Bimodal Multicast

And Cache Invalidation

Who/What/Where

- Bruce Spang
- Software Engineer/Student
- Fastly

Bimodal Multicast

KENNETH P. BIRMAN
Cornell University
MARK HAYDEN
Digital Equipment Corporation/Compaq
OZNUR OZKASAP and ZHEN XIAO
Cornell University
MIHAI BUDIU
Carnegie Mellon University
and
YARON MINSKY
Cornell University

There are many methods for making a multicast protocol "reliable." At one end of the spectrum, a reliable multicast protocol might offer atomicity guarantees, such as all-or-nothing delivery, delivery ordering, and perhaps additional properties such as virtually synchronous addressing. At the other are protocols that use local repair to overcome transient packet loss in the network, offering "best effort" reliability. Yet none of this prior work has treated stability of multicast delivery as a basic reliability property, such as might be needed in an internet radio, television, or conferencing application. This article looks at reliability with a new goal: development of a multicast protocol which is reliable in a sense that can be rigorously quantified and includes throughput stability guarantees. We characterize this new protocol as a "bimodal multicast" in reference to its reliability model, which corresponds to a family of bimodal probability distributions. Here, we introduce the protocol, provide a theoretical analysis of its behavior, review experimental results, and discuss some candidate applications. These confirm that bimodal multicast is reliable, scalable, and that the protocol provides remarkably stable delivery throughput.

Powderhorn

Outline

- Frame the problem
- The papers we looked at
- Bimodal Multicast
- What we built

