Data Dissemination: from Academia to Industry

João Leitão
NOVA-LINCS & NOVA University of Lisbon
Jordan West
BASHO Inc.

RICON
Las Vegas
October 2014

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

Classical Distributed Systems Challenge

How to disseminate information across a large number of participants?

- Some intuitive requirements:
  - Reliable.
  - Efficient.
  - Scalable.

Classical Distributed Systems Challenge

How to disseminate information across a large number of participants?

- Some intuitive requirements:
  - Reliable.
  - Efficient.
  - Scalable.

Applications

- Notification systems.
- Streaming multimedia content.
- *Cluster Management*.

In practice...

When I started to think about this problem, I was mostly focused on peer-to-peer systems.

Applications

- Notification systems.
- Streaming multimedia content.
- *Cluster Management*.

In practice...

When I started to think about this problem, I was mostly focused on peer-to-peer systems.

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

The Academic View: Data Dissemination

Design Alternatives: One to All

Lets start simple...

If you have information to disseminate, send it to everyone!

One to All

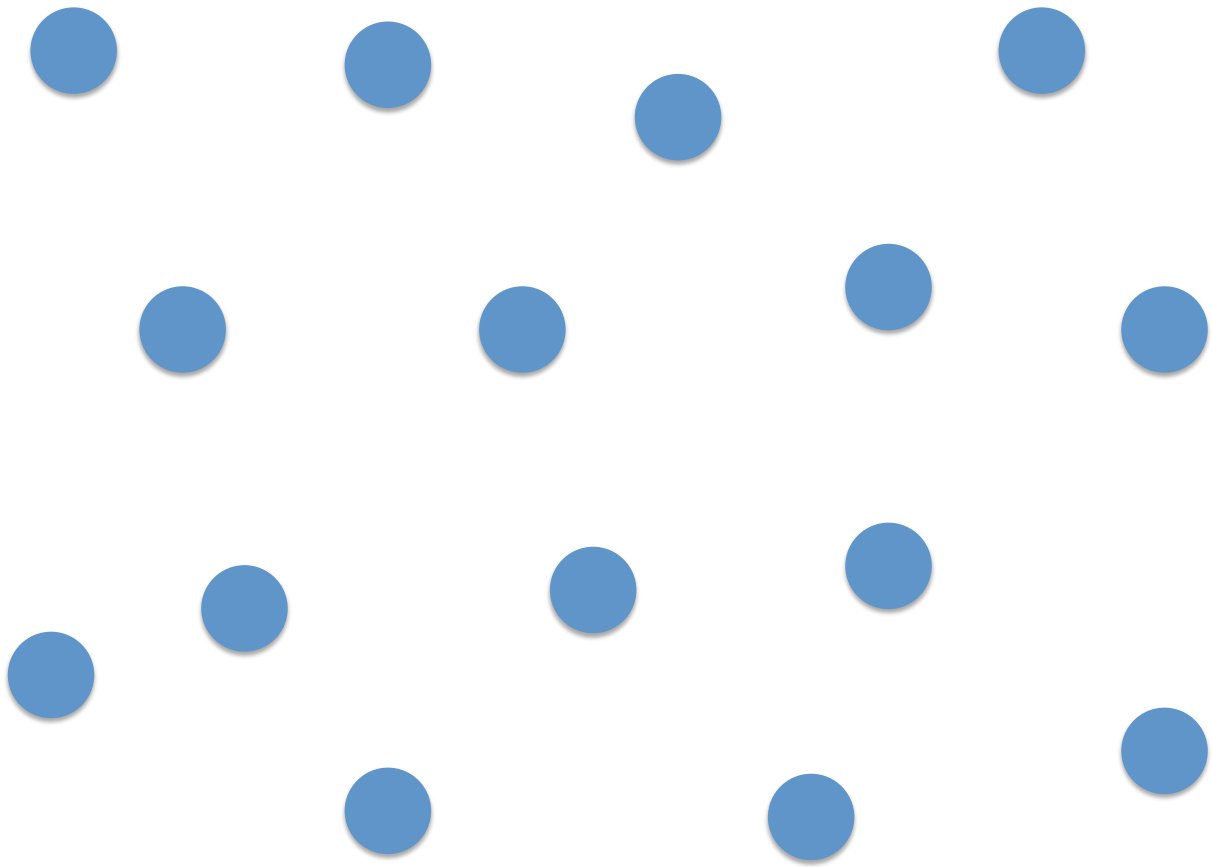
Lets start simple...

If you have information to disseminate, send it to everyone!

One to All

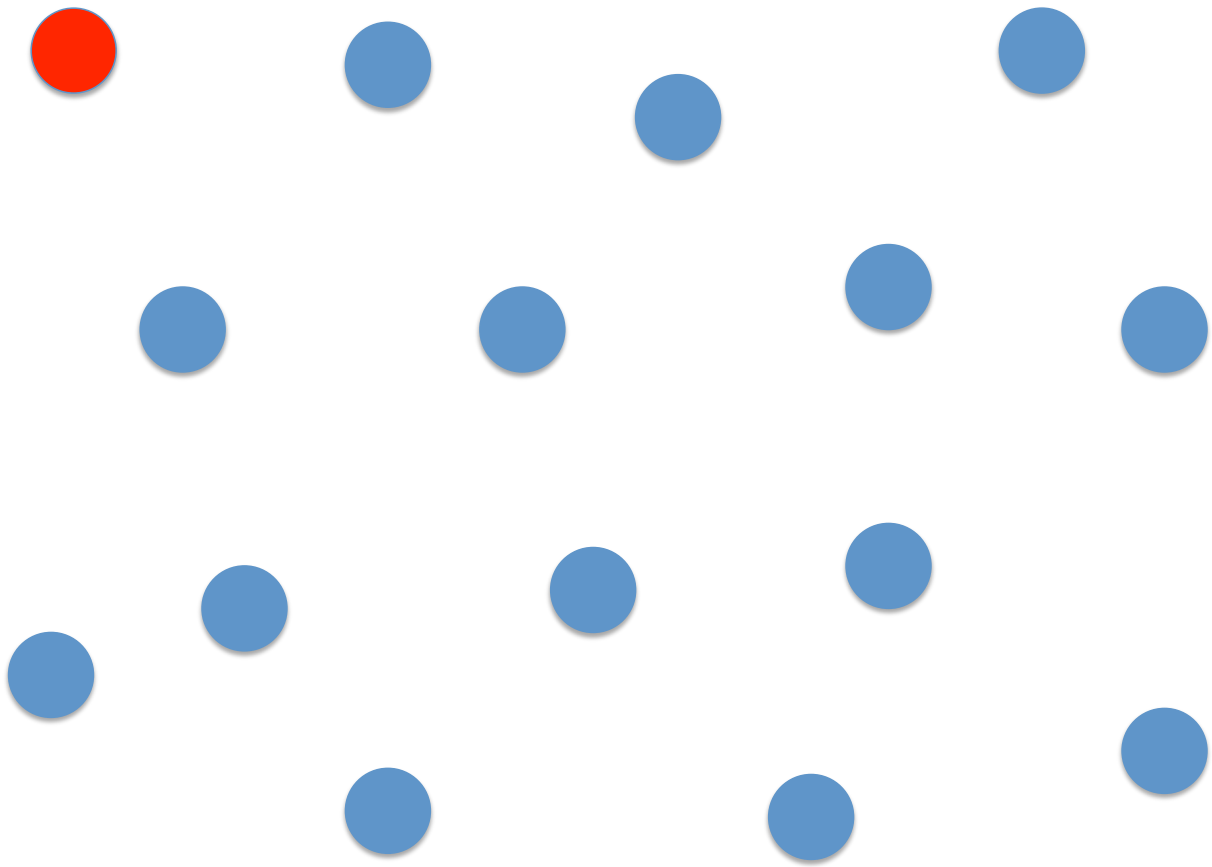
The Academic View: Data Dissemination

Design Alternatives: One to All



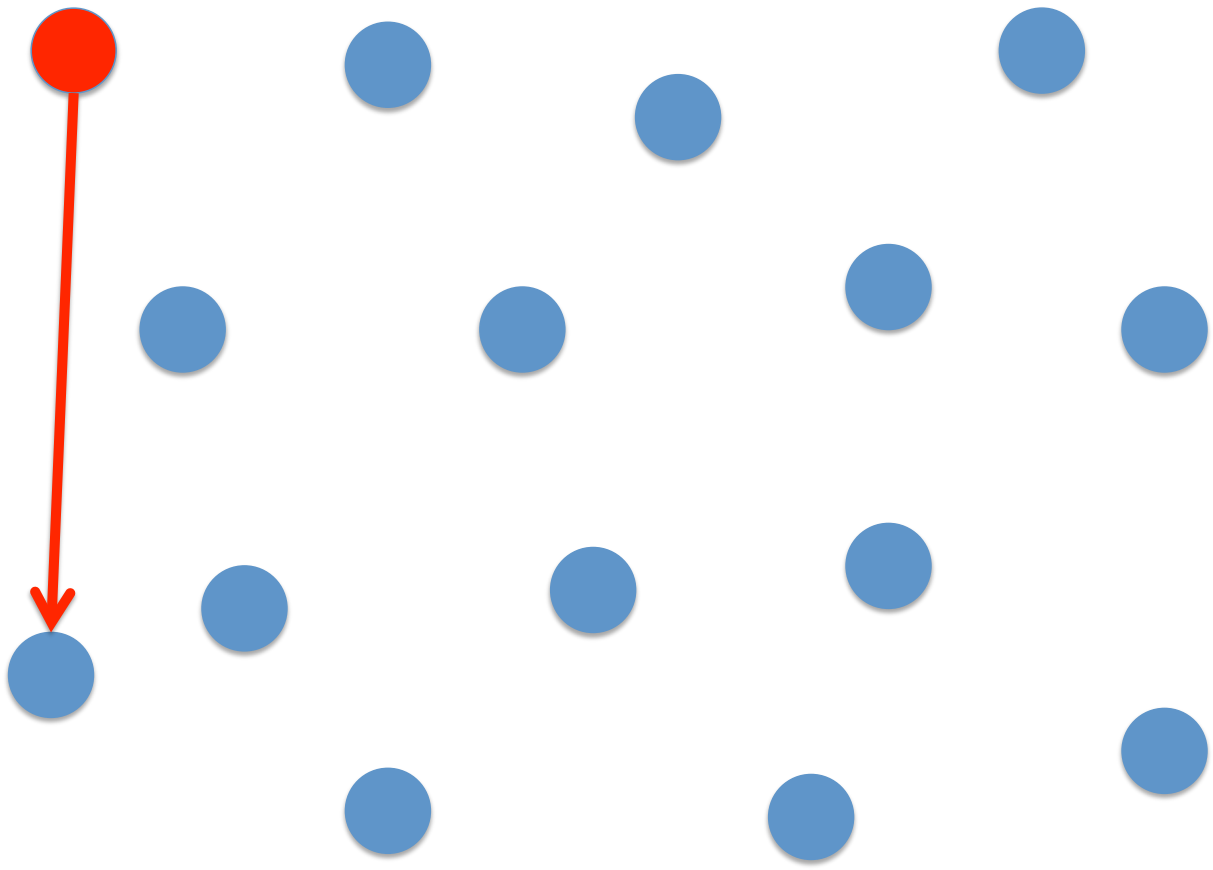
The Academic View: Data Dissemination

Design Alternatives: One to All



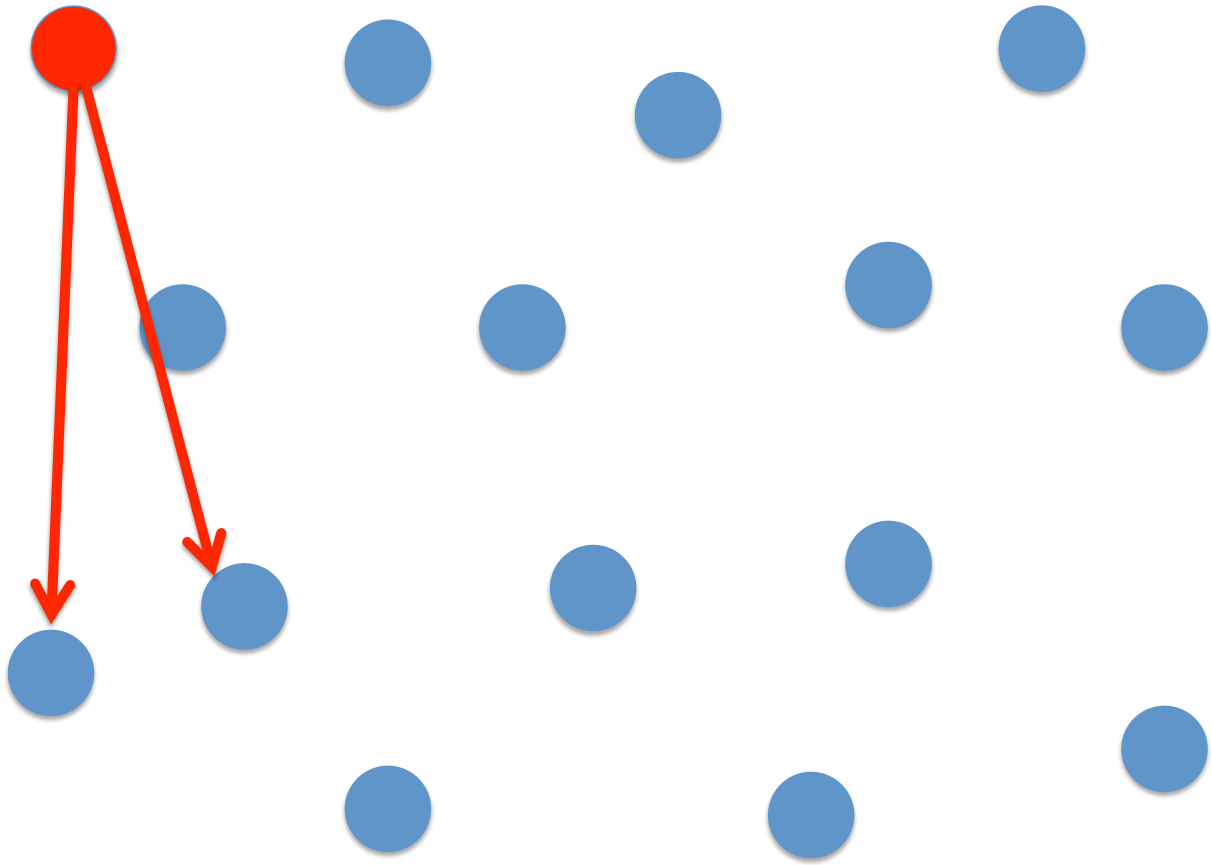
The Academic View: Data Dissemination

Design Alternatives: One to All



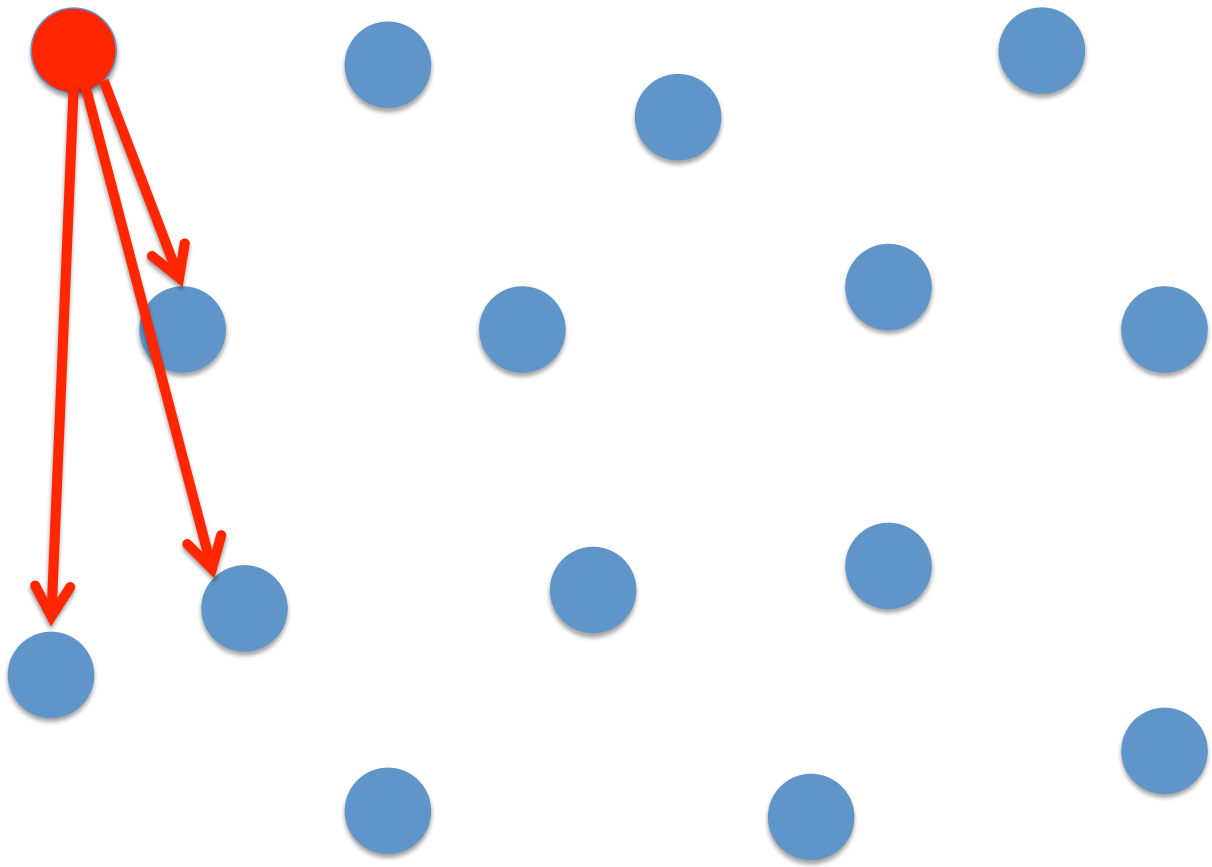
The Academic View: Data Dissemination

Design Alternatives: One to All



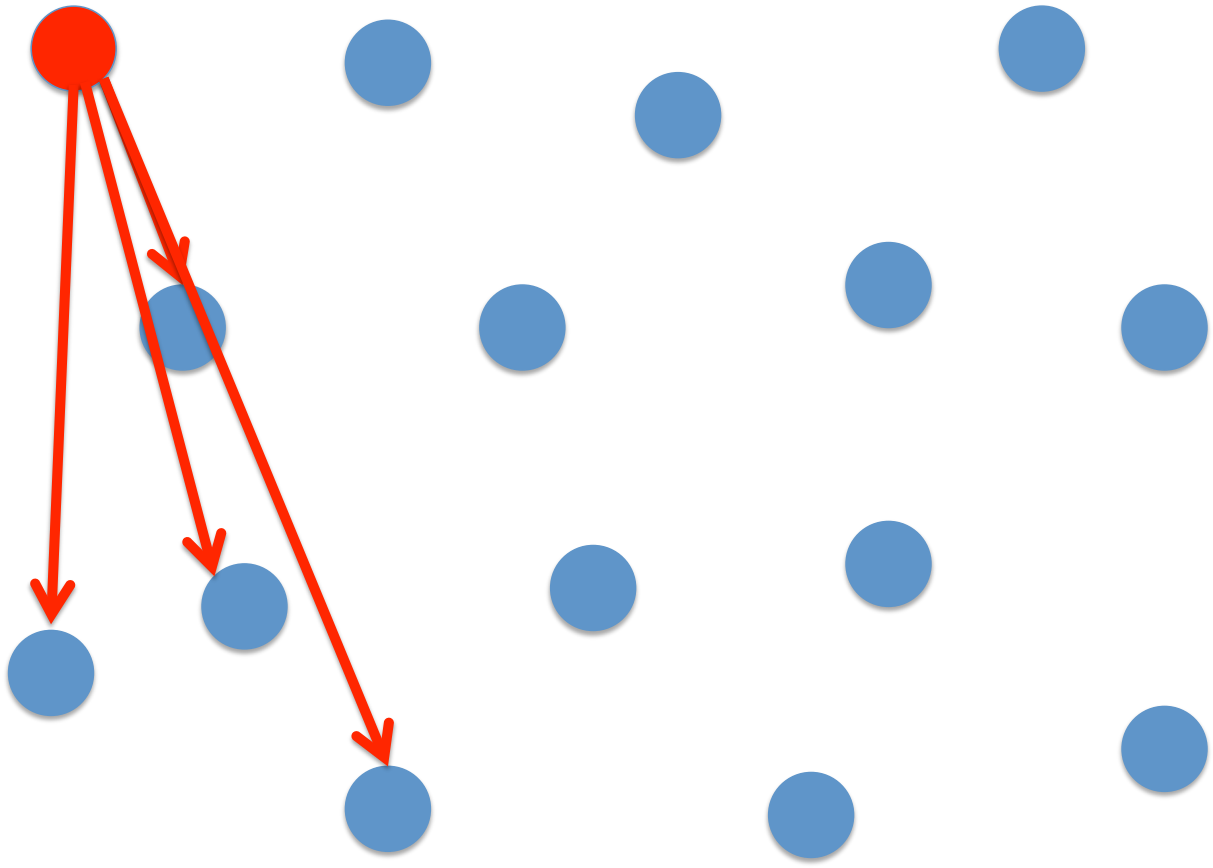
The Academic View: Data Dissemination

Design Alternatives: One to All



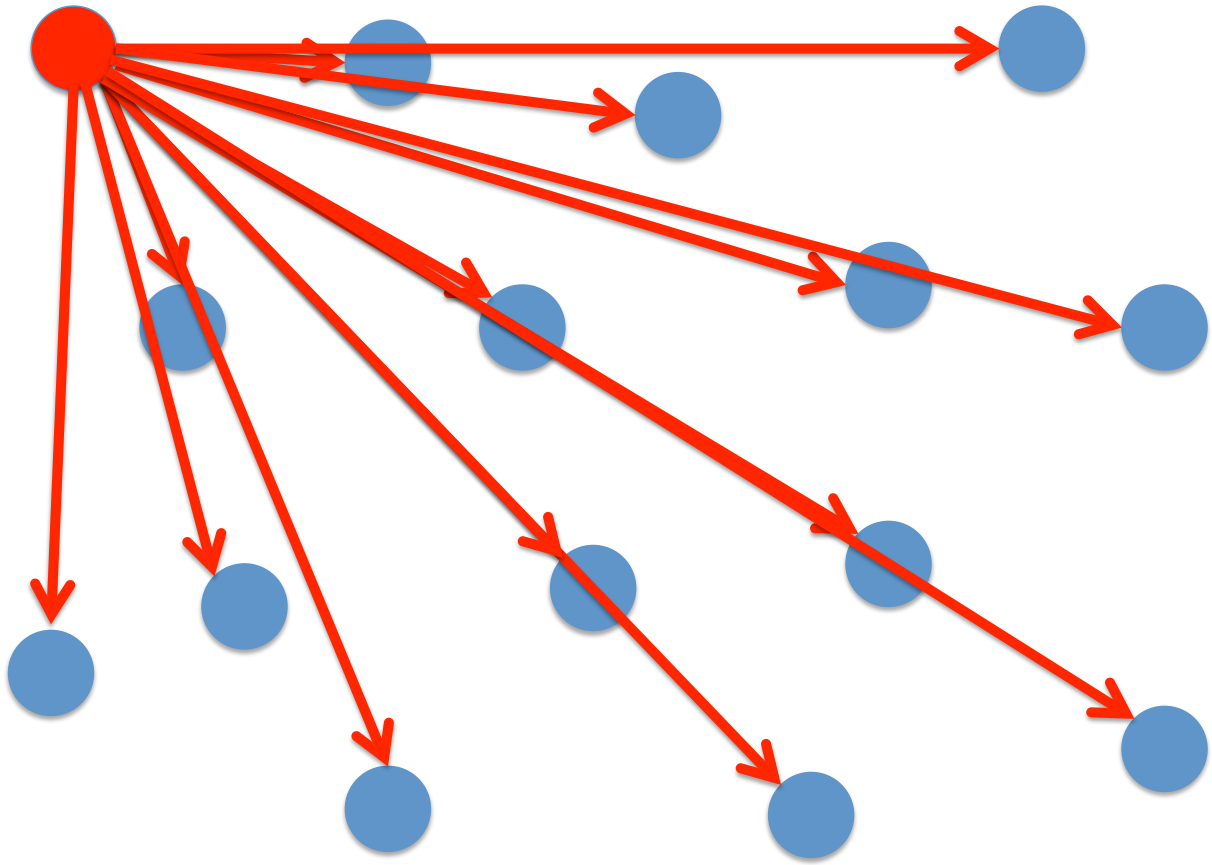
The Academic View: Data Dissemination

Design Alternatives: One to All



The Academic View: Data Dissemination

Design Alternatives: One to All



The Academic View: Data Dissemination

Design Alternatives: One to All

One to All

- Positive Aspects:
  - Straight forward.
  - No redundancy (i.e, each node receives each message a single time).
- Negative Aspects:
  - Requires each node to know the full membership.
  - Not Scalable (no load distribution).
  - Not Resilient to Faults.

The Academic View: Data Dissemination

Design Alternatives: One to All

One to All

- Positive Aspects:
  - Straight forward.
  - No redundancy (i.e, each node receives each message a single time).
- Negative Aspects:
  - Requires each node to know the full membership.
  - Not Scalable (no load distribution).
  - Not Resilient to Faults.

The Academic View: Data Dissemination

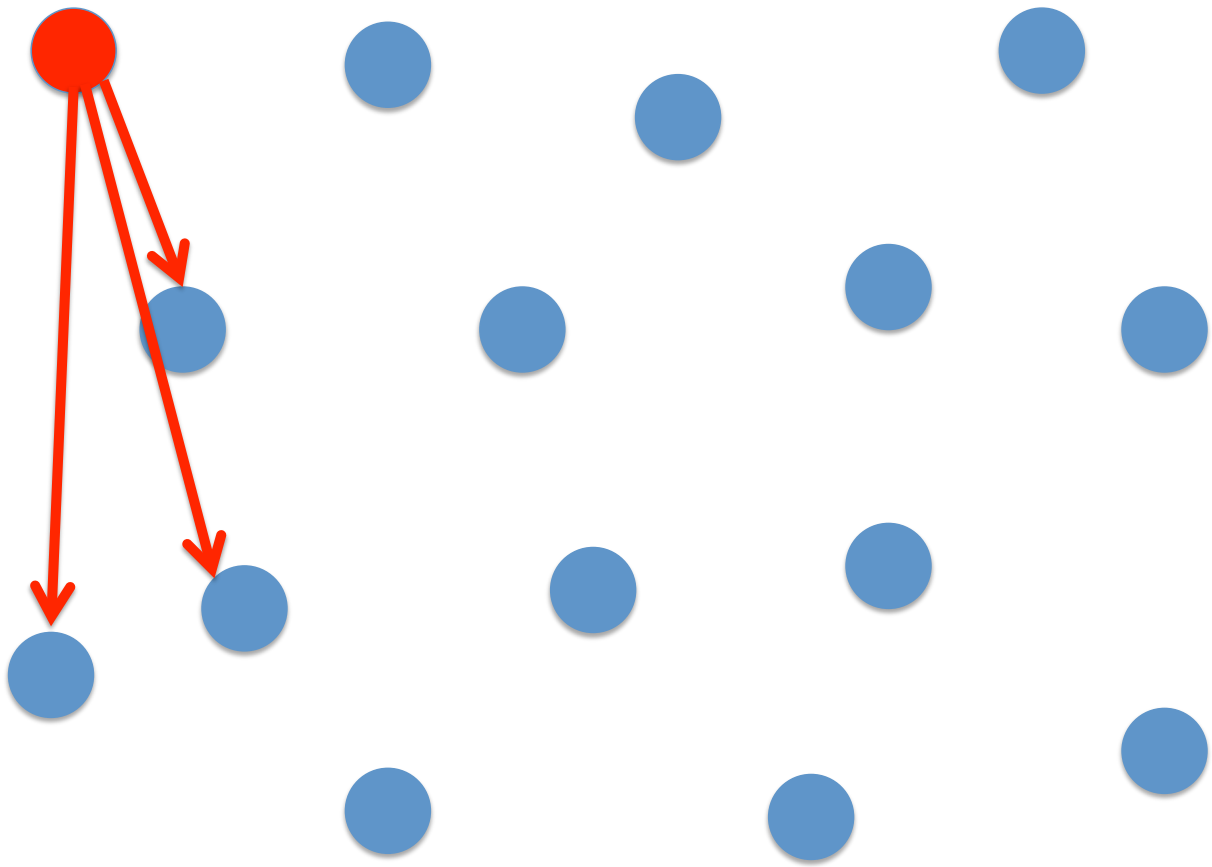
Design Alternatives: One to All

One to All

- Positive Aspects:
  - Straight forward.
  - No redundancy (i.e, each node receives each message a single time).
- Negative Aspects:
  - Requires each node to know the full membership.
  - Not Scalable (no load distribution).
  - Not Resilient to Faults.

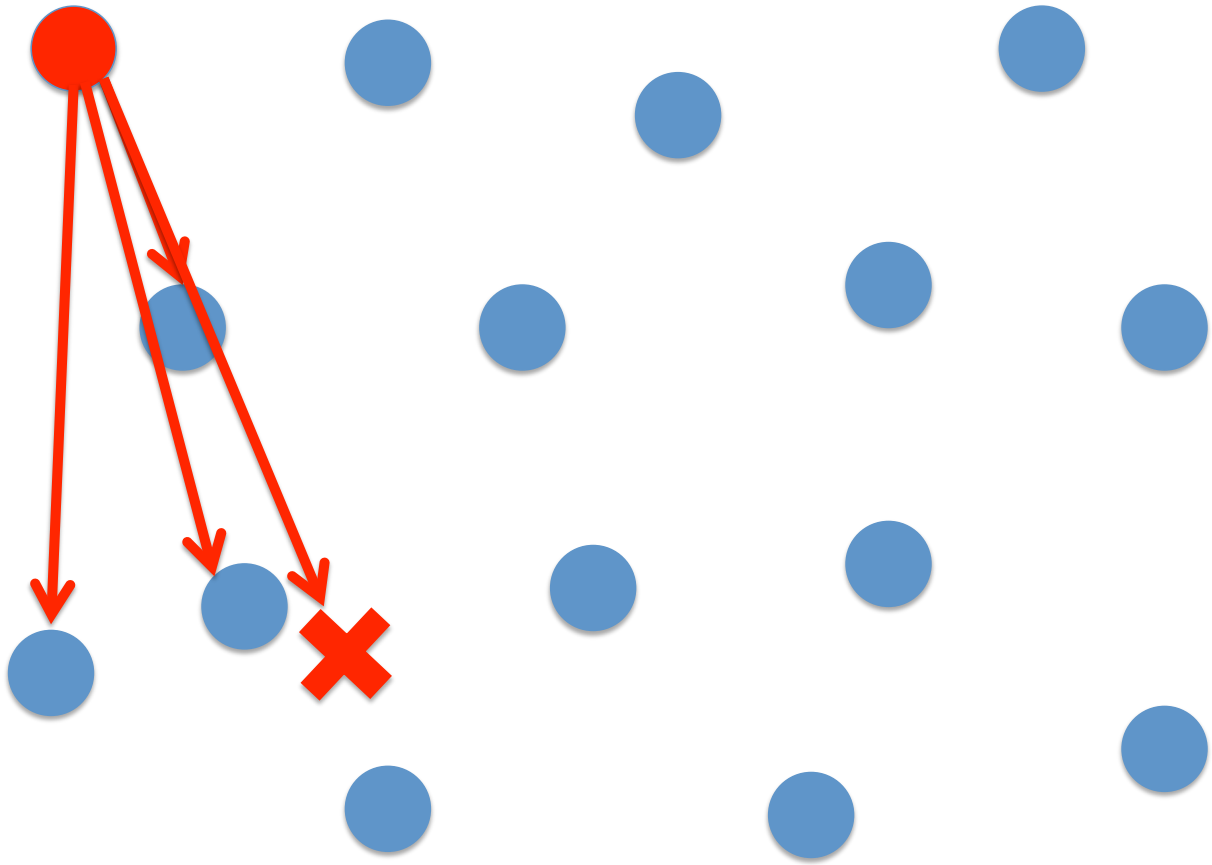
The Academic View: Data Dissemination

Design Alternatives: One to All



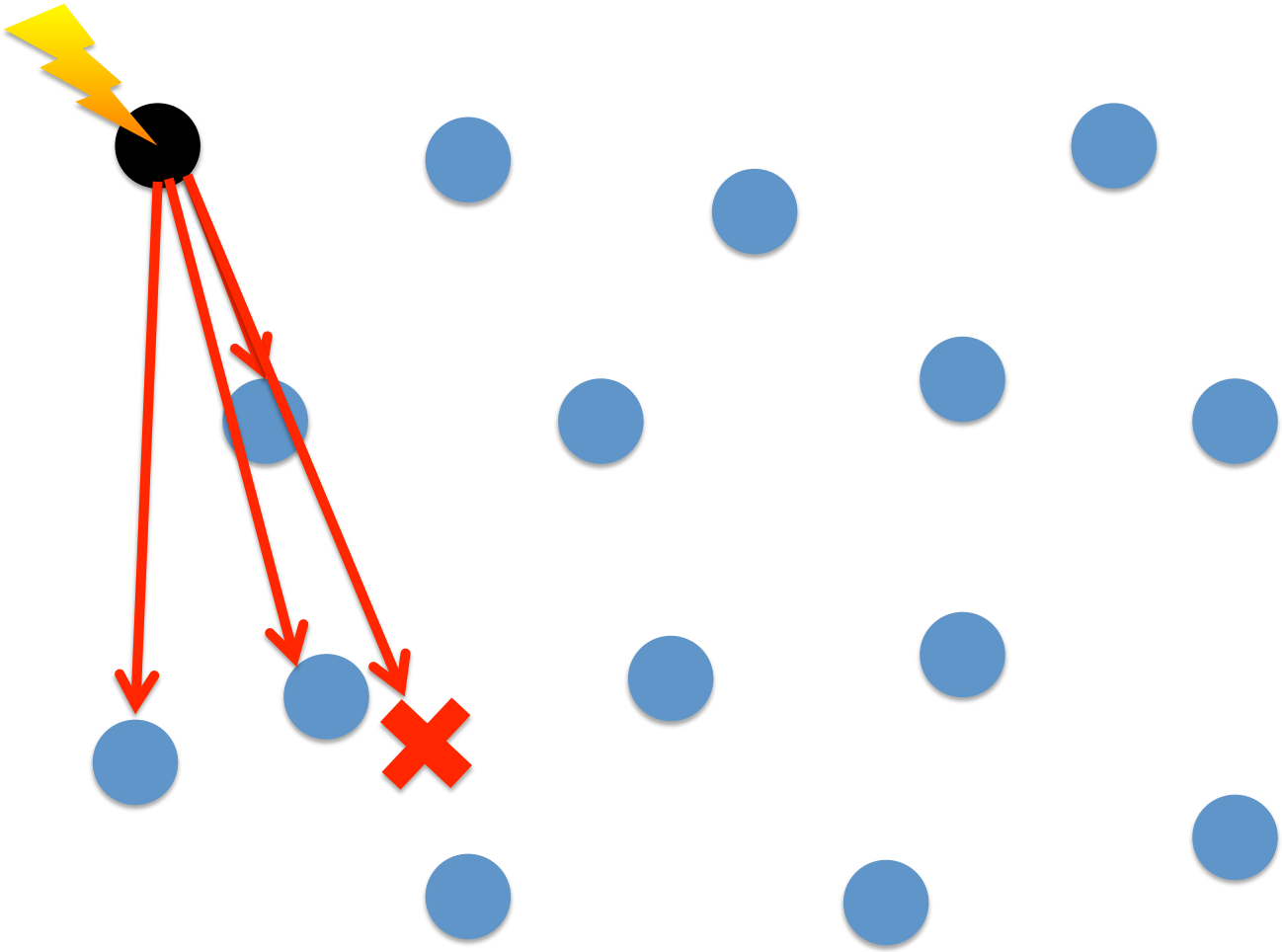
The Academic View: Data Dissemination

Design Alternatives: One to All



The Academic View: Data Dissemination

Design Alternatives: One to All



The Academic View: Data Dissemination

Design Alternatives: Spanning Tree

Dealing with Load Distribution

Organize participants/nodes in a tree and forward messages across the tree.

