Gossip and Self-Stabilization

Lonnie Princehouse

CS 5412

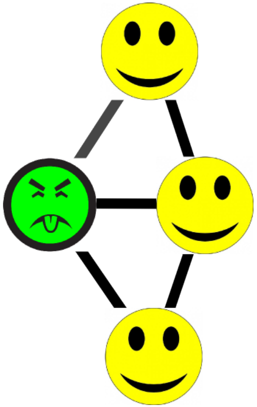
February 28, 2012

Gossip Protocols

Gossip is the family of protocols loosely characterized by

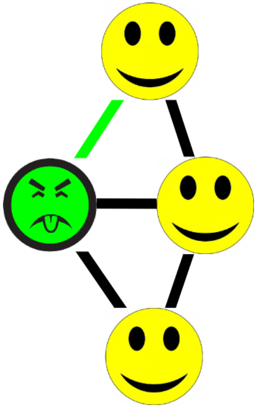
- Randomized peer selection
  - Probabilistic convergence
- Round-based execution
  - Not "reactive": messages only sent on a timer, not in response to stimuli
  - Predictable network load (good!) / high latency (bad!)
- Robust fault tolerance

AKA Epidemic Protocols



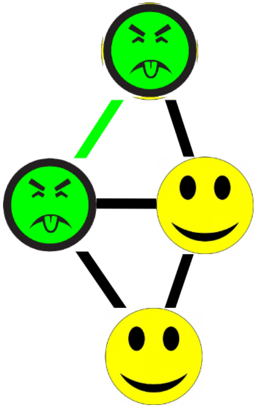
- Starting with an initial infected node

AKA Epidemic Protocols



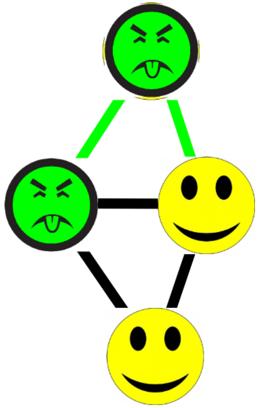
- Starting with an initial infected node
- Select a random neighbor

AKA Epidemic Protocols



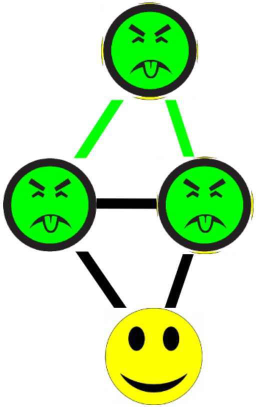
- Starting with an initial infected node
- Select a random neighbor
- Neighbor becomes infected

AKA Epidemic Protocols



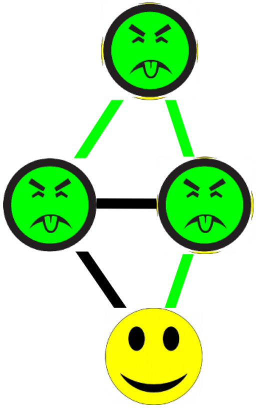
- Starting with an initial infected node
- Select a random neighbor
- Neighbor becomes infected
- Repeat

AKA Epidemic Protocols



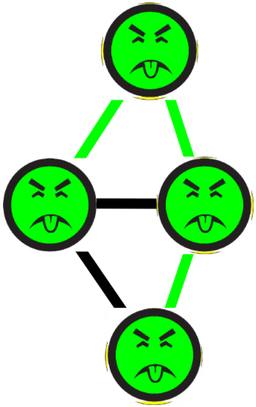
- Starting with an initial infected node
- Select a random neighbor
- Neighbor becomes infected
- Repeat

