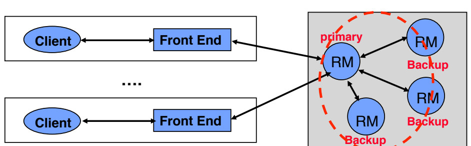
CSE 486/586 Distributed Systems
Gossiping

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap

- Consistency models
  - Linearizability
  - Sequential consistency
  - Causal consistency
  - Eventual consistency
- Depending on application scenarios, one consistency model makes more sense that others.
- As you relax consistency guarantees, you have more room for performance optimization.

Recall: Passive Replication



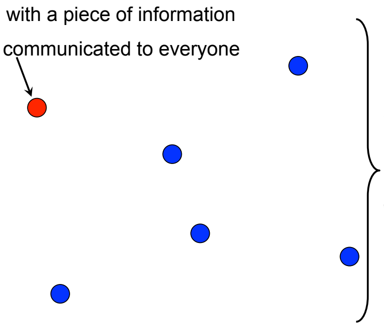
- Request Communication: the request is issued to the primary RM and carries a unique request id.
- Coordination: Primary takes requests atomically, in order, checks id (resends response if not new id.)
- Execution: Primary executes & stores the response
- Agreement: If update, primary sends updated state/ result, req-id and response to all backup RMs (1-phase commit enough).
- Response: primary sends result to the front end

Eager vs. Lazy Replication

- Eager replication, e.g., B-multicast, R-multicast, etc. (previously in the course)
  - Multicast request to all RMs immediately
  - (Roughly) replicating time-sensitive data, e.g., high-volume reads/writes
- Alternative: Lazy replication
  - Allow replicas to converge eventually and lazily
  - Propagate updates and queries lazily, e.g., when network bandwidth available
  - May provide weaker consistency than sequential consistency, but improves performance
  - (Roughly) replicating non-time-sensitive data, e.g., daily backup replication
- Lazy replication can be provided by using the gossiping

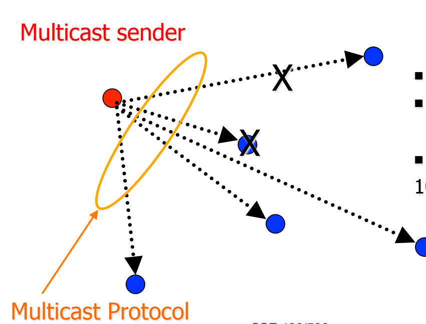
Revisiting Multicast

Node with a piece of information to be communicated to everyone



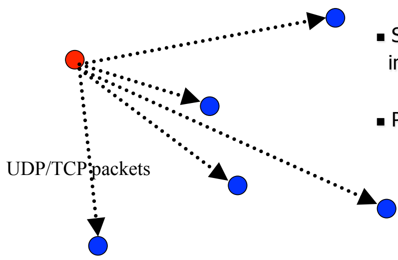
Distributed Group of "Nodes"= Processes at Internet-based hosts

Fault-Tolerance and Scalability



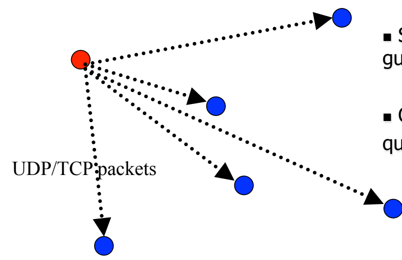
- Nodes may crash
- Packets may be dropped
- Possibly 1000's of nodes

B-Multicast



- Simplest implementation
- Problems?

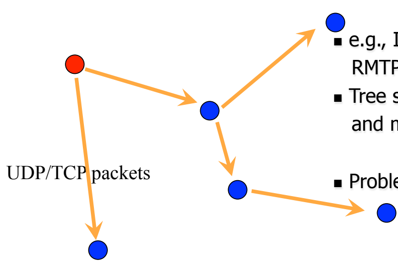
R-Multicast



- Stronger guarantees
- Overhead is quadratic in N

Any Other?