Spanning Tree

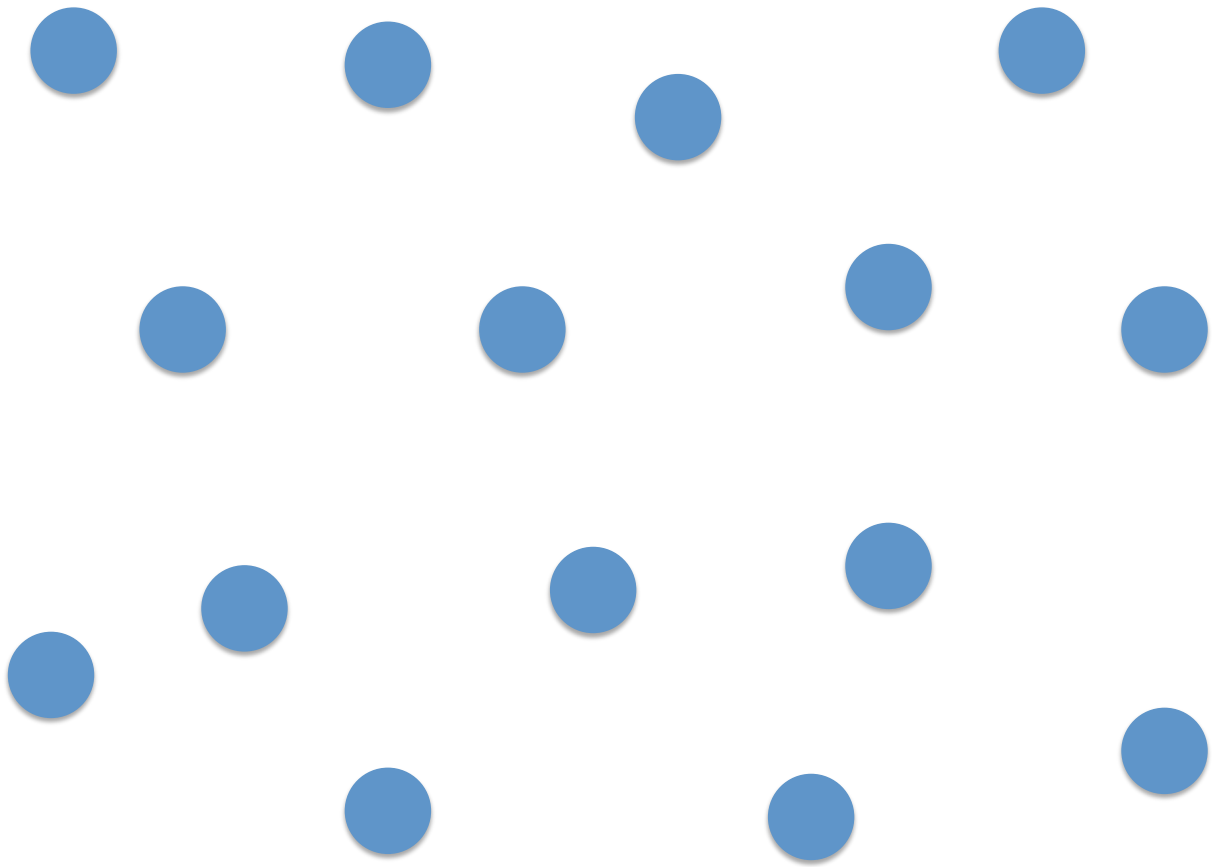
Dealing with Load Distribution

Organize participants/nodes in a tree and forward messages across the tree.

Spanning Tree

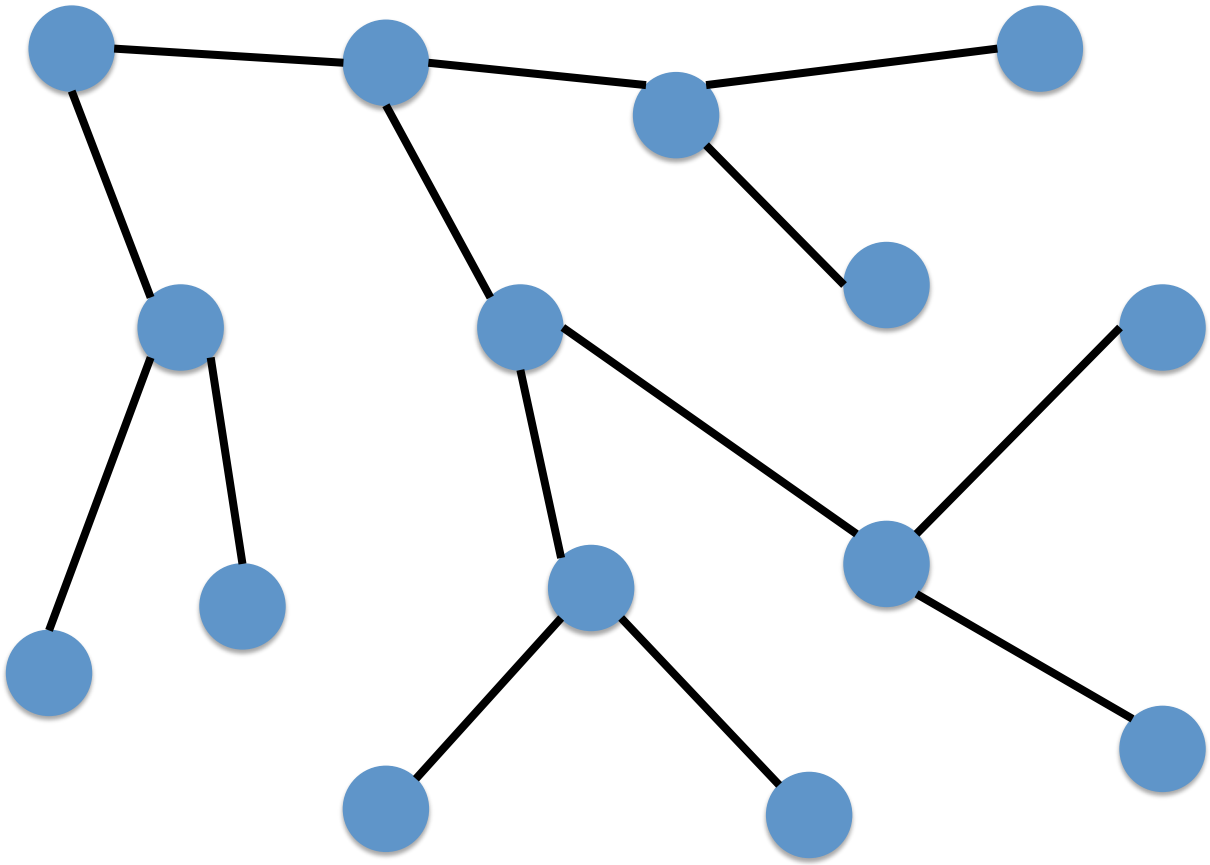
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



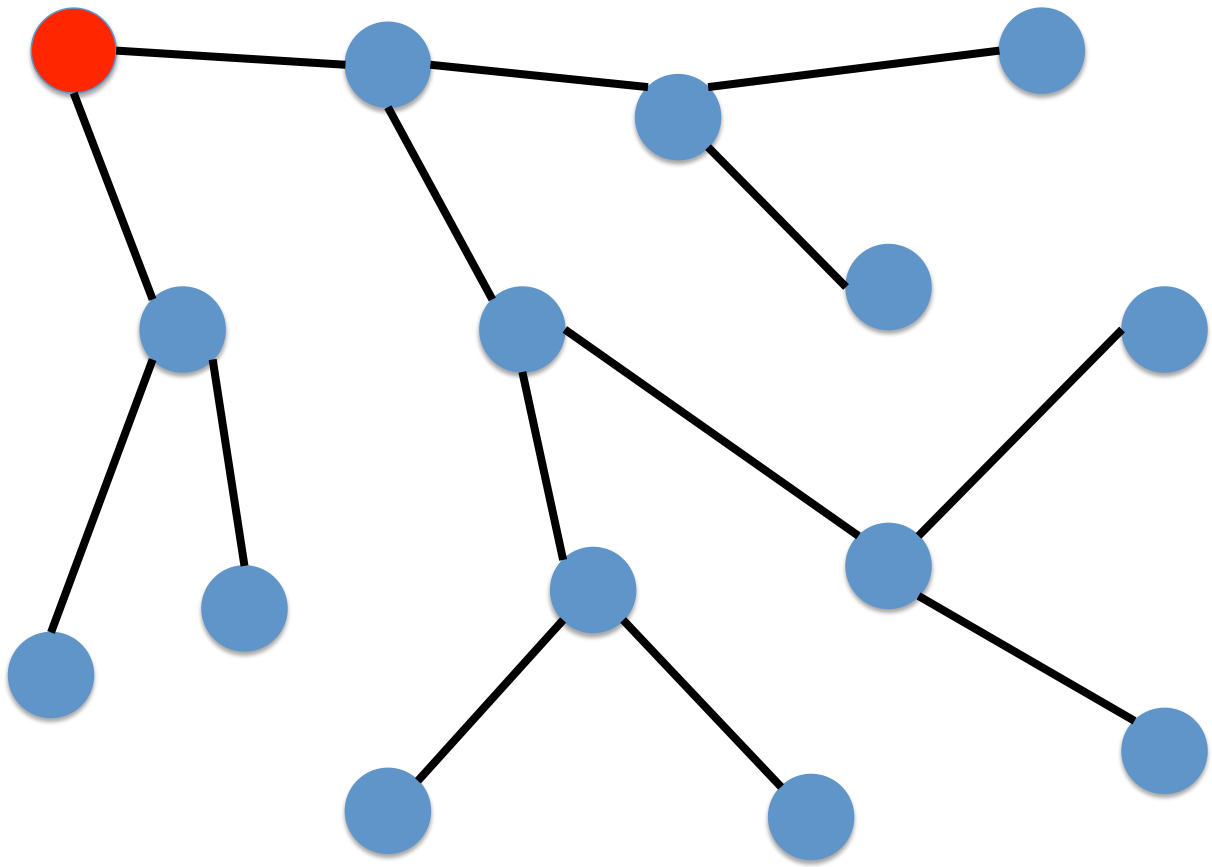
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



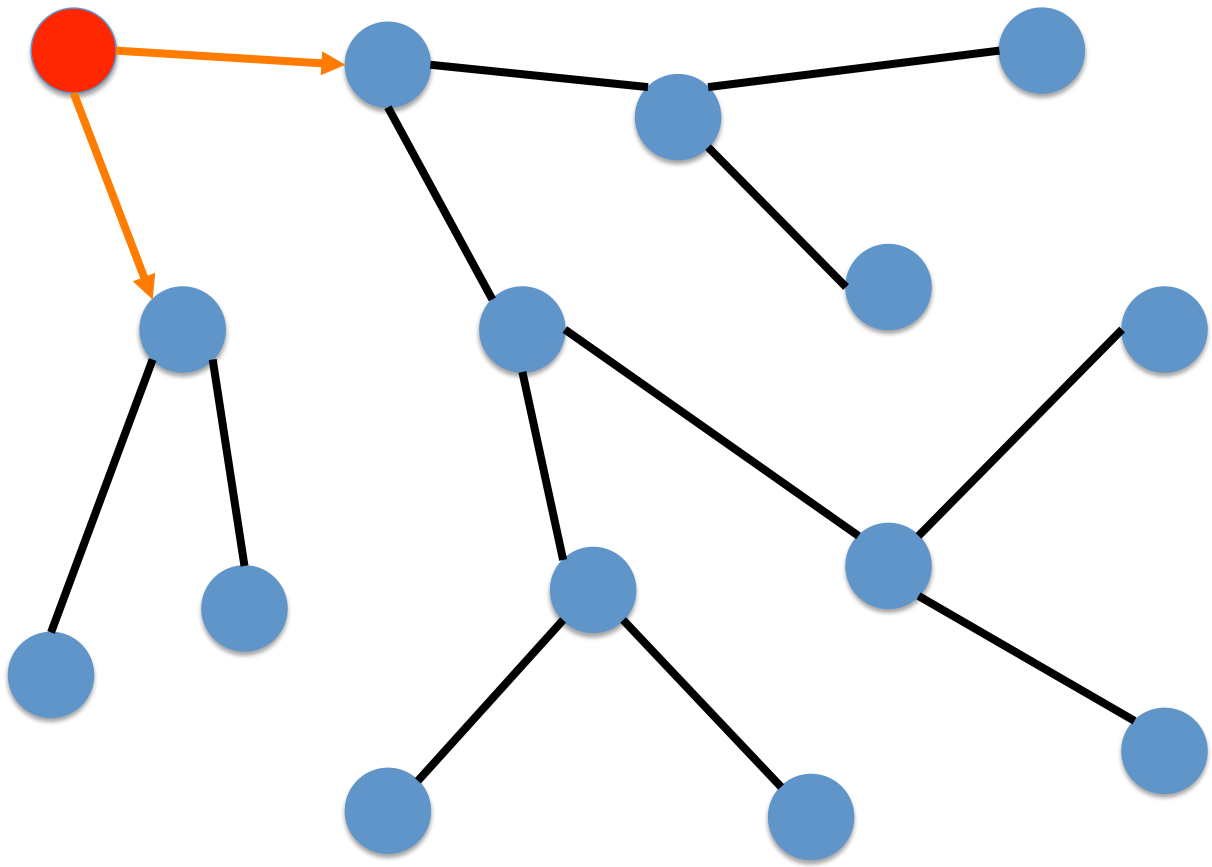
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



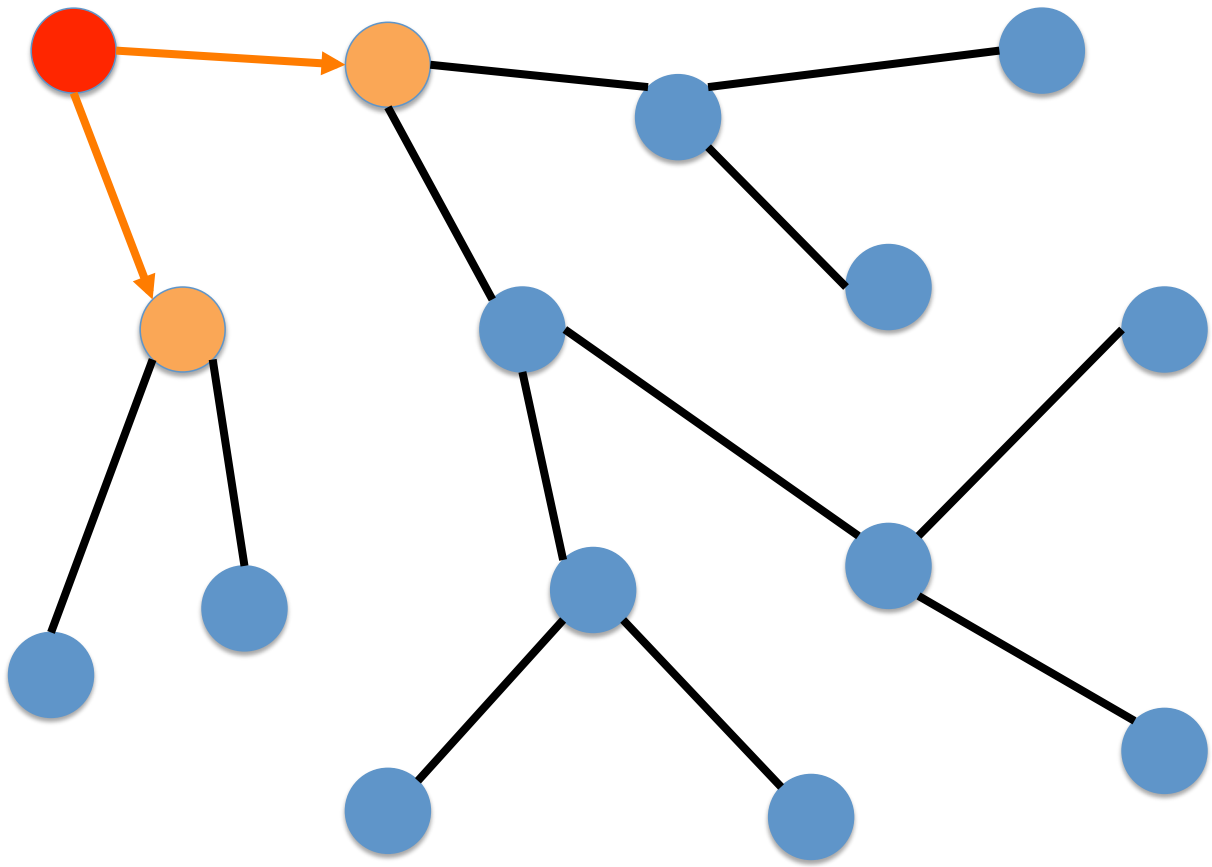
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



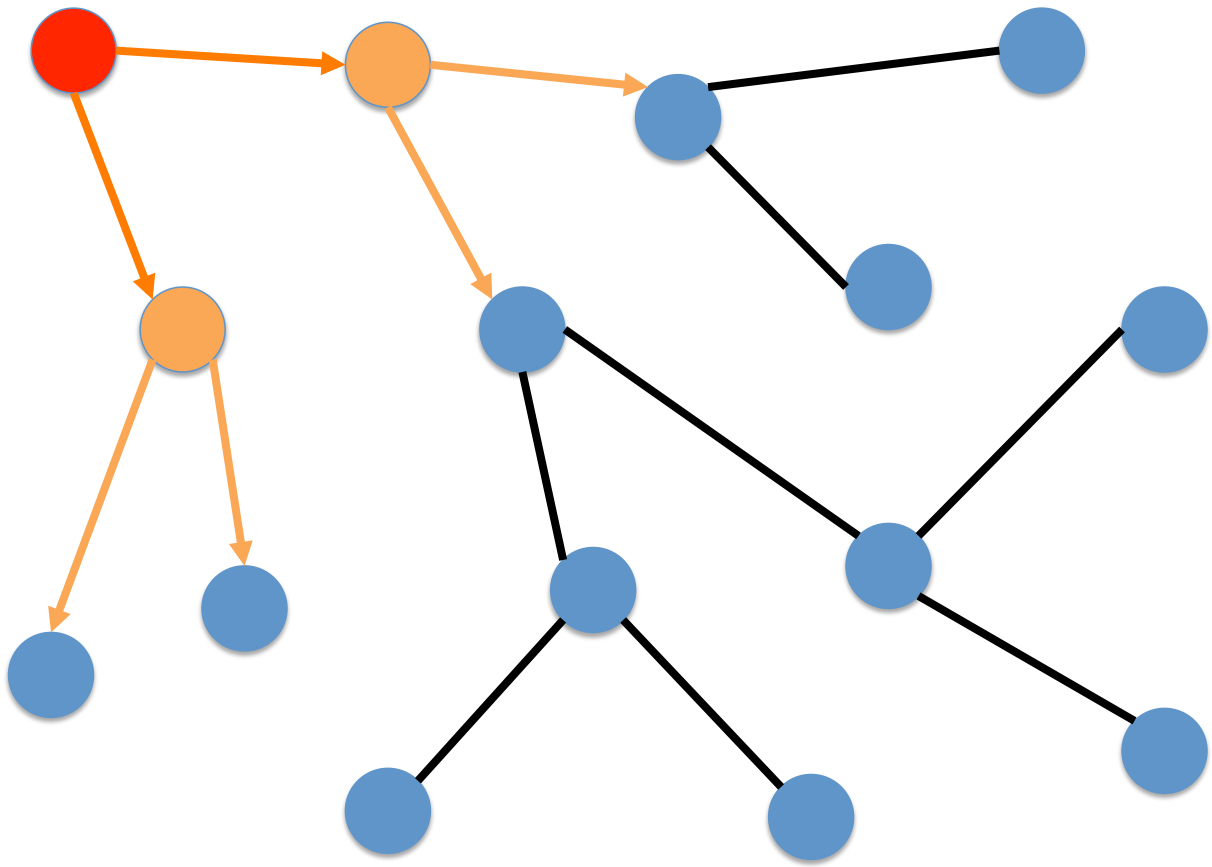
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



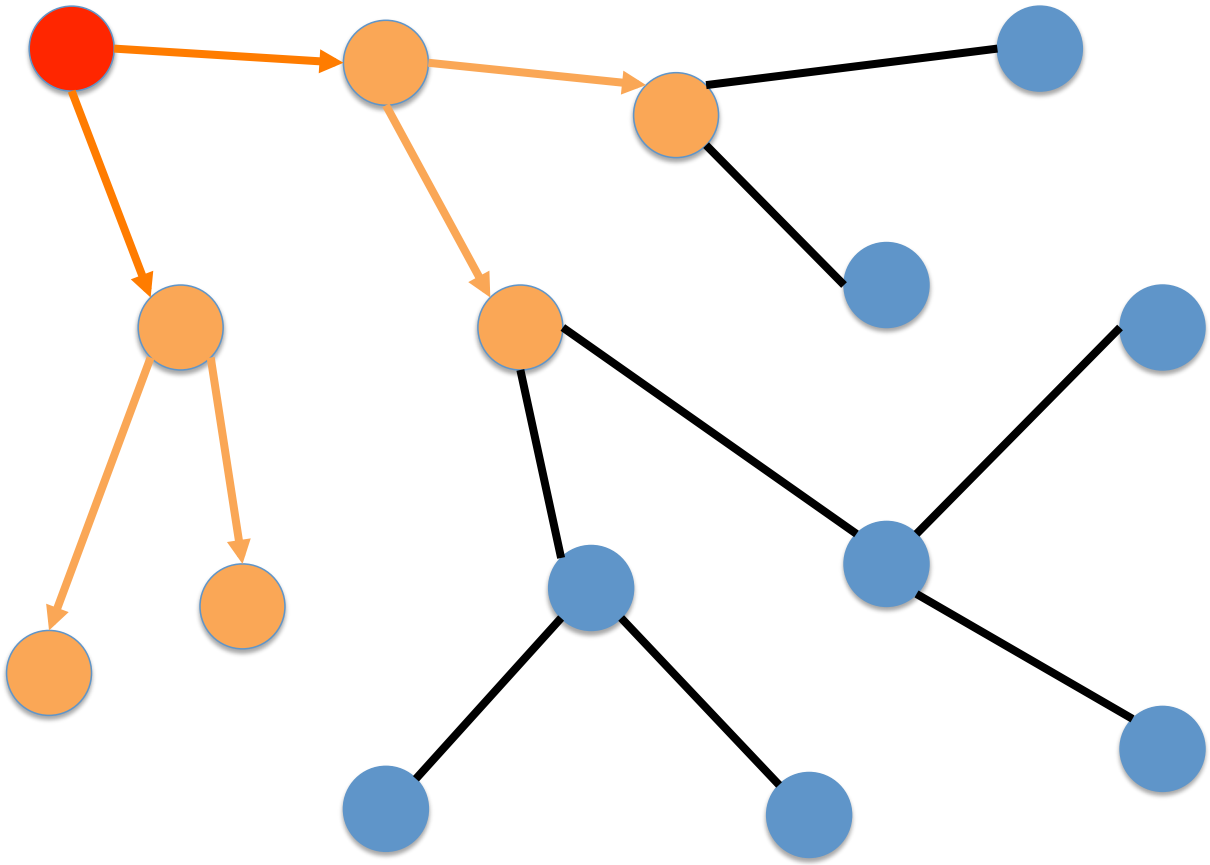
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



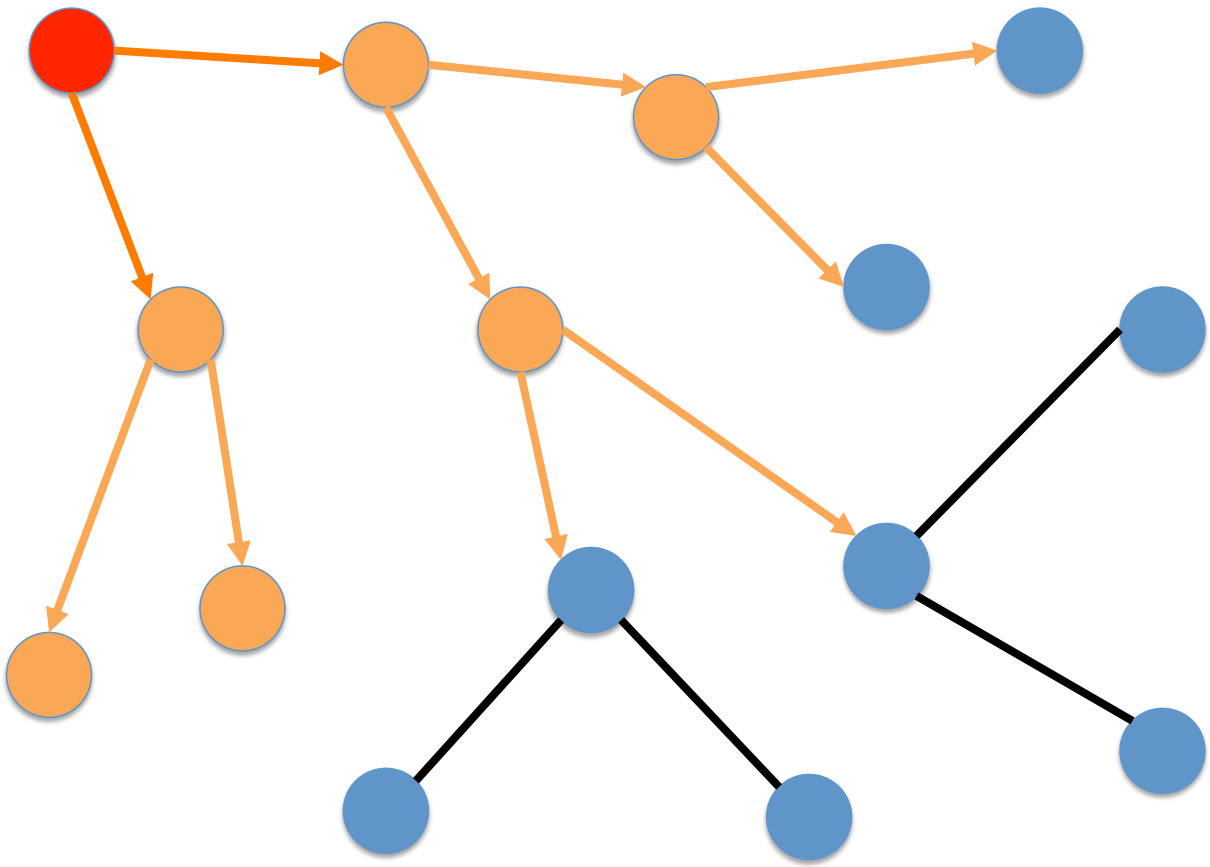
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



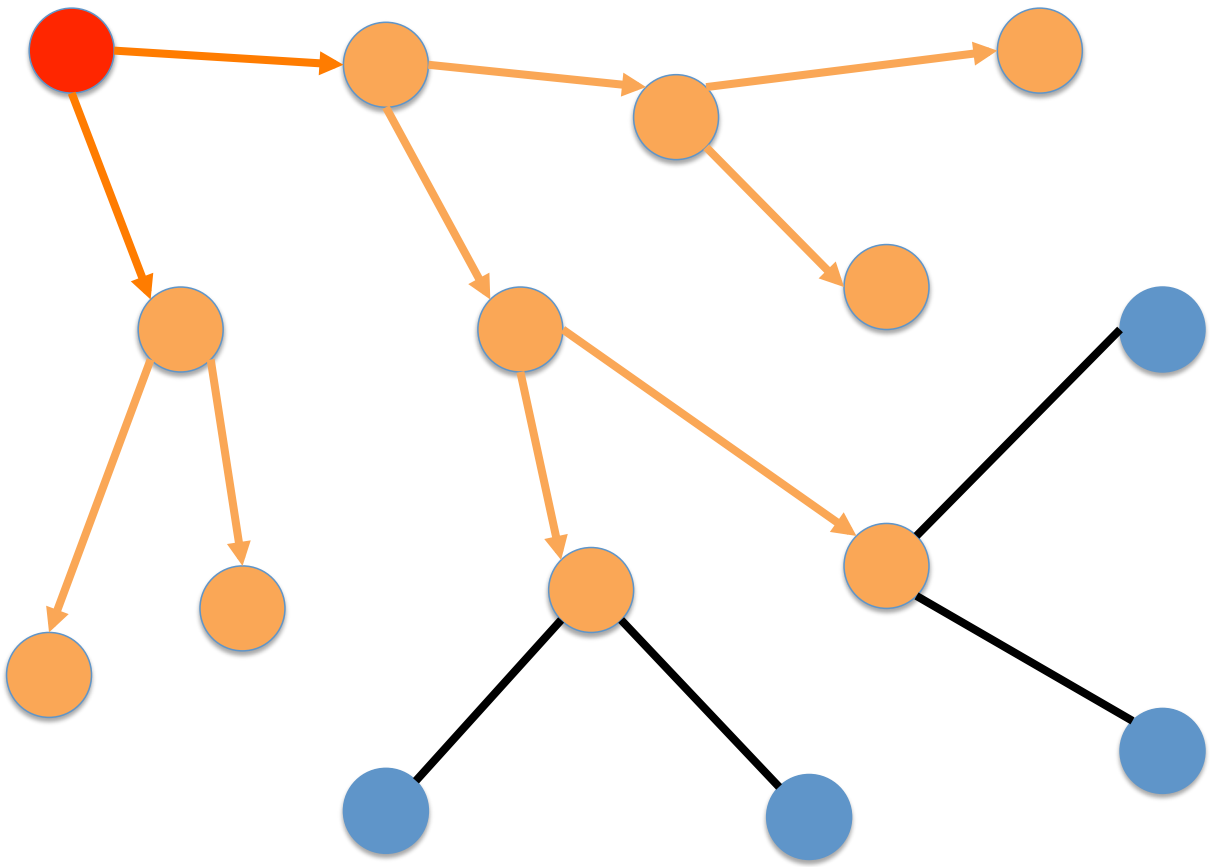
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



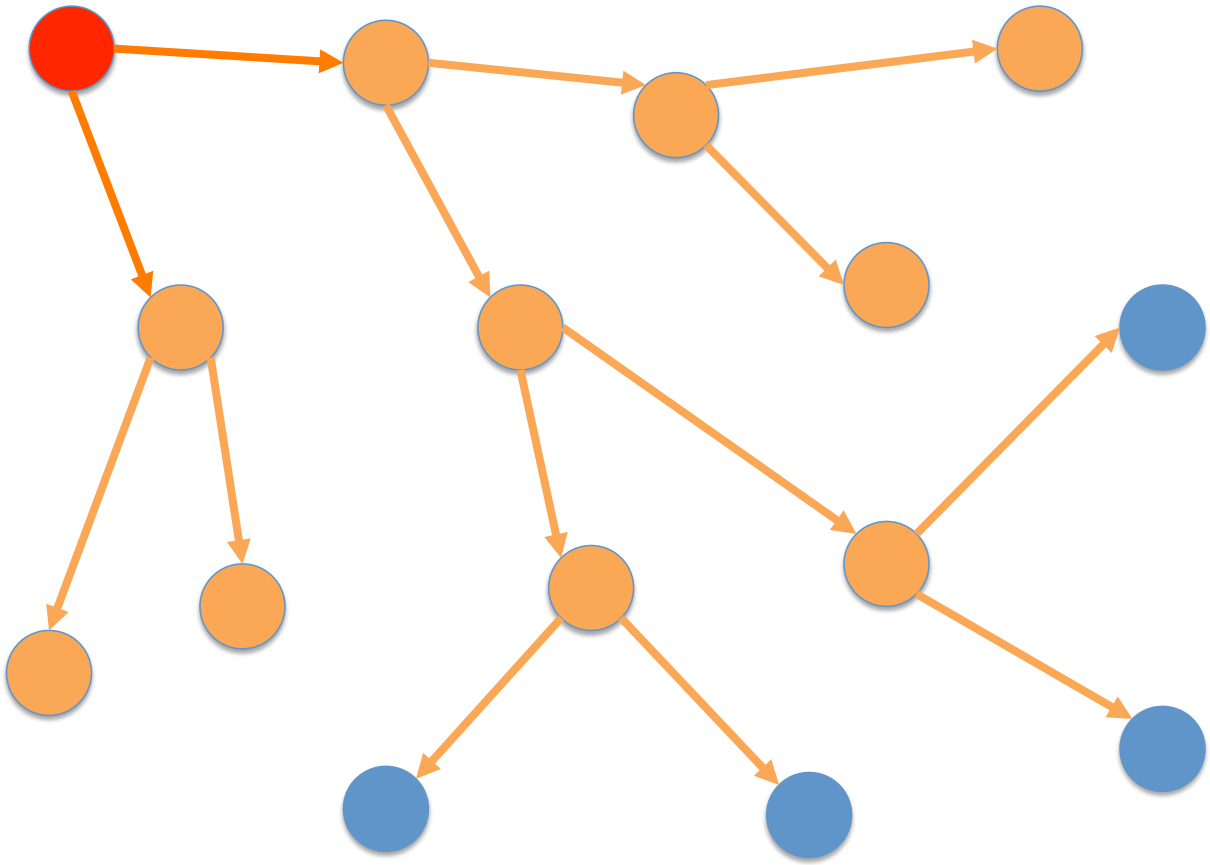
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



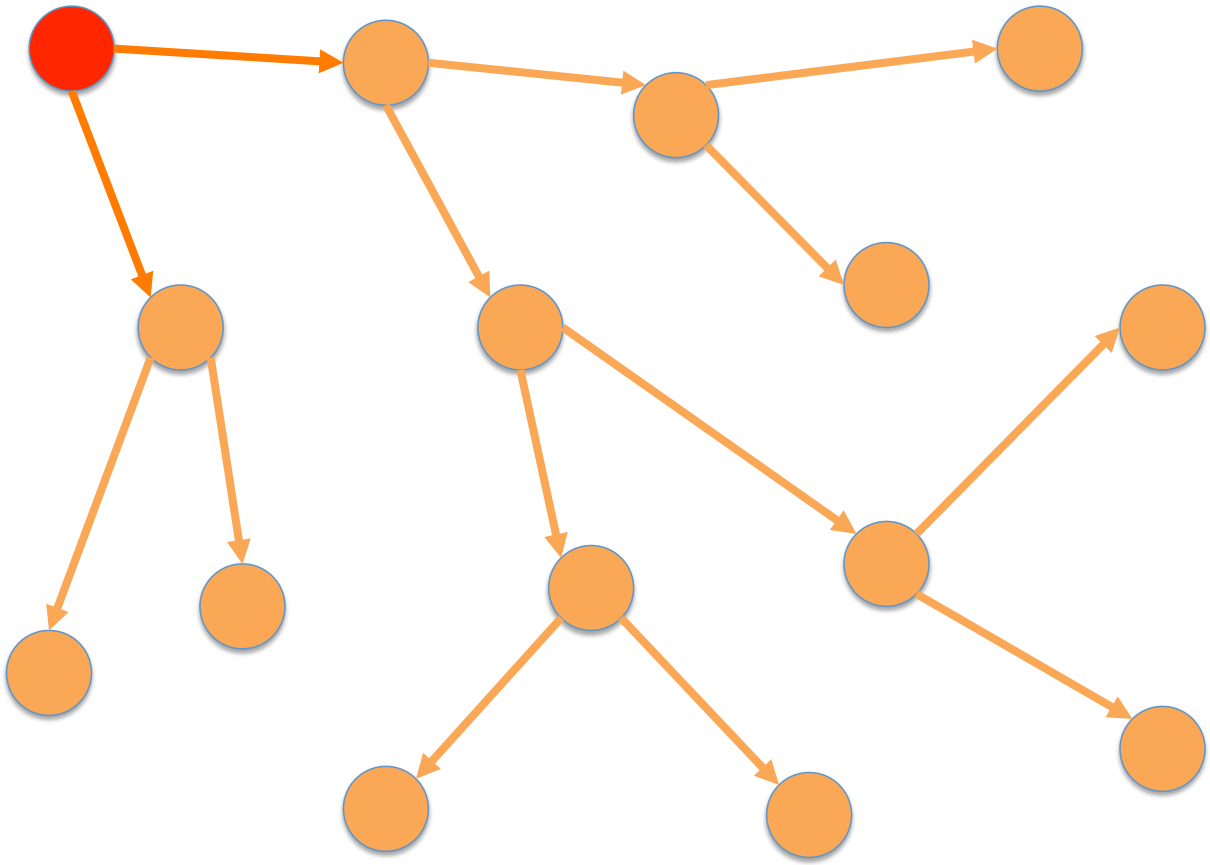
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



The Academic View: Data Dissemination

Design Alternatives: Spanning Tree

Spanning Tree

- Positive Aspects:
  - Load Distribution.
  - No redundancy (i.e, each node receives each message a single time).
- Negative Aspects:
  - Complexity in Managing the Topology (Hinders Scalability).
  - (Still) Not Resilient to Faults.

The Academic View: Data Dissemination

Design Alternatives: Spanning Tree

Spanning Tree

- Positive Aspects:
  - Load Distribution.
  - No redundancy (i.e, each node receives each message a single time).
- Negative Aspects:
  - Complexity in Managing the Topology (Hinders Scalability).
  - (Still) Not Resilient to Faults.