AKA Epidemic Protocols



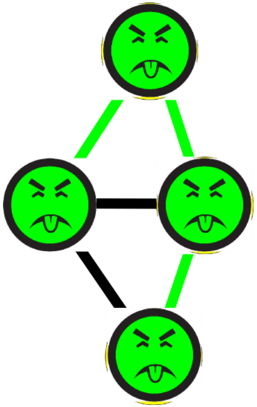
- Starting with an initial infected node
- Select a random neighbor
- Neighbor becomes infected
- Repeat

AKA Epidemic Protocols



- Starting with an initial infected node
- Select a random neighbor
- Neighbor becomes infected
- Repeat

AKA Epidemic Protocols



- Starting with an initial infected node
- Select a random neighbor
- Neighbor becomes infected
- Repeat

Intuition behind fault-tolerance:
Randomized peer selection makes it difficult to design gossip protocols that rely on a “critical path” of nodes

Simple Epidemic

- Assume a fixed population of size n
- Assume homogeneous spreading
  - Complete graph: Anyone can infect anyone with equal probability
- Assume k members already infected
- Infection occurs in rounds

Probability of Infection

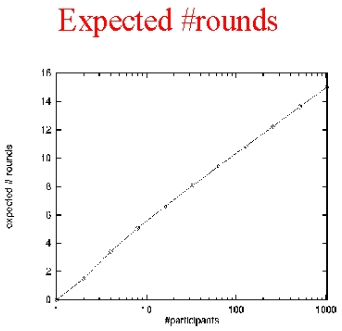
- Probability $P_{infect}(k,n)$ that a particular uninfected member is infected in a round if k are already infected

$$\begin{aligned} P_{infect}(k,n) &= 1 - P(\text{nobody infects members}) \\ &= 1 - (1 - 1/n)^k \end{aligned}$$

- $E(\#\text{ newly infected members}) = (n-k) \times P_{infect}(k,n)$

Rate of Simple Epidemic

- Infection
  - Initial growth factor very high
  - Exponential growth
- Number of rounds necessary to infect the entire population is $O(\log n)$
- For large n, $P_{infect}(n/2, n) \approx 1 - (1/e)^(1/2) \approx 0.4$



Expected # of Rounds vs. Participants [log scale]

Source: Ashish Motivala 2002

Gossip Applications

What are the commmon gossip applications?

- Rumor-Mongering
  - Broadcast and multicast
  - Sensor networks
    - Every node has a local sensor reading; the system records or aggregates these remote readings
  - Data center monitoring
- Anti-Entropy
  - Eventual consistency for sets of versioned objects
- Overlay maintenance and crash failure detection
  - E.g., "heartbeat" protocols

``...When an unauthorized movement is detected, an alert is sent to the base station which sends warning messages to the security office or whomever is responsible for that area. The security system relies on networks of cars constantly gossiping with their neighbors using the concealed wireless nodes. The cars raise the alarm when a thief tries to make a getaway...''

Anti-Entropy [Demers et. al '87]

Keeping a distributed database in sync with anti-entropy:

- Distributed database storing **versioned objects**
- Updates are (*key*, *value*, *version*) triplets
  - Broadcast update using gossip
  - Nodes update their stores when they receive an update with a newer version of a stored object

Overlay Maintenance

- Network overlays critical for many high performance distributed systems
- Must be maintained in the presence of churn: node arrival, departure, and failure
- Gossip's high latency often makes it a poor fit for the applications running on top of the overlay
- ... but ideally suited as a foundation for continually adjusting the overlay according to churn, due to its fault tolerance

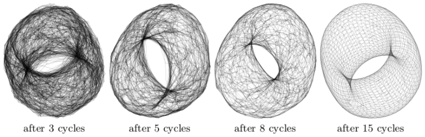


Fig. 2. Illustrative example of constructing a torus over $50 \times 50 = 2500$ nodes, starting from a uniform random topology with $c = 20$. For clarity, only the nearest 4 neighbors (out of 20) of each node are displayed.