- E.g., tree-based multicast

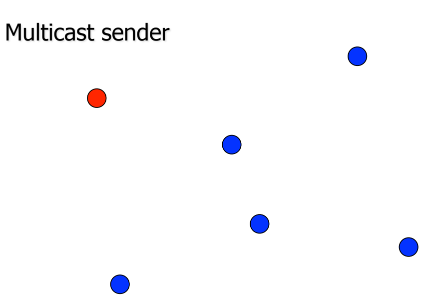


- e.g., IPmulticast, SRM RMTP, TRAM,TMTP
- Tree setup and maintenance
- Problems?

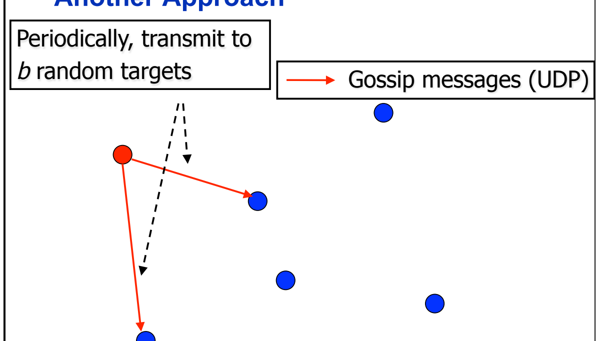
CSE 486/586 Administrivia

- PA4 will be released soon.

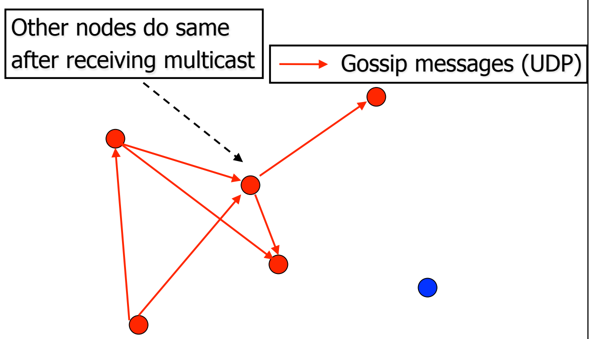
Another Approach



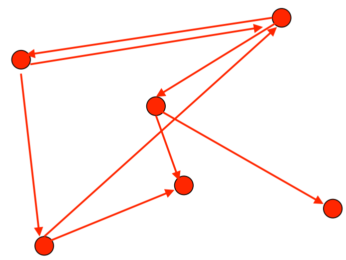
Another Approach



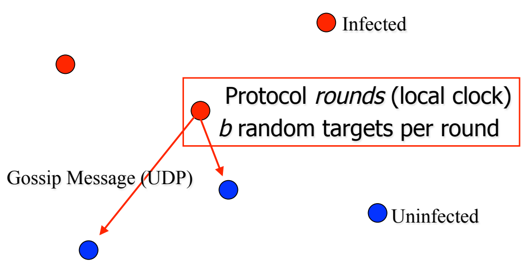
Another Approach



Another Approach



"Gossip" (or "Epidemic") Multicast



Properties

- Lightweight
- Quick spread
- Highly fault-tolerant
- Analysis from old mathematical branch of *Epidemiology* [Bailey 75]
- Parameters *c,b*:
  - *c* for determining rounds: (*c*log(n)*), *b*: # of nodes to contact
  - Can be small numbers independent of *n, e.g., c=2; b=2;*
- Within *c*log(n)* rounds, [low latency]
  - all but $\frac{1}{n^{cb-2}}$ of nodes receive the multicast [reliability]
  - each node has transmitted no more than *c*b*log(n)* gossip messages [lightweight]

Fault-Tolerance

- Packet loss
  - 50% packet loss: analyze with *b* replaced with *b/2*
  - To achieve same reliability as 0% packet loss, takes twice as many rounds
- Node failure
  - 50% of nodes fail: analyze with *n* replaced with *n/2* and *b* replaced with *b/2*
  - Same as above

