Cristina Nita-Rotaru

CS505: Distributed Systems

Gossip Protocols.

Slides prepared based on material by Prof. Ken Birman at Cornell University, available at http://www.cs.cornell.edu/ken/book/

Required reading for this topic…

- Bimodal multicast K. Birman, M. HaydenO. Ozkasap, Z. Xiao, M. Budiu, Y. Minsky

Reliable Multicast

- Ensures that a precise subset of processes/nodes in a group delivers a message (ideally none of the other processes receives the message)
- System environment characteristics
  - Large number of processes
  - No global network-level multicast protocol

Meaning of Reliability in Multicast

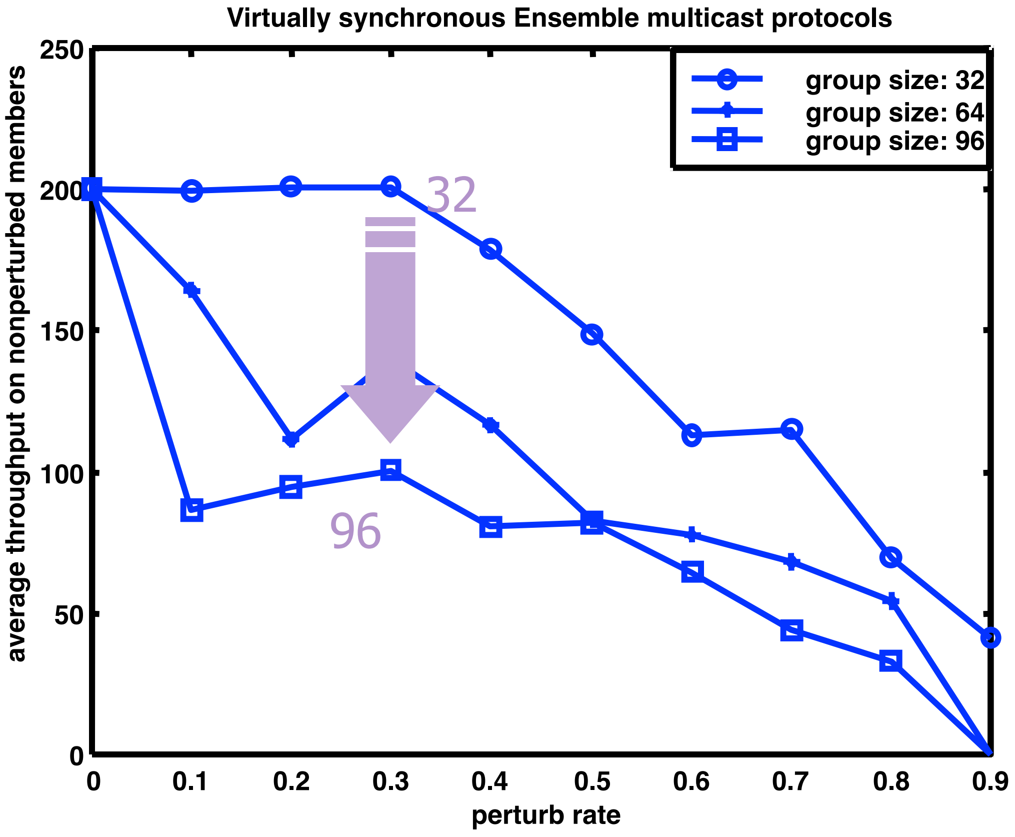
- Integrity: A correct process p delivers a message m at most once.
- Validity: If a correct process multicasts message m, then it will eventually deliver m.
- Agreement: If a correct process delivers message m, then all the other correct processes in the group will eventually deliver m.

Approaches

- Deterministic schemes
  - With strong reliability guarantees do not scale well (e.g., O(n2) msgs)
- Probabilistic, gossip-based, schemes
  - Every process periodically (every T ms) „talks" to a subset of (Fanout, F) processes about some messages
  - Good trade-off between reliability and scalability
  - Very resilient to arbitrary crash failures

Limitations of Classical Reliable Multicast

- With classical reliable multicast, throughput collapses as the system scales up!
- Even if we have just one slow receiver… as the group gets larger (hence more *healthy* receivers), impact of a performance perturbation is more and more evident!



Gossip Overview

“Did you hear that Sally and John are going out?”

