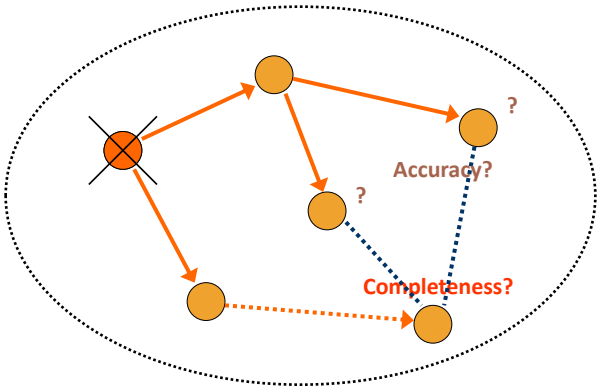
GROUP MEMBERSHIP (1/3)

82

82

Failure Detection



FLP Impossibility result: It is impossible to design a failure detector that is both complete and accurate in an asynchronous network [Chandra and Toueg 1990]

83

83

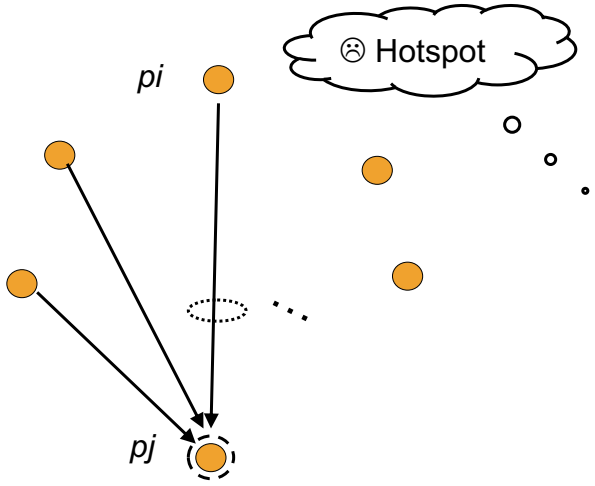
How to Proceed?

- Approximate ◊W with sufficiently long timeouts
  - Problem: latency
- Use probabilistic protocols
  - Solve consensus with high probability
- Change problem e.g. to group membership
  - Process group approach
- Accept consensus protocol that terminates with high probability
  - Paxos algorithm

84

84

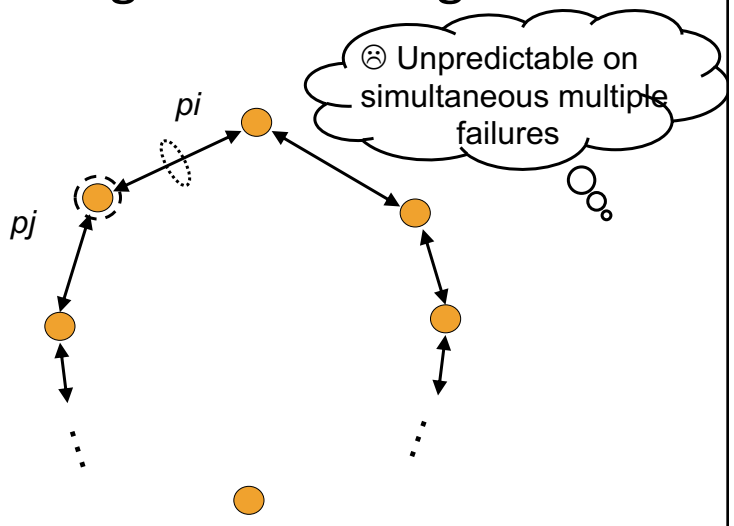
Centralized Heartbeating



85

85

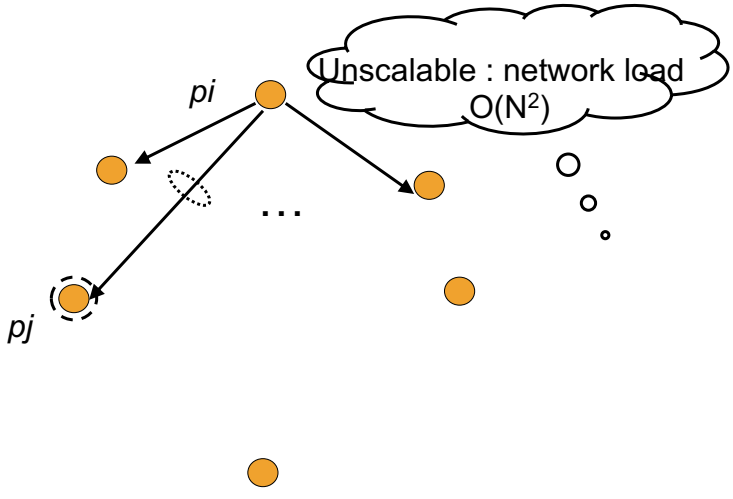
Ring Heartbeating



86

86

All-to-All Heartbeating



87

87

Gossip "epidemics"

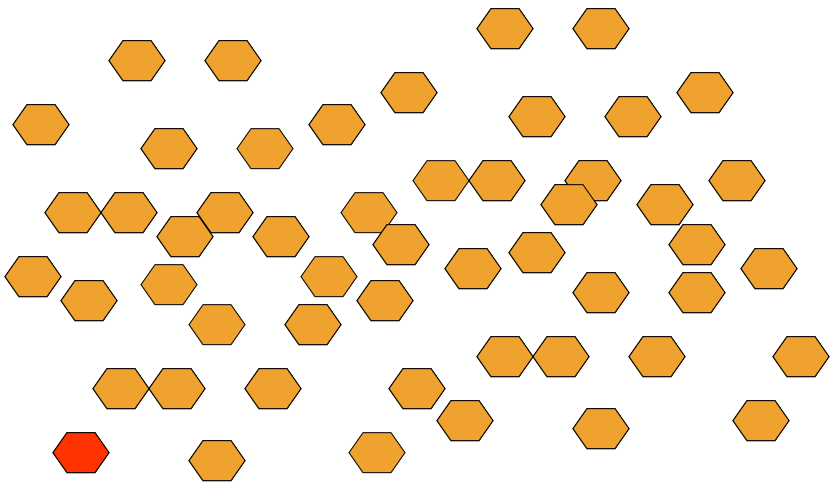
- [t=0] Suppose that I know something
- [t=1] I pick you… Now two of us know it.
- [t=2] We each pick … now 4 know it…