The Academic View: Data Dissemination

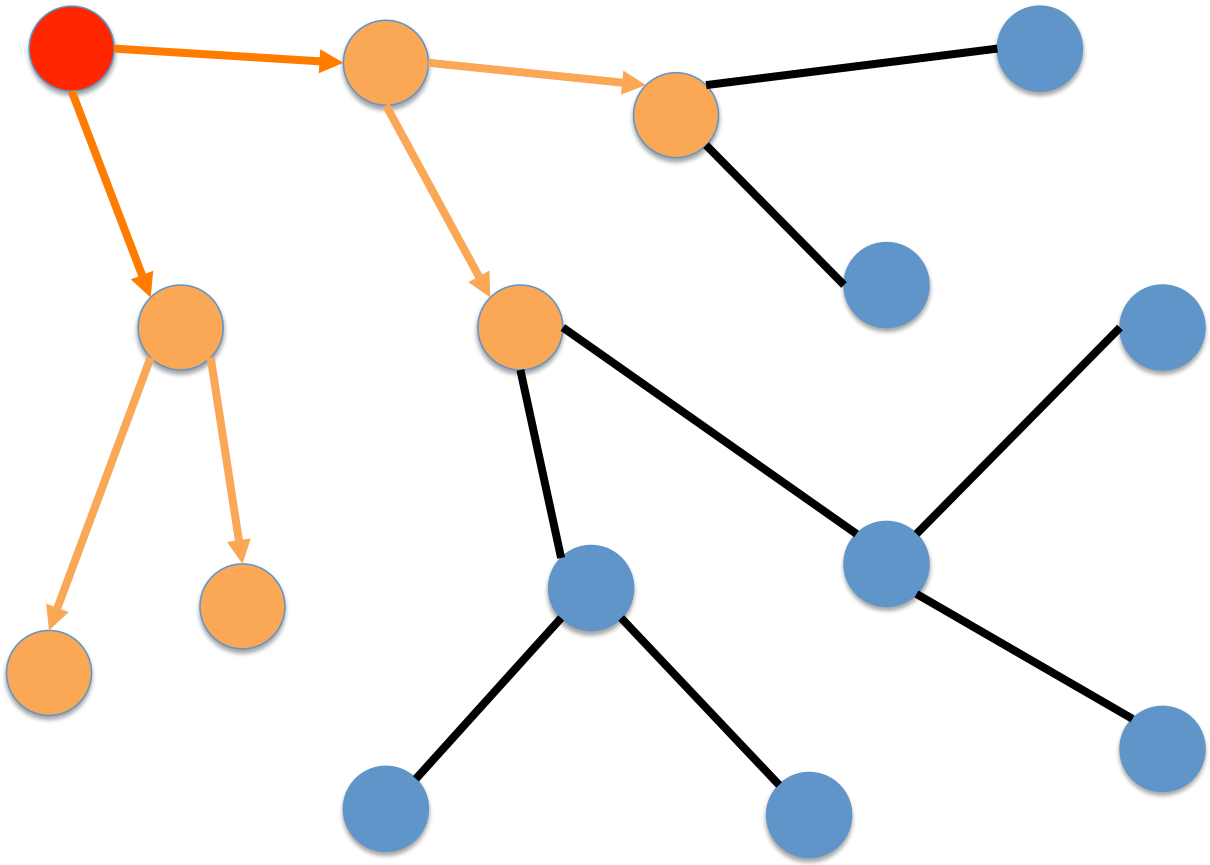
Design Alternatives: Spanning Tree

Spanning Tree

- Positive Aspects:
  - Load Distribution.
  - No redundancy (i.e, each node receives each message a single time).
- Negative Aspects:
  - Complexity in Managing the Topology (Hinders Scalability).
  - (Still) Not Resilient to Faults.

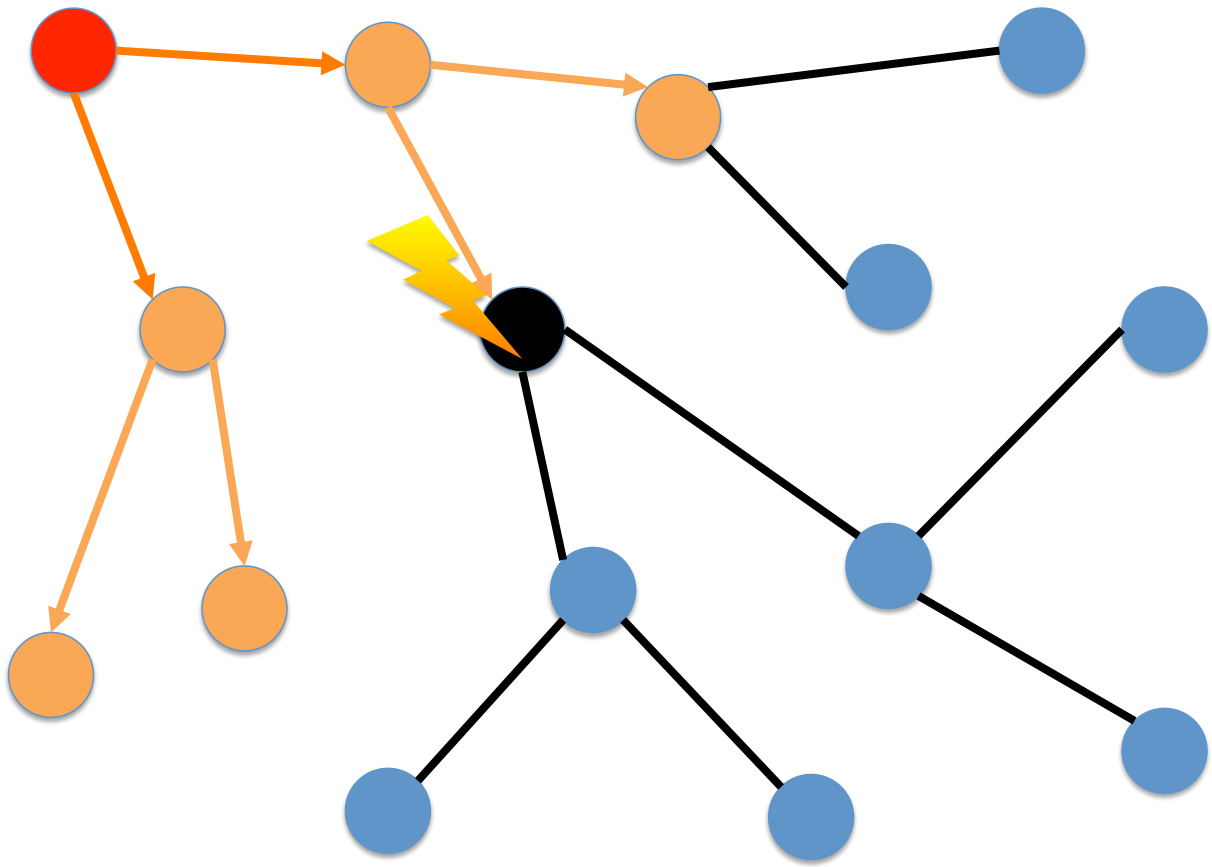
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



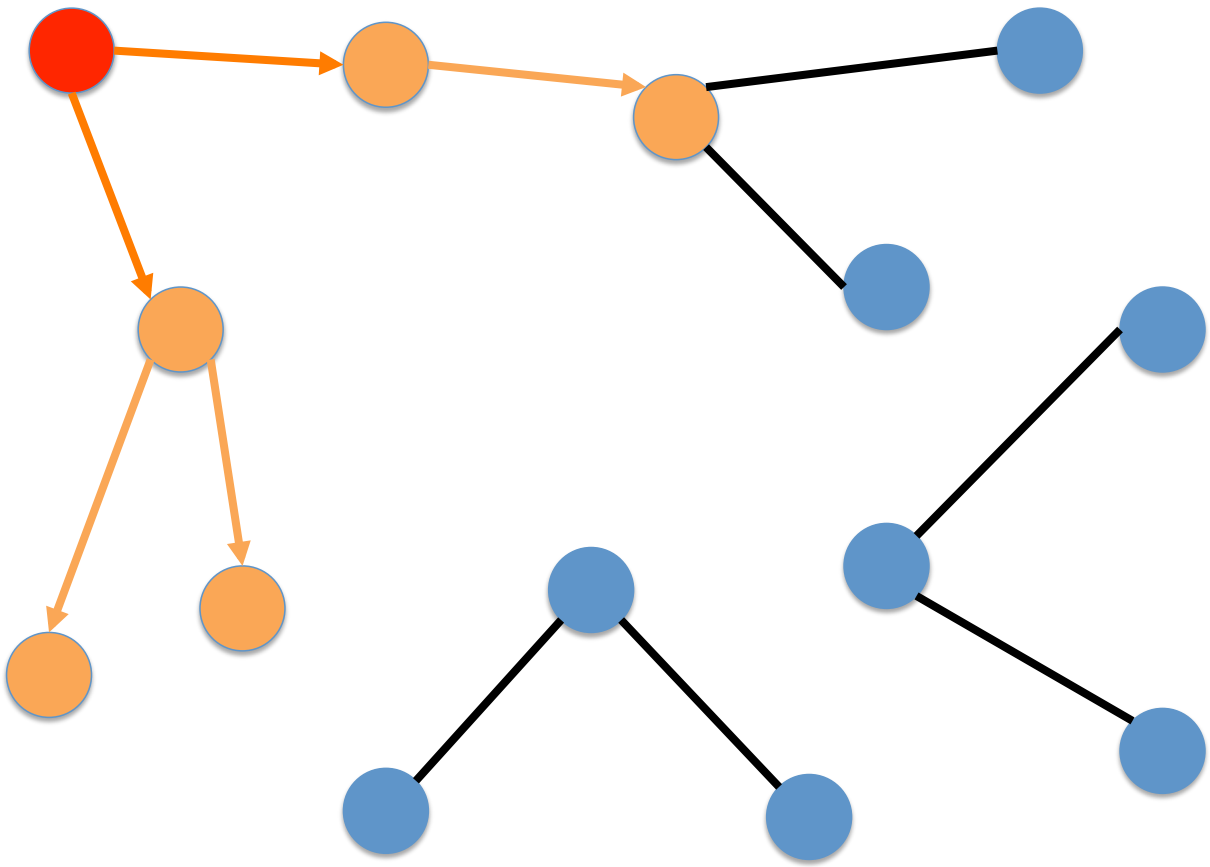
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



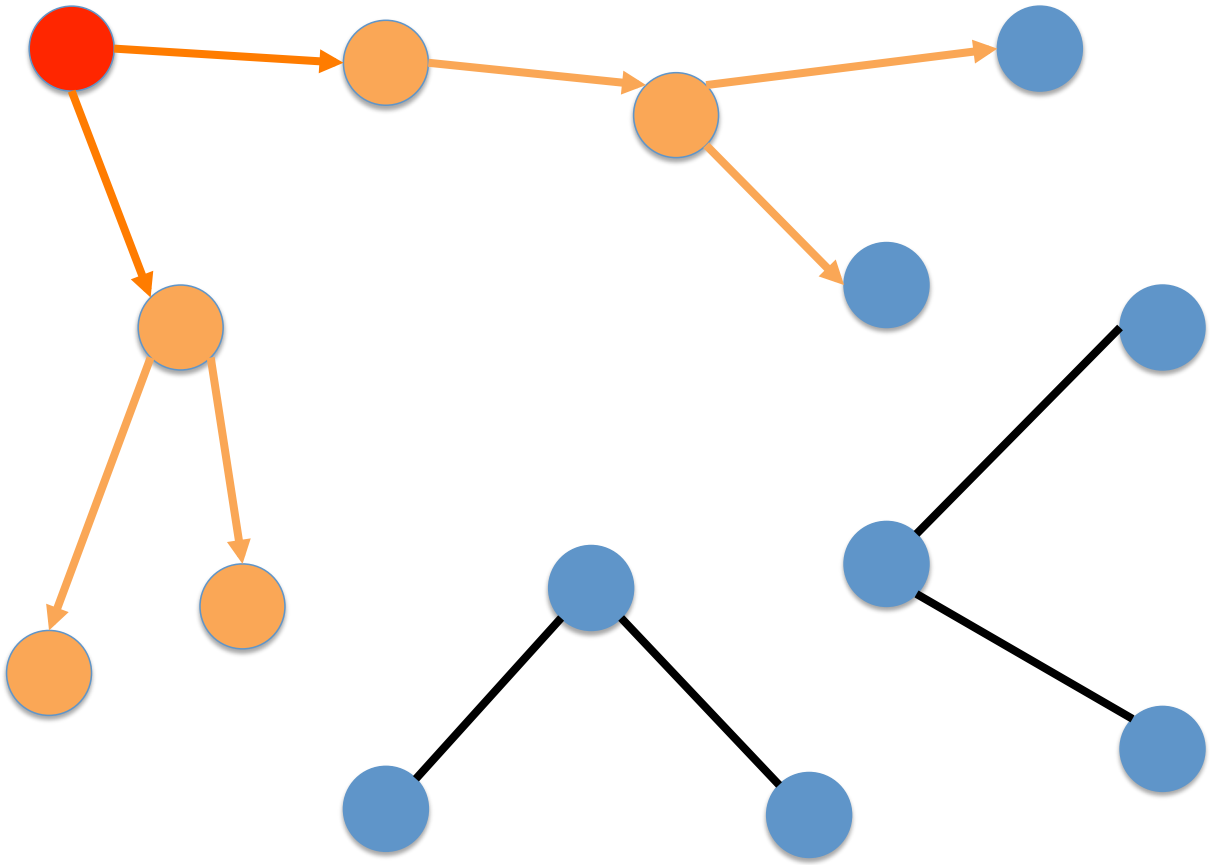
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



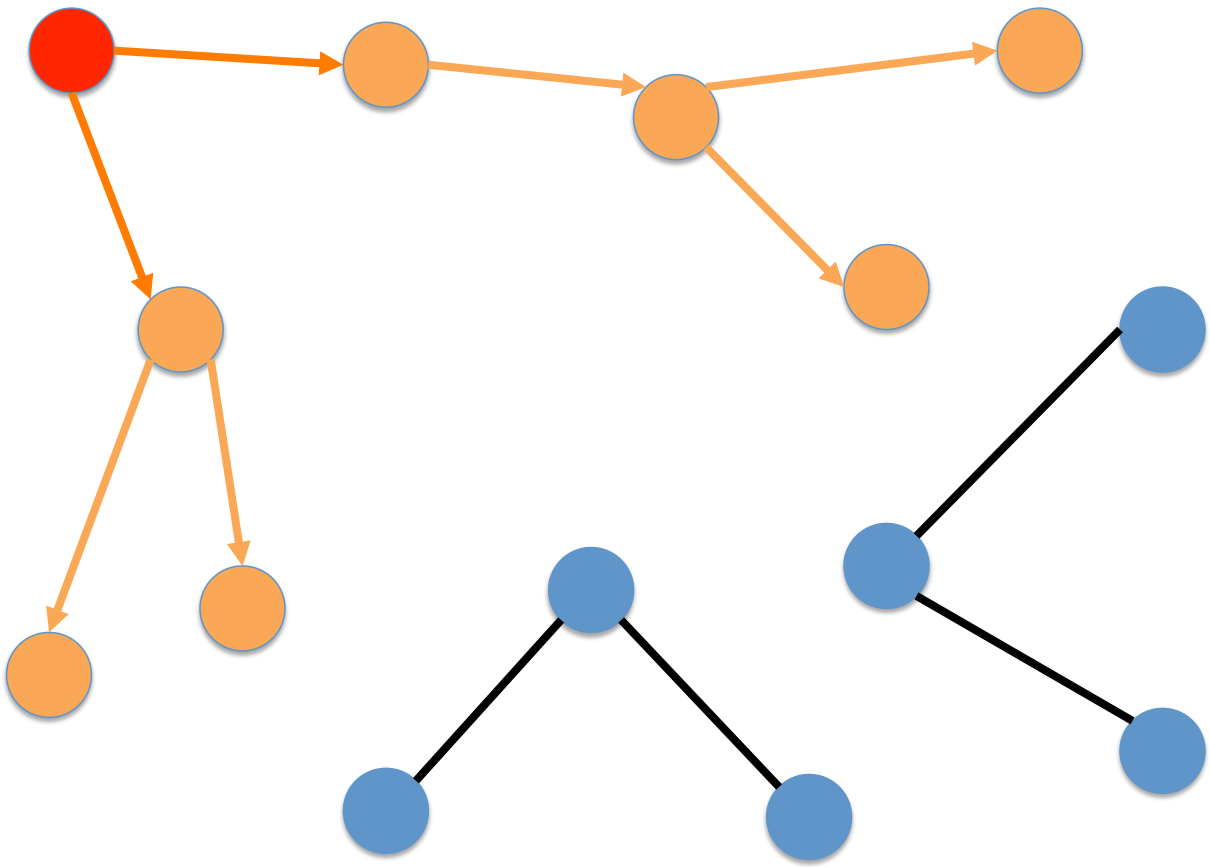
The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



The Academic View: Data Dissemination

Design Alternatives: Spanning Tree



The Academic View: Data Dissemination

Design Alternatives: Flood

Dealing with Fault Tolerance

Organize participants/nodes in a random, highly connected, graph and forward messages across all links.

Flood

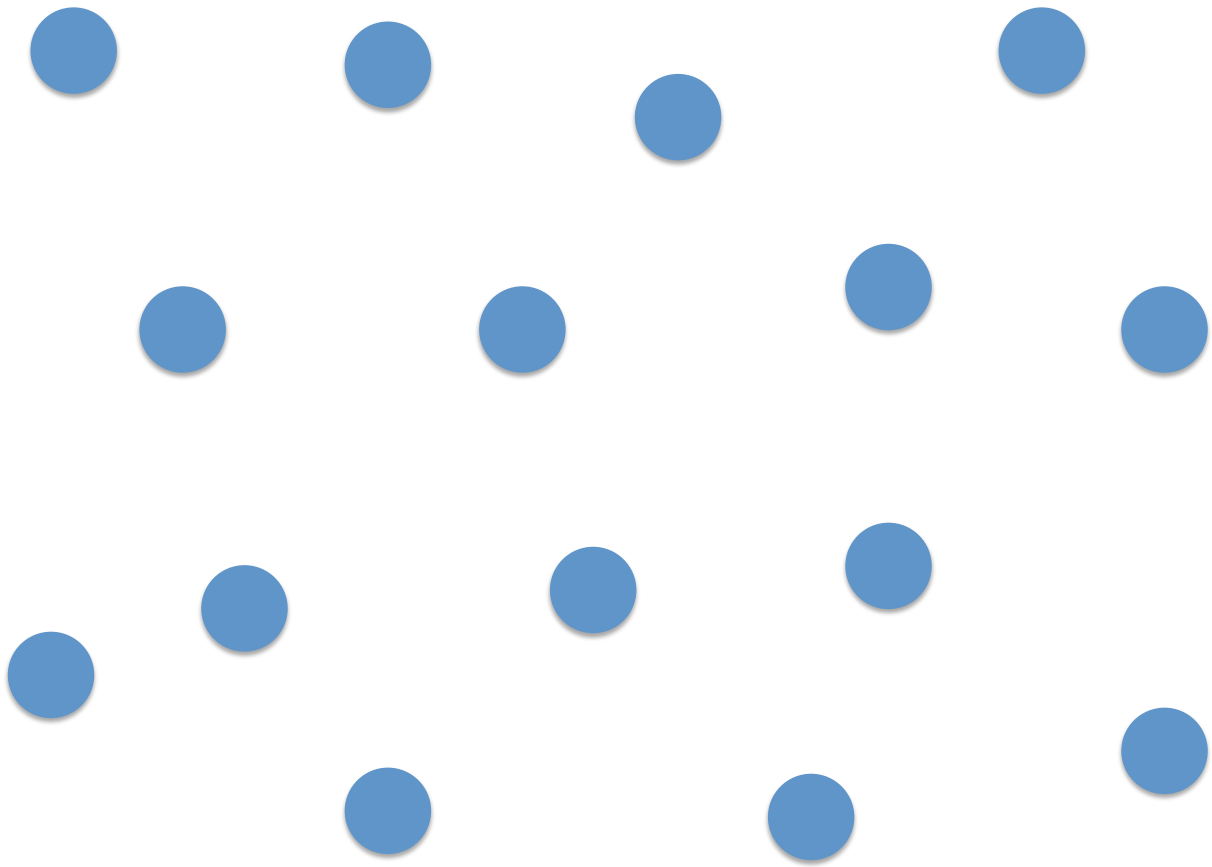
Dealing with Fault Tolerance

Organize participants/nodes in a random, highly connected, graph and forward messages across all links.