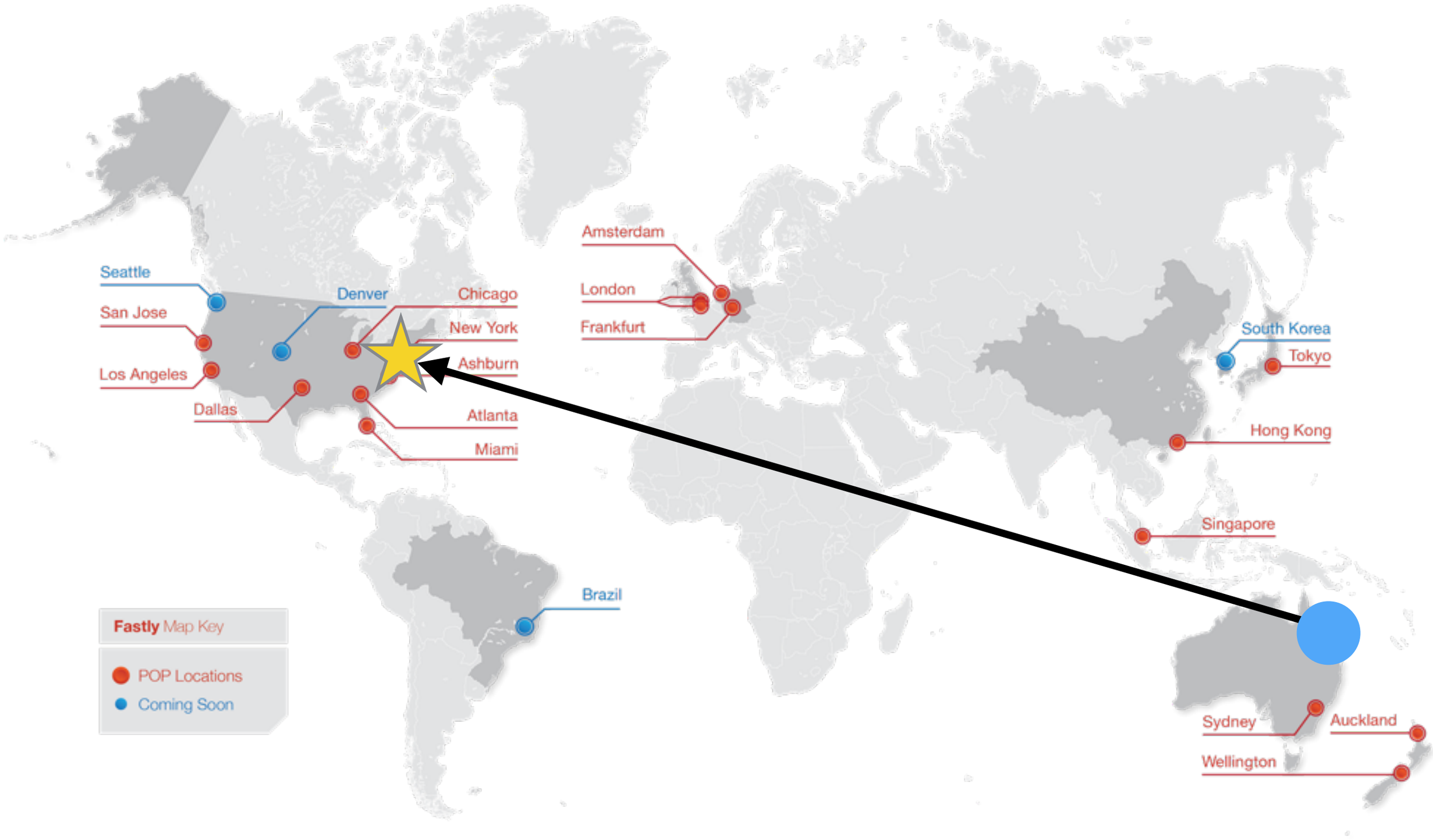
Content Delivery Networks

FASTLY GLOBAL CONTENT DELIVERY NETWORK

FASTLY GLOBAL CONTENT DELIVERY NETWORK

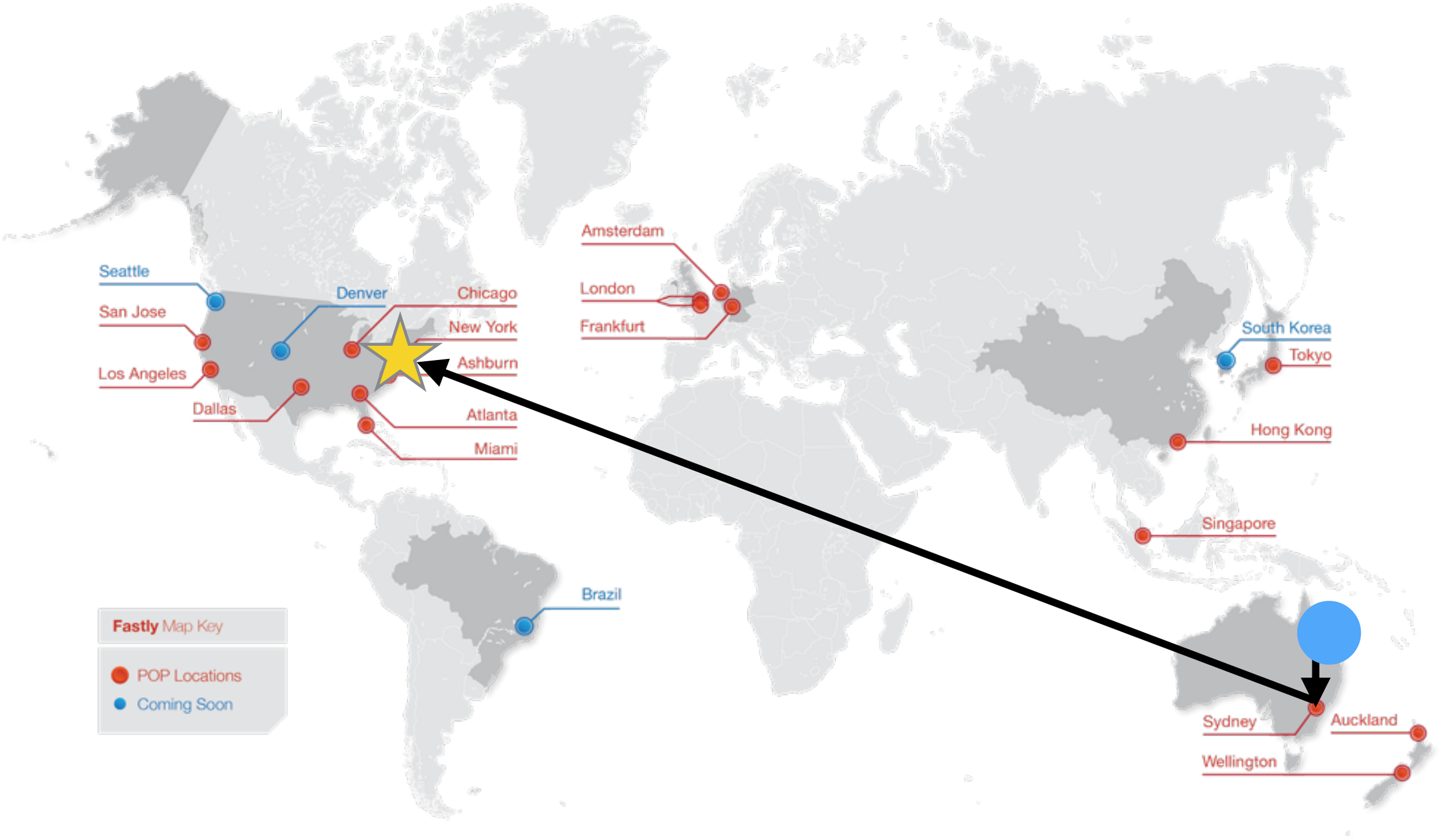
FASTLY GLOBAL CONTENT DELIVERY NETWORK