- Information spread: exponential rate.
  - Due to re-infection (gossip to an infected node) spreads as 1.8^k after *k* rounds
  - But in O(log(N)) time, N nodes are infected

88

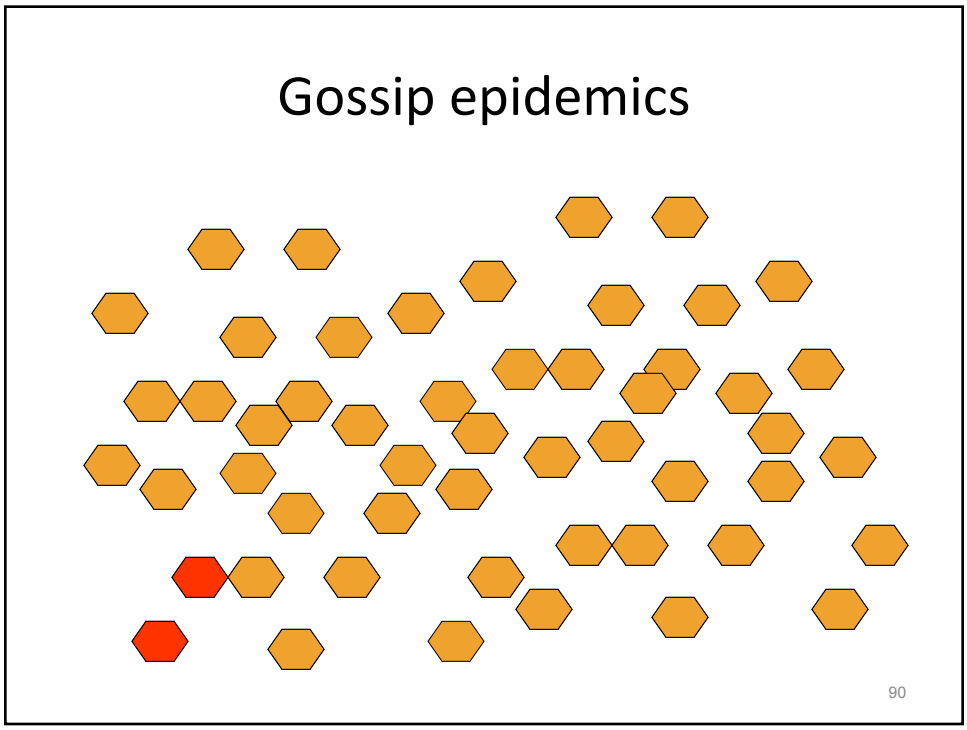
88

Gossip epidemics

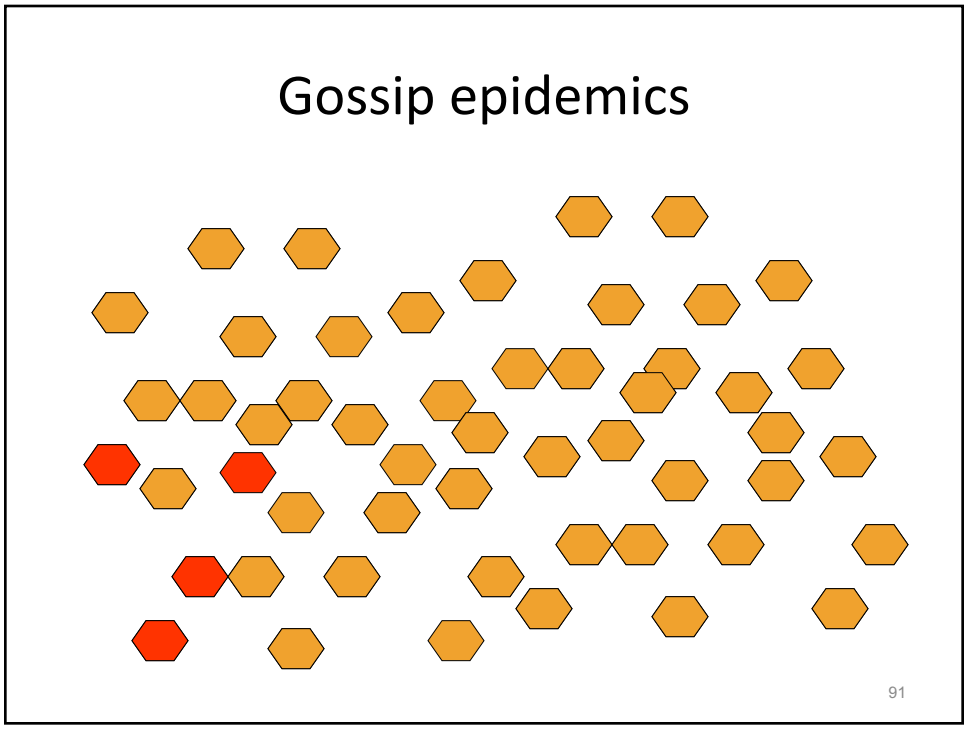


89

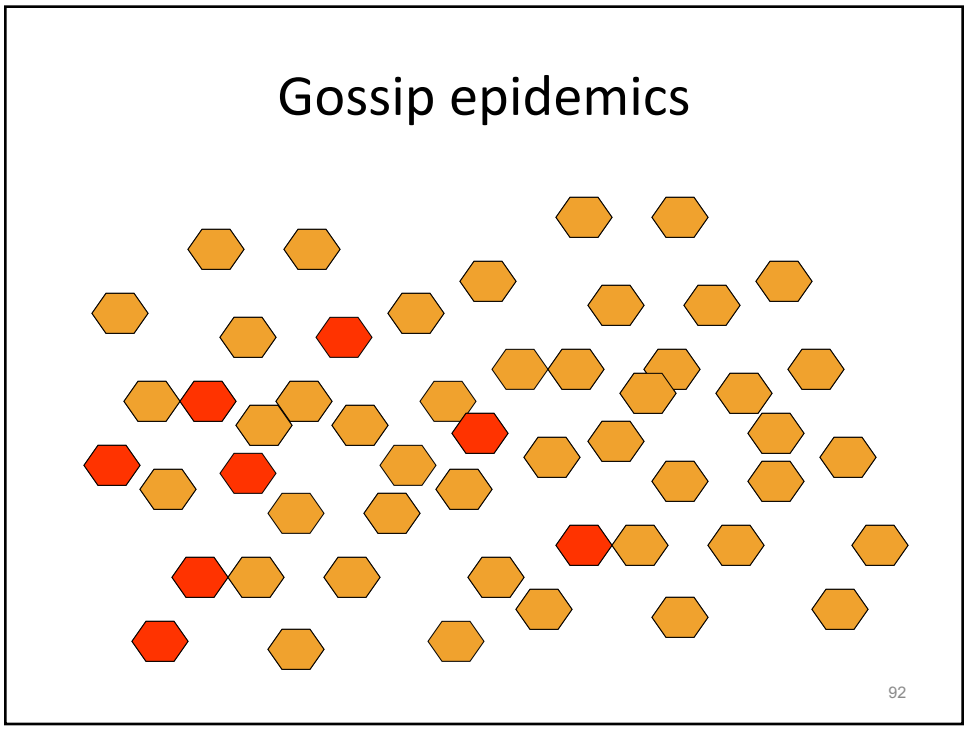
89



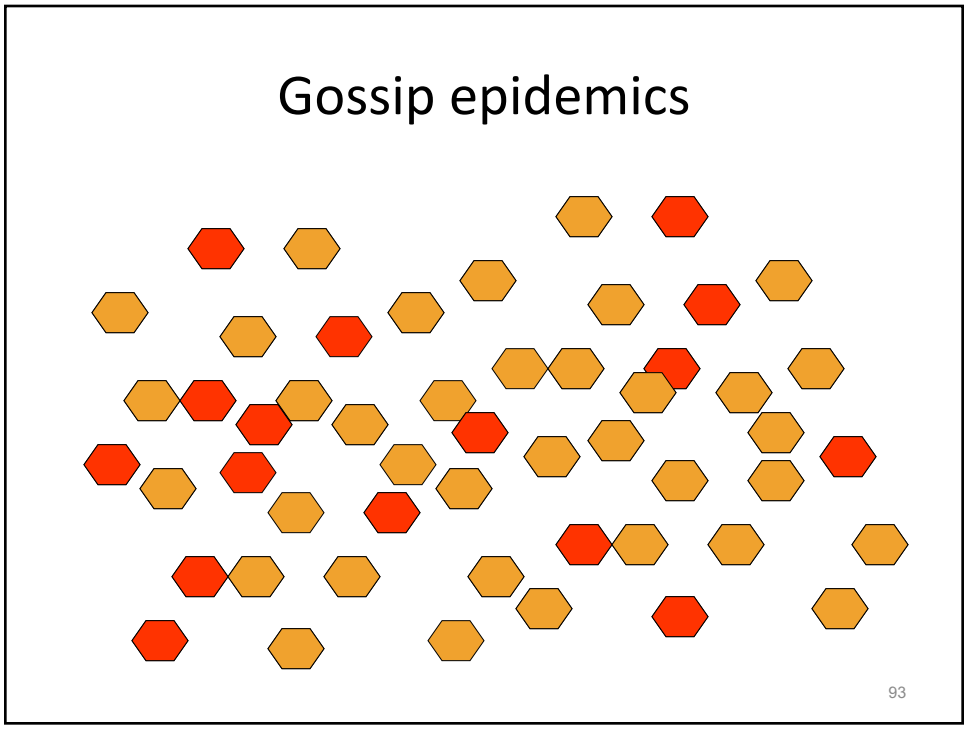
90



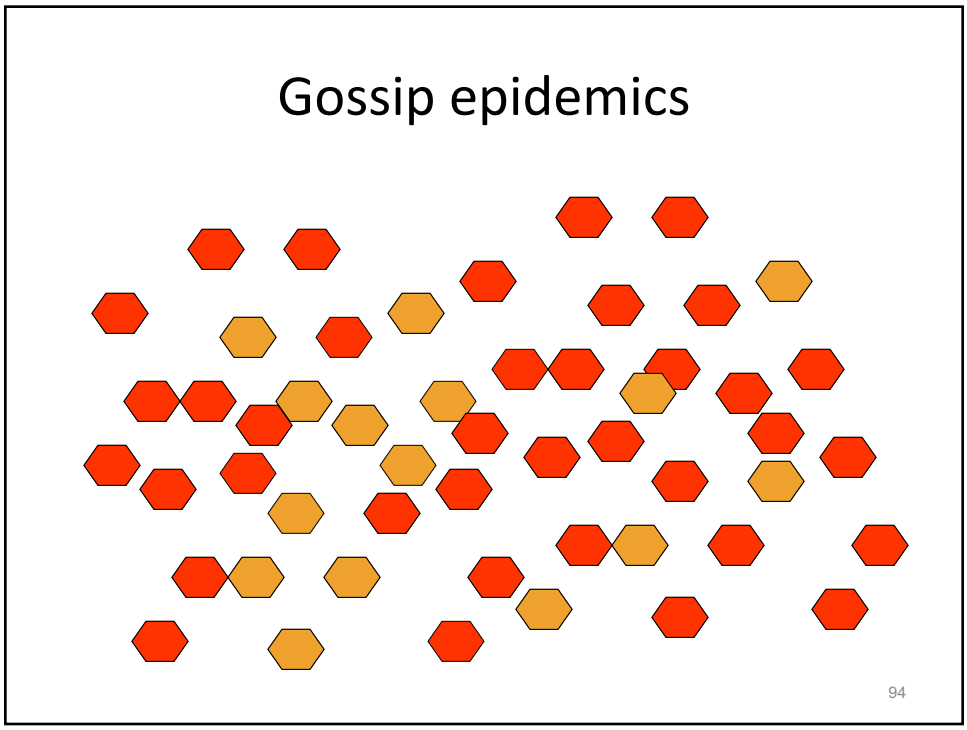
91



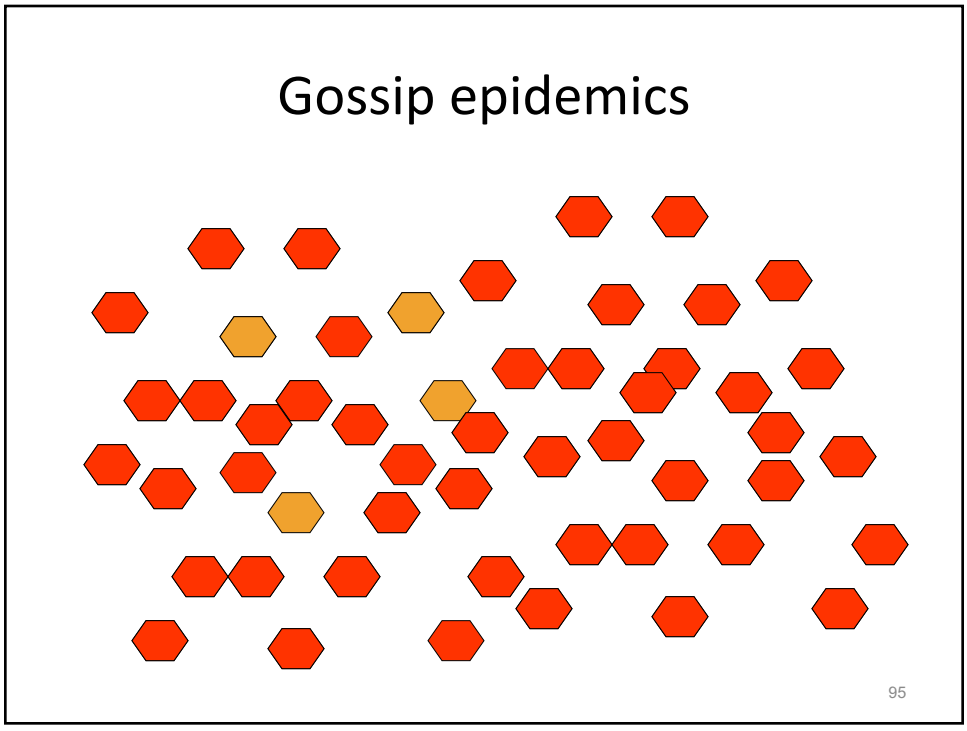
92



93

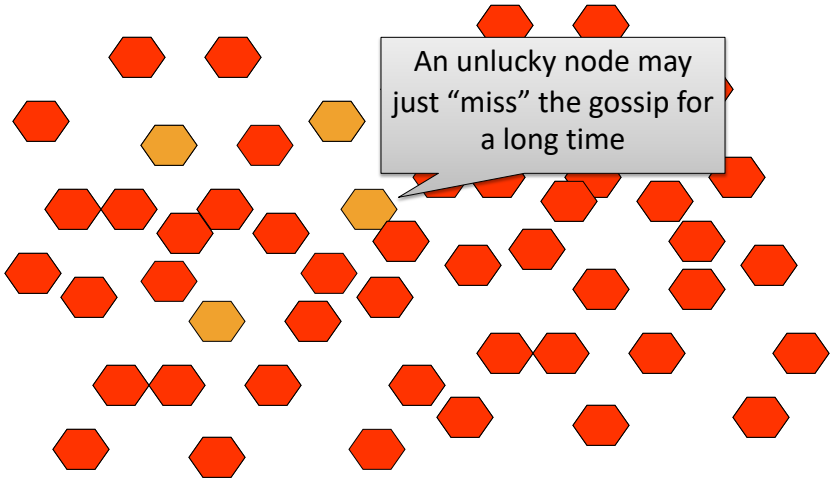


94



95

Gossip epidemics

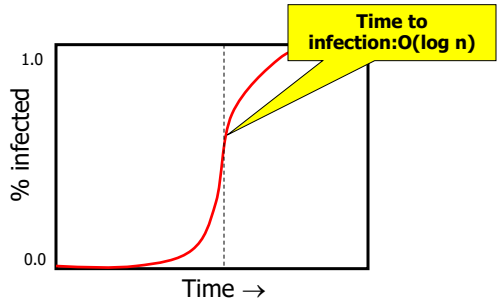


96

96

Gossip: scales nicely

- Participants’ loads independent of size
- Network load linear in system size
- Data spreads in log(system size) time



97

97

Facts about gossip epidemics

- Extremely robust