Flood

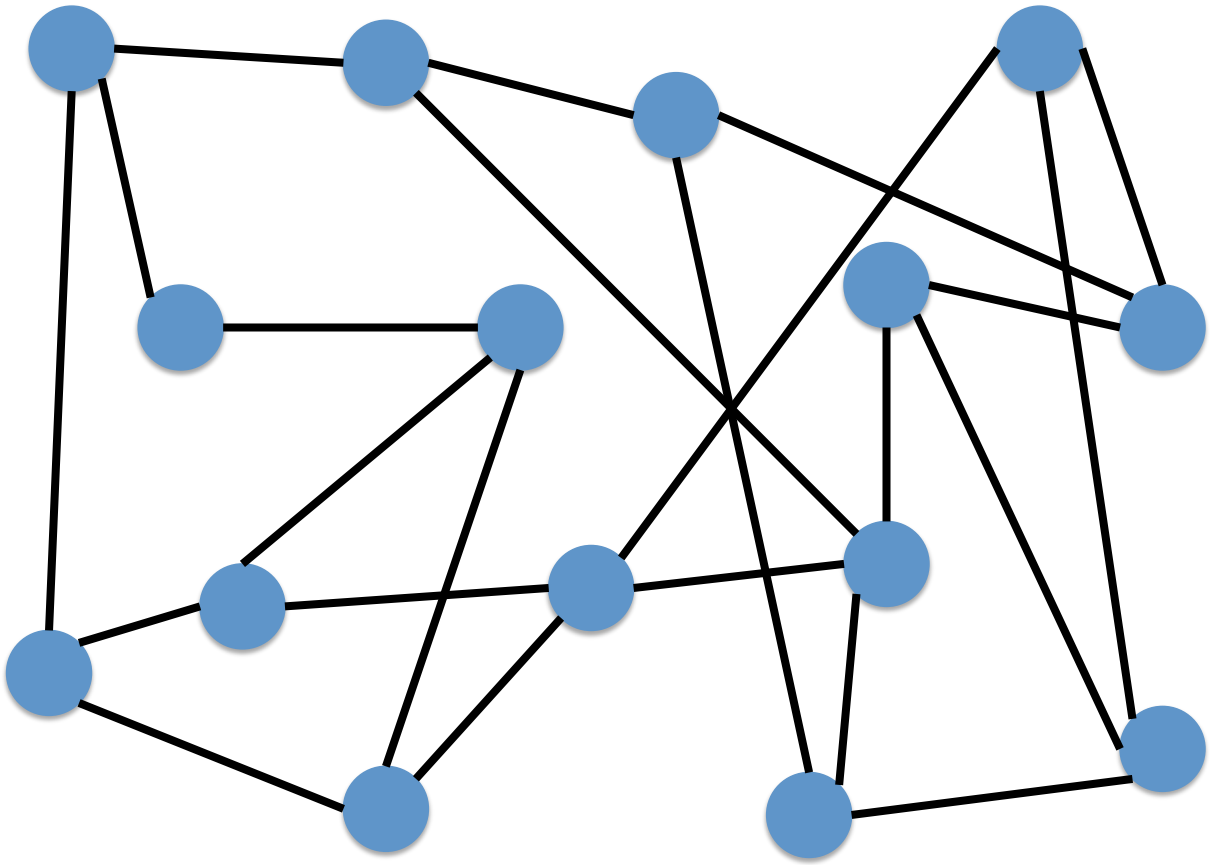
The Academic View: Data Dissemination

Design Alternatives: Flood



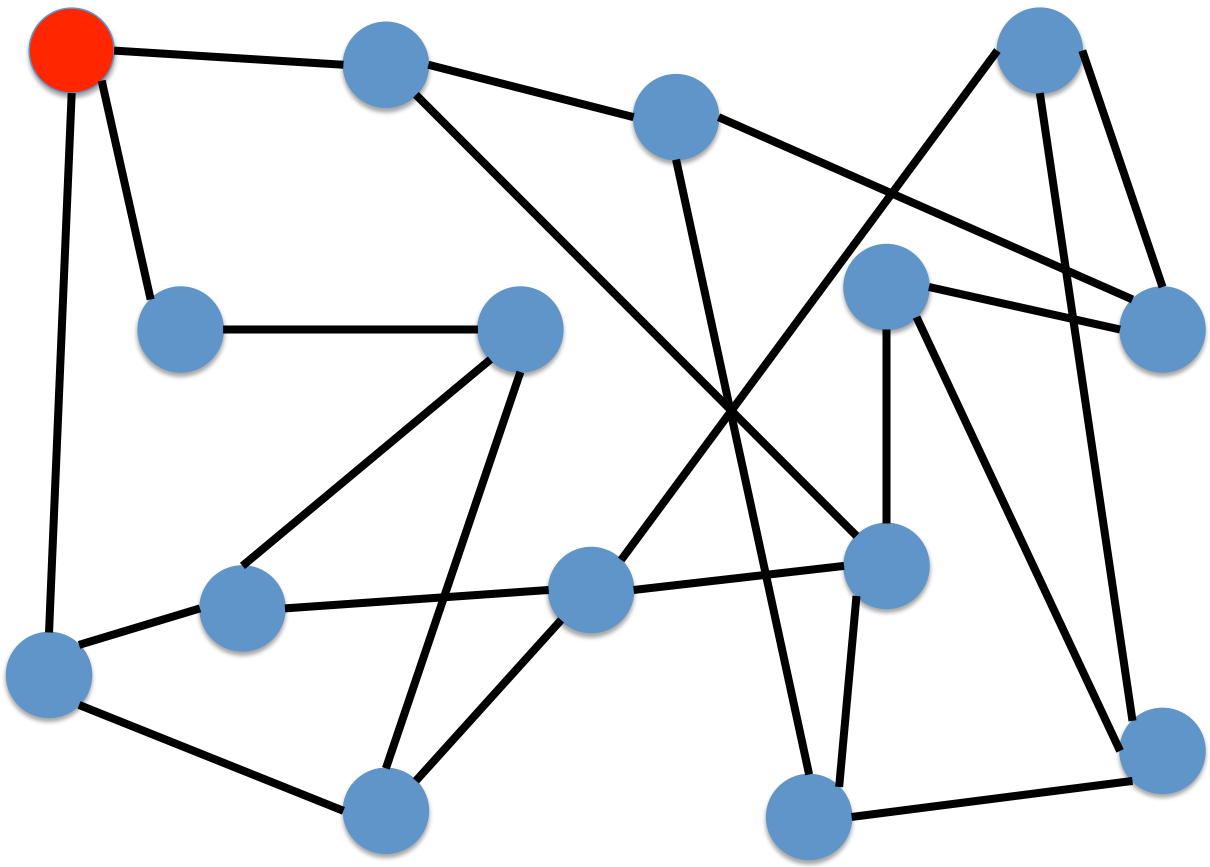
The Academic View: Data Dissemination

Design Alternatives: Flood



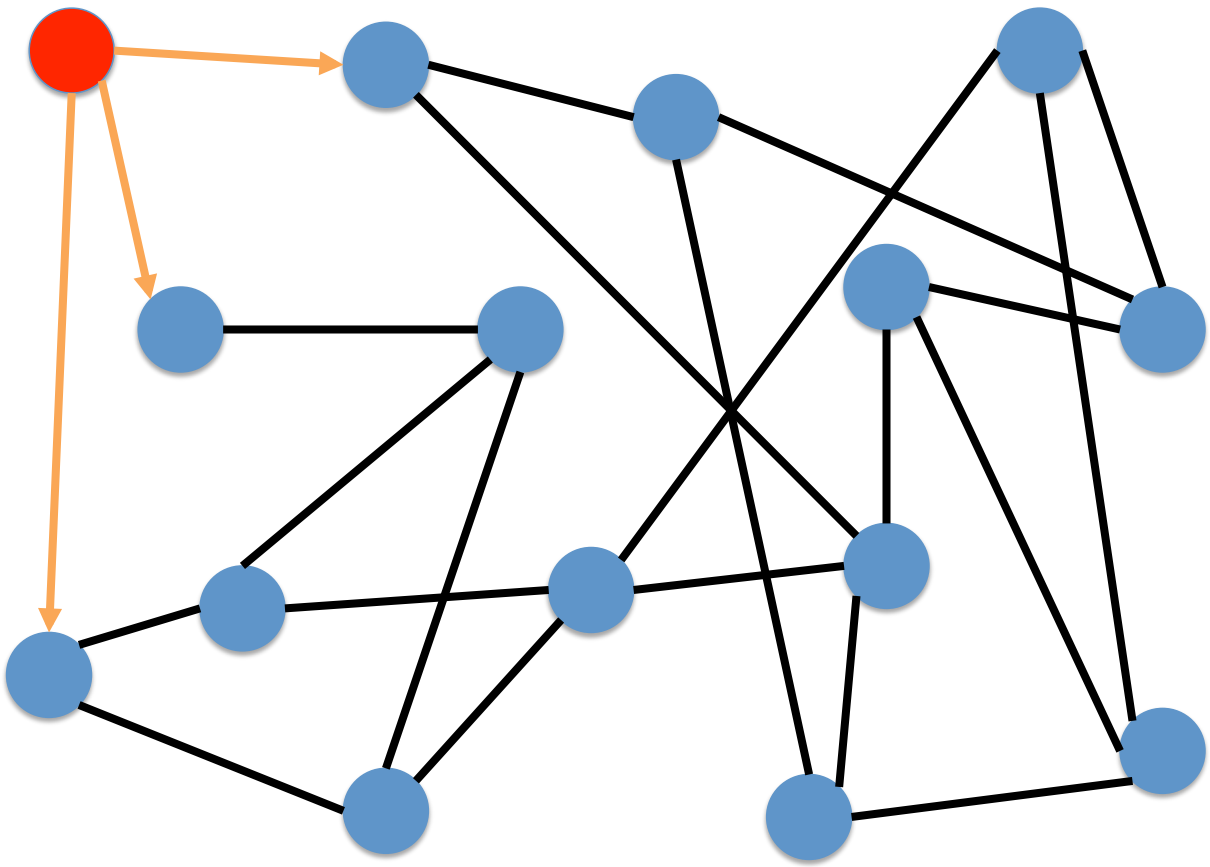
The Academic View: Data Dissemination

Design Alternatives: Flood



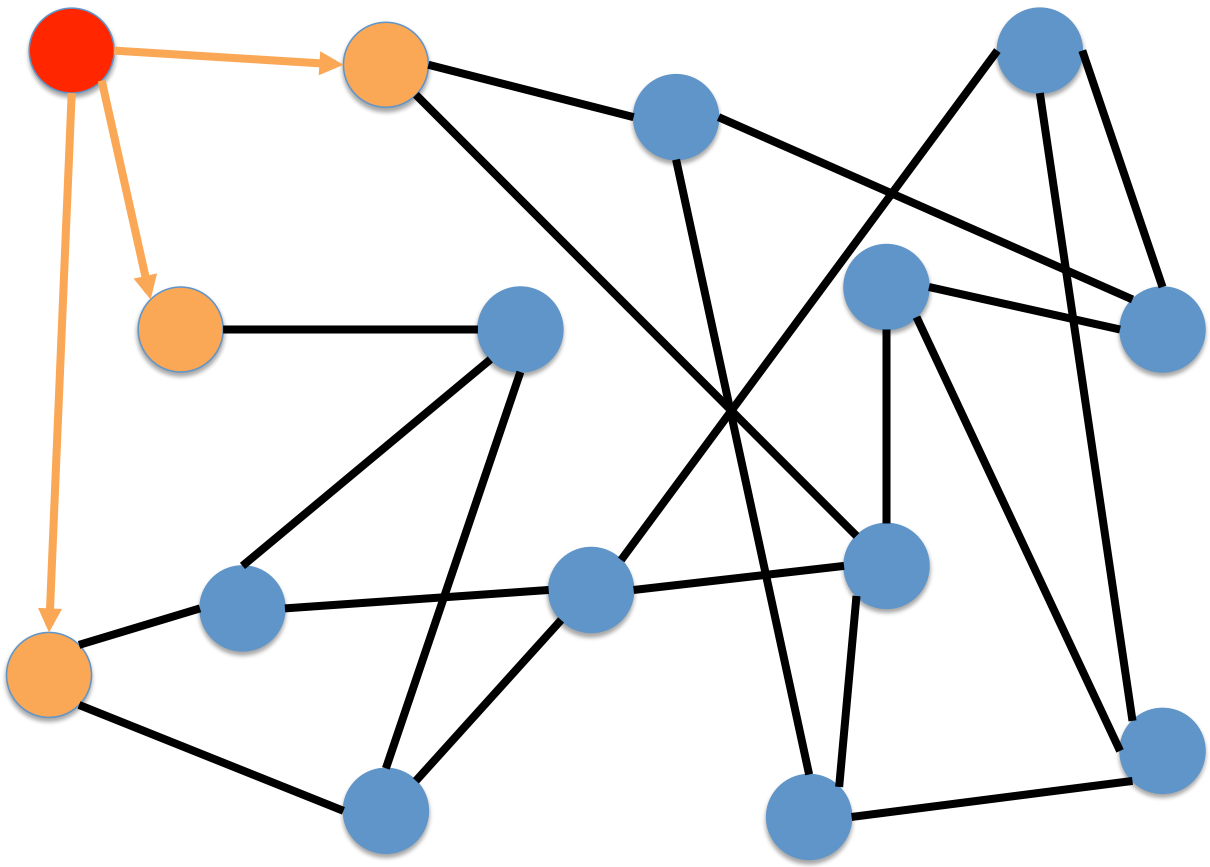
The Academic View: Data Dissemination

Design Alternatives: Flood



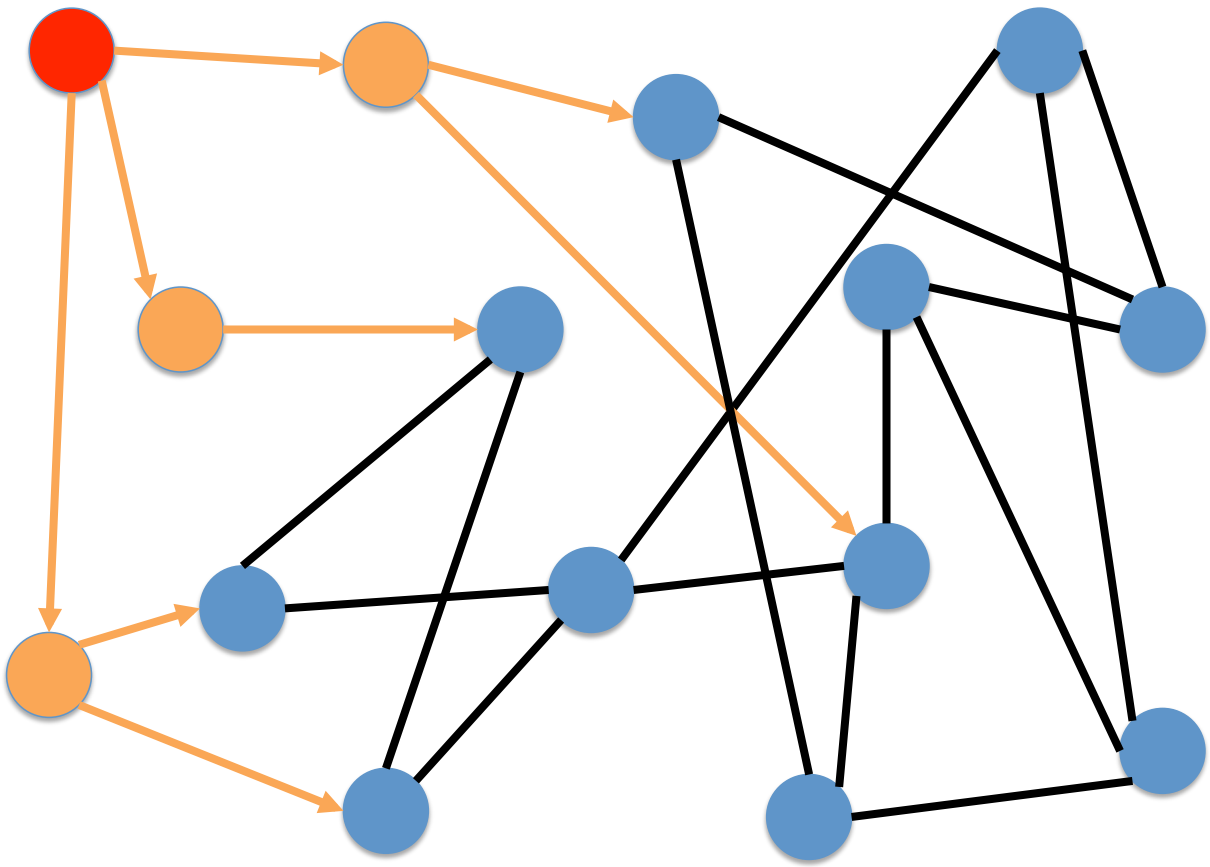
The Academic View: Data Dissemination

Design Alternatives: Flood



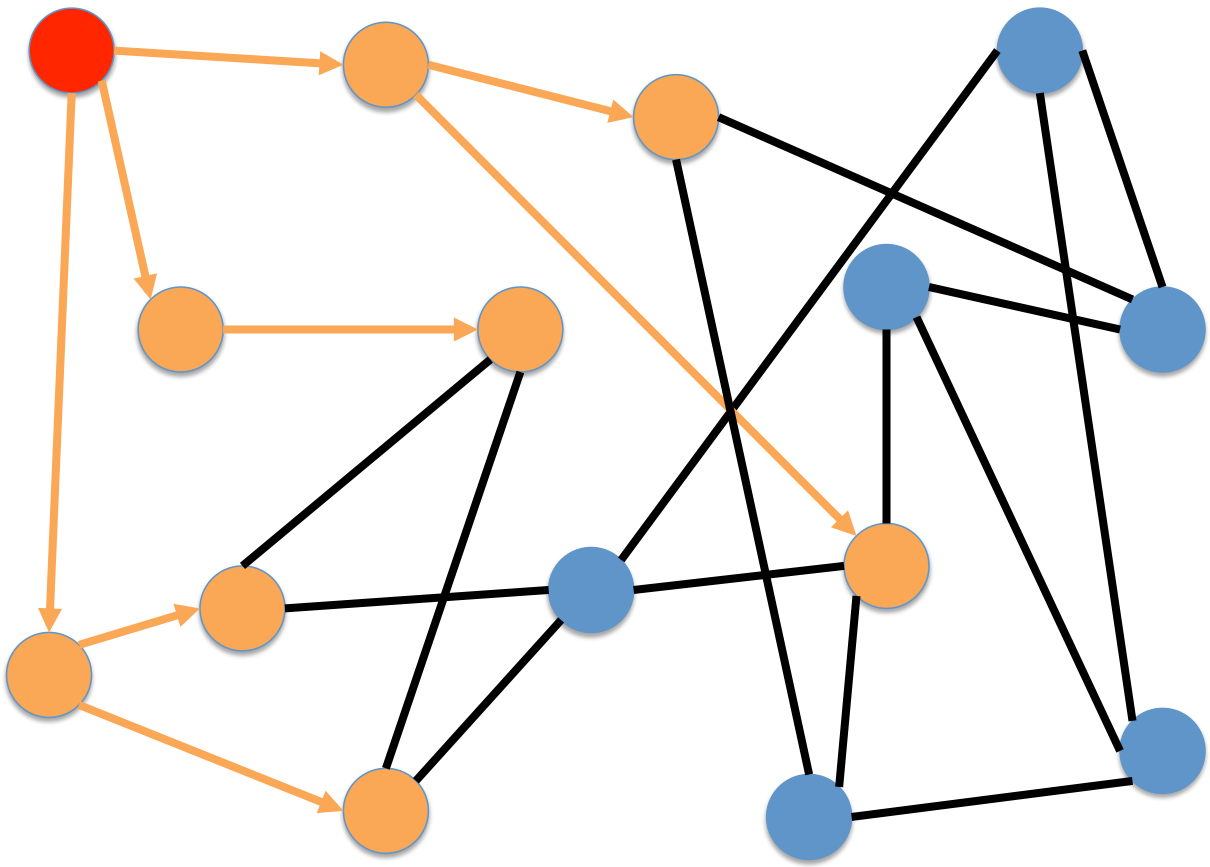
The Academic View: Data Dissemination

Design Alternatives: Flood



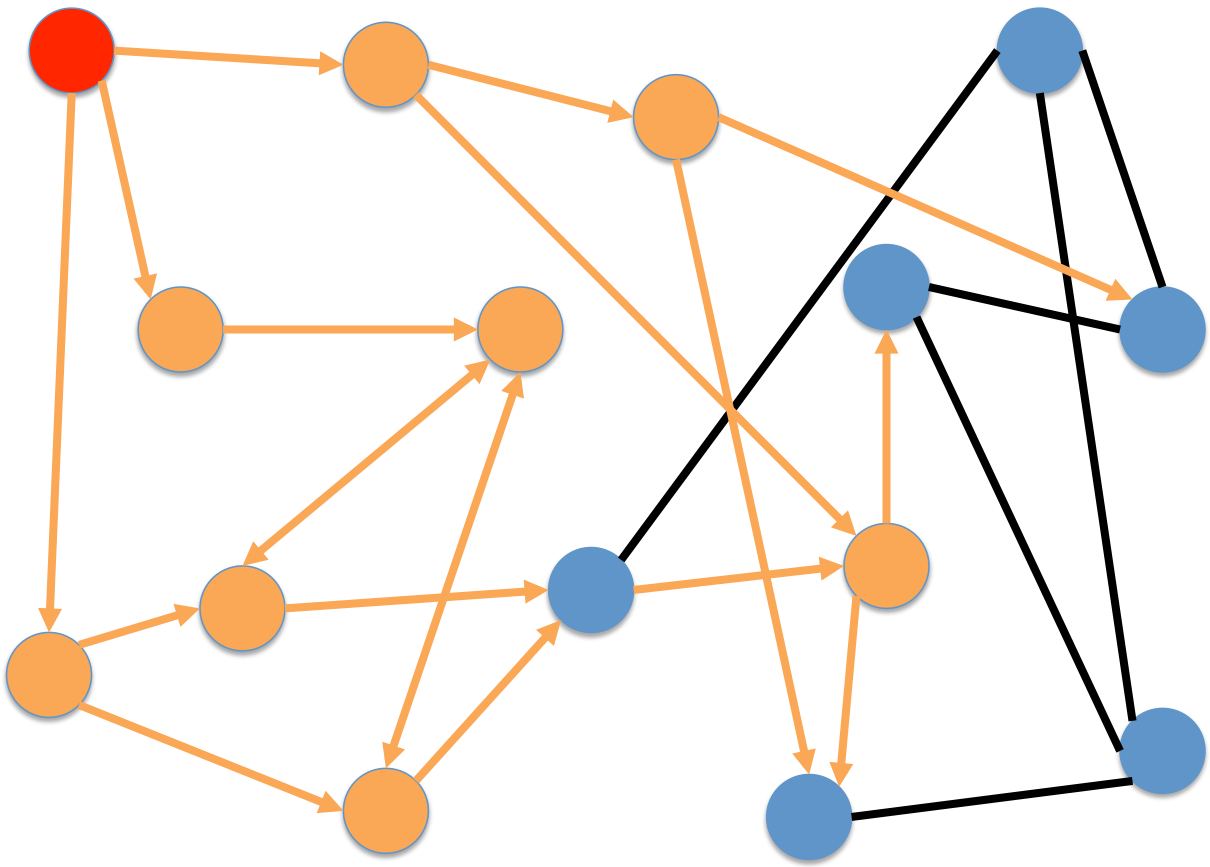
The Academic View: Data Dissemination

Design Alternatives: Flood



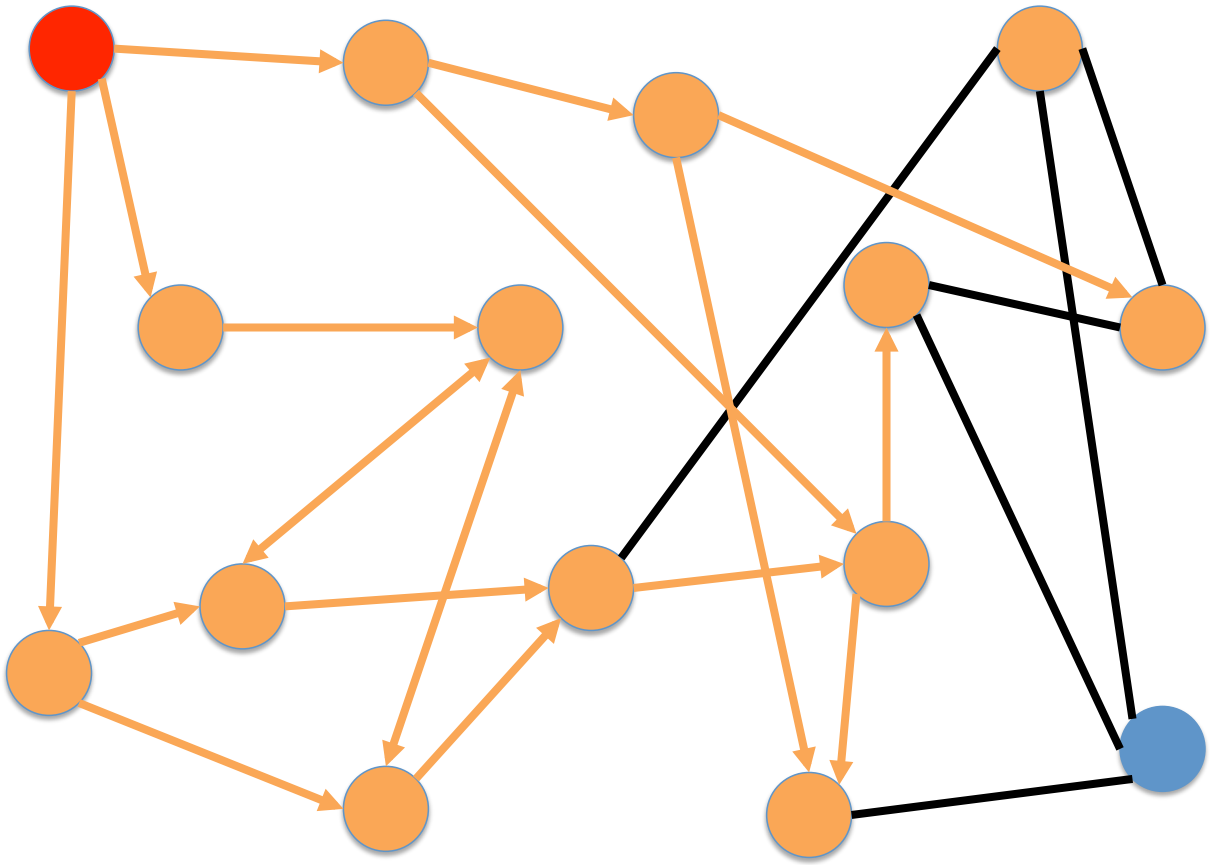
The Academic View: Data Dissemination

Design Alternatives: Flood



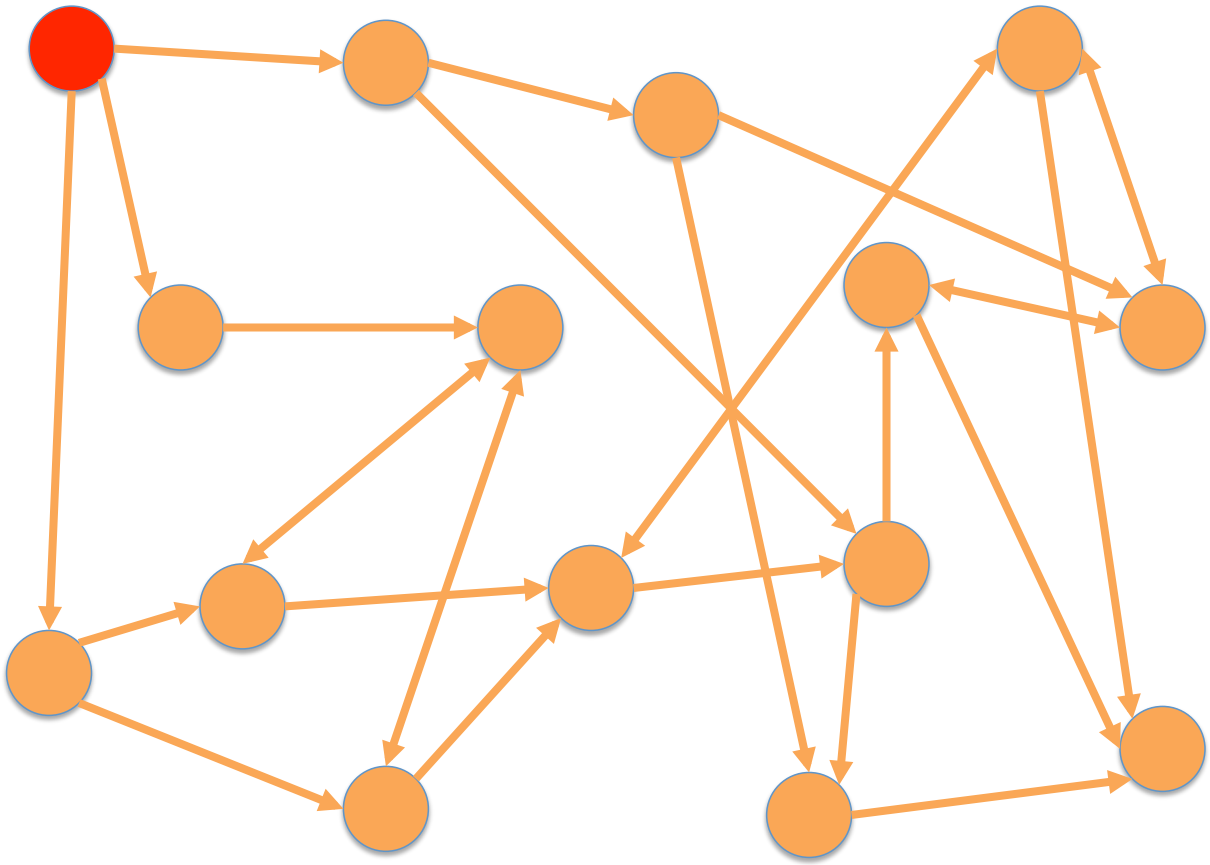
The Academic View: Data Dissemination

Design Alternatives: Flood



The Academic View: Data Dissemination

Design Alternatives: Flood



The Academic View: Data Dissemination

Design Alternatives: Flood

Flood

- Positive Aspects:
  - Load Distribution.
  - Simple Design, and a random overlay is easier to maintain than a tree.
  - Robust (due to inherent redudancy).
- Negative Aspects:
  - No longer that efficient (nodes receive and are required to process several copies of each message).

The Academic View: Data Dissemination

Design Alternatives: Flood

Flood

- Positive Aspects:
  - Load Distribution.
  - Simple Design, and a random overlay is easier to maintain than a tree.
  - Robust (due to inherent redudancy).
- Negative Aspects:
  - No longer that efficient (nodes receive and are required to process several copies of each message).

The Academic View: Data Dissemination

Design Alternatives: Gossip

Addressing Efficiency

Organize participants/nodes in a random, highly connected, graph and have nodes forward each message across a subset (f) of the links.

Gossip

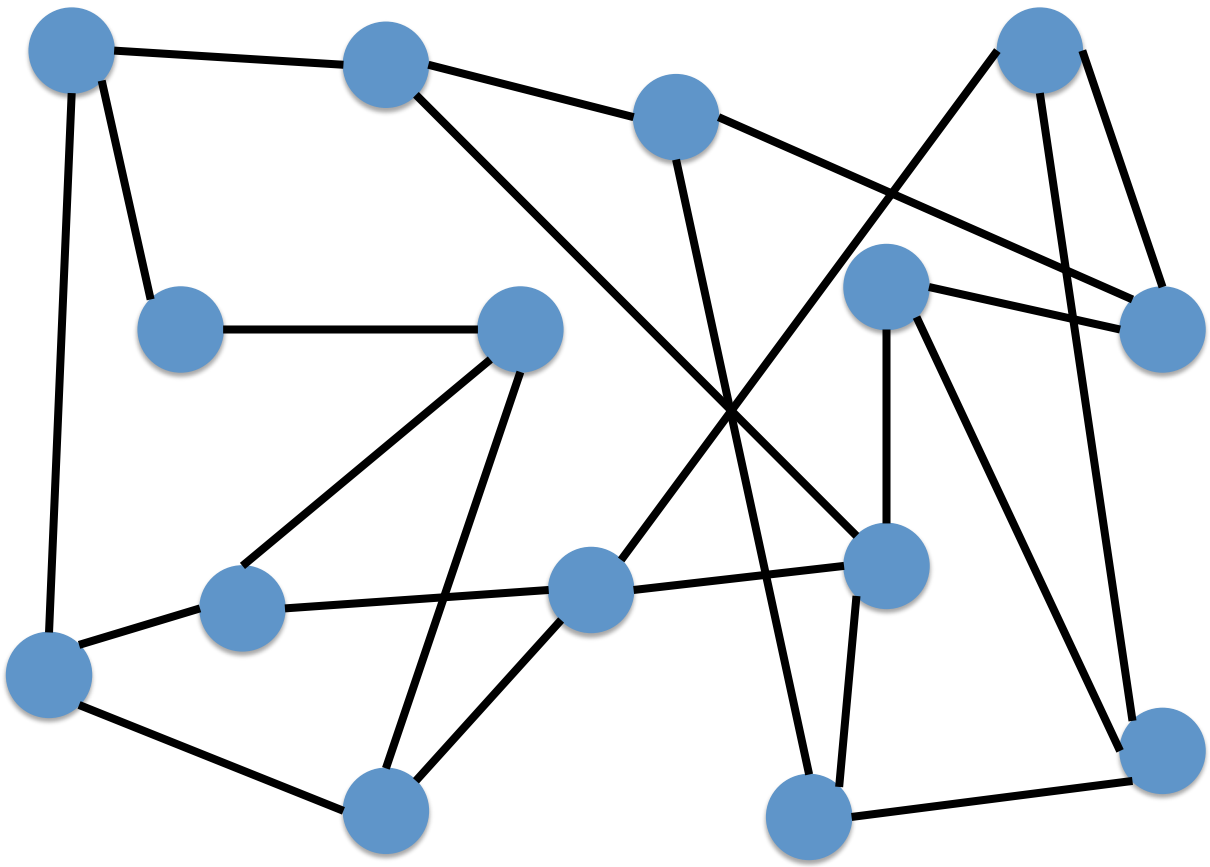
Addressing Efficiency

Organize participants/nodes in a random, highly connected, graph and have nodes forward each message across a subset (f) of the links.

Gossip

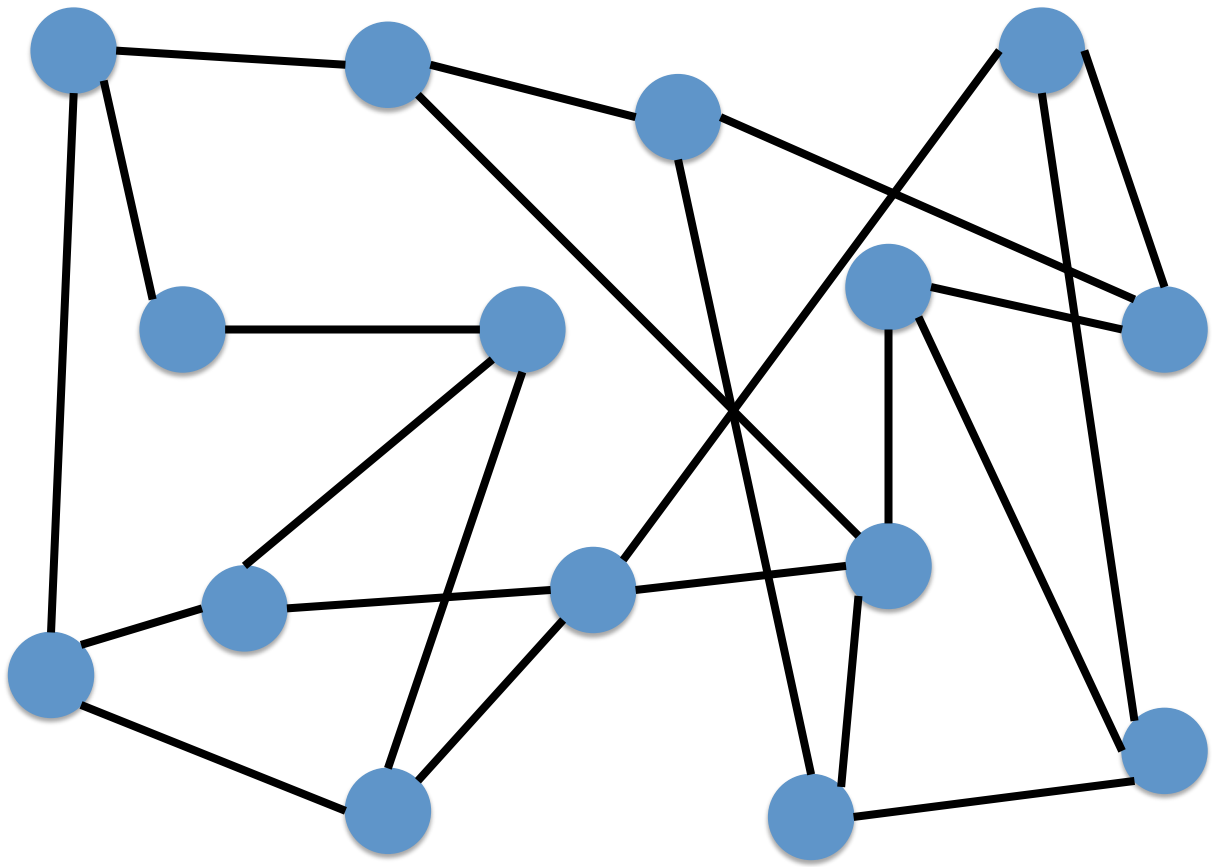
The Academic View: Data Dissemination

Design Alternatives: Gossip



The Academic View: Data Dissemination

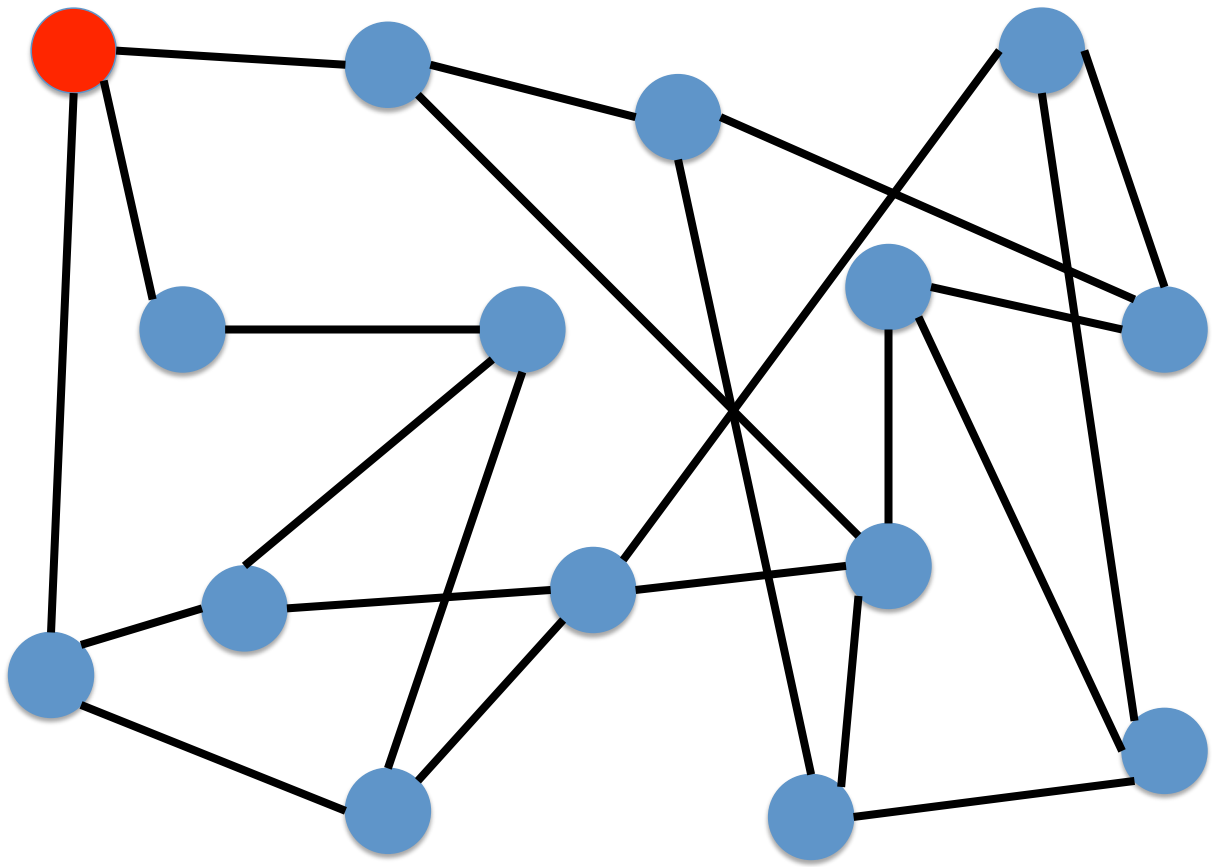
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

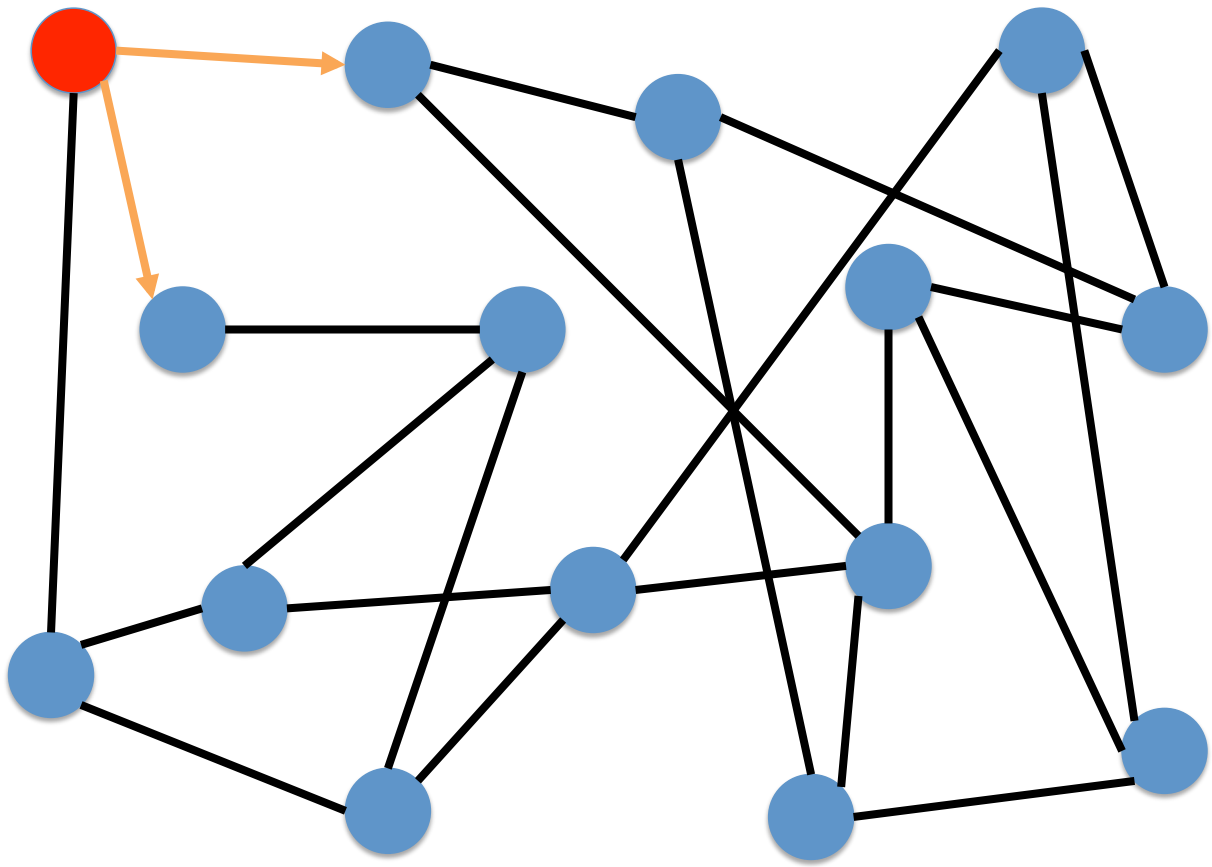
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

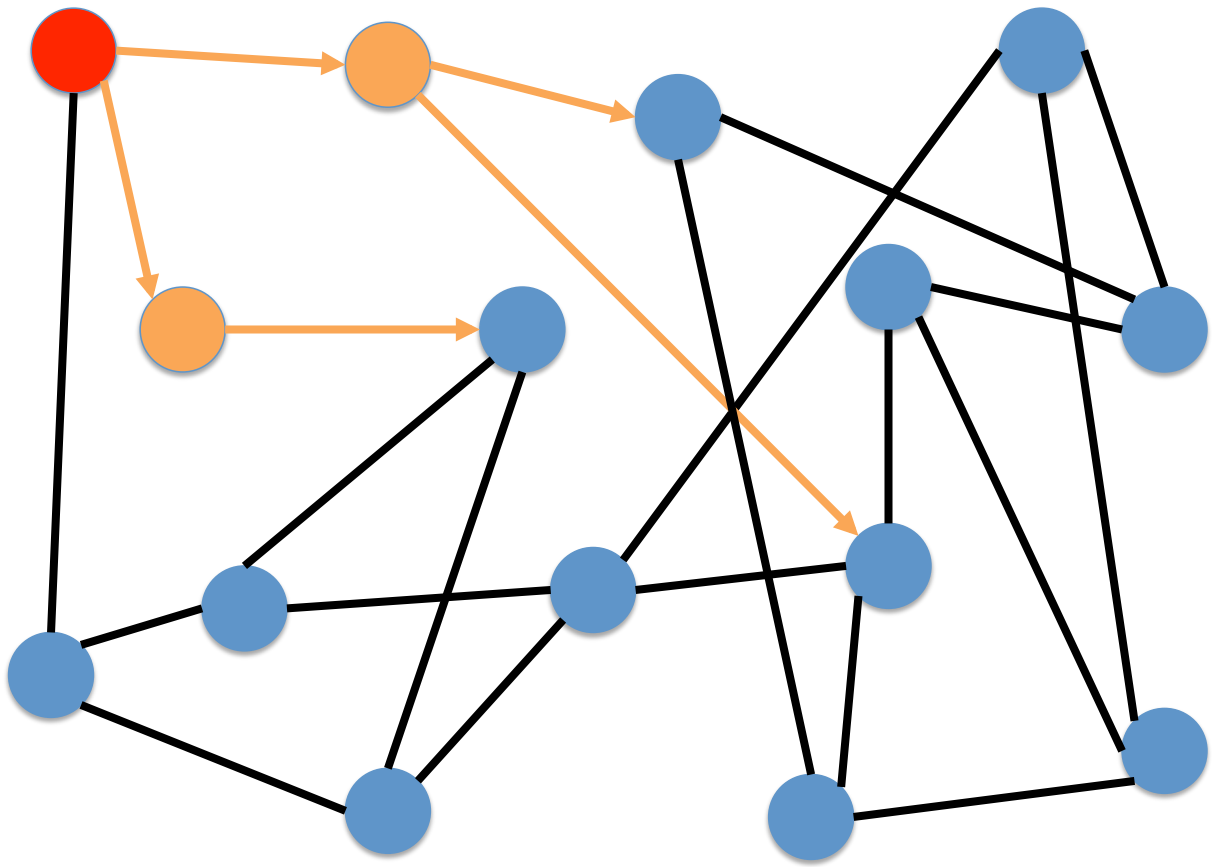
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

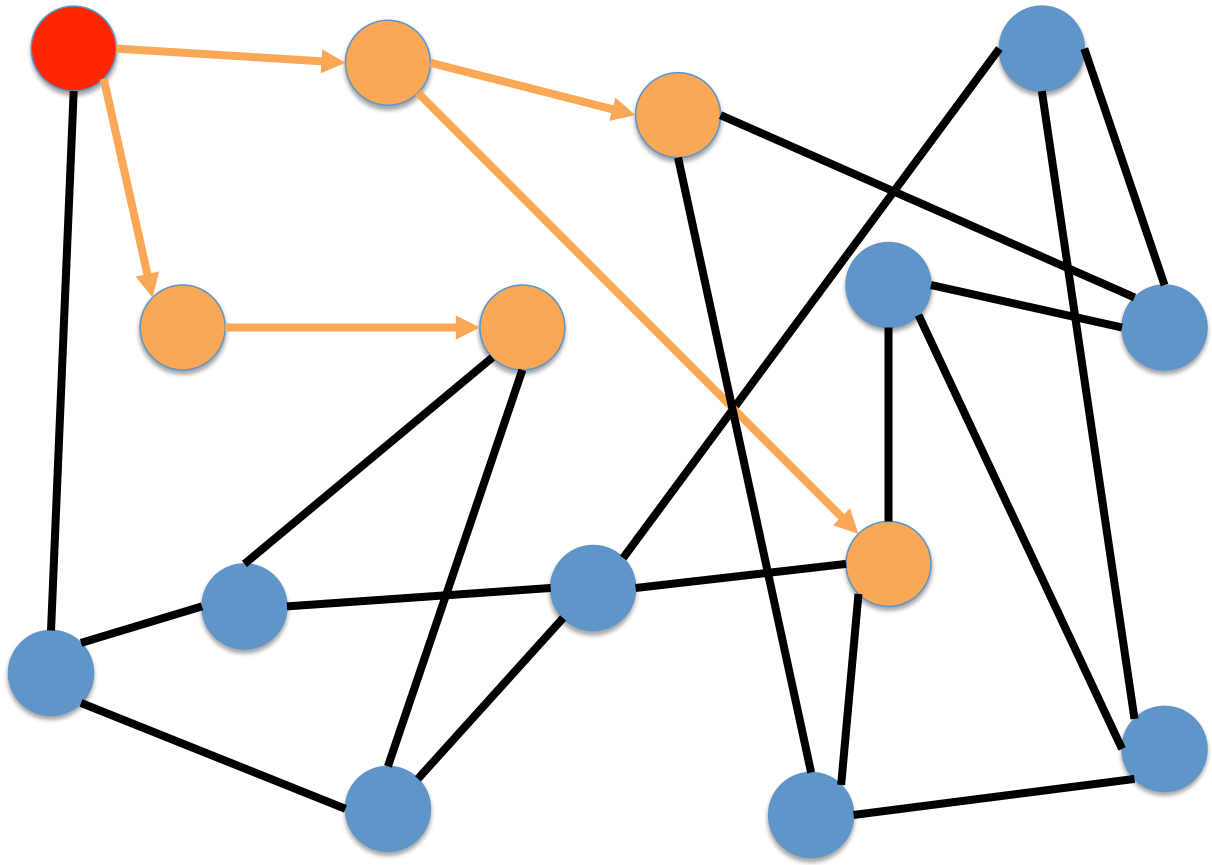
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

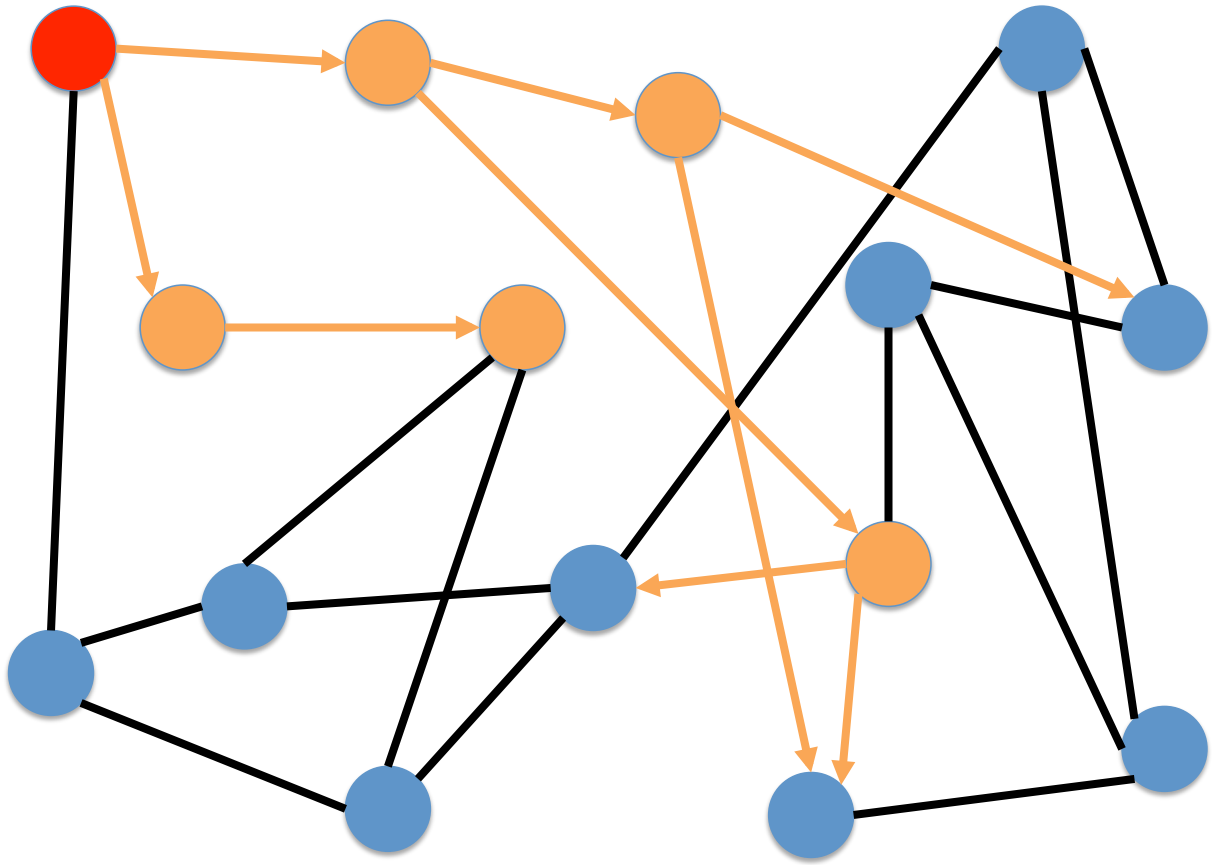
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

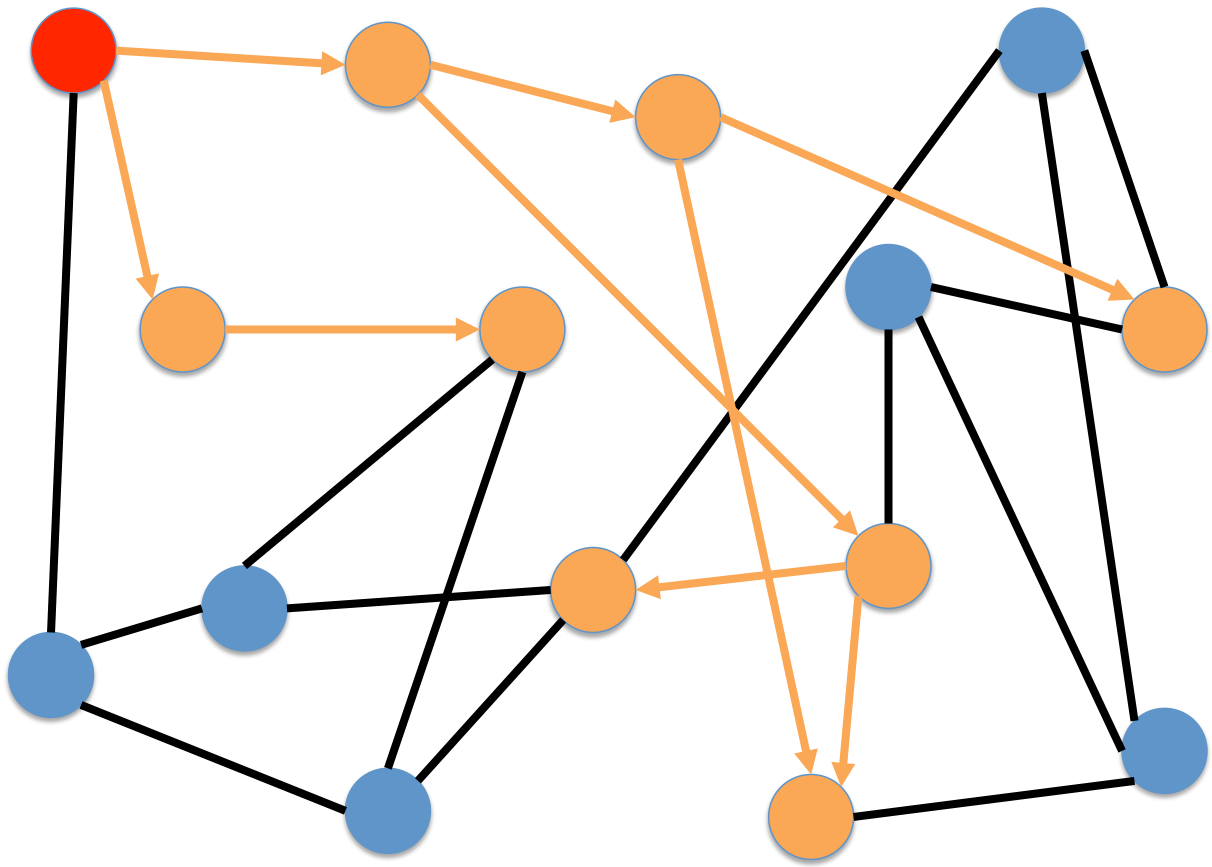
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