`T-Man [Jelasity et. al] builds overlays according to custom biased weighting functions for neighbor preference.`

`This shows a toroidal overlay as it converges.`

Scaling Gossip

A Convenient Assumption

"Gossip with a random node, chosen from all nodes in the system"

- On the scale of P2P internet systems, or even large cloud computing datacenters, constant churn makes it impractical for every node to be aware of all other currently participating nodes.
- Instead, typically a node will know only about its **view** — those nodes adjacent to it in the communication graph.
- Generally, the view size is fixed or at most $log(n)$

Can we approximate truly uniform peer selection with only a subset of global membership?

Scaling Gossip

A Convenient Assumption

"Gossip with a random node, chosen from all nodes in the system"

- On the scale of P2P internet systems, or even large cloud computing datacenters, constant churn makes it impractical for every node to be aware of all other currently participating nodes.
- Instead, typically a node will know only about its **view** — those nodes adjacent to it in the communication graph.
- Generally, the view size is fixed or at most $log(n)$

Can we approximate truly uniform peer selection with only a subset of global membership? Yes. No. Maybe. (depends on the application)

Peer Sampling [Kermarrec et. al]

Random walk sampling

- Instead of choosing a neighbor directly, send out a random walk probe
- When the probe stops, its current location is the sampled peer
- **Discrete Time Random Walk**
  - Probes take a predetermined number of steps
- **Continuous Time Random Walk**
  - Probes flip a coin to decide if they should stop or keep going
  - Coin may be weighted, possibly even by properties of the current location, e.g., node degree
- Can be used for general sampling of any sensor data; not just view-building

Self-Stabilizing Protocols

"[Distributed sytems] have been designed, but all such designs I was familiar with were not "self-stabilizing" in the sense that, when once (erroneously) in an illegitimate state, they could – and usually did!– remain so forever."

- — Edsger Dijkstra proposed several *self-stabilizing* distributed systems in 1974
- (This was mostly ignored)
- Until 1983, when Leslie Lamport delivered a distributed computing keynote address concerning self-stabilization

Transient Faults in Distributed Systems

Transient Faults

Category of faults that affect the system only temporarily. After a transient fault, system is left with an arbitrary initial state

How can we handle transient faults?

Transient Faults in Distributed Systems

Transient Faults

Category of faults that affect the system only temporarily. After a transient fault, system is left with an arbitrary initial state

How can we handle transient faults?

- Ignore?

Transient Faults in Distributed Systems

Transient Faults

Category of faults that affect the system only temporarily. After a transient fault, system is left with an arbitrary initial state

How can we handle transient faults?

- Ignore?
  - ...and leave our system in a perpetually broken state?!

Transient Faults in Distributed Systems

Transient Faults

Category of faults that affect the system only temporarily. After a transient fault, system is left with an arbitrary initial state

How can we handle transient faults?

- Ignore?
  - ...and leave our system in a perpetually broken state?!
- Detect and repair?

Transient Faults in Distributed Systems

Transient Faults

Category of faults that affect the system only temporarily. After a transient fault, system is left with an arbitrary initial state

How can we handle transient faults?

- Ignore?
  - ...and leave our system in a perpetually broken state?!
- Detect and repair?
  - Harder than it sounds! (see next slide)

Transient Faults in Distributed Systems

Transient Faults

Category of faults that affect the system only temporarily. After a transient fault, system is left with an arbitrary initial state

How can we handle transient faults?

- Ignore?
  - ...and leave our system in a perpetually broken state?!
- Detect and repair?
  - Harder than it sounds! (see next slide)
- Design our systems to gracefully tolerate them

Transient Faults in Distributed Systems

Transient Faults

Category of faults that affect the system only temporarily. After a transient fault, system is left with an arbitrary initial state

How can we handle transient faults?