  - Data travels on exponentially many paths!
  - Hard to even slow it down…
    - Suppose 50% of packets are lost…
    - … 1 additional round!
  - Push-pull works best.
  - Many optimizations are needed in practice…

98

98

GROUP MEMBERSHIP (2/3)

99

99

Completeness & Accuracy

- Trivial algorithms
- Completeness :
  - declare all as failed (always)
- Accuracy :
  - declare all as alive (always)

100

100

Completeness & Accuracy

- In practice, most applications require
  - Completeness to <u>always</u> be guaranteed
    - *Eventual* consistency absolutely required
  - Accuracy guaranteed <u>most of the time</u> (probabilistically)
    - Performance degradation can be tolerated

101

101

Gossip-Based Failure Detection

- Scalable failure detection
  - Detection time : O(N log(N))
  - Network load per node : O(1)
- Detects all faulty nodes within a time bound
  - Time-bounded completeness
- Has a rate of false positives (probabilistic)

Failure Detection Protocol

- System Assumptions
  - No bound on message delivery
    - Most messages delivered in reasonable time
  - Failure model: Crash stop
  - Low clock drift
- Bird's Eye Protocol:
  - each member M_i sends out a heartbeat
  - heartbeat is disseminated using gossip
  - failure detection when time out waiting for M_i's next heartbeat

Basic Protocol

- Each member maintains a list (O(N)) of
  - $< M_i, H_i, T_{last,i} >$
  - M_i : member address
  - H_i : heartbeat count
  - $T_{last,i}$: last time of heartbeat increase
- Every T_{gossip}, each member
  - Increments its heartbeat
  - Selects a random target member (from its list) and sends to it a constant number of $<M_i, H_i>$ entries