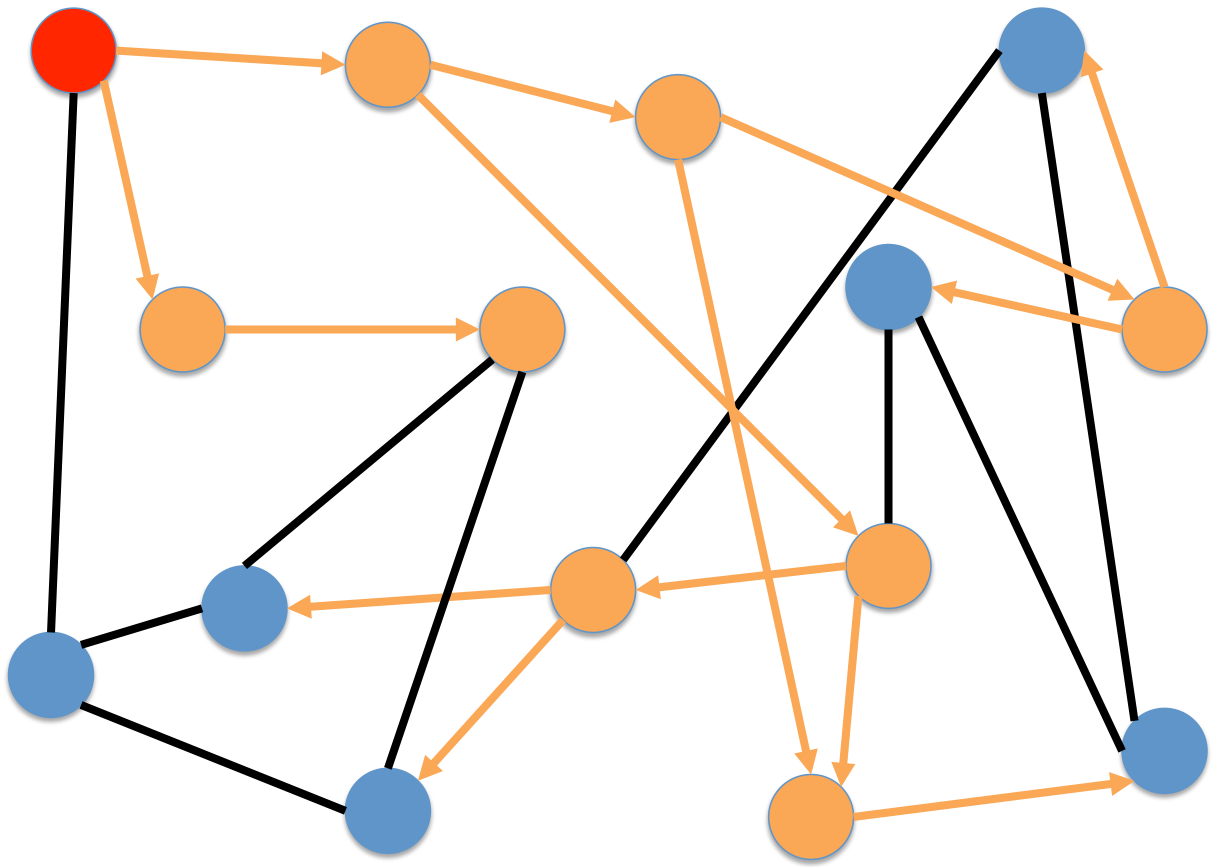
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

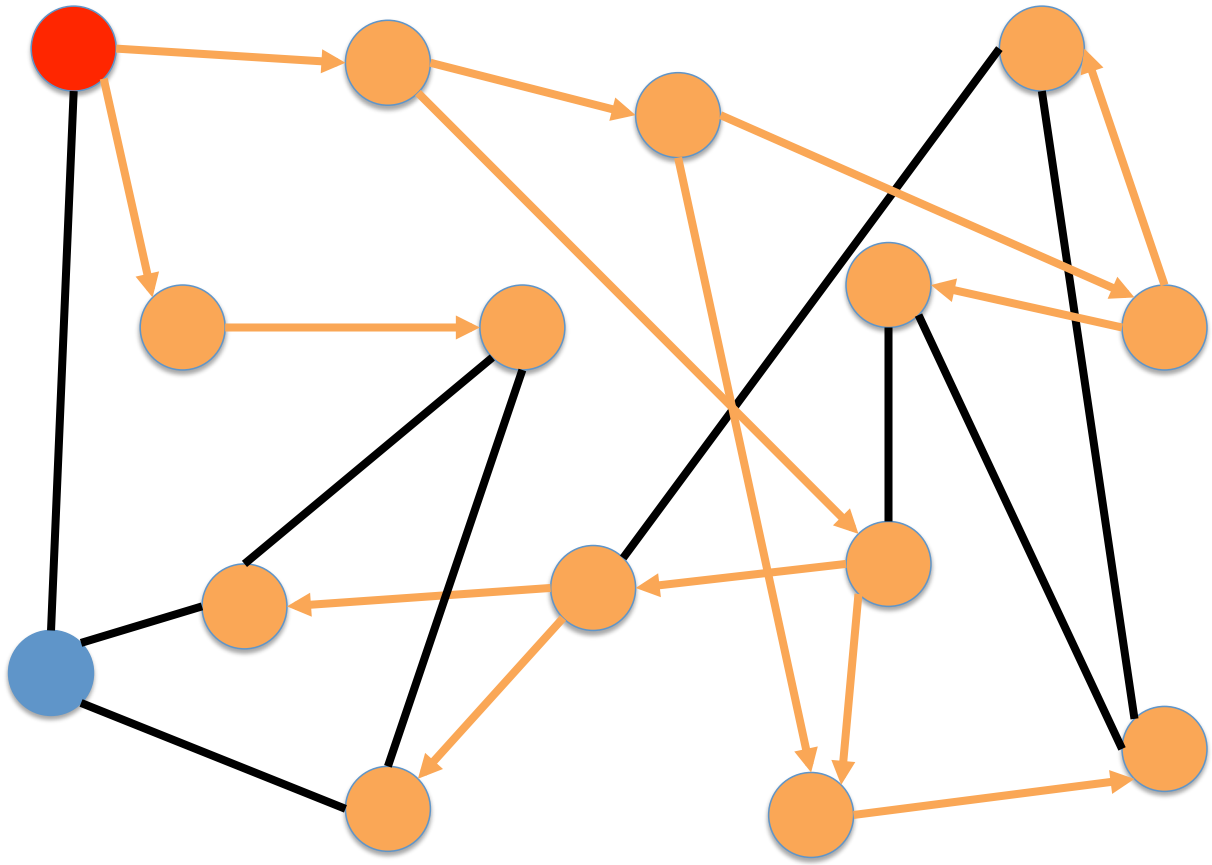
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

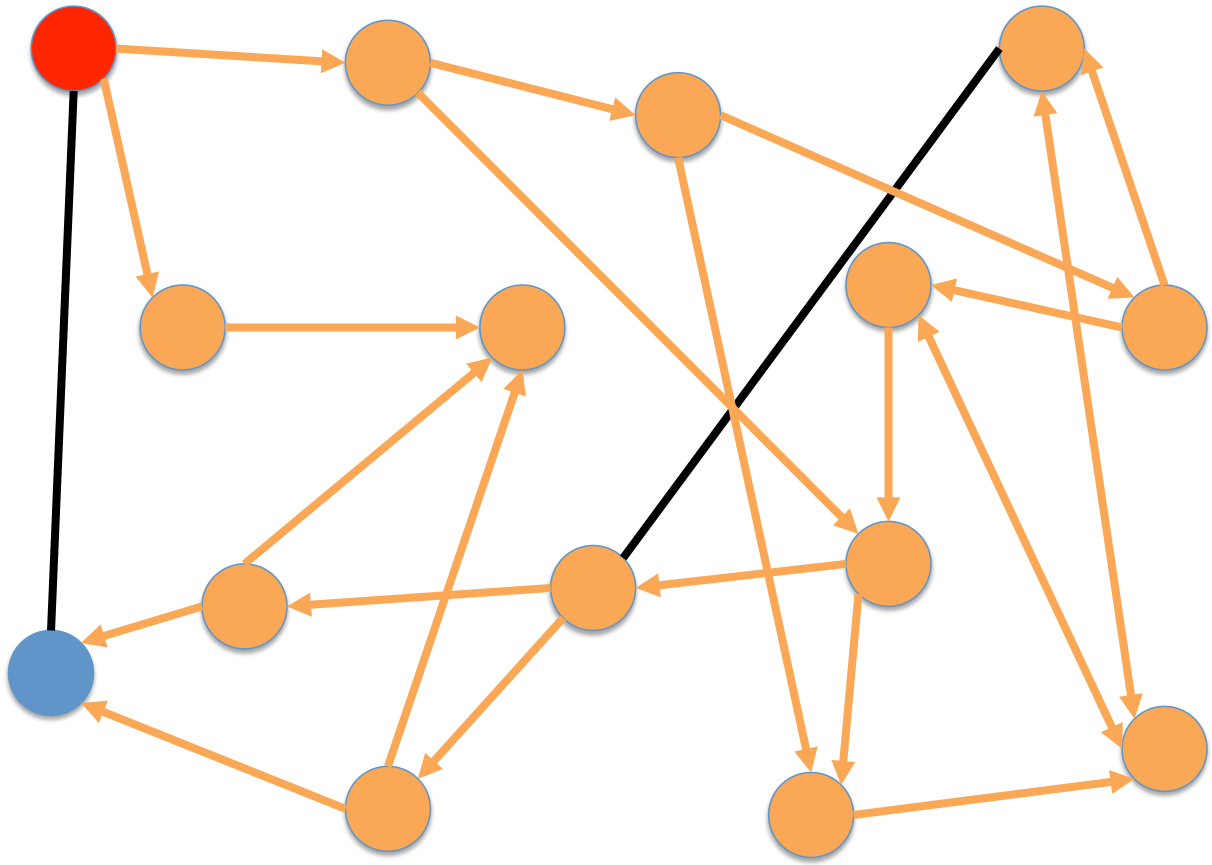
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

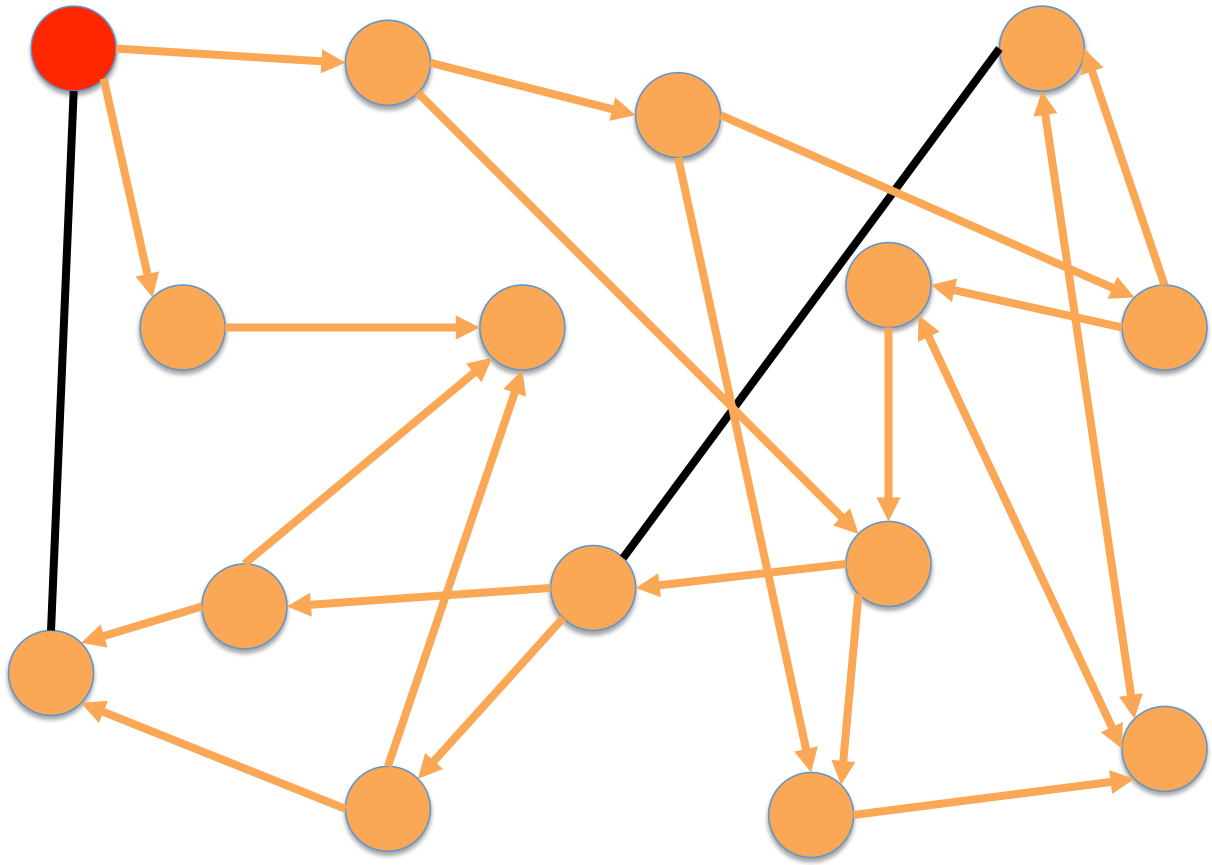
Design Alternatives: Gossip



Gossip *fanout* = 2

The Academic View: Data Dissemination

Design Alternatives: Gossip



Gossip *fanout* = 2

Gossip

- Positive Aspects:
  - Load Distribution.
  - Simple Design, and a random overlay is easier to maintain than a tree.
  - Robust (due to inherent redudancy).
- Negative Aspects:
  - Slightly inefficient (but better than flood).
  - No longer predictable (each message will be disseminated across different links).

The Academic View: Data Dissemination

Design Alternatives: Gossip

Gossip

- Positive Aspects:
  - Load Distribution.
  - Simple Design, and a random overlay is easier to maintain than a tree.
  - Robust (due to inherent redudancy).
- Negative Aspects:
  - Slightly inefficient (but better than flood).
  - No longer predictable (each message will be disseminated across different links).

Gossip

- Positive Aspects:
  - Load Distribution.
  - Simple Design, and a random overlay is easier to maintain than a tree.
  - Robust (due to inherent redudancy).
- Negative Aspects:
  - Slightly inefficient (but better than flood).
  - No longer predictable (each message will be disseminated across different links).

The Academic View: Data Dissemination

Design Alternatives: Embedded Tree

All these design alternatives:

- One-to-All.
- Spanning Tree.
- Flood.
- Gossip.

The Academic View: Data Dissemination

Design Alternatives: Embedded Tree

All these design alternatives:

- Spanning Tree.
- Flood.
- Gossip.

The Academic View: Data Dissemination

Design Alternatives: Embedded Tree

All these design alternatives:

- **Spanning Tree [Efficiency].**
- Flood.
- Gossip.

The Academic View: Data Dissemination

Design Alternatives: Embedded Tree

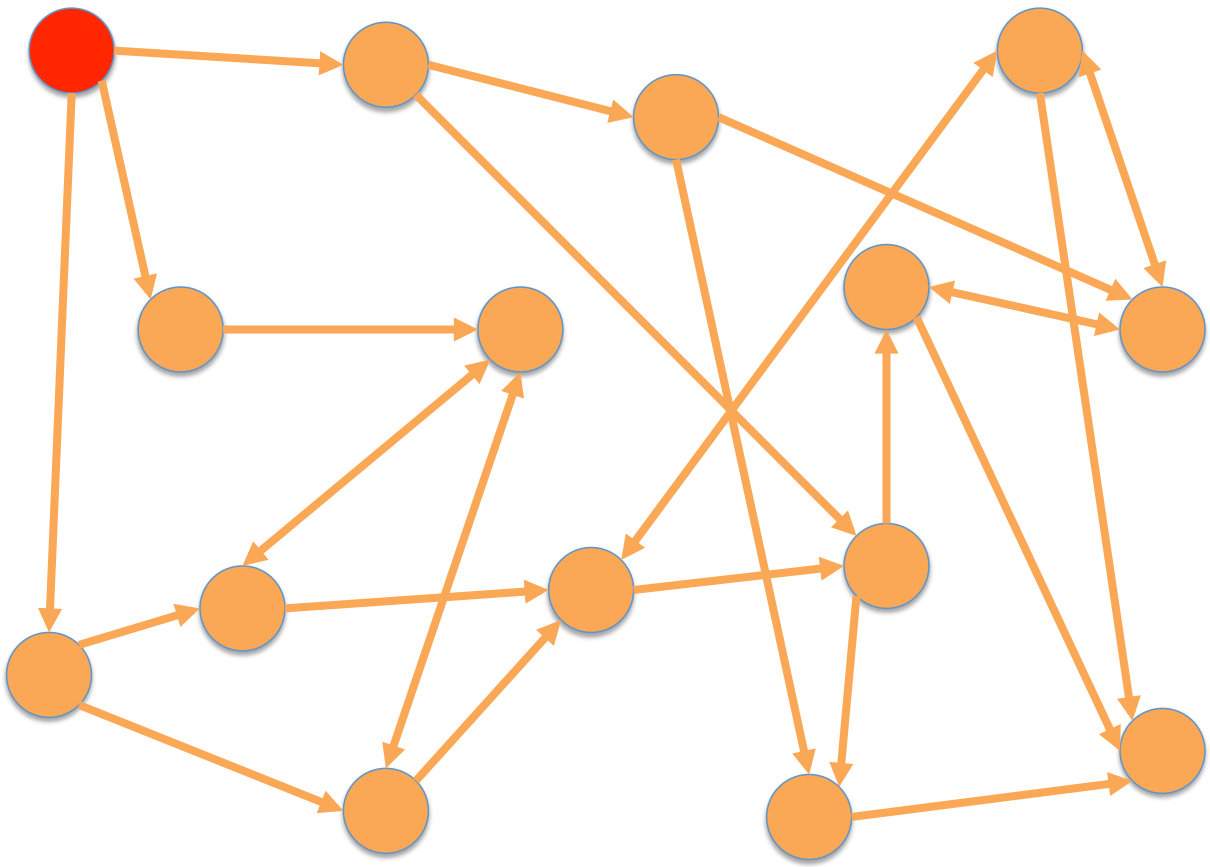
All these design alternatives:

- **Spanning Tree [Efficiency].**
- **Flood. [Robustness].**
- **Gossip. [Robustness].**

At some point you start to see **trees** everywhere...