FASTLY GLOBAL CONTENT DELIVERY NETWORK



FASTLY GLOBAL CONTENT DELIVERY NETWORK

FASTLY GLOBAL CONTENT DELIVERY NETWORK



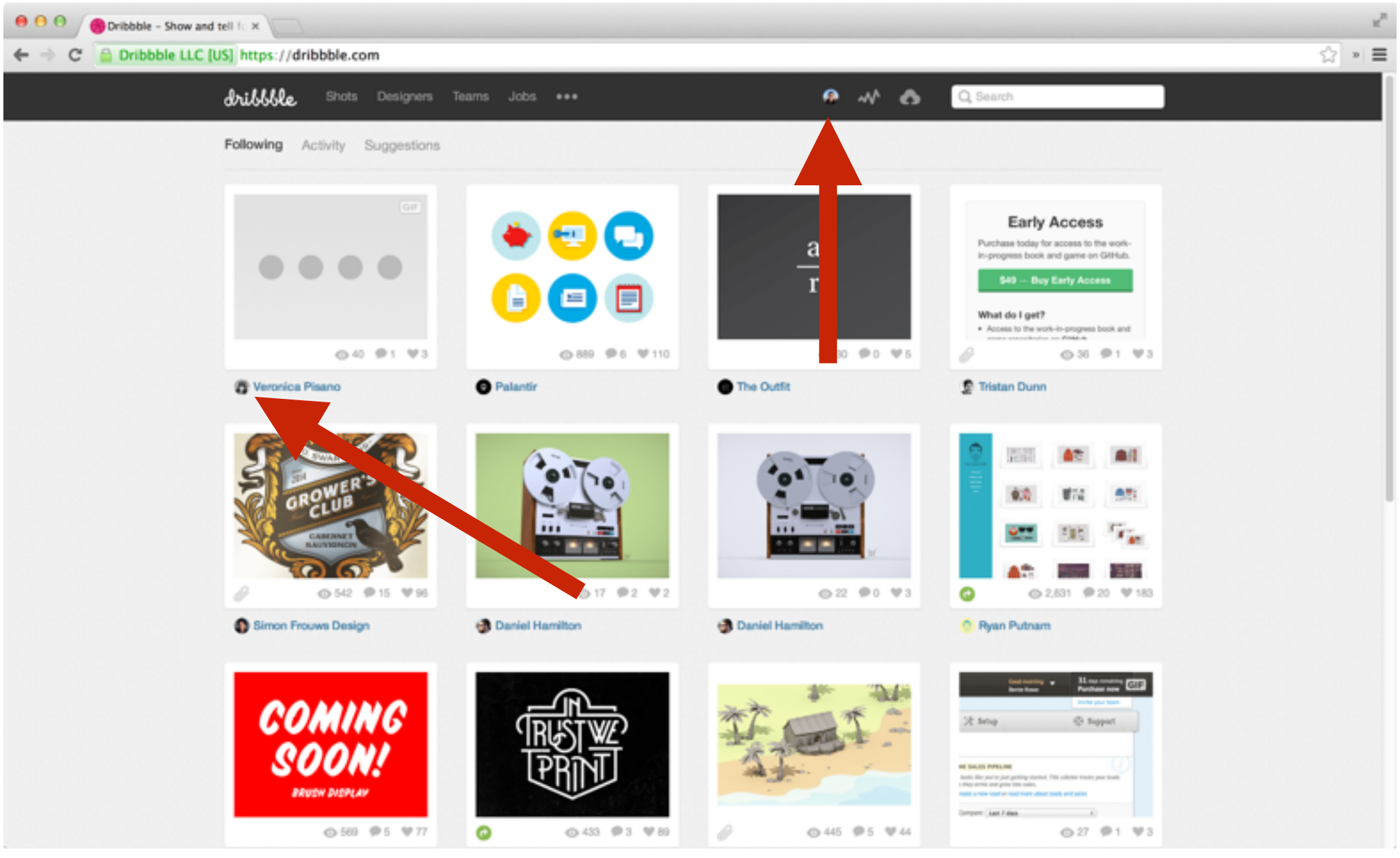
FASTLY GLOBAL CONTENT DELIVERY NETWORK

Cache Invalidation