104

104

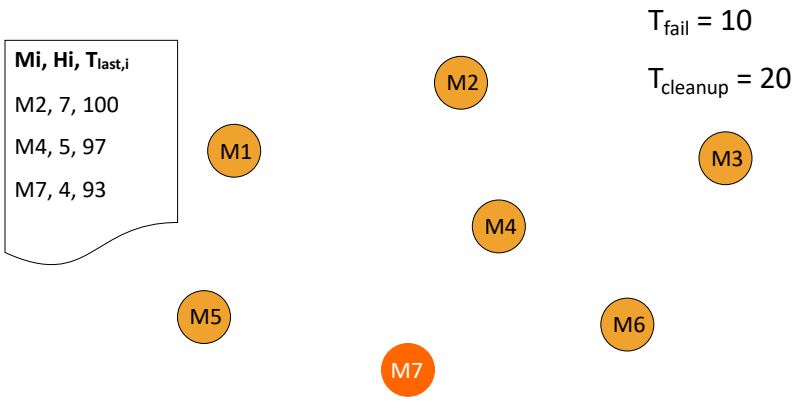
Basic Protocol

- A member, upon receiving gossip message,
  - Merges the list (maximum heartbeat)
- If $T_{last,i} + T_{fail} < T_{now}$,
  - Member M_i is considered failed
  - But remember M_i for $T_{cleanup}$ (~ $2*T_{fail}$), to prevent resurrection