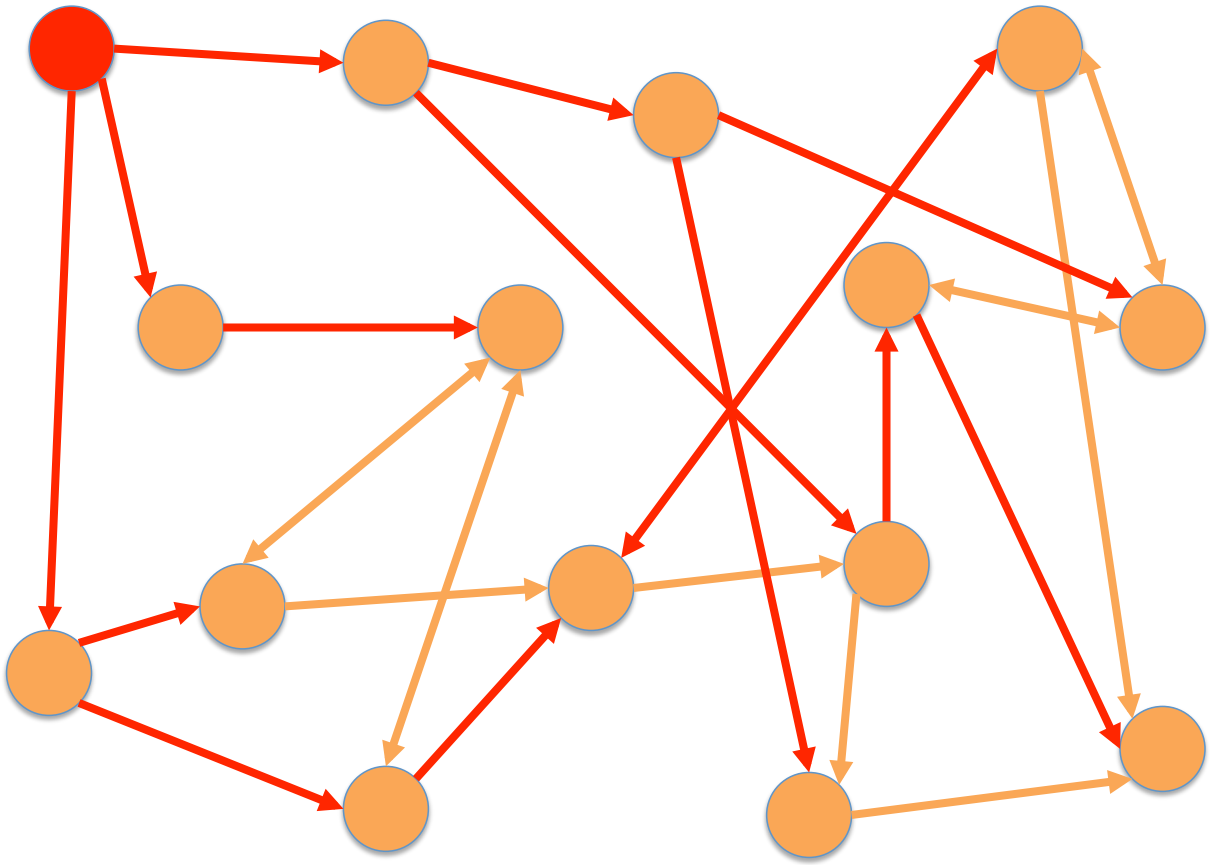
The Academic View: Data Dissemination

Design Alternatives: Embedded Tree



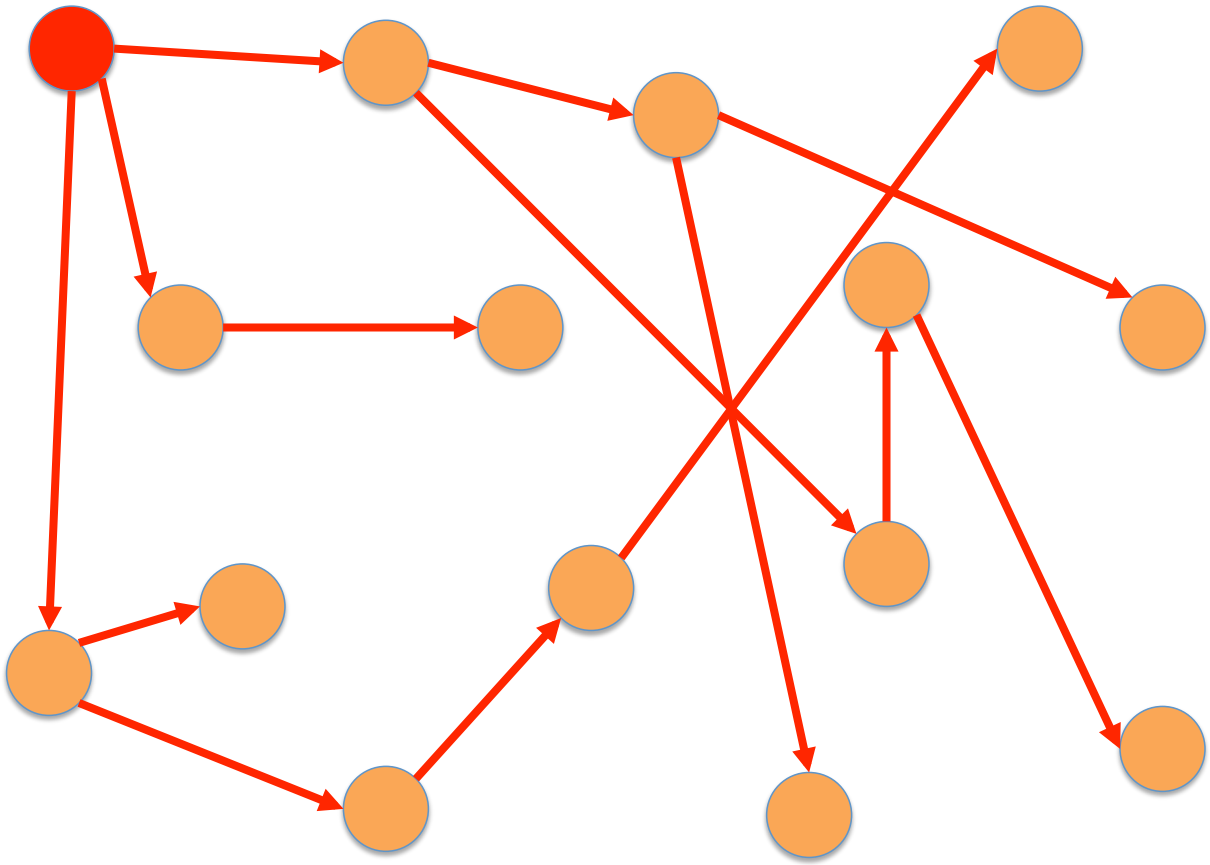
The Academic View: Data Dissemination

Design Alternatives: Embedded Tree



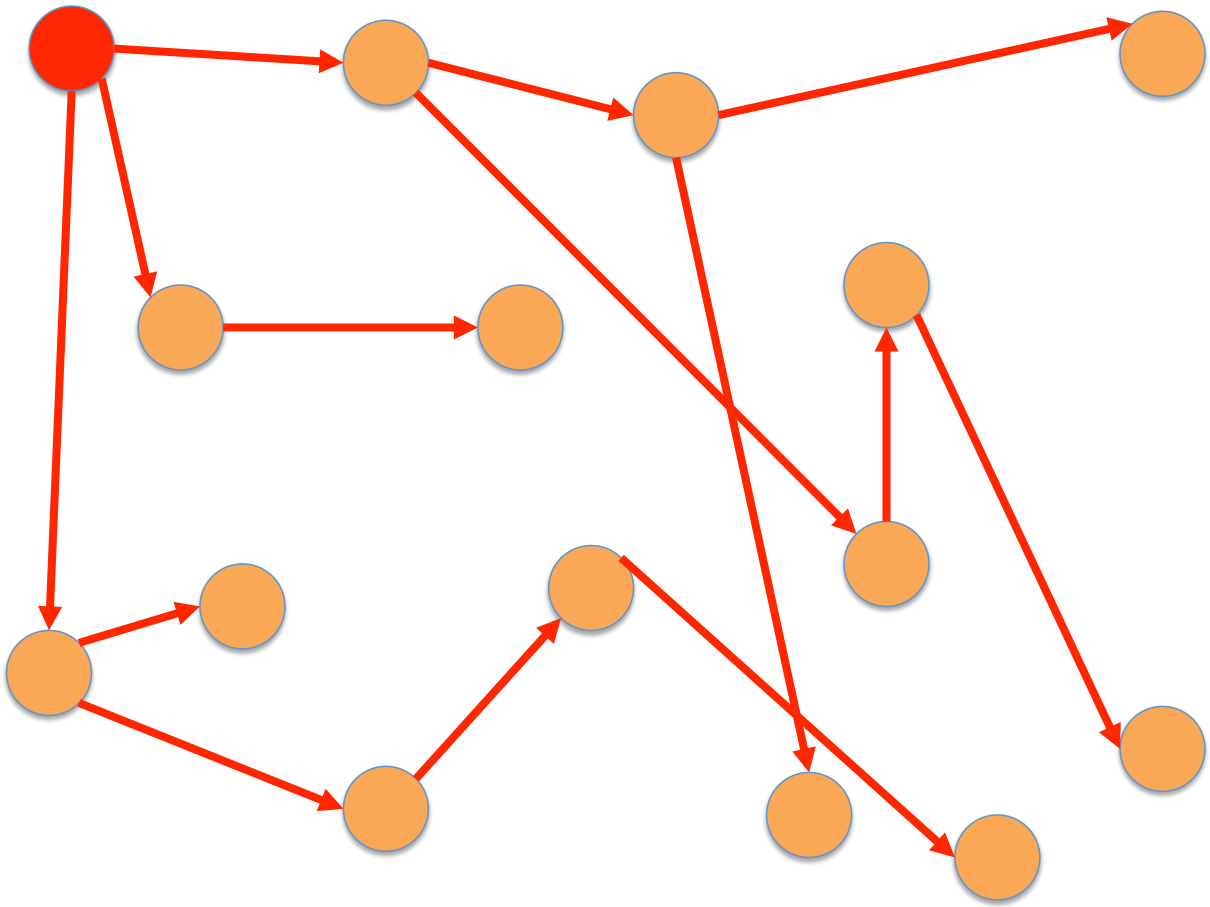
The Academic View: Data Dissemination

Design Alternatives: Embedded Tree



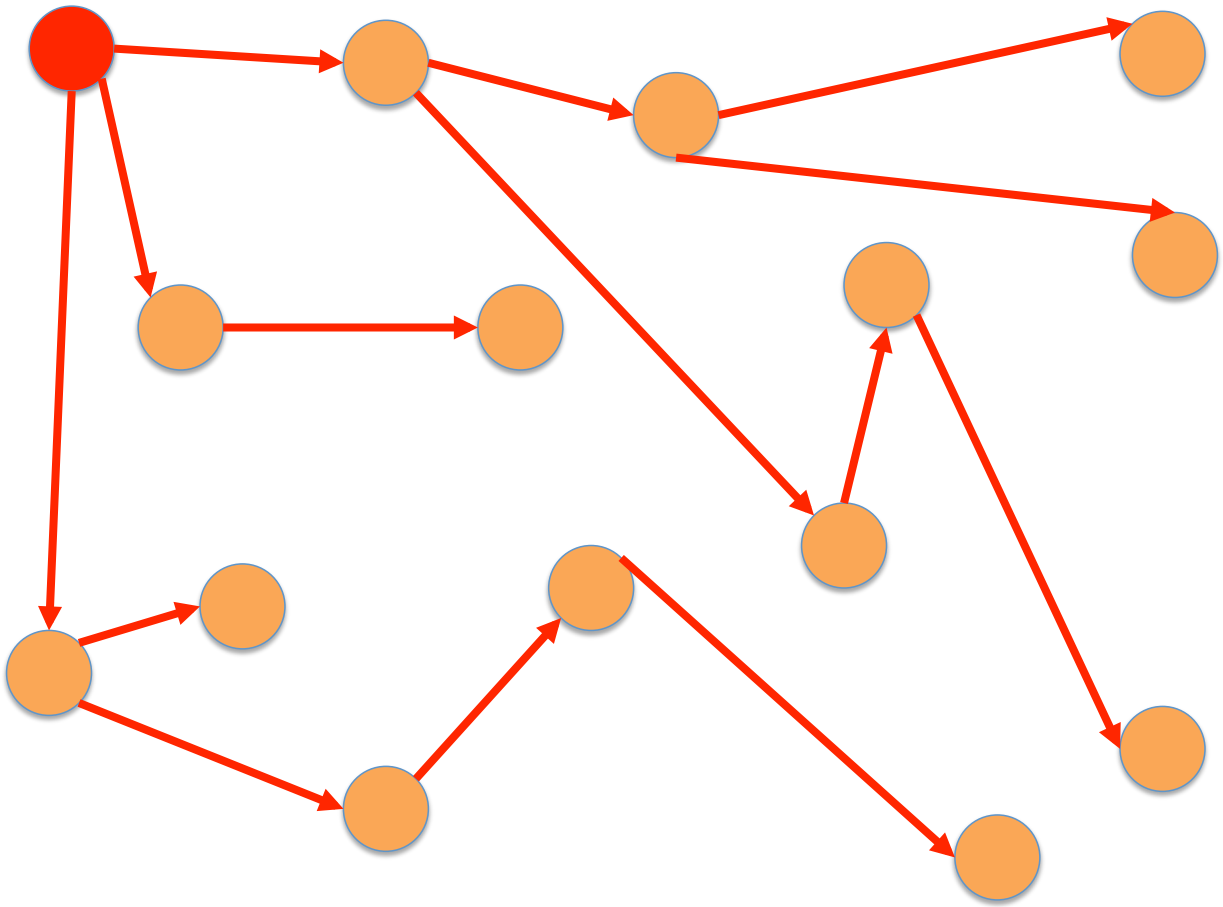
The Academic View: Data Dissemination

Design Alternatives: Embedded Tree



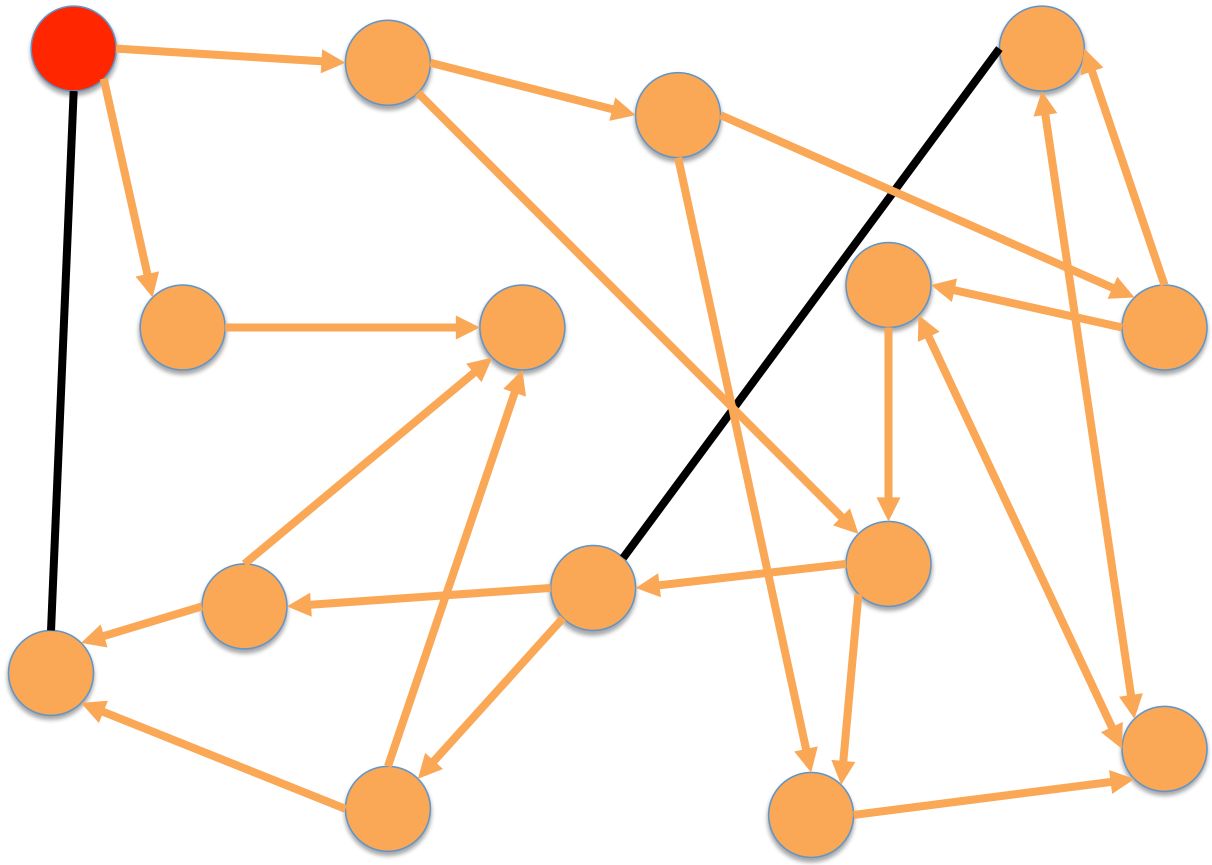
The Academic View: Data Dissemination

Design Alternatives: Embedded Tree



The Academic View: Data Dissemination

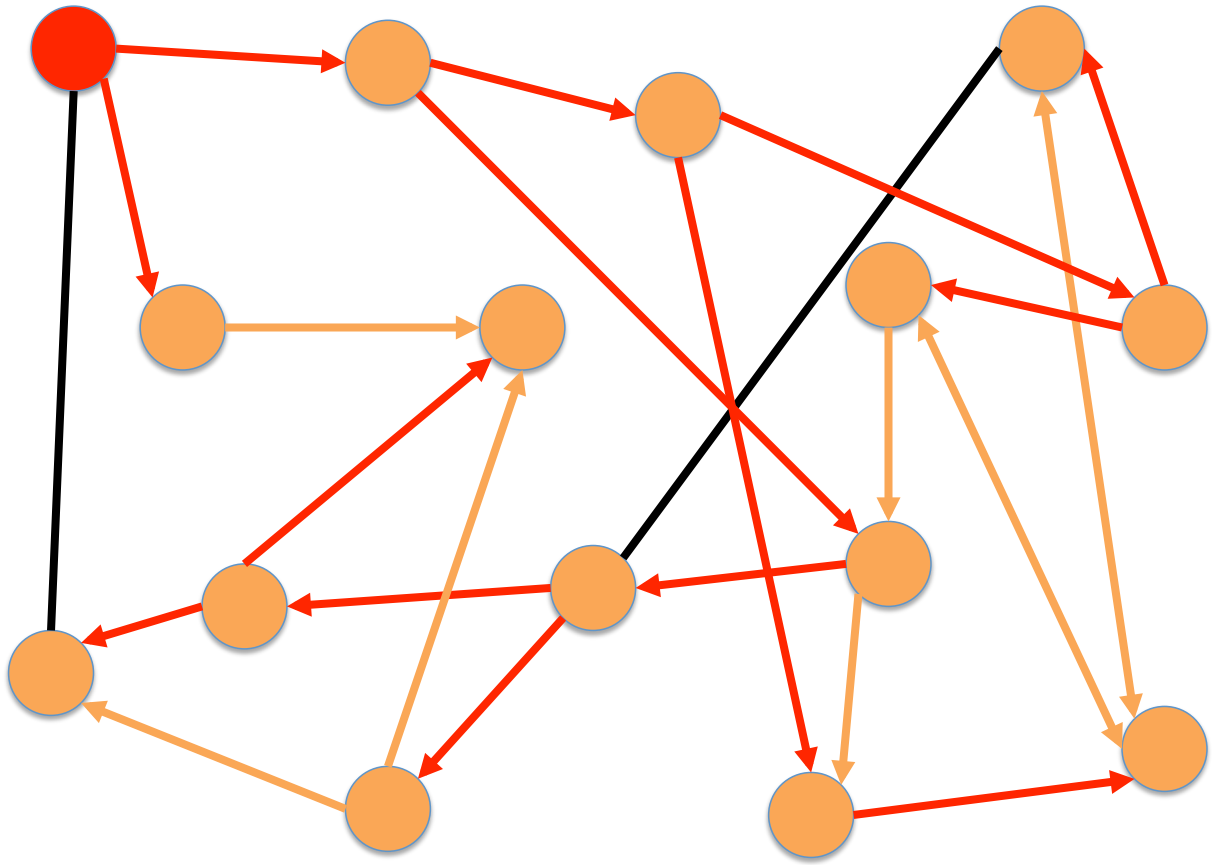
Design Alternatives: Embedded Tree



Gossip *fanout* = 2

The Academic View: Data Dissemination

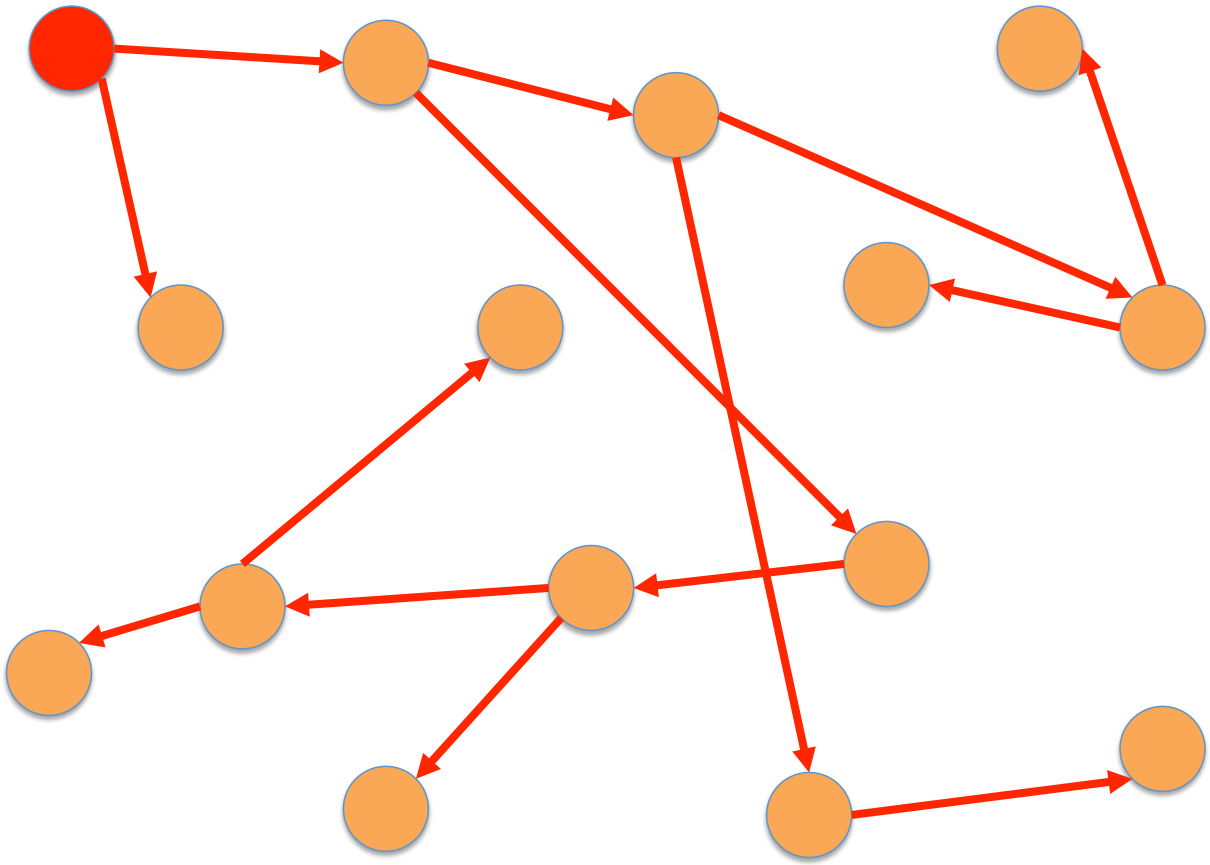
Design Alternatives: Embedded Tree



Gossip *fanout* = 2

The Academic View: Data Dissemination

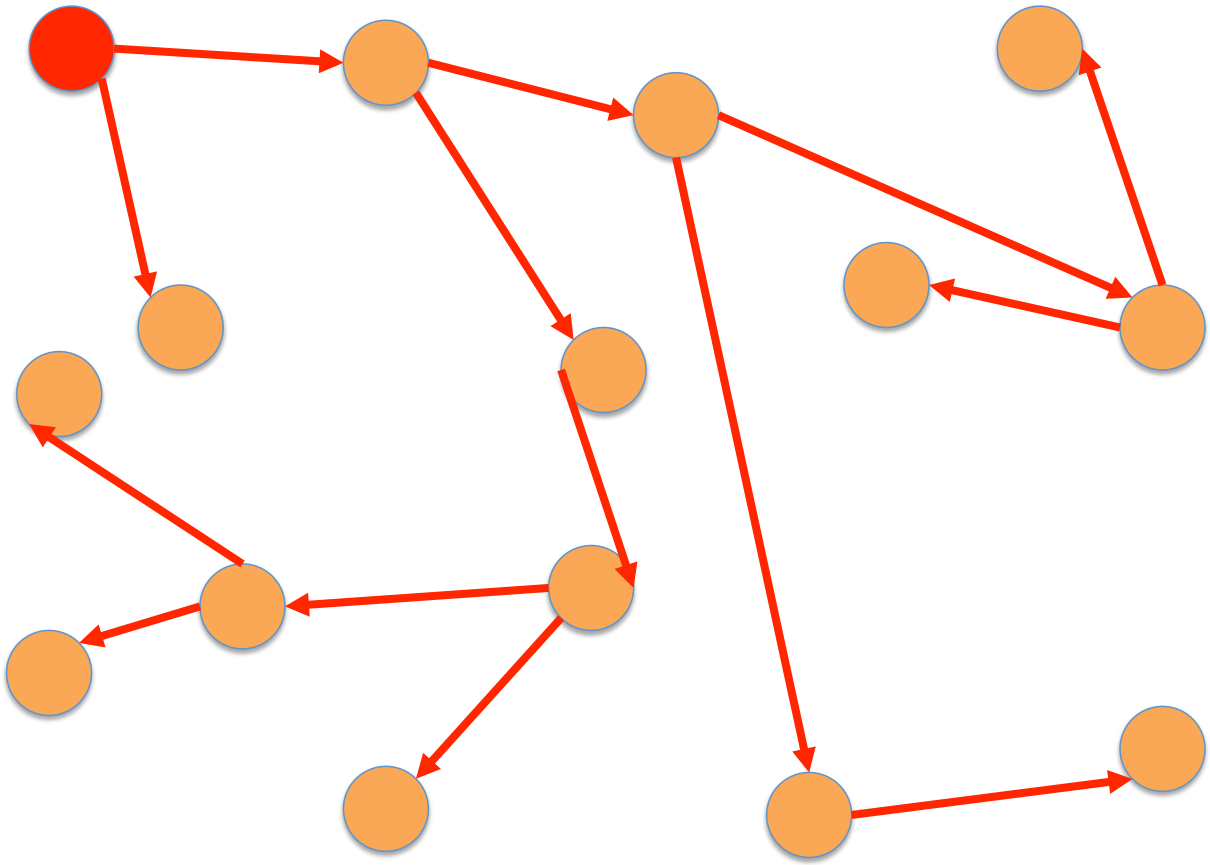
Design Alternatives: Embedded Tree



Gossip *fanout* = 2

The Academic View: Data Dissemination

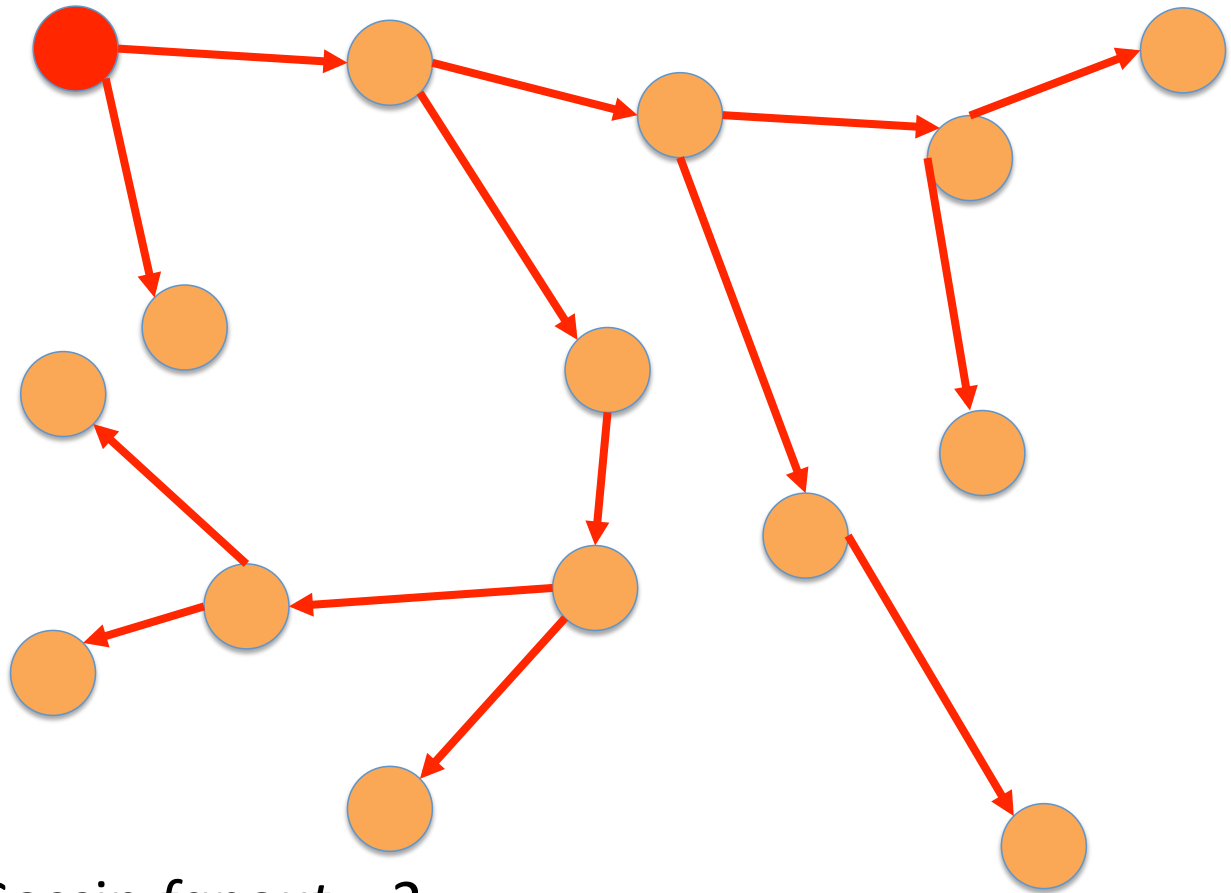
Design Alternatives: Embedded Tree



Gossip *fanout* = 2

The Academic View: Data Dissemination

Design Alternatives: Embedded Tree



Gossip *fanout* = 2

The Academic View: Data Dissemination

Design Alternatives: Embedded Tree

Observation:

The closure of the paths the lead to the first delivery of a message to each node forms a tree.

This observation:

Lead us to design the **Plumtree protocol**.

The Academic View: Data Dissemination

Design Alternatives: Embedded Tree

Observation:

The closure of the paths the lead to the first delivery of a message to each node forms a tree.

This observation:

Lead us to design the **Plumtree protocol**.

Embedded Trees: Plumtree Protocol

Epidemic Broadcast Trees.
João Leitão, José Pereira and Luís Rodrigues
Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, Beijing, China, October, 2007.


Embedded Trees: Plumtree Protocol

Gossip modes of operation

- There are 1001 ways to Gossip...

Eager push

Nodes send the message payload to random selected peers as soon as they receive a message for the first time.

Lazy push

When a node receive a message for the first time, they only send that message identifier to random selected peers. If those peers have not received the message, they make an explicit request.

Embedded Trees: Plumtree Protocol

Gossip modes of operation

- There are 1001 ways to Gossip...

Eager push

Nodes send the message payload to random selected peers as soon as they receive a message for the first time.

Lazy push

When a node receive a message for the first time, they only send that message identifier to random selected peers. If those peers have not received the message, they make an explicit request.

Embedded Trees: Plumtree Protocol

Gossip modes of operation

- There are 1001 ways to Gossip...

Eager push

Nodes send the message payload to random selected peers as soon as they receive a message for the first time.

Lazy push

When a node receive a message for the first time, they only send that message identifier to random selected peers. If those peers have not received the message, they make an explicit request.

Embedded Trees: Plumtree Protocol

Overview

Plumtree: **p**ush-**l**azy-p**u**sh **m**ulticast **tree**.

- It operates as any gossip protocol, each node gossips with f neighbors on top of a random overlay.
- Decentralized.
- Creates (and maintain) an embedded tree on a random overlay.
- It combines two distinct gossip strategies:
  - Eager Push: In the random overlay links that belong to the embedded spanning tree.
  - Lazy Push: In the remaining random overlay links.
- TCP is used between nodes as it offers a better reliability and an additional fault detection mechanism.

Embedded Trees: Plumtree Protocol

Overview

Plumtree: **p**ush-**l**azy-p**u**sh **m**ulticast **tree**.

- It operates as any gossip protocol, each node gossips with f neighbors on top of a random overlay.
- Decentralized.
- Creates (and maintain) an embedded tree on a random overlay.
- It combines two distinct gossip strategies:
  - Eager Push: In the random overlay links that belong to the embedded spanning tree.
  - Lazy Push: In the remaining random overlay links.
- TCP is used between nodes as it offers a better reliability and an additional fault detection mechanism.

Embedded Trees: Plumtree Protocol

Overview

Plumtree: **p**ush-**l**azy-p**u**sh **m**ulticast **tree**.

- It operates as any gossip protocol, each node gossips with f neighbors on top of a random overlay.
- Decentralized.
- Creates (and maintain) an embedded tree on a random overlay.
- It combines two distinct gossip strategies:
  - Eager Push: In the random overlay links that belong to the embedded spanning tree.
  - Lazy Push: In the remaining random overlay links.
- TCP is used between nodes as it offers a better reliability and an additional fault detection mechanism.

Embedded Trees: Plumtree Protocol

Building the Tree

- Intuition: Use the links in the overlay that generated a message delivery.
- After initialization all links in the overlay are candidates do belong to the spanning tree.
- When a message is broadcasted, all links used to disseminate a redundant message are pruned from the tree.
- If the random overlay is connected, after the dissemination of a message the tree will cover all nodes.

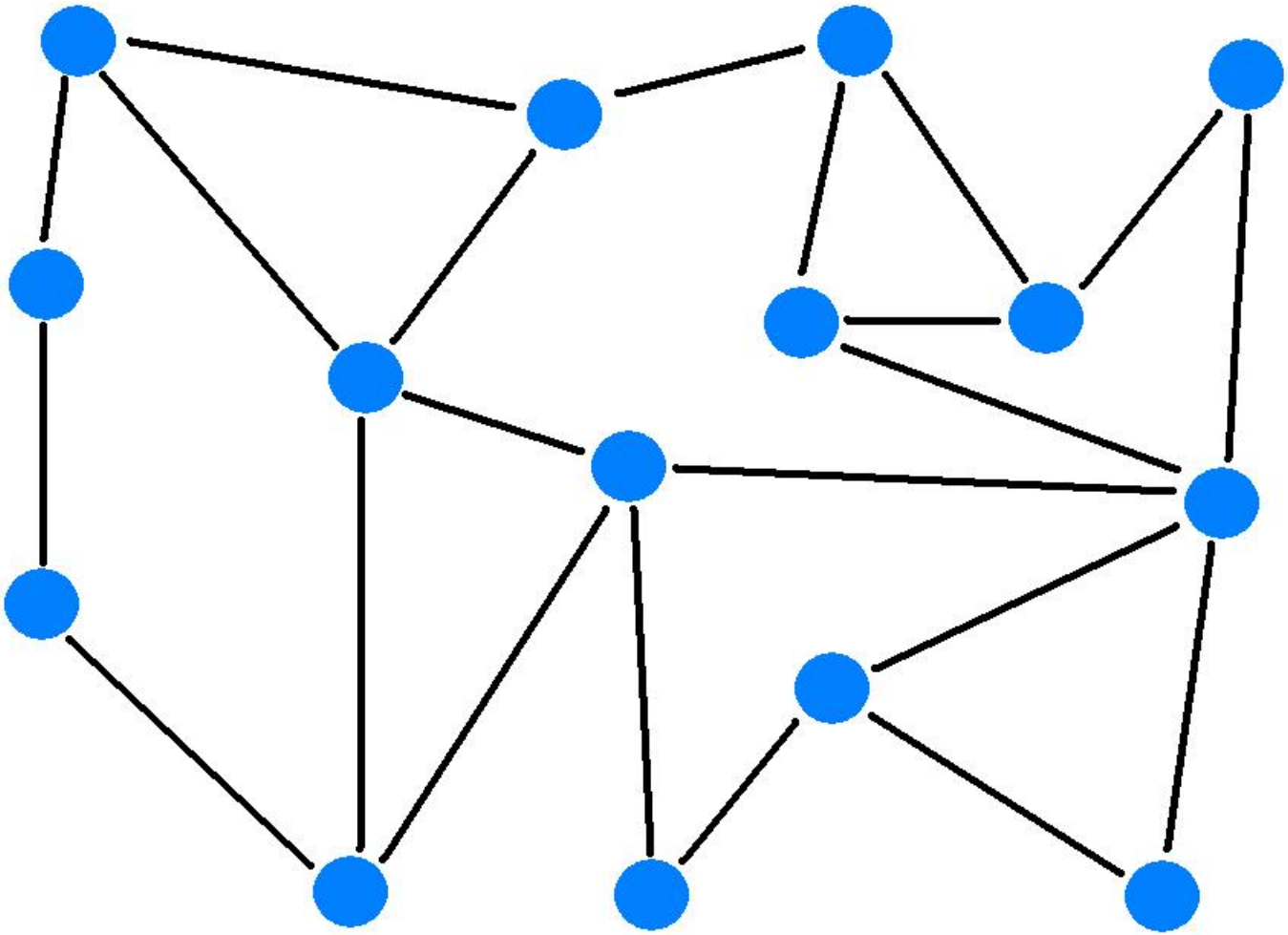
Embedded Trees: Plumtree Protocol

Building the Tree

- Intuition: Use the links in the overlay that generated a message delivery.
- After initialization all links in the overlay are candidates do belong to the spanning tree.
- When a message is broadcasted, all links used to disseminate a redundant message are pruned from the tree.
- If the random overlay is connected, after the dissemination of a message the tree will cover all nodes.