- Ignore?
  - ...and leave our system in a perpetually broken state?!
- Detect and repair?
  - Harder than it sounds! (see next slide)
- Design our systems to gracefully tolerate them
  - Self-stabilizing systems are always moving towards a correct state
  - System isn't "aware" of faults, but repairs damage nonetheless

The Trouble with Error Detection

- Using only local knowledge—a node and its immediate neighbors—we may not be able to detect faulty global state
- Trying to track properties of global state in a distributed system is impractical
  - Does not scale

Self-Stabilizing System: Definition

Define a set of *legitimate* system states. The two defining properties of a self-stabilizing system are:

Convergence

Starting from an arbitrary initial state, the system eventually reaches a legitimate state.

Self-Stabilizing System: Definition

Define a set of *legitimate* system states. The two defining properties of a self-stabilizing system are:

Convergence

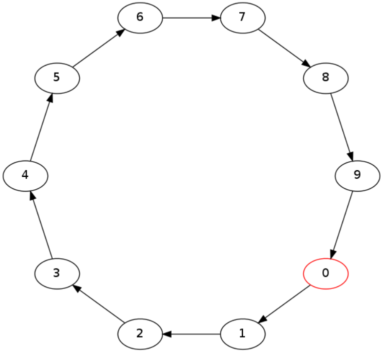
Starting from an arbitrary initial state, the system eventually reaches a legitimate state. Worst-case convergence time $O(n^2)$ rounds

Closure

Once in a legitimate state, the system remains in a legitimate state in the absence of faults.

Example: Dijkstra's Token Ring Mutual Exclusion

- $N+1$ processes labeled $0, ..., N$
- processes are arranged in a ring, such that each node i can only see its predecessor $i - 1 \ mod \ N$
- Each process i has a counter C_i in the range $\{0, ..., K\}$ for $K \geq N+1$
- For each process i, define a boolean function *privilege*$_i(C_{i-1}, C_i)$
    - Goal: *privilege*$_i$ true for only one process at a time, and it rotates around the ring
- Legitimate states: *privilege*$_i$ true for exactly one process i
- Legal executions: Privilege moves in the ring from process i to its successor $(i+1) \ mod \ K$



Example: Dijkstra's Token Ring Mutual Exclusion

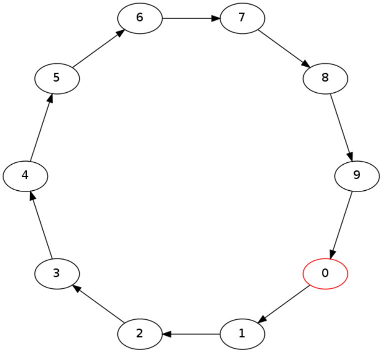
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1) \; mod \; K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

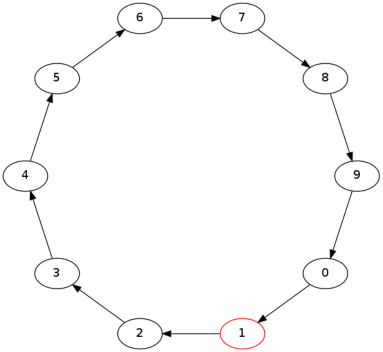
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

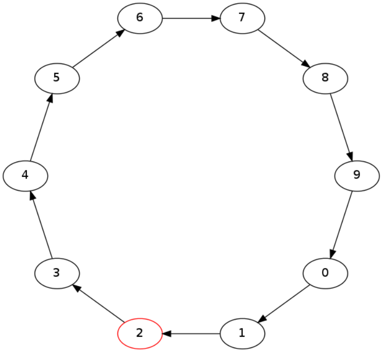
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

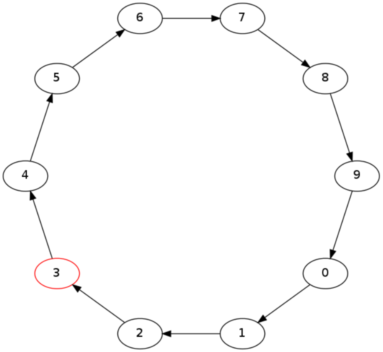
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

