  - This is just because there are more routers and more links involved
  - The loss rate is probably constant for each link and router…
- So SRM overhead rises… as $O(n^2)$!

Is $O(n^2)$ overhead inevitable?

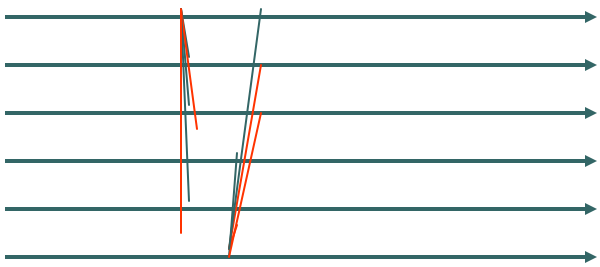
CS514

- The phenomenon is common!
  - RMTP suffers a similar problem
  - Jim Gray encountered something very much like this in parallel databases
  - We saw that virtual synchrony has this same issue too
- But there are some ways to multicast that don't degrade so quickly

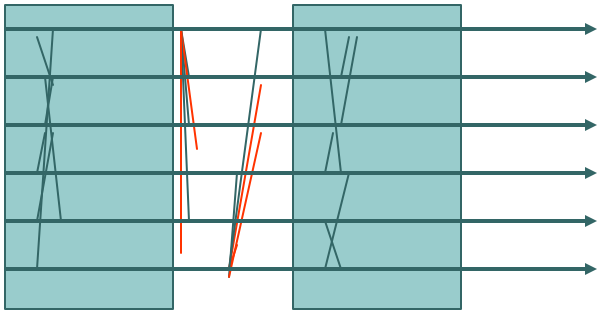
Bimodal Multicast

CS514

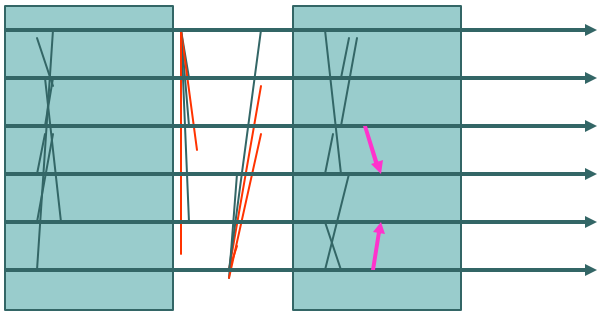
- Uses some sort of best effort dissemination protocol to get the message "seeded"
  - E.g. IP multicast, or our own tree-based scheme running on TCP but willing to drop packets if congestion occurs
- But some nodes log messages
  - We use a DHT scheme
  - Detect a missing message? Recover from a log server that should have it…



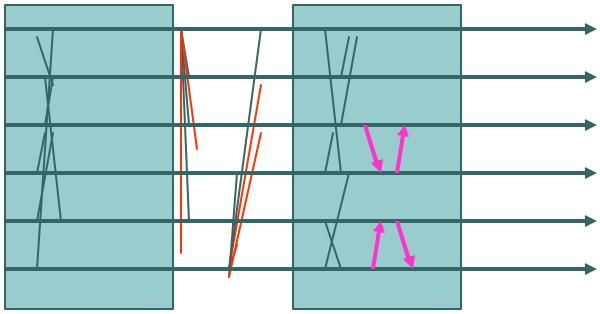
Start by using *unreliable* multicast to rapidly distribute the message. But some messages may not get through, and some processes may be faulty. So initial state involves partial distribution of multicast(s)



Periodically (e.g. every 100ms) each process sends a *digest* describing its state to some randomly selected group member. The digest identifies messages. It doesn't include them.



Recipient checks the gossip digest against its own history and *solicits* a copy of any missing message from the process that sent the gossip



Processes respond to solicitations received during a round of gossip by retransmitting the requested message. The round lasts much longer than a typical RPC time.