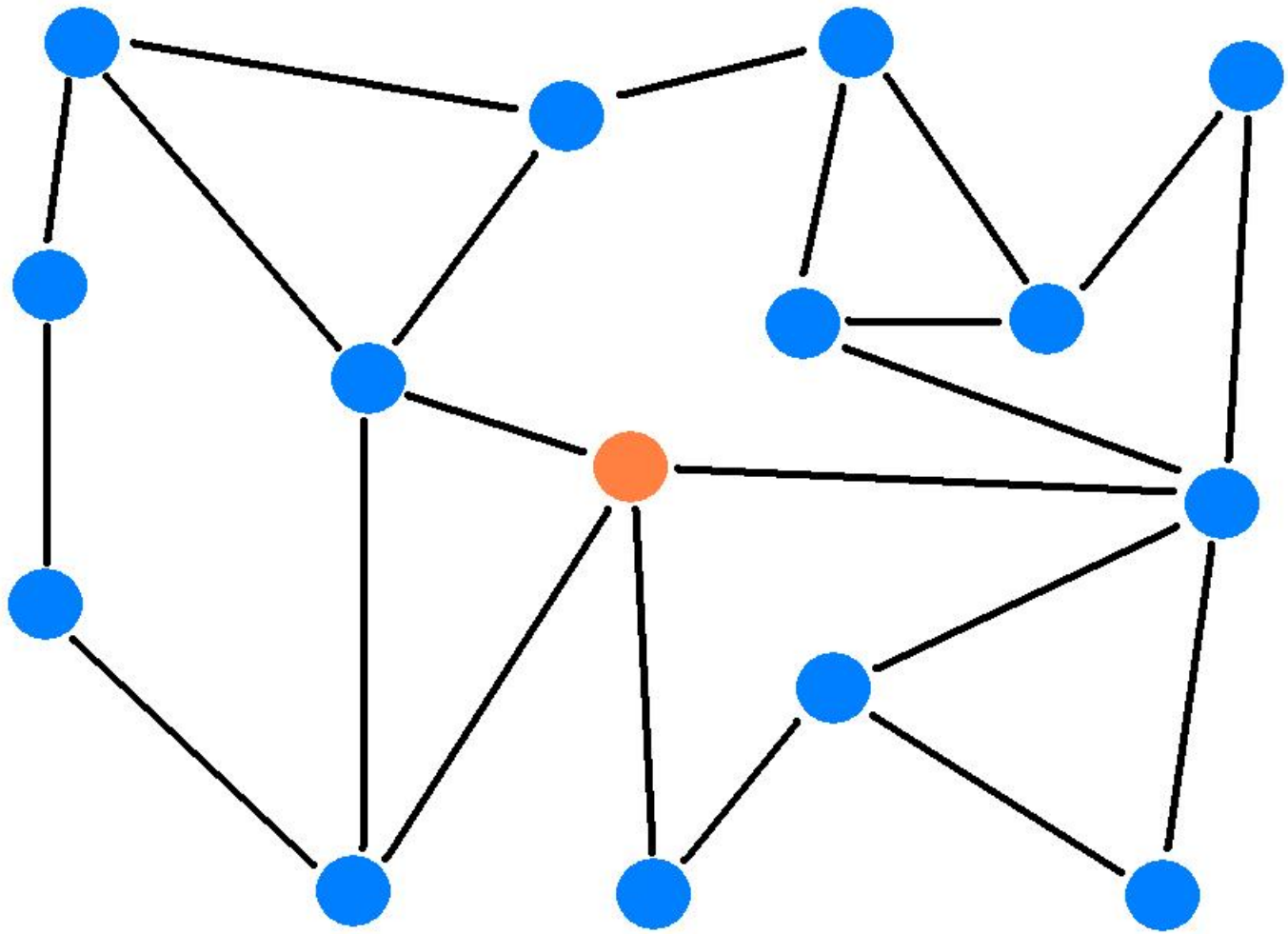
Embedded Trees: Plumtree Protocol

Building the Tree



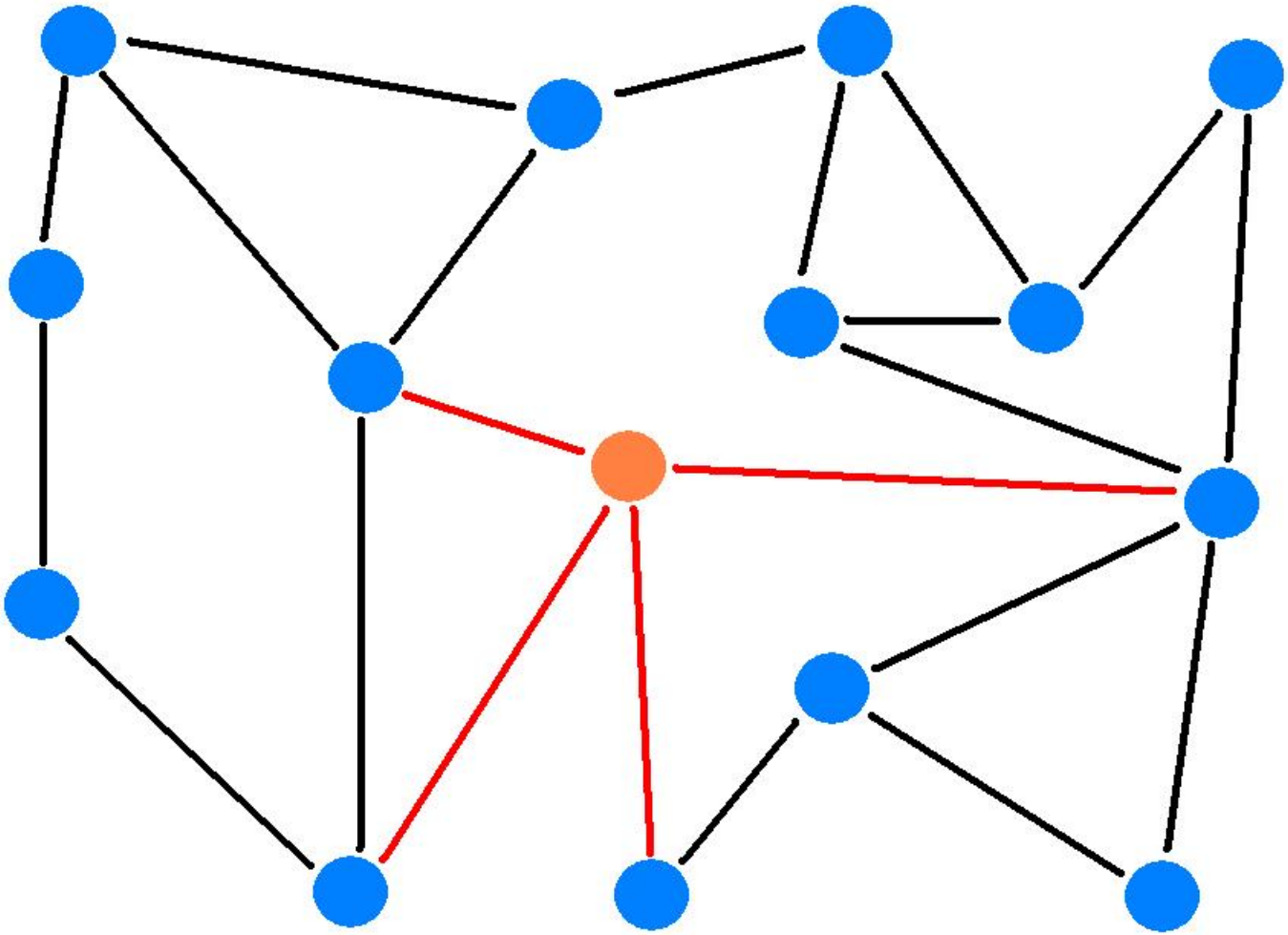
Embedded Trees: Plumtree Protocol

Building the Tree



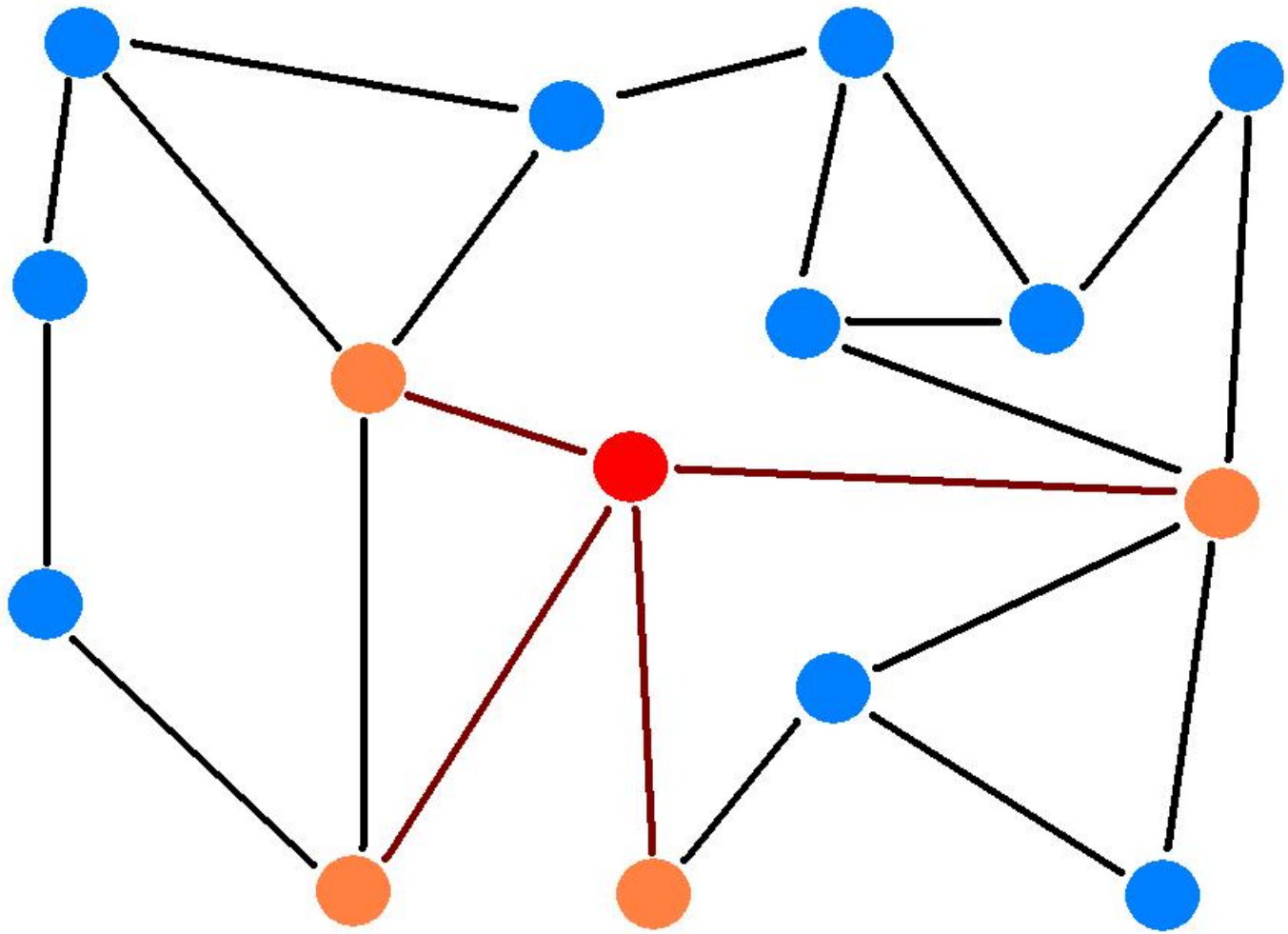
Embedded Trees: Plumtree Protocol

Building the Tree



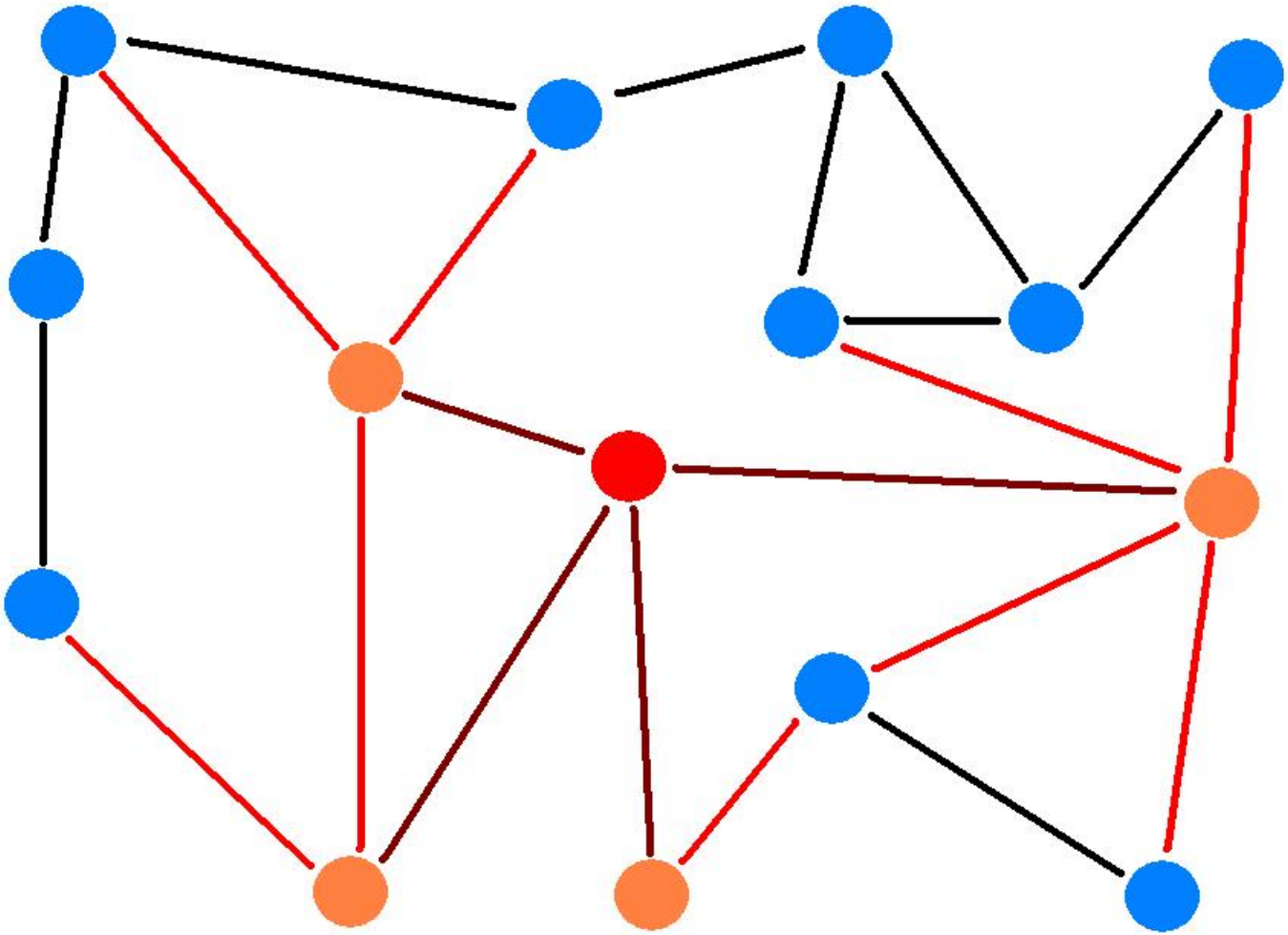
Embedded Trees: Plumtree Protocol

Building the Tree



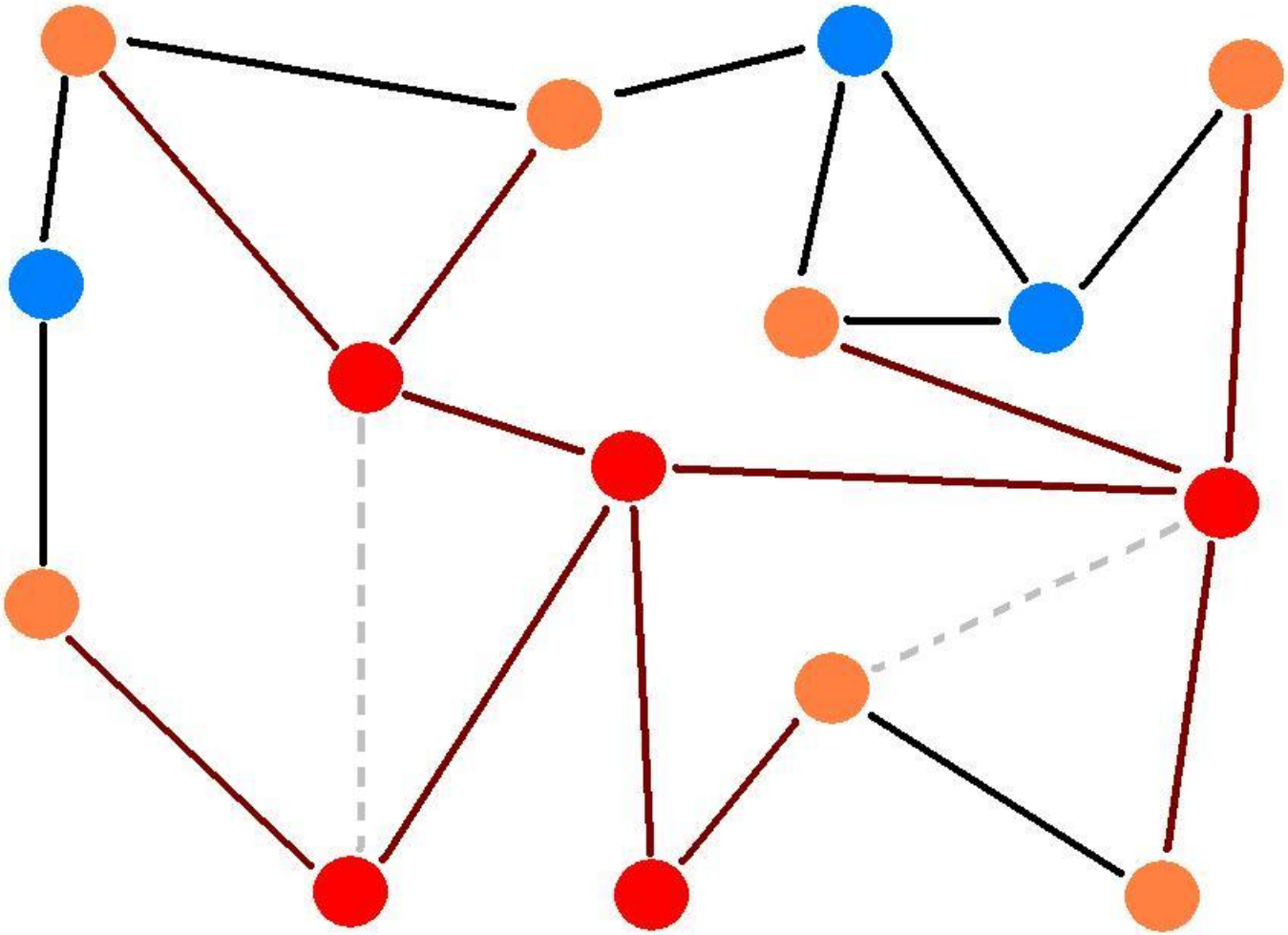
Embedded Trees: Plumtree Protocol

Building the Tree



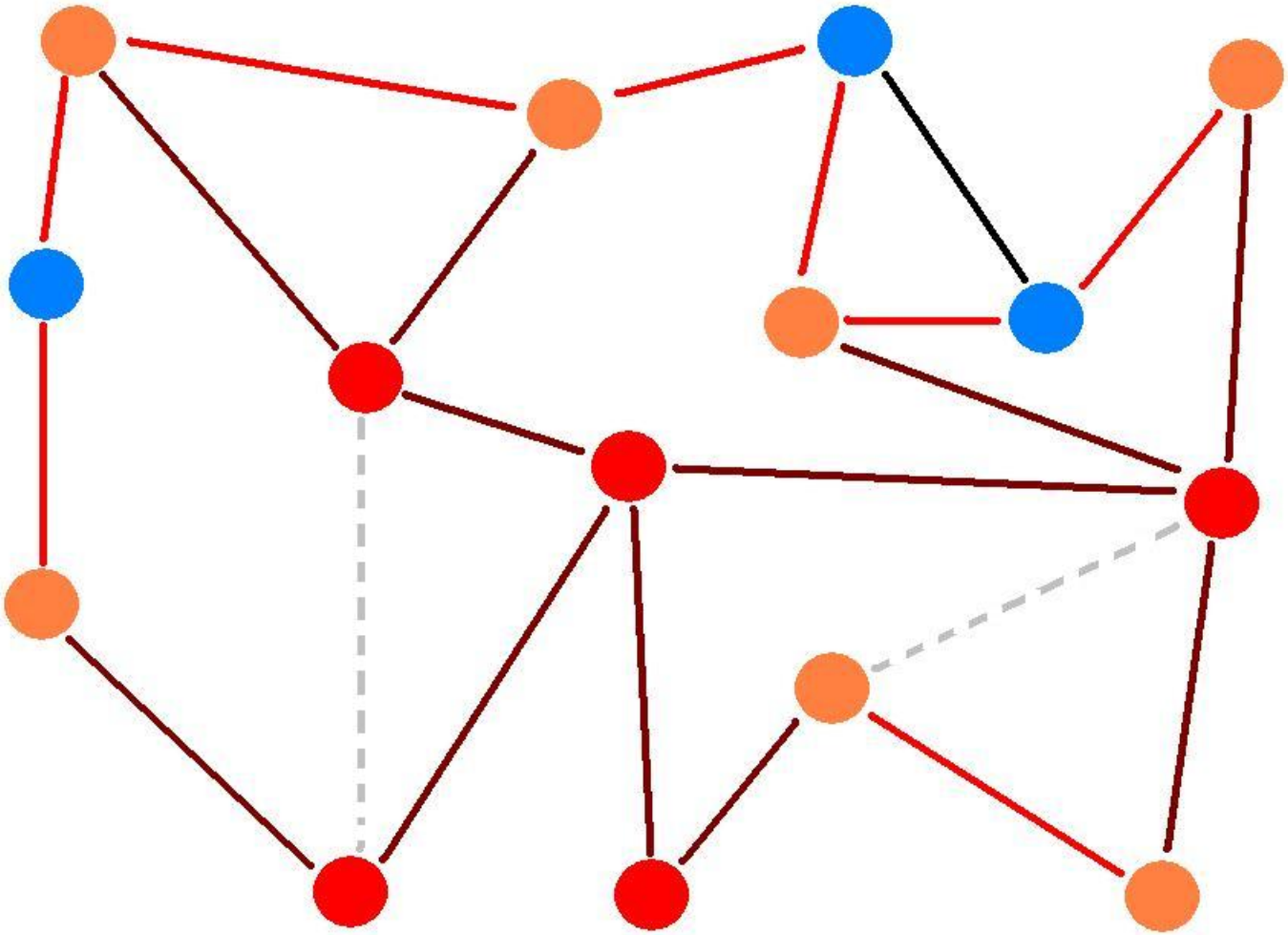
Embedded Trees: Plumtree Protocol

Building the Tree



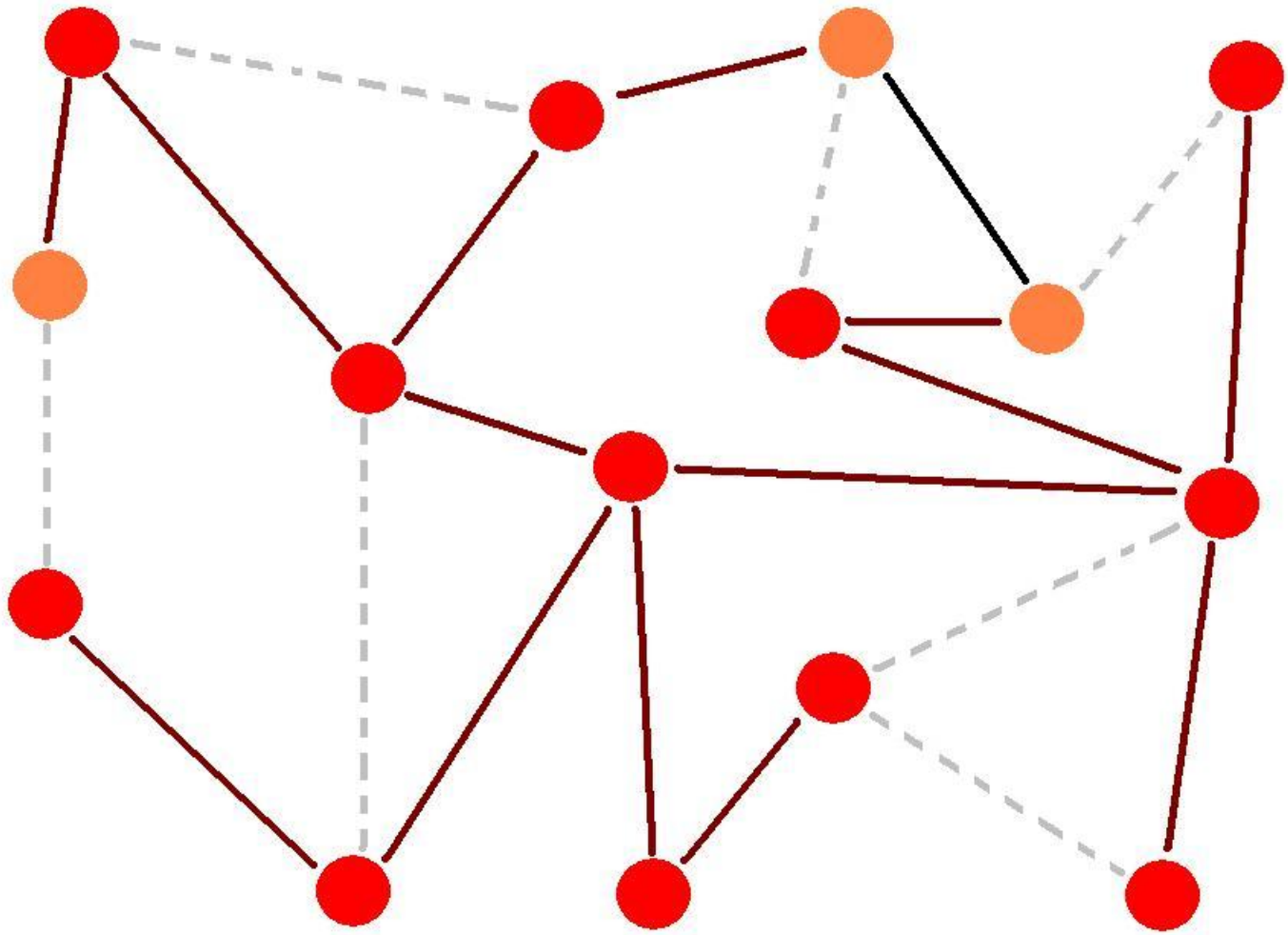
Embedded Trees: Plumtree Protocol

Building the Tree



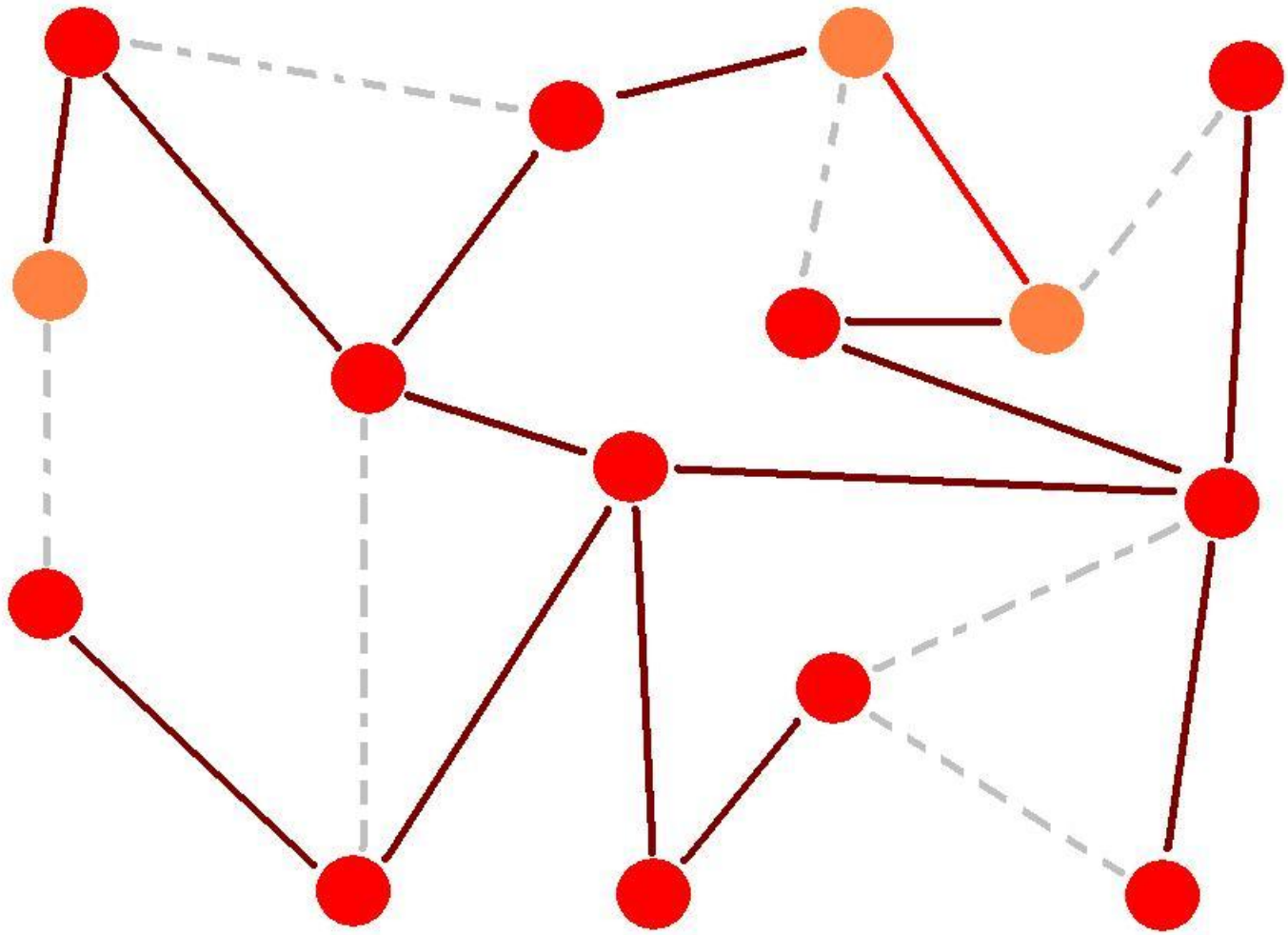
Embedded Trees: Plumtree Protocol

Building the Tree



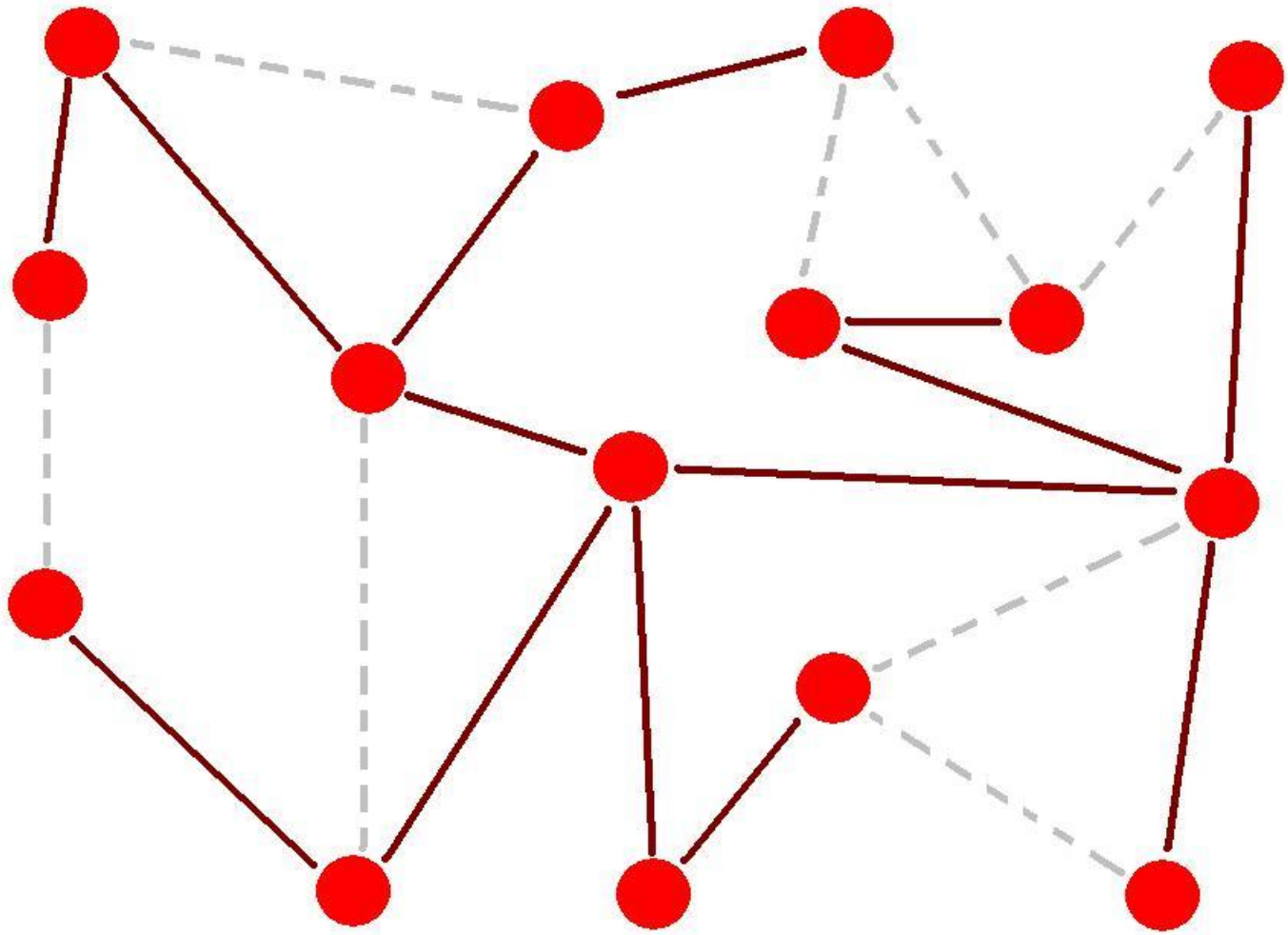
Embedded Trees: Plumtree Protocol

Building the Tree



Embedded Trees: Plumtree Protocol

Building the Tree



Embedded Trees: Plumtree Protocol

Recovering the Tree

- If a node fails some nodes might become disconnected from the embedded tree.
- When a node receives a announcement for a message it did not received it starts a timer.
- If the timer expires before receiving the message the node explicitly requests it to the node that sent the announcement.
- Reintroduces the link to that node into the embedded tree.
- The node that had sent the announcement sends the message when requested.
- It also reintroduces that link into the embedded tree.
- At the end of this process the embedded tree is repaired.

Embedded Trees: Plumtree Protocol

Recovering the Tree

- If a node fails some nodes might become disconnected from the embedded tree.
- When a node receives a announcement for a message it did not received it starts a timer.
- If the timer expires before receiving the message the node explicitly requests it to the node that sent the announcement.
- Reintroduces the link to that node into the embedded tree.
- The node that had sent the announcement sends the message when requested.
- It also reintroduces that link into the embedded tree.
- At the end of this process the embedded tree is repaired.

Embedded Trees: Plumtree Protocol

Recovering the Tree

- If a node fails some nodes might become disconnected from the embedded tree.
- When a node receives a announcement for a message it did not received it starts a timer.
- If the timer expires before receiving the message the node explicitly requests it to the node that sent the announcement.
- Reintroduces the link to that node into the embedded tree.
- The node that had sent the announcement sends the message when requested.
- It also reintroduces that link into the embedded tree.
- At the end of this process the embedded tree is repaired.

Embedded Trees: Plumtree Protocol

Recovering the Tree

- If a node fails some nodes might become disconnected from the embedded tree.
- When a node receives a announcement for a message it did not received it starts a timer.
- If the timer expires before receiving the message the node explicitly requests it to the node that sent the announcement.
- Reintroduces the link to that node into the embedded tree.
- The node that had sent the announcement sends the message when requested.
- It also reintroduces that link into the embedded tree.
- At the end of this process the embedded tree is repaired.

Embedded Trees: Plumtree Protocol

Experimental Evaluation

- Evaluation was conducted with the Peersim simulator.
- 10.000 nodes system.
- Hundreds and hundreds of Simulations.
- Compare the performance between:
  - A standard (push) Gossip Protocol named *Eager*.
  - Several configurations of Plumtree.

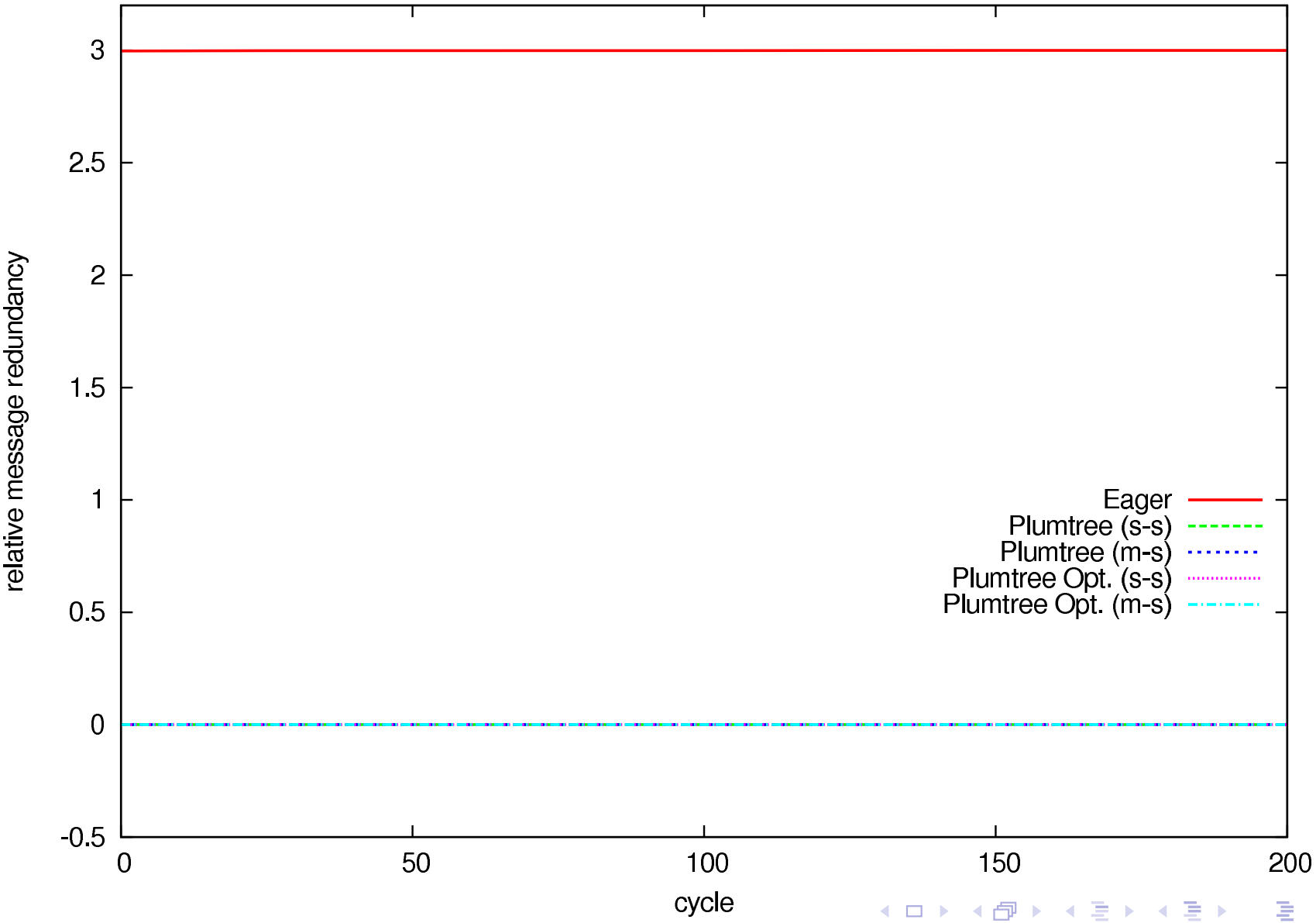
Embedded Trees: Plumtree Protocol

Experimental Evaluation

- Evaluation was conducted with the Peersim simulator.
- 10.000 nodes system.
- Hundreds and hundreds of Simulations.
- Compare the performance between:
  - A standard (push) Gossip Protocol named *Eager*.
  - Several configurations of Plumtree.

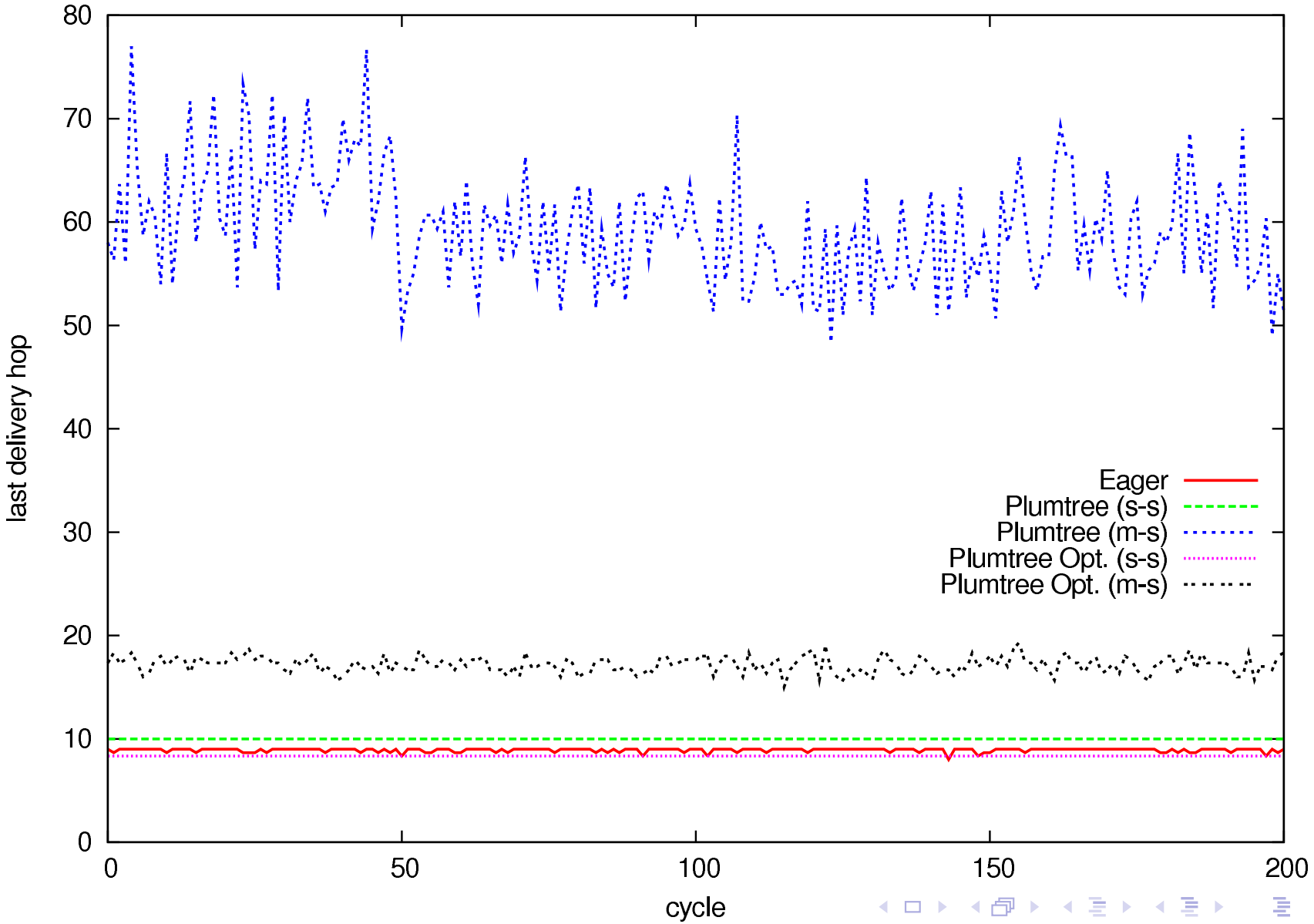
Embedded Trees: Plumtree Protocol

Experimental Evaluation



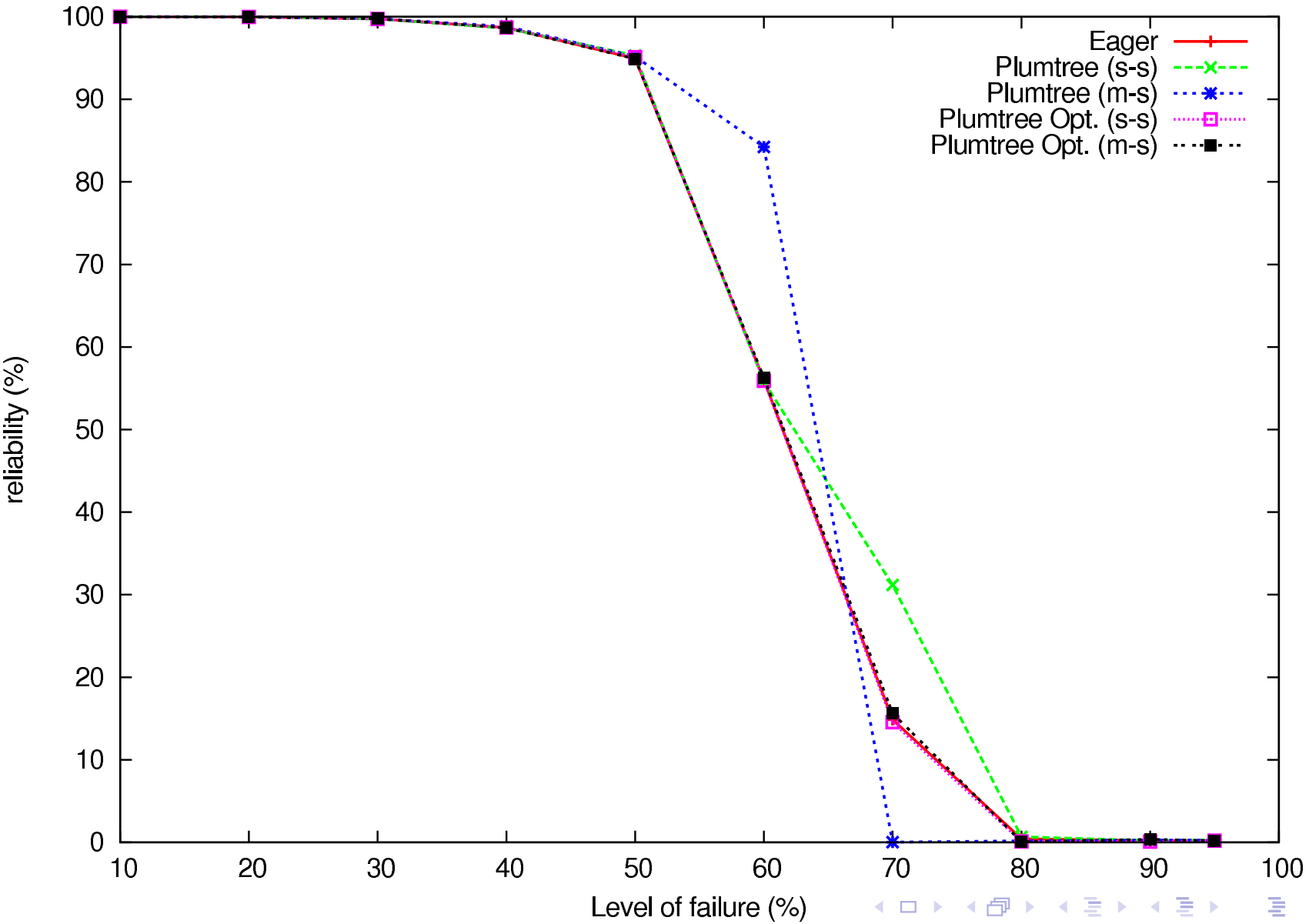
Embedded Trees: Plumtree Protocol

Experimental Evaluation



Embedded Trees: Plumtree Protocol

Experimental Evaluation



Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

The Industry View: Cluster Metadata Management

Global Information, Needed Locally

Operational Information

Mebership, Ownership, Transfers, Capabilities

Configuration

Replica Counts, Backends, Is Feature X Enabled?

Monitoring Data

Transfer Statistics, Background Work Tracking, Self-Monitoring

The Industry View: Cluster Metadata Management

Requirements

- Failure Tolerance
  - Disable malicious clients despite network partitions
  - Add new nodes despite others being down
- Quick Convergence
  - Become aware of new nodes ASAP
  - Keep monitoring data as fresh as possible
- Minimal Resource Usage
  - Goal is to speed up, not slow down, foreground work
  - Network is needed for the data!
- AP typically, CP rarely necessary

The Industry View: Cluster Metadata Management

Requirements

- Failure Tolerance
  - Disable malicious clients despite network partitions
  - Add new nodes despite others being down
- Quick Convergence
  - Become aware of new nodes ASAP
  - Keep monitoring data as fresh as possible
- Minimal Resource Usage
  - Goal is to speed up, not slow down, foreground work
  - Network is needed for the data!
- AP typically, CP rarely necessary

The Industry View: Cluster Metadata Management

Requirements

- Failure Tolerance
  - Disable malicious clients despite network partitions
  - Add new nodes despite others being down
- Quick Convergence
  - Become aware of new nodes ASAP
  - Keep monitoring data as fresh as possible
- Minimal Resource Usage
  - Goal is to speed up, not slow down, foreground work
  - Network is needed for the data!
- AP typically, CP rarely necessary

The Industry View: Cluster Metadata Management

Requirements

- Failure Tolerance
  - Disable malicious clients despite network partitions
  - Add new nodes despite others being down
- Quick Convergence
  - Become aware of new nodes ASAP
  - Keep monitoring data as fresh as possible
- Minimal Resource Usage
  - Goal is to speed up, not slow down, foreground work
  - Network is needed for the data!
- AP typically, CP rarely necessary

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

Cluster Metadata Management In Riak

Implementations

0.x

- Peer Service: Classic Anti-Entropy Protocol
- Data Dissemination: Same

1.x

- Peer Service: Static Graph Overlay w/ Anti-Entropy
- Data Dissemination: Same

2.x

- Peer Service: Static Graph Overlay w/ Anti-Entropy
- Data Dissemination: Sender-Based Embedded Trees

Cluster Metadata Management In Riak

Motivations

0.x to 1.0

- Anti-Entropy is Reliable but Slow to Converge
- Improve Convergence Time, Specifically in Failure Scenarios

1.x to 2.0

- Static Graph Overlay/Anti-Entropy have several issues:
  - Too Much Network Overhead
  - Improved Convergence Only Needed for Subset of Data
  - Cold Rumor Mongering
- Improve Resource Usage and Stability

Connectivity

"all nodes should have in their partial views...another correct node" is not an invariant we can maintain

System Model

Must consider dropped, delayed, duplicated* messages in addition to node failures

Topology

Peer Service is Fully Connected (as is Erlang)

Membership

Operator-controlled instead of reactive (failure detector is implemented separately from peer service)

What happens if Lazy Messages are Dropped?

Add to Outstanding Set. Remove on Graft or Ack

Handling Extreme Failures

Back protocol by random anti-entropy with peers that are not in eager or lazy sets

Shared vs. Sender-Based Trees

Sender-Based. Minimize overhead w/ Lazy Construction

Cluster Metadata Management In Riak

Measurements

- Reduced convergence time (measured in LDH) by 66%
- Reduced needless message overhead in stable state 99.9%
- % Failures before needing to rely on anti-entropy good for small clusters (near 90%) but decreased quickly for larger clusters (under 50%)

Cluster Metadata Management In Riak

Peer Service Tuning

- Want to increase percent failures we can tolerate before relying on anti-entropy
- Fanout of tree given to us by peer service matters
- For clusters > 10 nodes, fanout $= \ln(\text{cluster size}) + 1$
- With our use of multiple trees, more peer service tuning may be necessary

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

The Academic View: When One Tree is not Enough

Understanding The Problem

We had Plumtree:

- Efficient.
- Very Robust.
- Could even rebalance itself to improve latency (height of the tree).

The reality about trees...

Trees do not distribute the load evenly among participants.

The Academic View: When One Tree is not Enough

Understanding The Problem

We had Plumtree:

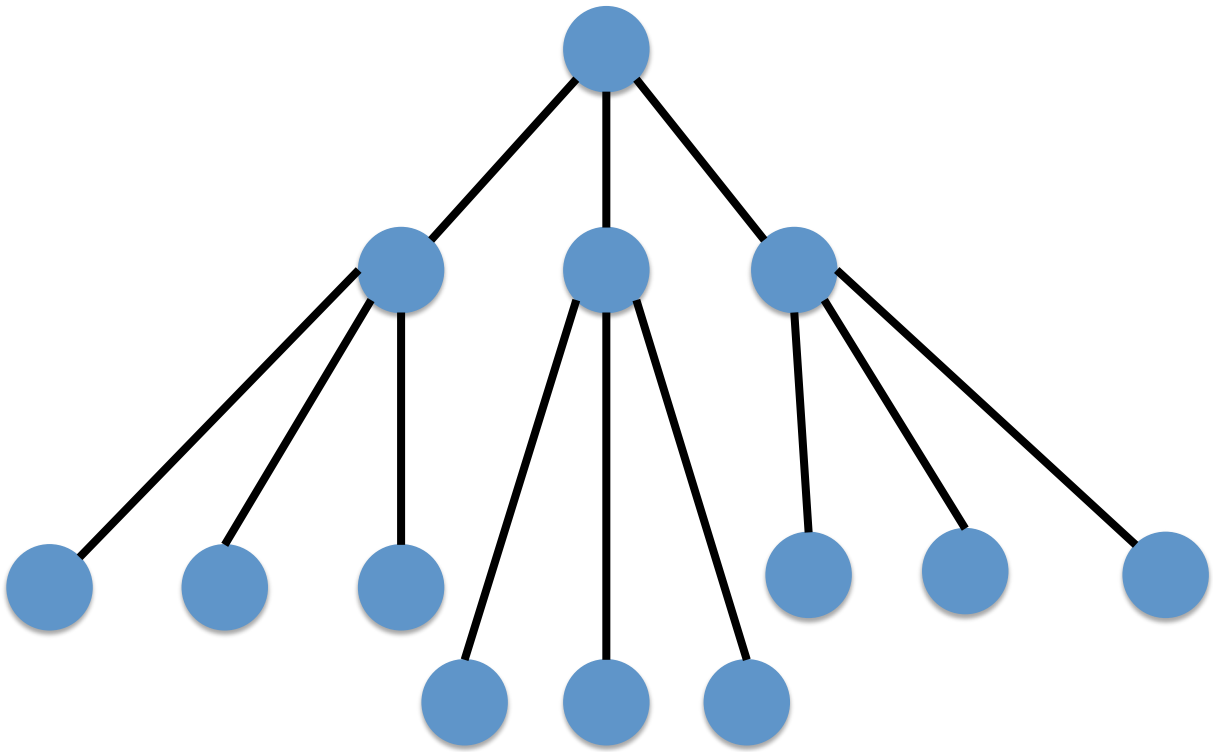
- Efficient.
- Very Robust.
- Could even rebalance itself to improve latency (height of the tree).