- Node A encounters “randomly selected” node B (might not be *totally* random)
  - Push: A tells B something B doesn’t know
  - Pull: A asks B for something it is trying to “find”
  - Push-pull: Combines both mechanisms

Definition: A Gossip Protocol…

- Uses random pairwise state merge
- Runs at a steady rate (and this rate is much slower than the network RTT)
- Uses bounded-size messages
- Does not depend on messages getting through reliably

Gossip Benefits

- Information flows around disruptions
- Scales very well
- Typically reacts to new events in log(N), N is number of processes
- Can be made self-repairing

… and Limitations

- Rather slow
- Very redundant
- Guarantees are at best probabilistic
- Depends heavily on the randomness of the peer selection

Typical Push-Pull Protocol

- Nodes have some form of database of participating machines
  - Could have a hacked bootstrap, then use gossip to keep this up to date!
- Set a timer and when it goes off, select a peer within the database
  - Send it some form of “state digest”
  - Peer responds with data you need and its own state digest
  - Respond with data peer needs

Gossip Implementation

- Recall that UDP is an "unreliable" datagram protocol supported in internet
  - Unlike for TCP, data can be lost
  - Also packets have a maximum size, usually 4k or 8k bytes (you can control this)
  - Larger packets are more likely to get lost!
- What if a packet would get too large?
  - Gossip layer needs to pick the most valuable stuff to include, and leave out the rest!

Use of Gossip Protocols

- Notify applications about some event
- Track the status of applications in a system
- Organize the nodes in some way (like into a tree, or even sorted by some index)
- Find “things” (like files)

Probabilistic Multicast

- Validity: If a correct process multicasts a message m, then some correct process in Dest(m) eventually delivers m
- Integrity: For any message m, every correct process p delivers m at most once, and only if m was previously multicast by Sender(m)
- **Probabilistic Agreement**: If a correct process in Dest(m) delivers message m, then every correct process in Dest(m) eventually delivers m with known, high, probability ω.

Scalable Reliable Multicast

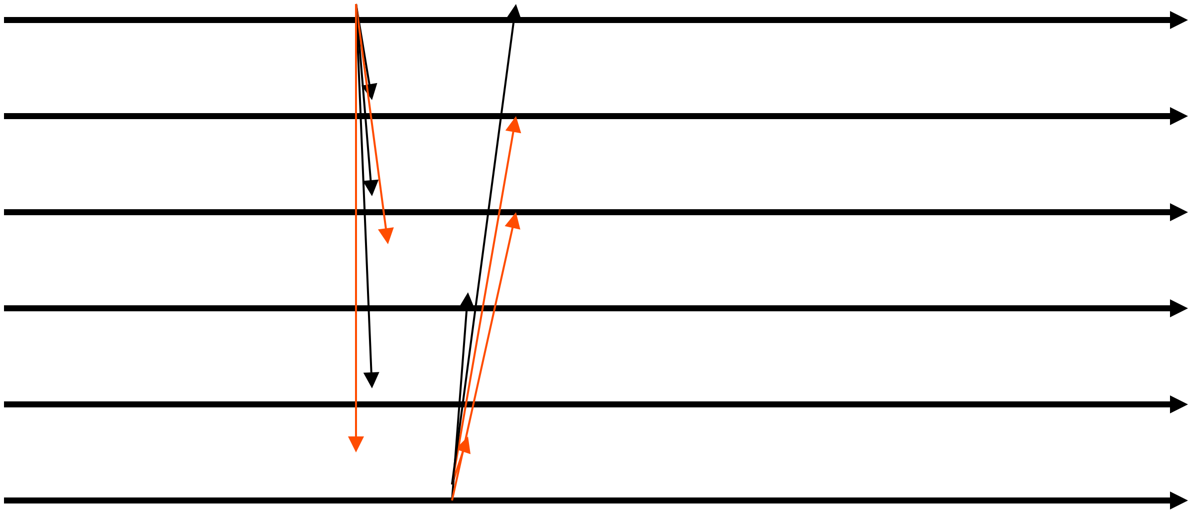
- Heartbeats: Each member periodically sends out a heartbeat including the sequence number of the latest sent packet. Members detect packet loss by comparing the sequence number in the heartbeat and the sequence number of the last data-packet received.
- NACKS: When a packet is lost, a negative acknowledgment (NACK) is sent to all members using the same method of transportation as the original data.
- Repair: Each member if he sees a NACK for a packet they have in their cache, they retransmit that packet to the whole group as a repair.
- To minimize the number of NACKs and repairs, these two operations are preceded by exponential back-off.