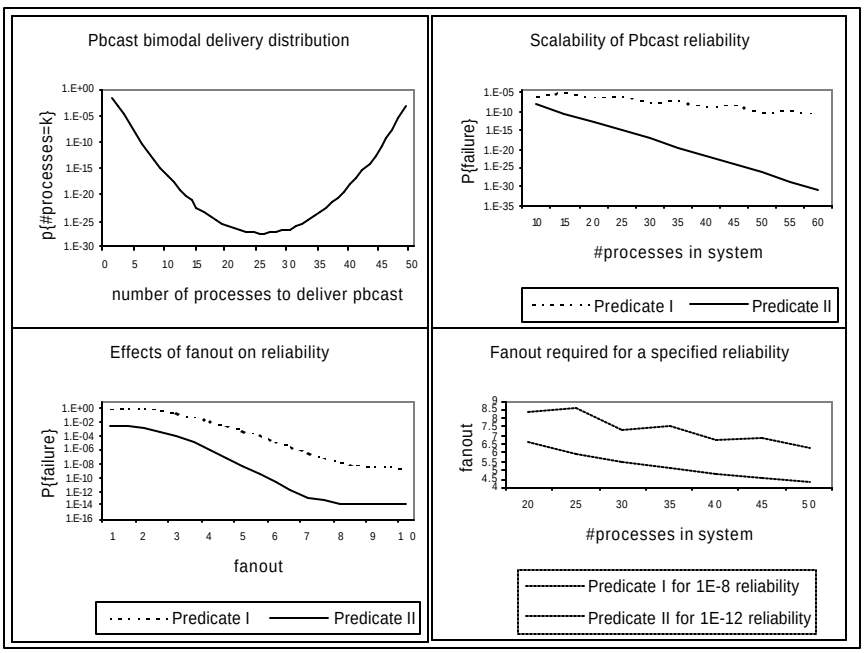
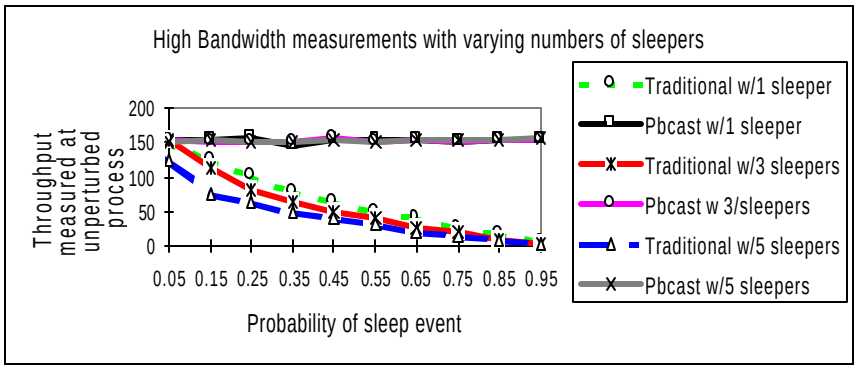


Figure 5: Graphs of analytical results

Our original degradation scenario



Making scalable multicast user friendly

CS514

- Publish-subscribe or "content addressing" concept
  - publish("subject", msg)
  - subscribe("subject", rcv_routine)
  - void rcv_routine(msg)
- Idea is that when a message matching subject(s) you registered for is published, you'll get a copy

Short history of pub-sub

CS514

- Original idea in a paper by David Cheriton and Willy Zwanepoel on "Process Groups in V System"
- Later used in many systems. Best known products include IBM MQ, TIB, DEC Message Queue

Recent directions?

CS514

- IBM Gryphon router puts mechanism into hardware
- J2EE "JMS" messaging subsystem uses this model
- Content Routing idea treats subscription as a query on the message body (but this is *slow….)*

All are basically doing reliable multicast

CS514

- No matter how you cut or slice it…
  - … a pub-sub architecture is doing multicast
  - …. and trying to be reliable
- So the scalability comes down to the same questions discussed earlier
- The interface has little to do with scalability!