The reality about trees...

Trees do not distribute the load evenly among participants.

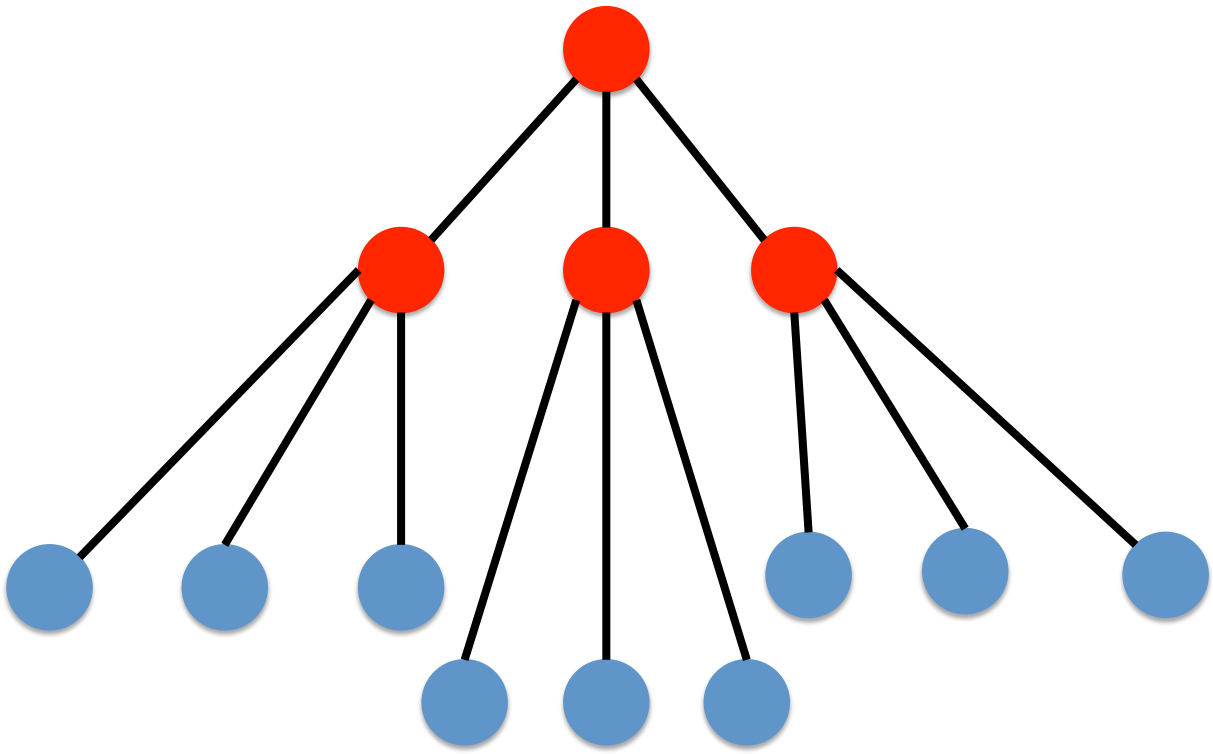
The Academic View: When One Tree is not Enough

Understanding The Problem



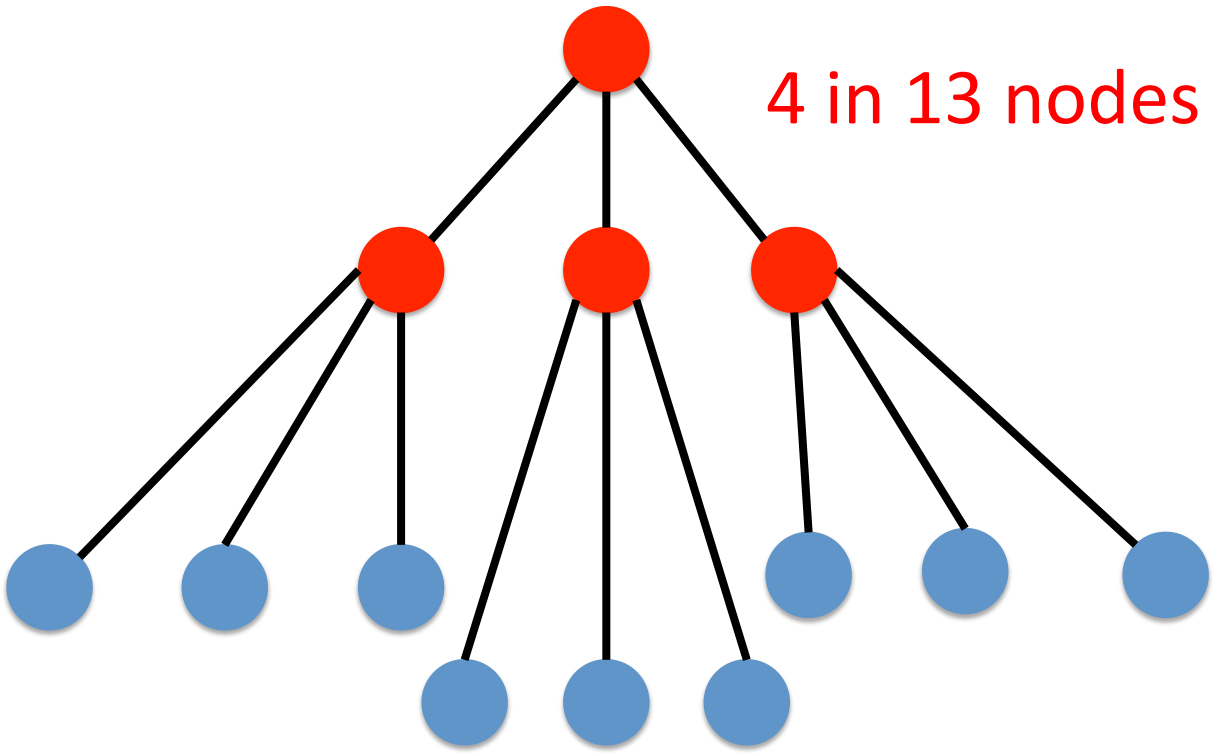
The Academic View: When One Tree is not Enough

Understanding The Problem



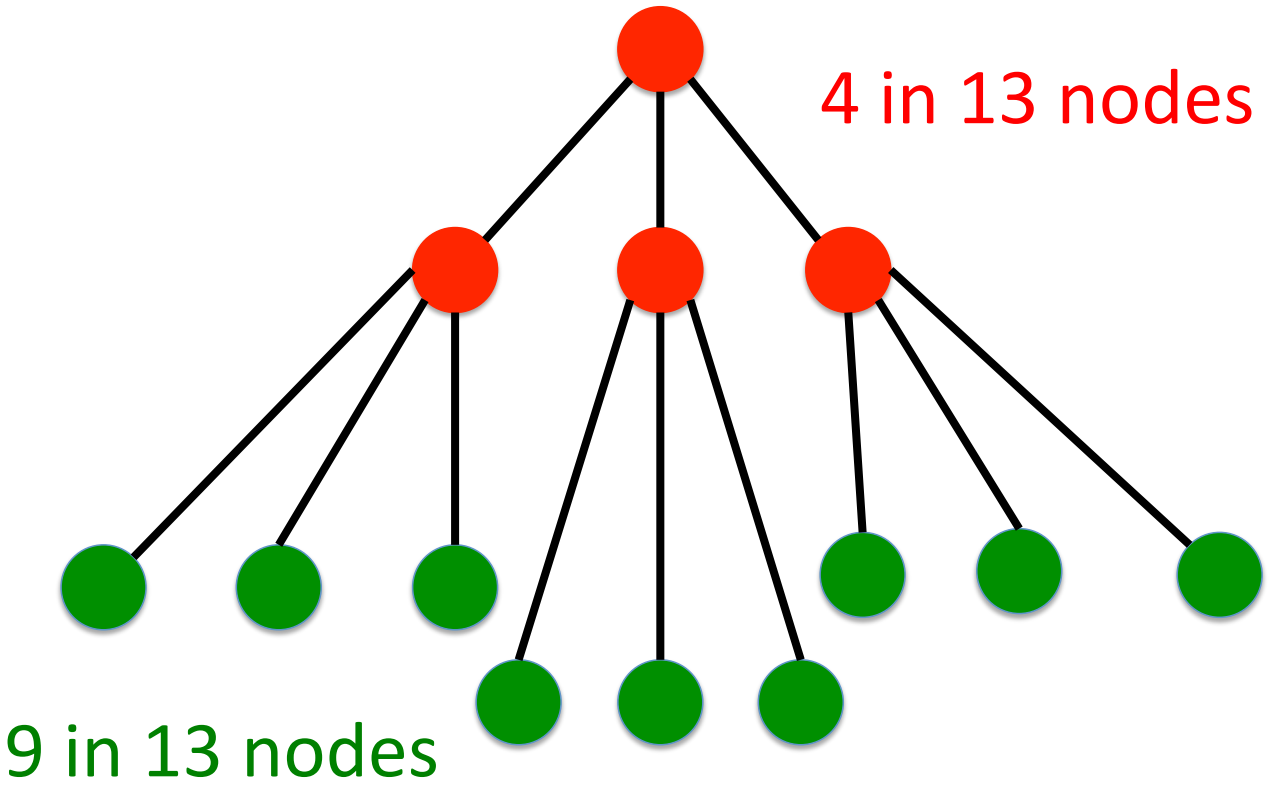
The Academic View: When One Tree is not Enough

Understanding The Problem



The Academic View: When One Tree is not Enough

Understanding The Problem



The Academic View: When One Tree is not Enough

Naive Solutions

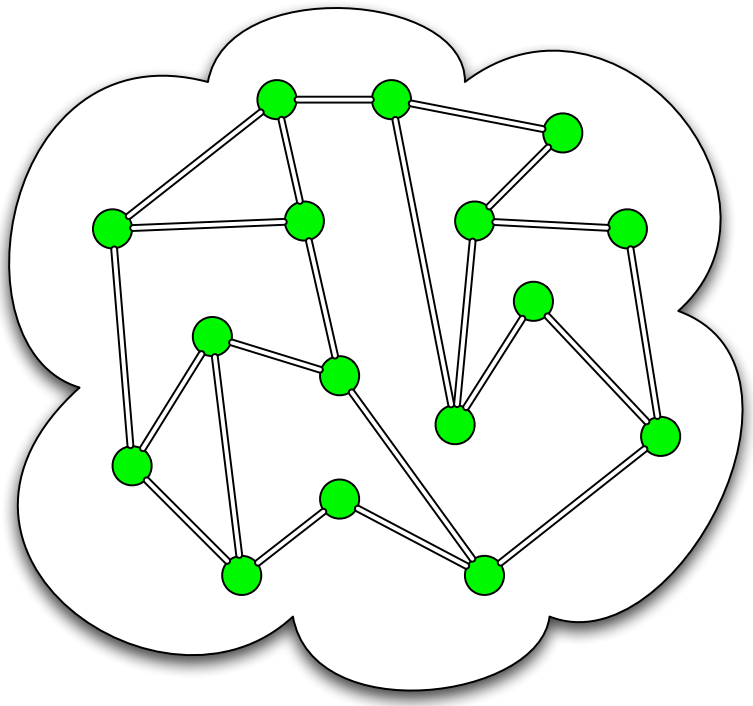
How hard can this be?

- We started by experimenting with two simple approaches:
- NUTS: Naive Unstructured spliTStream
- BOLTS: Basic multiple Overlay-TreeS

The Academic View: When One Tree is not Enough

Naive Solutions: NUTS

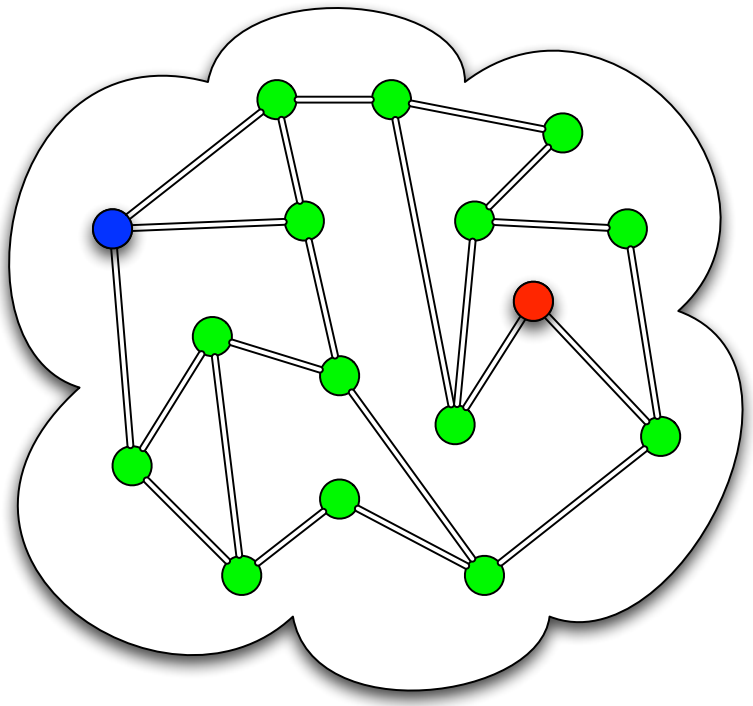
- Pick t nodes at random and create t Plumtrees; each rooted at a different node.



The Academic View: When One Tree is not Enough

Naive Solutions: NUTS

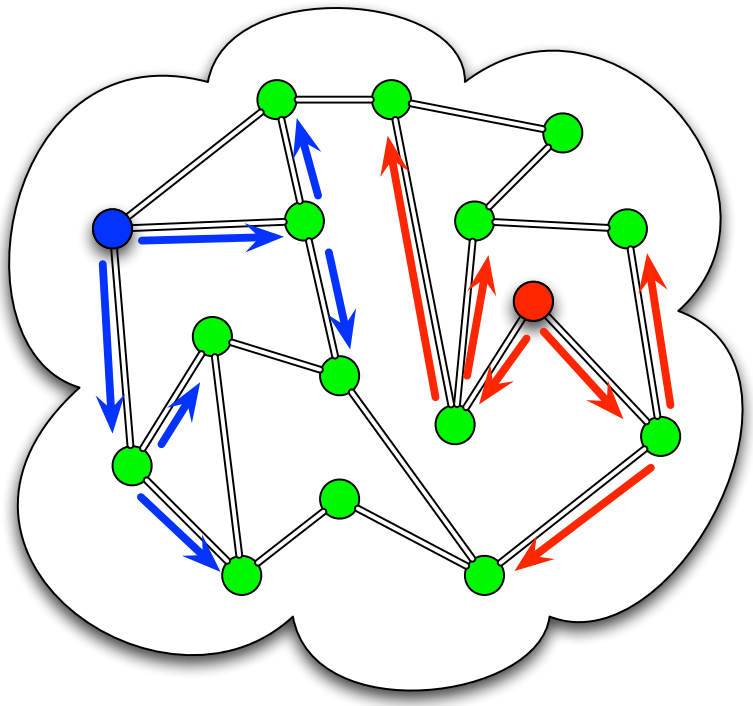
- Pick t nodes at random and create t Plumtrees; each rooted at a different node.



The Academic View: When One Tree is not Enough

Naive Solutions: NUTS

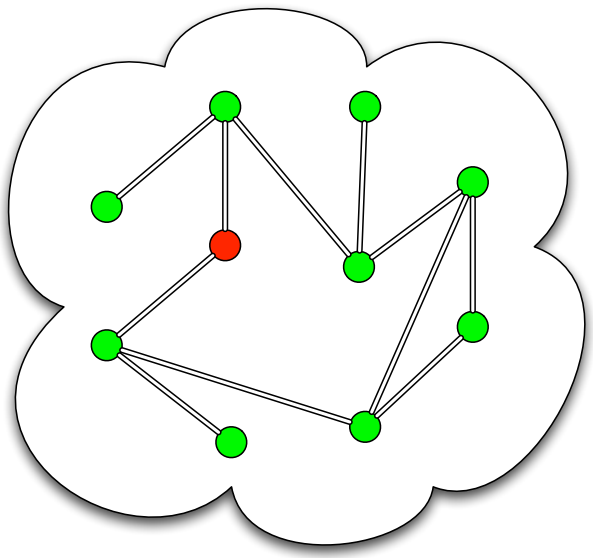
- Pick t nodes at random and create t Plumtrees; each rooted at a different node.

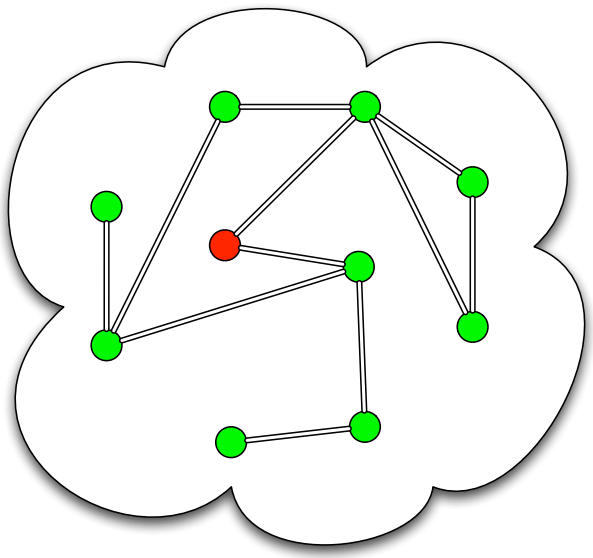


The Academic View: When One Tree is not Enough

Naive Solutions: BOLTS

- Create t different unstructured overlays and create a Plumtree using each overlay.

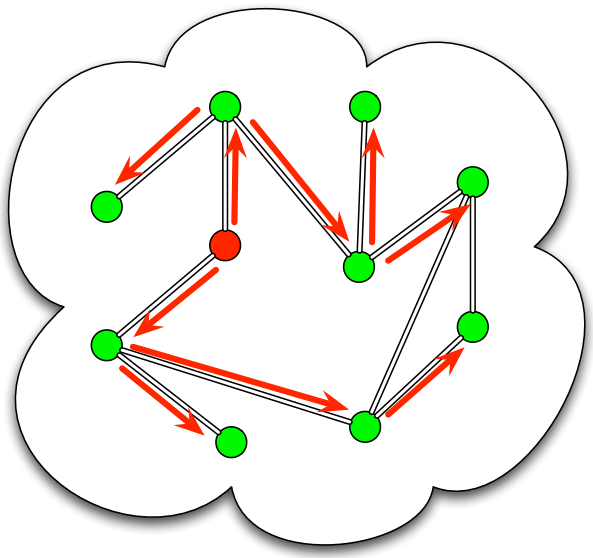


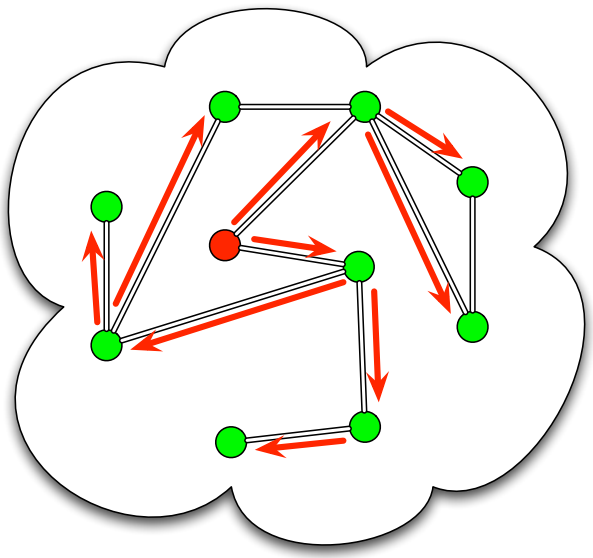


The Academic View: When One Tree is not Enough

Naive Solutions: BOLTS

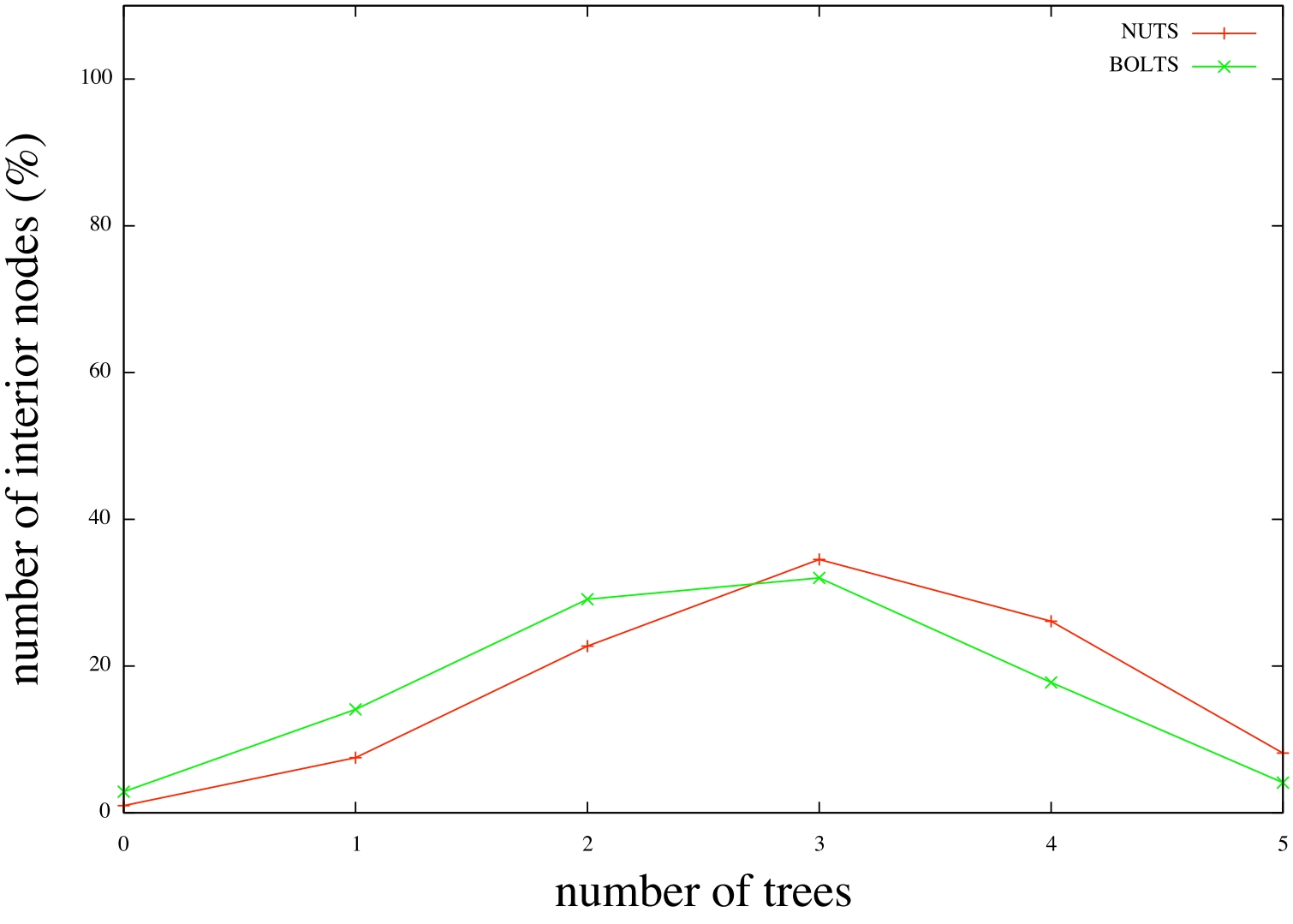
- Create t different unstructured overlays and create a Plumtree using each overlay.





The Academic View: When One Tree is not Enough

Naive Solutions: NUTS & BOLTS



The Academic View: When One Tree is not Enough

Understanding The Goal

From one tree to multiple trees

- Load balancing:
  - every node should be interior on the same number of trees
- Use all resources:
  - each node should be interior in at least one tree
- Fault-tolerance:
  - each node should be interior in at most one tree

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

The Thicket Protocol

Thicket: A Protocol for Building and Maintaining Multiple Trees in a P2P Overlay.

Mário Ferreira, João Leitão, and Luís Rodrigues

Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems (SRDS), New Delhi, India, 31 October-3 November 2010.

The Thicket Protocol
Deploying the Trees

- We use the same principle as in Plumtree to define trees, however:
  - Overlay links are explicitly added to each tree (instead of removed).
  - Trees are deployed by taking into consideration the remaining trees.
  - Finding the minimal coordination necessary to achieve our goal is tricky.
  - Tree coverage is impossible at tree deployment time.

The Thicket Protocol
Deploying the Trees

- We use the same principle as in Plumtree to define trees, however:
  - Overlay links are explicitly added to each tree (instead of removed).
  - Trees are deployed by taking into consideration the remaining trees.
  - Finding the minimal coordination necessary to achieve our goal is tricky.
  - Tree coverage is impossible at tree deployment time.

The Thicket Protocol
Deploying the Trees

- We use the same principle as in Plumtree to define trees, however:
  - Overlay links are explicitly added to each tree (instead of removed).
  - Trees are deployed by taking into consideration the remaining trees.
  - Finding the minimal coordination necessary to achieve our goal is tricky.
  - Tree coverage is impossible at tree deployment time.

The Thicket Protocol
Recovering Trees

- Global coverage of trees is provided by the repair mechanism.
- All messages exchanged among nodes convey information about the load of the sender.
  - How many messages it sends per time unit.
  - How many child nodes in each tree.
- Tree repair uses less loaded nodes.

The Thicket Protocol

Recovering Trees

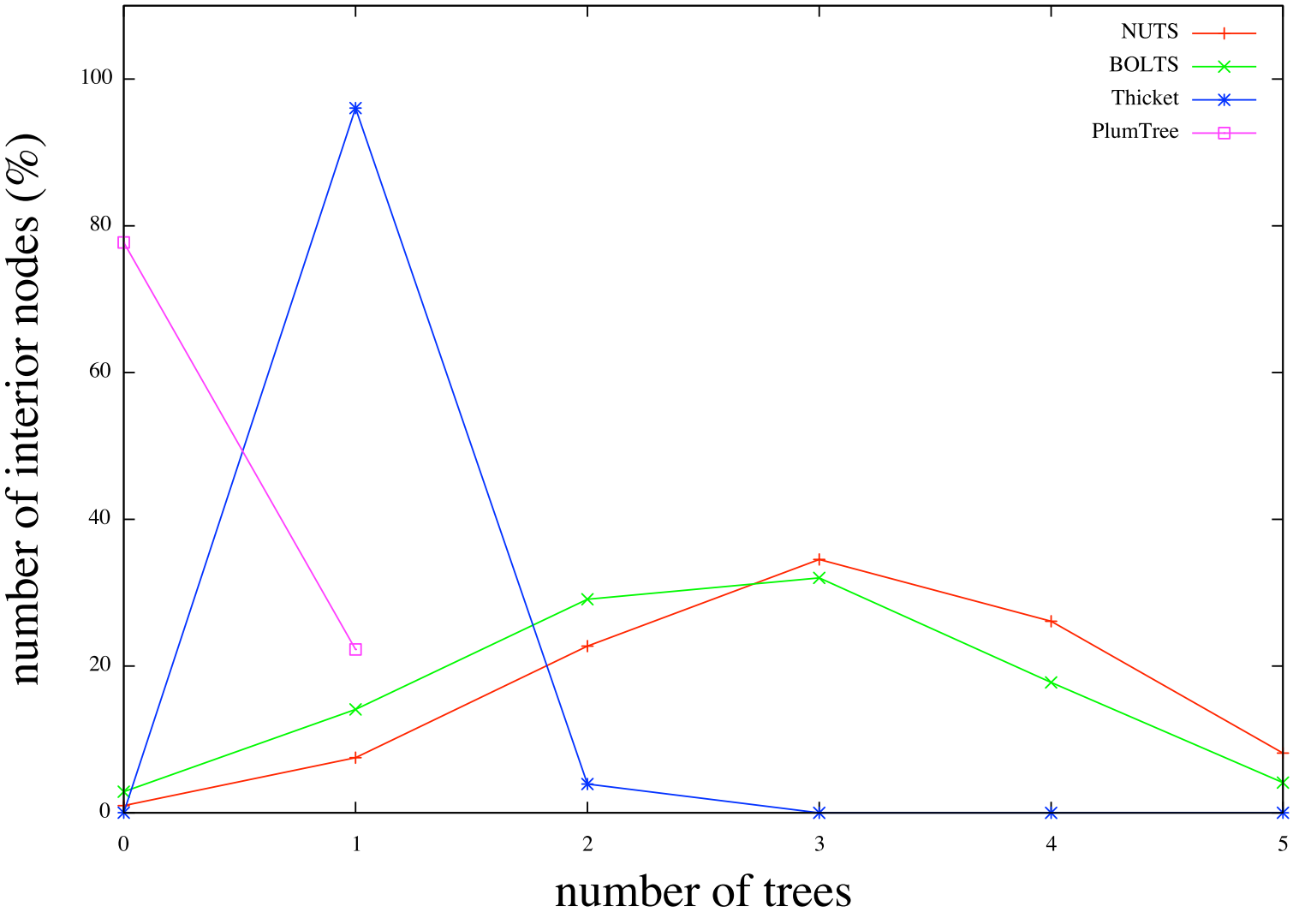
- Global coverage of trees is provided by the repair mechanism.
- All messages exchanged among nodes convey information about the load of the sender.
  - How many messages it sends per time unit.
  - How many child nodes in each tree.
- Tree repair uses less loaded nodes.

The Thicket Protocol
Experimental Results

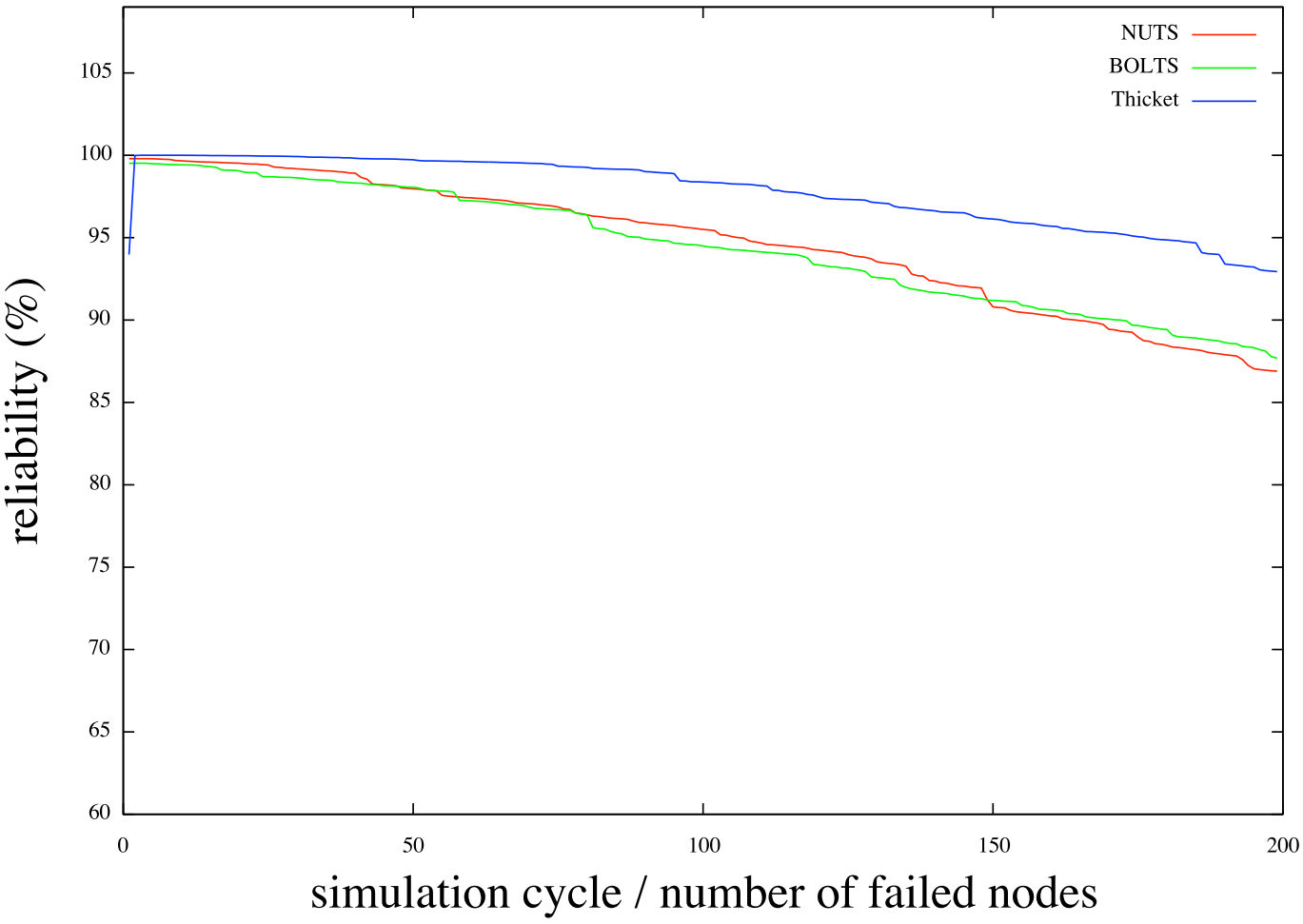
- Simulated a 10.000 overlay using PeerSim.
- Comparison with NUTS and BOLTS (and Plumtree).
- Same underlying unstructured overlay network.
- Assumes reliable FIFO channels.

The Thicket Protocol

Experimental Results



The Thicket Protocol
Experimental Results



Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

The Industry View: Improving Cluster Metadata

Measuring Interior Nodes

10 Nodes

- 4 of 10 nodes are interior in 3 of 10 trees
- 3 of 10 nodes are interior in 4 of 10 trees
- 3 of 10 nodes are interior in 5 of 10 trees

20 Nodes

- 4 nodes are interior only when roots
- 16 nodes are interior in 5 or more trees

80 Nodes

- 4 nodes are interior only when roots
- 36 of 80 nodes are interior in $> 25\%$ of trees

The Industry View: Improving Cluster Metadata

Measuring Interior Nodes

10 Nodes

- 4 of 10 nodes are interior in 3 of 10 trees
- 3 of 10 nodes are interior in 4 of 10 trees
- 3 of 10 nodes are interior in 5 of 10 trees

20 Nodes

- 4 nodes are interior only when roots
- 16 nodes are interior in 5 or more trees

80 Nodes

- 4 nodes are interior only when roots
- 36 of 80 nodes are interior in $> 25\%$ of trees

The Industry View: Improving Cluster Metadata

Improving Interior Node Counts

- Coupling between fanout, number of trees, and how many times a node must be interior
- Height of initial trees constructed by peer service

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

Exploring Thicket

- Research Area

Migration & New Data

- Move data from 1.x subsystem to 2.x subsystem
- Store data we couldn't before

WAN Replication

- Integrate with Riak's Multi-Site Replication
- Abstract Messaging Layer

The Future of Cluster Metadata in Riak

Expand Usage

Exploring Thicket

- Research Area

Migration & New Data

- Move data from 1.x subsystem to 2.x subsystem
- Store data we couldn't before

WAN Replication

- Integrate with Riak's Multi-Site Replication
- Abstract Messaging Layer

The Future of Cluster Metadata in Riak

Expand Usage

Exploring Thicket

- Research Area

Migration & New Data

- Move data from 1.x subsystem to 2.x subsystem
- Store data we couldn't before

WAN Replication

- Integrate with Riak's Multi-Site Replication
- Abstract Messaging Layer

Outline

1 Motivation

2 The Academic View: Data Dissemination

3 Embedded Tree: Plumtree Protocol

4 The Industry View: Cluster Metadata Management

5 Cluster Metadata Management In Riak

6 The Academic View: When One Tree is not Enough

7 The Thicket Protocol

8 The Industry View: Improving Cluster Metadata

9 The Future of Cluster Metadata in Riak

10 Summary

Summary

- Joining academia and industry is powerful.
- Riak's implementation has largely followed academic research, inadvertently...
- ...even when the research was on sort of a different field...
- ...and we have now begun to work together with academia.

Summary

- Joining academia and industry is powerful.
- Riak's implementation has largely followed academic research, inadvertently...
- ...even when the research was on sort of a different field...
- ...and we have now begun to work together with academia.

Summary

- Joining academia and industry is powerful.
- Riak's implementation has largely followed academic research, inadvertently...
- ...even when the research was on sort of a different field...
- ...and we have now begun to work together with academia.

Thanks.

- Jordan West – @_jrwest
- João Leitão – @jcaleitao