CSE 486/586 Distributed Systems
Gossiping

Slides by 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

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

- Simplest implementation
- Problems?

UDP/TCP packets
R-Multicast

- Stronger guarantees
- Overhead is quadratic in $N$
Any Other?

• E.g., tree-based multicast

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

Multicast sender
Another Approach

Periodically transmit to

$b$ random targets

Gossip messages (UDP)
Another Approach

Other nodes repeat after receiving multicast

Gossip messages (UDP)
Another Approach

Iterate several times
“Gossip” (or “Epidemic”) Multicast

c protocol rounds (local clock)
b random targets per round

Gossip Message (UDP)
Properties

• Lightweight
• Rapid dissemination of information
• 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 [reliable]
  – 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, could 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
• The same applicable to:
  – Rumors
  – Infectious diseases
  – An Internet worm
• Some implementations
  – Amazon Web Services EC2/S3 (rumored)
  – Usenet NNTP (Network News Transport Protocol)
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

Comparatively, gossip has:
- Lower overhead
- Better accuracy
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 for $P_i$ has not increased for more than $T_{\text{fail}}$ seconds (according to local time), $P_i$ is considered failed.
- But *don’t delete it right away*.
- Wait another $T_{\text{cleanup}}$ seconds, then delete $P_i$ from the list.
Gossip-Style Failure Detection

• What if an entry pointing to a failed process is deleted immediately at $T_{\text{fail}}$ seconds?

<table>
<thead>
<tr>
<th></th>
<th>Value 1</th>
<th>Value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10120</td>
<td>66</td>
</tr>
<tr>
<td>2</td>
<td>10103</td>
<td>62</td>
</tr>
<tr>
<td>3</td>
<td>10098</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>10111</td>
<td>65</td>
</tr>
</tbody>
</table>

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

$T_{\text{fail}} = 25$

Current time at P2: 75
Gossip Consistency

- Consistency guarantees are chosen *to suit the application*.
  - Sound familiar?
- The protocol discussed here *does not consider ordering*.
  - Could be: causal, FIFO, total, […]
- Higher consistency, ordering guarantees *increase cost*.
  - More packets exchanged
  - Updates may be delayed
  - (One may be traded for the other in some cases)
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
References


Acknowledgements

• These slides originally by Steve Ko, used by permission with minor modifications by Ethan Blanton
• These slides contain material developed and copyrighted by Indranil Gupta (UIUC).