Other important trends

CS514

- Side-by-side multicast trees
  - Could send data with different qualities on each tree (e.g. main MPEG frames and fine-grained inner frames)
  - Or could just send identical data for better reliability and real-time
  - Or could even use erasure coding
- Sometimes called “mesh routing”
  - Digital fountains work this way

Mesh issues

CS514

- A mesh or even multiple side-by-side trees are hard to coordinate and manage
  - Recall the NY Stock Exchange talk
  - These approaches become "implausibly complex" – do they work?
- Also need to keep the data stream itself loosely in sync.

What happens next?

CS514

- Big push now is for world-wide pub-sub solutions
  - Some demand from massive potential applications, like stock exchanges and military
  - Some demand from gaming industry, other end-user applications
- Cornell "newswire" aims at at latter…

Air Force JBI Project

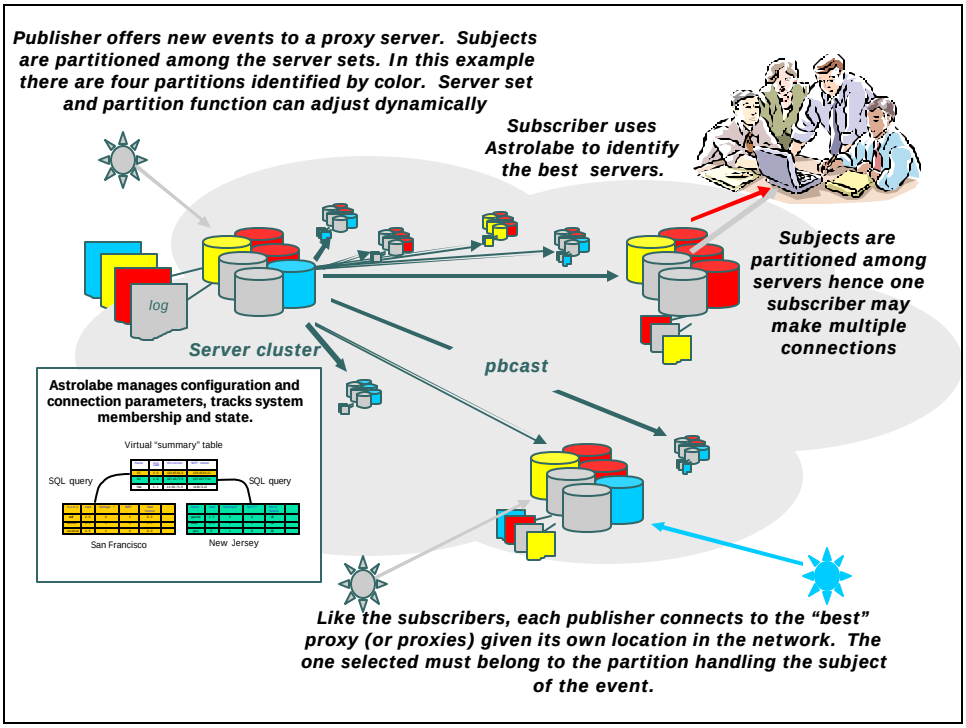
CS514

- Goal is to support an incredibly scalable pub-sub architecture
  - Perhaps could allow every computer in the military to subscribe
  - And could connect every data source they own as a publisher
- Start small… maybe assume thousands of devices and users

Air Force JBI

CS514

- Other features
  - Use Web Services standards where possible
  - Don’t send big objects in the messages themselves (instead can point to “external payload”)
  - Must be able to query a repository of high-value, older messages
  - Good security model


Publisher offers new events to a proxy server. Subjects are partitioned among the server sets. In this example there are four partitions identified by color. Server set and partition function can adjust dynamically
Subscriber uses Astrolabe to identify the best servers.
log
Server cluster
pbcast
Subjects are partitioned among servers hence one subscriber may make multiple connections
Astrolabe manages configuration and connection parameters, tracks system membership and state.
Virtual "summary" table
SQL query
SQL query
San Francisco
New Jersey
Like the subscribers, each publisher connects to the "best" proxy (or proxies) given its own location in the network. The one selected must belong to the partition handling the subject of the event.