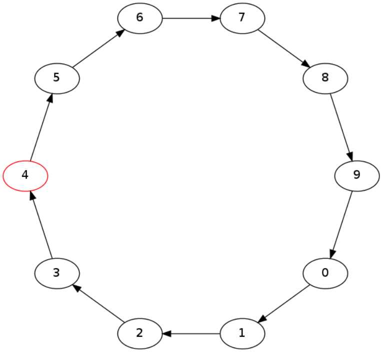
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

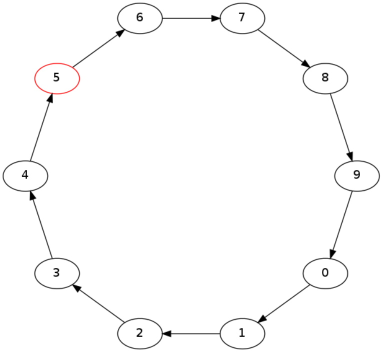
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)$ *mod K*

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

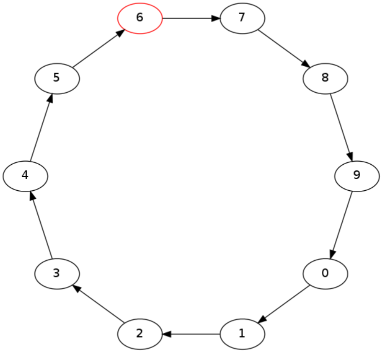
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

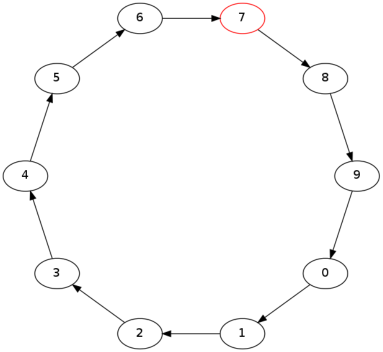
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1) \ mod \ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

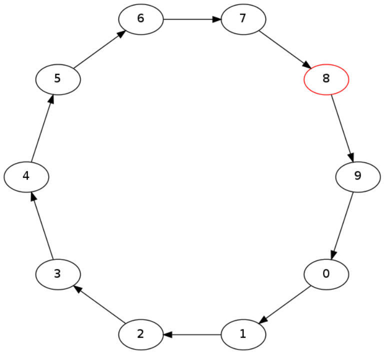
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

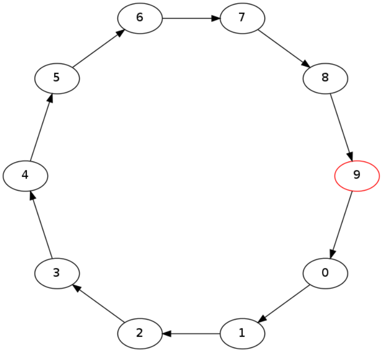
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

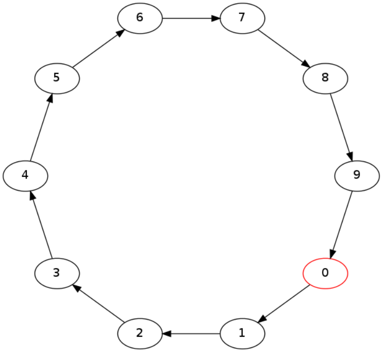
Execution

Process 0

if $C_0 = C_N$ then $C_0 \leftarrow (C_0 + 1)\ mod\ K$

All other processes

if $C_i \neq C_{i-1}$ then $C_i \leftarrow C_{i-1}$



Example: Dijkstra's Token Ring Mutual Exclusion

Convergence

Does it converge from an arbitrary initial state, in the absence of faults?

Example: Dijkstra's Token Ring Mutual Exclusion

Convergence

Does it converge from an arbitrary initial state, in the absence of faults?