Problems with ACK/NACK Schemes

- As number of receivers gets large ACKS/NAKS pile up (sender has more and more work to do)
  - Hence it needs longer to discover problems
  - And this causes it to buffer messages for longer and longer… hence flow control kicks in!
  - So the whole group slows down

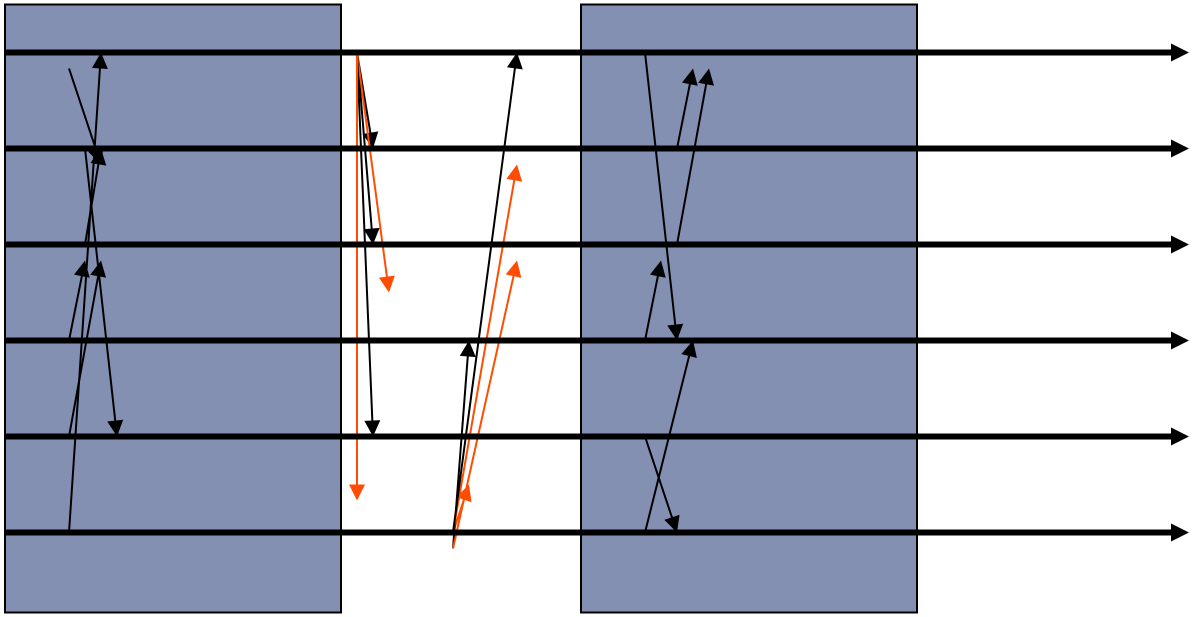
Bimodal Multicast: First Phase

- Combines gossip with IP multicast
- Start by using *unreliable UDP* multicast to rapidly distribute the message.
- Some messages may not get through, and some processes may be faulty: initial state involves partial distribution of multicast(s)



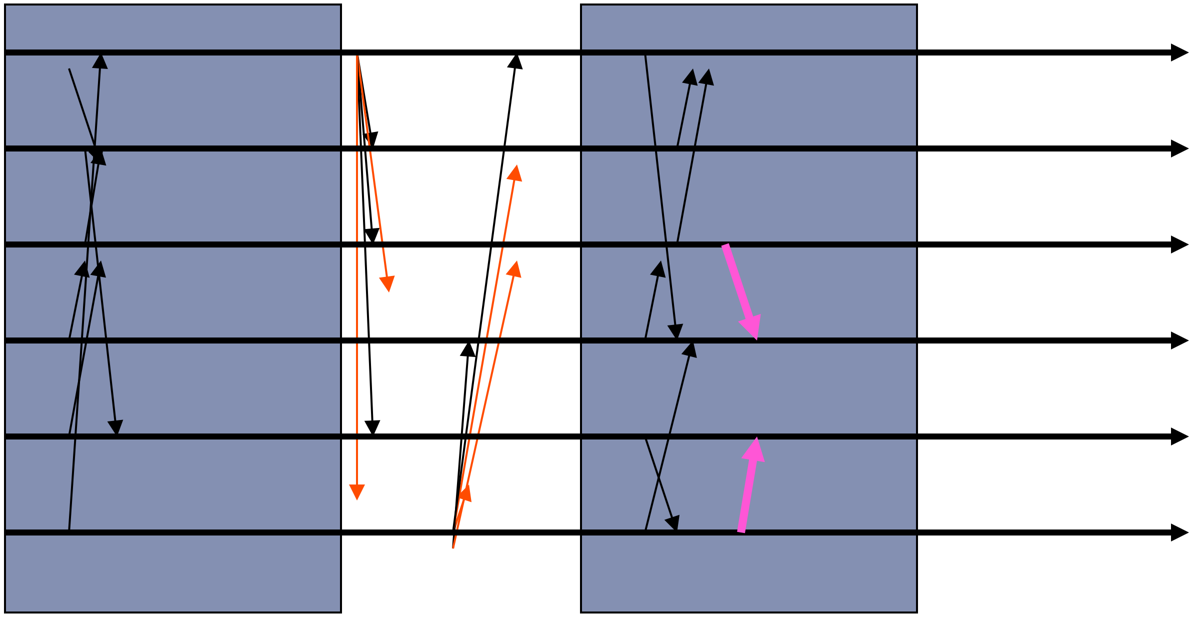
Finding out what is missing

- Periodically (e.g. every 100ms) each process sends a *digest* describing its state to some randomly selected group member. The digest identifies messages.