105

105

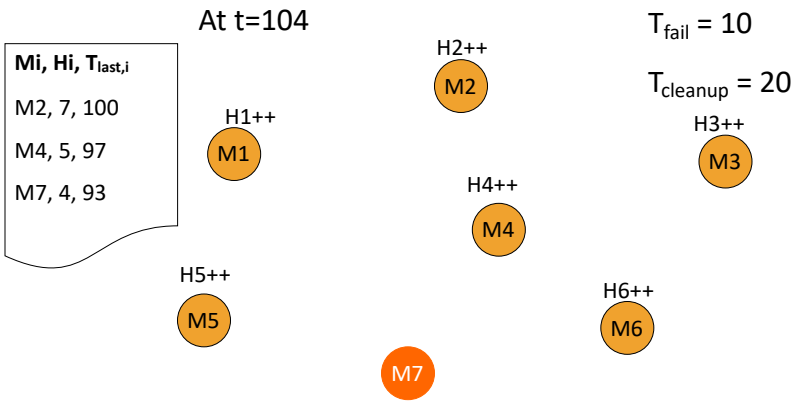
Basic Protocol



106

106

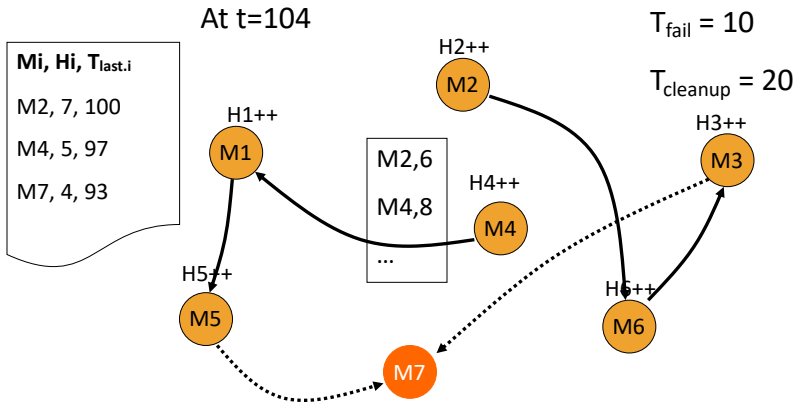
Basic Protocol



107

107

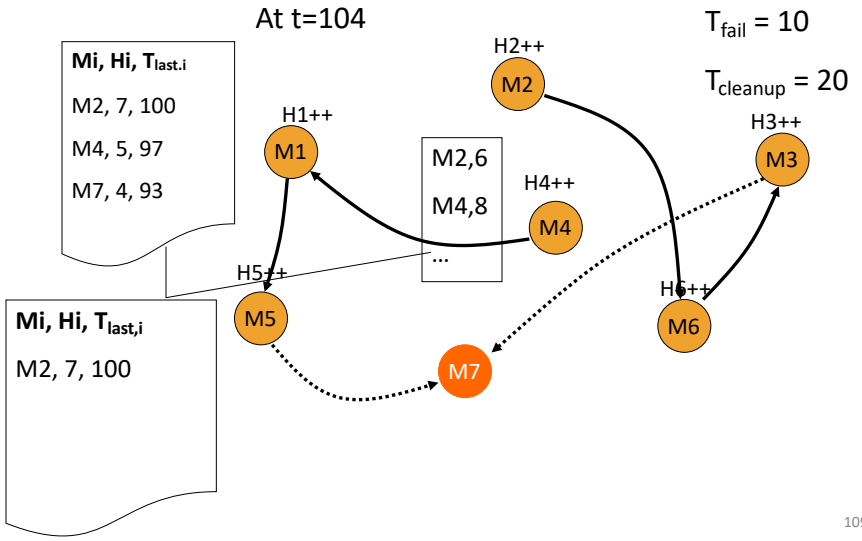
Basic Protocol



108

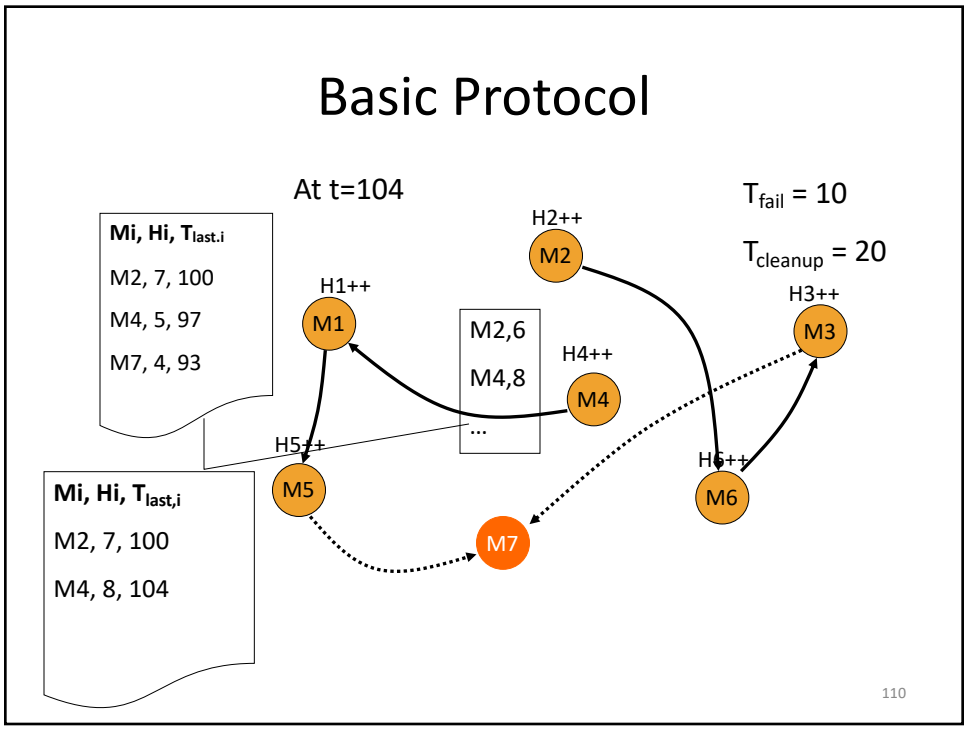
108

Basic Protocol

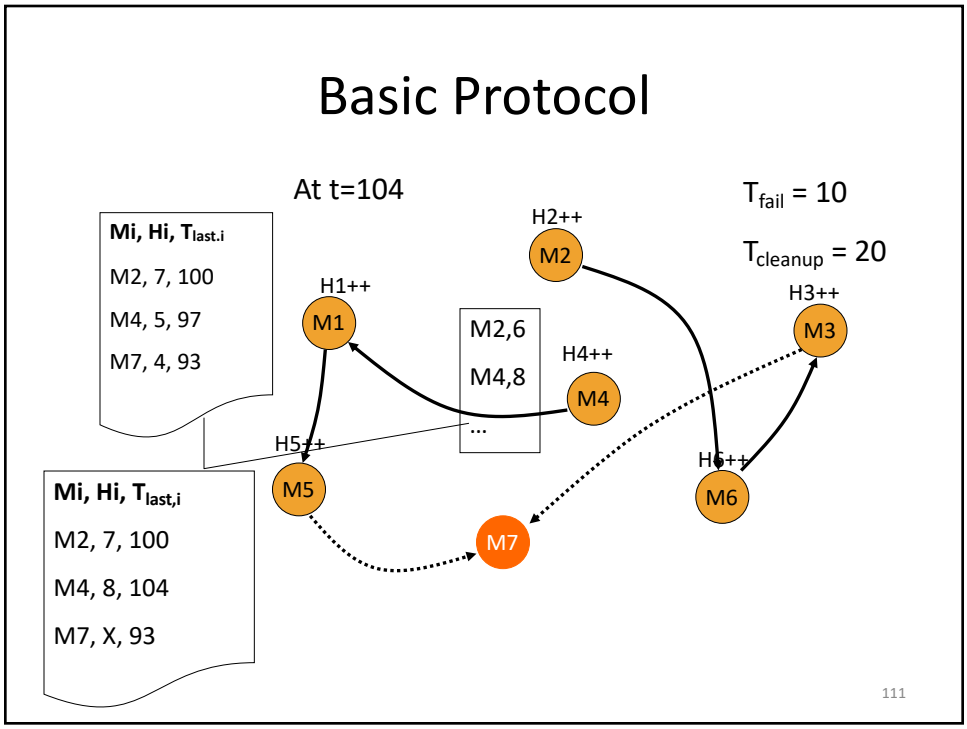


109

109


Basic Protocol
At t=104
T_{fail} = 10
$T_{cleanup}$ = 20
Mi, Hi, $T_{last,i}$
M2, 7, 100
M4, 5, 97
M7, 4, 93
H1++
M1
H2++
M2
H3++
M3
M2,6
M4,8
...
H4++
M4
H5++
M5
H6++
M6
M7
Mi, Hi, $T_{last,i}$
M2, 7, 100
M4, 8, 104
110

110


Basic Protocol
At t=104
T_{fail} = 10
$T_{cleanup}$ = 20
Mi, Hi, $T_{last,i}$
M2, 7, 100
M4, 5, 97
M7, 4, 93
H1++
M1
H2++
M2
H3++
M3
M2,6
M4,8
...
H4++
M4
H5++
M5
H6++
M6
M7
Mi, Hi, $T_{last,i}$
M2, 7, 100
M4, 8, 104
M7, X, 93
111

111

Analysis

- Detection Time = time to spread a gossip in a group of N nodes
  - O(log(N)) for a single gossip
  - But N such gossips being multicast
    - one heartbeat from each node
  - Since the actual message can carry only a constant number of heartbeats, the total dissemination is O(N log(N)).

Summary

Completeness	Eventual detection Expected detection time with known mistake
Accuracy	Probabilistic
Speed	Detection time : O(N log(N))
Scalability	Detection time : O(N log(N)) Network load : O(N) Per node overhead : O(1)
Resilience	Basic : resilient to message loss, # of failures Hierarchical : resilient to network partitions, large # of failures

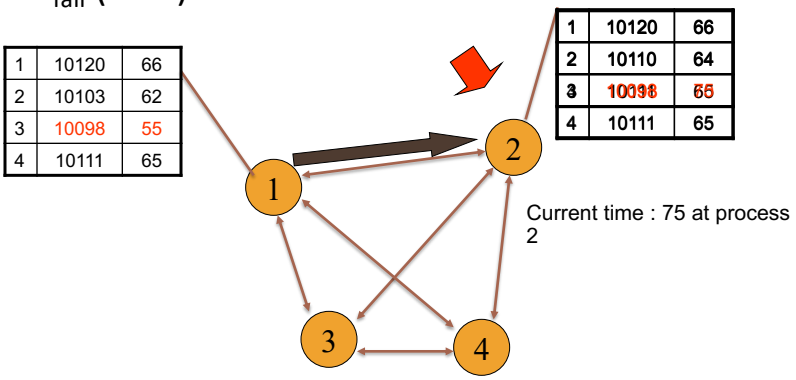
GROUP MEMBERSHIP (3/3)

Review: Gossip Protocol

- A member, upon receiving gossip message,
  - Merges the list (maximum heartbeat)
- If $T_{last,i} + T_{fail} < T_{now}$,
  - Member M_i is considered failed
  - But remember M_i for $T_{cleanup}$ (~ $2*T_{fail}$), to prevent resurrection

Review: Gossip Protocol

- What if an entry for failed process is deleted right after T_{fail} (= 24) seconds?

