CSE 486/586 Distributed Systems

Gossiping

Steve Ko
Computer Sciences and Engineering
University at Buffalo

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

Multicast sender

Multicast Protocol

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

B-Multicast

UDP/TCP packets

- Simplest implementation
- Problems?

R-Multicast

UDP/TCP packets

- Stronger guarantees
- Overhead is quadratic in N

Any Other?

- E.g., tree-based multicast

UDP/TCP packets

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

Multicast sender

Periodically, transmit to \( b \) random targets

Gossip messages (UDP)

Other nodes do same after receiving multicast

Gossip messages (UDP)

Uninfected

Infected

Protocol rounds (local clock)

\( b \) random targets per round

Gossip Message (UDP)

“Gossip” (or “Epidemic”) Multicast

CSE 486/586 Administrivia

- PA2-B is due in ~2 weeks.
- PA1 grades are posted.
- PA2-A grading is in progress.
Properties

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

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)

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

Using Gossip for Failure Detection: Gossip-style Heartbeating

- Each process sends out heartbeat to every other process
- Con: Slow process/link causes false positives
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)