- ✓ *Yes.* Eventually, C_0 will increment to a value not contained in the arbitrary initial state. This value will be copied all around the ring, at which point we reach a legitimate state with process 0 holding the token.

Example: Dijkstra's Token Ring Mutual Exclusion

Convergence

Does it converge from an arbitrary initial state, in the absence of faults?

- ✓ *Yes.* Eventually, C_0 will increment to a value not contained in the arbitrary initial state. This value will be copied all around the ring, at which point we reach a legitimate state with process 0 holding the token.

Closure

Is it impossible to reach an illegimate state from a legitimate state in the absence of faults?

Example: Dijkstra's Token Ring Mutual Exclusion

Convergence

Does it converge from an arbitrary initial state, in the absence of faults?

- ✓ *Yes.* Eventually, C_0 will increment to a value not contained in the arbitrary initial state. This value will be copied all around the ring, at which point we reach a legitimate state with process 0 holding the token.

Closure

Is it impossible to reach an illegimate state from a legitimate state in the absence of faults?

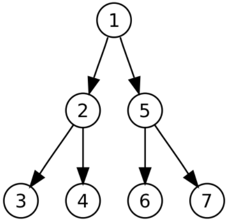
- ✓ *Yes.* Execution always moves the token one step forward on the ring.

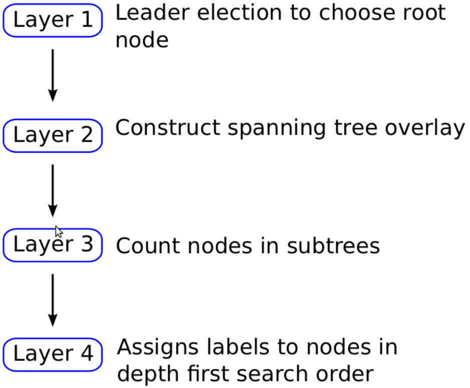
Further Self-Stabilization

- Dijkstra's 1974 paper offered two more self-stabilizing examples
- He speculated that there is no uniform solution, i.e., there is no distinguished process like process 0 in the example
  - Actually, it's possible for rings of prime size [Burns-Pachl]

Self-Stabilizing Layers

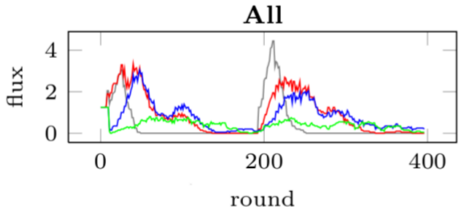
We can layer self-stabilizing protocols. If protocol P_2's convergence is predicated on P_1, running them both together results in a composite self-stabilizing protocol.




Layer 1
Leader election to choose root node
Layer 2
Construct spanning tree overlay
Layer 3
Count nodes in subtrees
Layer 4
Assigns labels to nodes in depth first search order

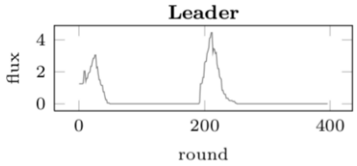
Self-Stabilizing Layers

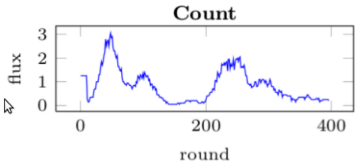
Worst-case time to converge is the sum of each layer's convergence time, but average convergence time is much better

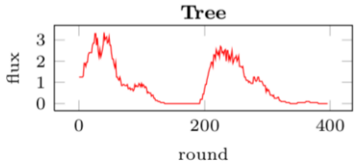


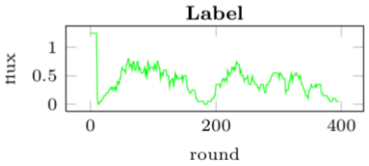
flux=rate of state change (nodes/round)

Correlated merge yields 9% fewer messages









End

Questions?

End

Questions?
Quiz