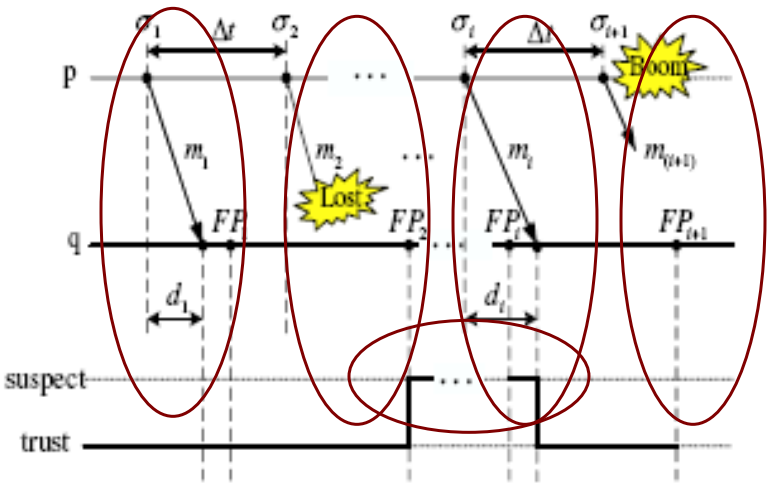


- Fix: remember for another T_{fail}

Suspicion Mechanism

- Goal: Reduce the frequency of false positives that might occur due to:
  - Network packet losses
  - Slow and unresponsive processes
- Key:
  - When a process is first detected as having failed, do not declare it as having failed
  - Instead, suspect the process first
  - Allow time to fix mistake

Suspicion Mechanism



118

118

Accrual Failure Detector

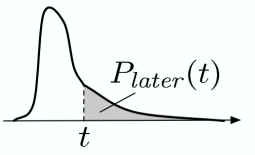
- Accrual Failure Detector
  - $\varphi(t)$: suspicion level at time t (for a node)
- Application sets a max suspicion level
  - Node declared failed otherwise
- Example: Cassandra/Dynamo
  - Set $\varphi(t) = 5 \Rightarrow$ 10-15 sec detection time
- Calculate $\varphi(t)$
  - Consider historical inter-arrival time of heartbeats

119

119

Accrual Failure Detector

- φ(t): suspicion level at time t



- $P_{later}(t)$: probability of heartbeat after *t* secs
- $P_{later}(t_{now} - t_{last})$: probability after "*now*"

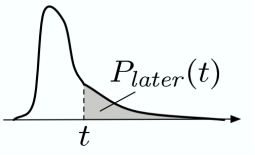
- Threshold φ: $P_{later}(t - t_{last}) < 1/10^{\varphi(t)}$
  - Threshold = 1 ⇒ 10% chance of mistake
  - Threshold = 2 ⇒ 1% chance of mistake
  - Threshold = 3 ⇒ 0.1% chance of mistake

120

120

Accrual Failure Detector

- φ(t): suspicion level at time t



- $P_{later}(t)$: probability of heartbeat after *t* secs
- $P_{later}(t_{now} - t_{last})$: probability after "*now*"

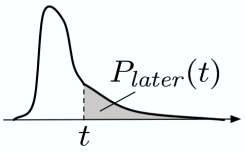
- Threshold $\varphi(t_{now}) = -\log_{10}(P_{later}(t_{now} - t_{last}))$
  - Threshold = 1 ⇒ 10% chance of mistake
  - Threshold = 2 ⇒ 1% chance of mistake
  - Threshold = 3 ⇒ 0.1% chance of mistake

121

121

Accrual Failure Detector

- φ(t): suspicion level at time t
- $P_{later}(t)$: probability of heartbeat after time *t*
  - Based on sampling of previous heartbeat timestamps
  - Assume normally distributed



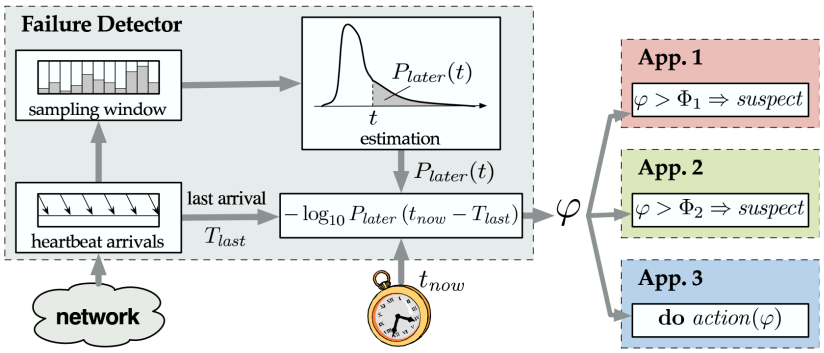
$$\varphi(t_{now}) = -log_{10}(P_{later}(t_{now} - T_{last}))$$

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_{t}^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = 1 - F(t)$$

122

122

Implementing Failure Detector



123

123

Summary

- To deal with failures:
  - What do you assume about the network
    - Asynchronous
    - Synchronous
  - What kinds of failures will you deal with
    - Fail-stop
    - Crash-stop
    - Byzantine
  - Know what is possible
- Practical applications
  - NoSQL databases e.g. Amazon Dynamo, Cassandra