Fault-Tolerance

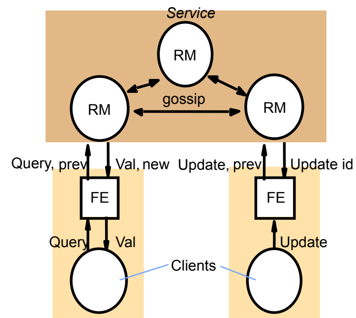
- With failures, is it possible that the epidemic might die out quickly?
- Possible, but improbable:
  - Once a few nodes are infected, with high probability, the epidemic will not die out
  - So the analysis we saw in the previous slides is actually behavior *with high probability*

  [Galey and Dani 98]
- The same applicable to:
  - Rumors
  - Infectious diseases
  - An Internet worm
- Some implementations
  - Amazon Web Services EC2/S3 (rumored)
  - Usenet NNTP (Network News Transport Protocol)

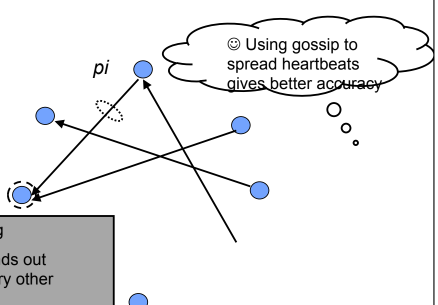
Gossiping Architecture

- The RMs exchange "gossip" messages
  - Periodically and amongst each other.
  - Gossip messages convey updates they have each received from clients, and serve to achieve convergence of all RMs.
- Objective: provisioning of highly available service. Guarantee:
  - Each client obtains a consistent service over time: in response to a query, an RM may have to wait until it receives "required" updates from other RMs. The RM then provides client with data that at least reflects the updates that the client has observed so far.
  - Relaxed consistency among replicas: RMs may be inconsistent at any given point of time. Yet all RMs eventually receive all updates and they apply updates with ordering guarantees. Can be used to provide sequential consistency.

Gossip Architecture

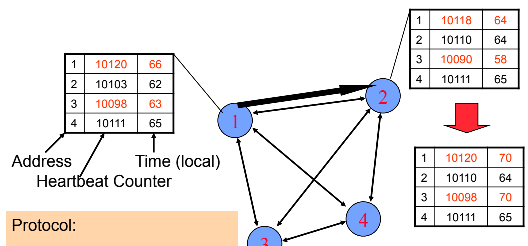


Using Gossip for Failure Detection: Gossip-style Heartbeating



All-to-all heartbeating
- Each process sends out heartbeats to every other process
- Con: Slow process/link causes false positives

Gossip-Style Failure Detection



Protocol:
- Processes periodically gossip their membership list
- On receipt, the local membership list is updated

Current time : 70 at process 2

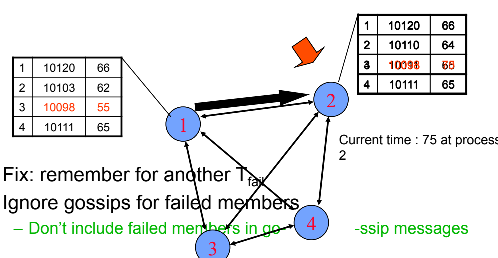
(asynchronous clocks)

Gossip-Style Failure Detection

- If the heartbeat has not increased for more than T_{fail} seconds (according to local time), the member is considered failed
- But don't delete it right away
- Wait another $T_{cleanup}$ seconds, then delete the member from the list

Gossip-Style Failure Detection

- What if an entry pointing to a failed process is deleted right after T_{fail} seconds?



- Fix: remember for another T_{fail}
- Ignore gossips for failed members
  - Don't include failed members in gossip messages

Summary

- Eager replication vs. lazy replication
  - Lazy replication propagates updates in the background
- Gossiping
  - One strategy for lazy replication
  - High-level of fault-tolerance & quick spread
- Another use case for gossiping
  - Failure detection

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).