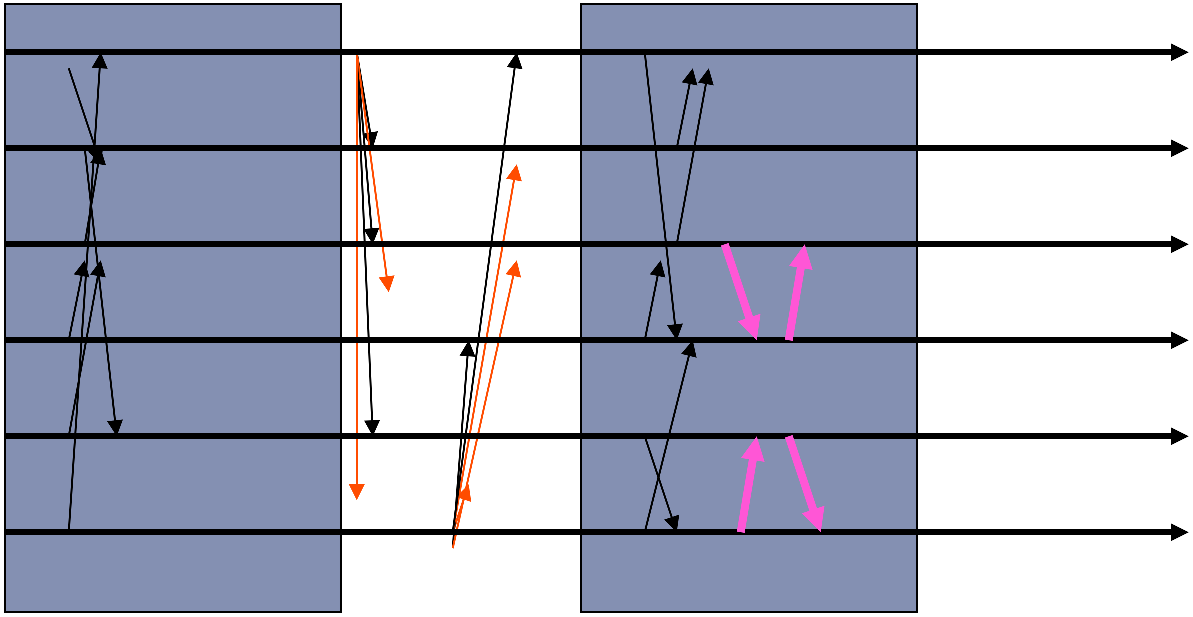
Soliciting missed messages

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



Sending out missed packets

- 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.



Delivery? Garbage Collection?

- Deliver a message when it is in FIFO order
  - Report an unrecoverable loss if a gap persists for so long that recovery is deemed "impractical"
- Garbage collect a message when you believe that no "healthy" process could still need a copy (we used to wait 10 rounds, but now are using gossip to detect this condition)
- Match parameters to intended environment

Need to bound costs

- Worries:
  - Someone could fall behind and never catch up, endlessly loading everyone else
  - What if some process has lots of stuff others want and they bombard him with requests?
  - What about scalability in buffering and in list of members of the system, or costs of updating that list?

Scalability

- Protocol is scalable except for its use of the membership of the full process group
- Updates could be costly
- Size of list could be costly
- In large groups, would also prefer not to gossip over long high-latency links

Router Overload Problem

- Random gossip can overload a central router
- Yet information flowing through this router is of diminishing quality as rate of gossip rises
- Insight: constant rate of gossip is achievable and adequate

Hierarchical Gossip

- Weight gossip so that probability of gossip to a remote cluster is smaller
- Can adjust weight to have constant load on router
- Now propagation delays rise… but just increase *rate* of gossip to compensate

How to Analyze such Protocols?

- Can use the mathematics of epidemic theory to predict reliability of the protocol
- Assume an initial state
- Now look at result of running B rounds of gossip: converges exponentially quickly towards atomic delivery

Summary

- Gossip is a valuable tool for addressing some of the needs of modern autonomic computing
- Often paired with other mechanisms, eg anti-entropy paired with UDP multicast
- Solutions scale well (if well designed!)