Approaches to Membership

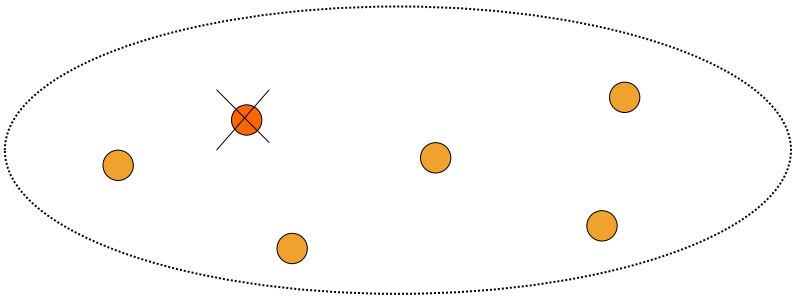
- Centralized
  - Doesn't scale
- Gossip
  - Slow dissemination
- SWIM*
  - Explicit detection and dissemination phases
  - Detection: ping/ack
  - Dissemination: epidemic spread piggybacked on pings and acks

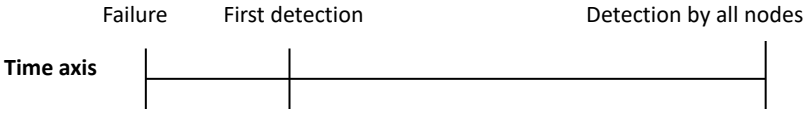
* Scalable weakly-consistent infection-style membership

126

126

Speed



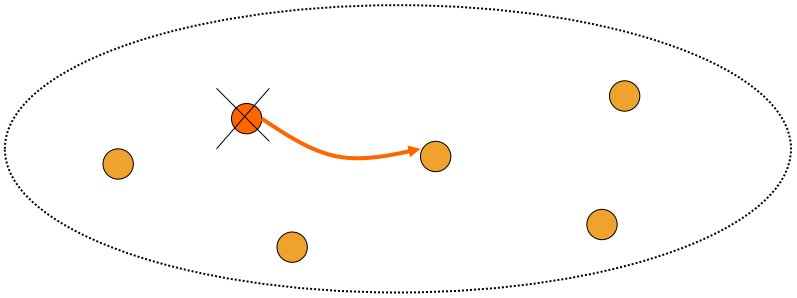


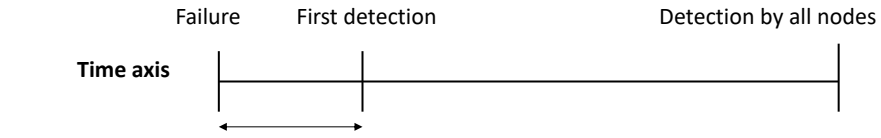
Detection time =

127

127

Speed



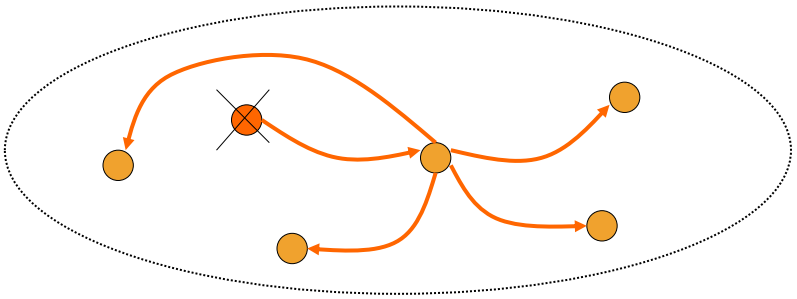


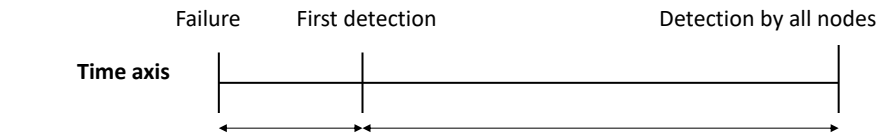
Detection time = First detection time

128

128

Speed





Detection time = First detection time + Dissemination time

129

129

SWIM* Failure Detector

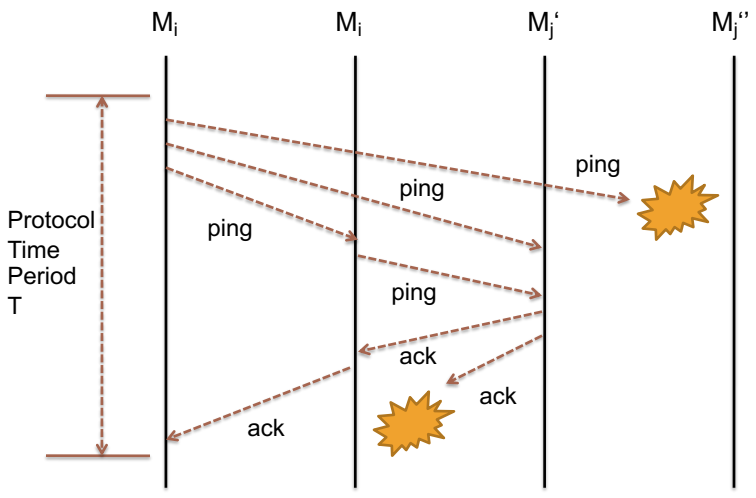
- Detection Phase (node M_i):
  - Each M_i node pings K random nodes, waits for acks
  - If no response from M_j, disseminate (M_i suspects M_j) messages
- Dissemination Phase (node M_k):
  - Piggyback messages on pings/acks (epidemic spread)
  - Mark M_j suspected if receive (M_i suspects M_j)
  - Disseminate (M_k knows M_j alive) if M_j acks ping
  - Disseminate (M_k declares M_j faulty) if M_j suspected and timeout before any ping-ack or keep-alive messages
    - Overrides any keep-alive messages at other nodes

* Scalable weakly-consistent infection-style membership

130

130

SWIM Failure Detection



131

131

Suspicion Mechanism

- Goal: Reduce the frequency of false positives that might occur due to:
  - Network packet losses
  - Slow and unresponsive processes
- Key:
  - When a process is first detected as having failed, do not declare it as having failed
  - Instead, suspect the process first
  - Allow time to fix mistake

132

132

Summary

Completeness	• Guaranteed (time-bounded) • Round-Robin probing strategy
Accuracy	• Partially / probabilistically guaranteed • Suspicion mechanism reduces the false positive rate
Speed	• First Detection time : expected constant time
Scalability	• Per node overhead : constant overhead • Dissemination latency grows slowly (logarithmically) with the group size.

133

133