We would like to be able to update a piece of content globally

Cache Invalidation

Cache Invalidation



Cache Invalidation

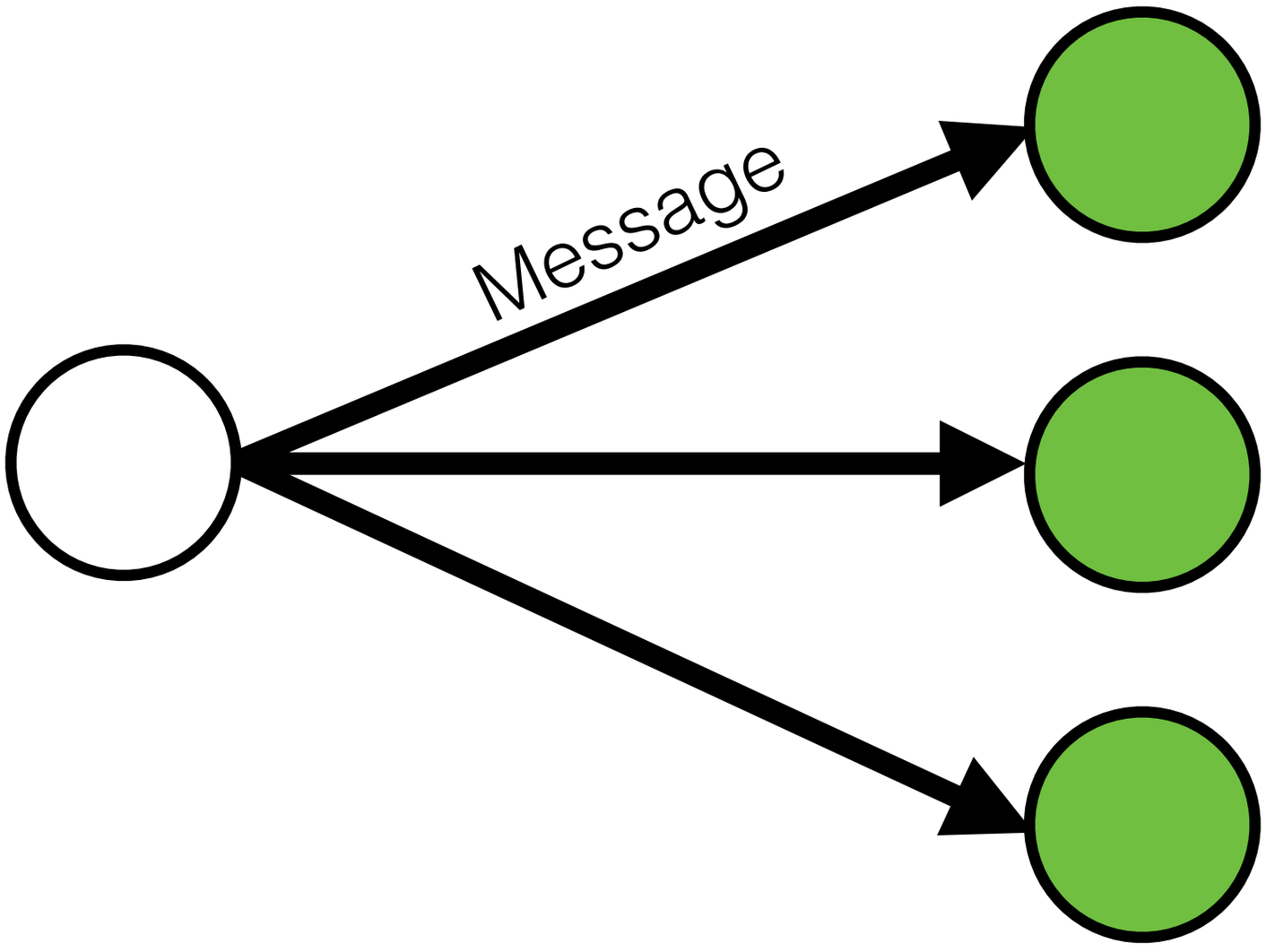
Approach

Notify all servers to remove a piece of content

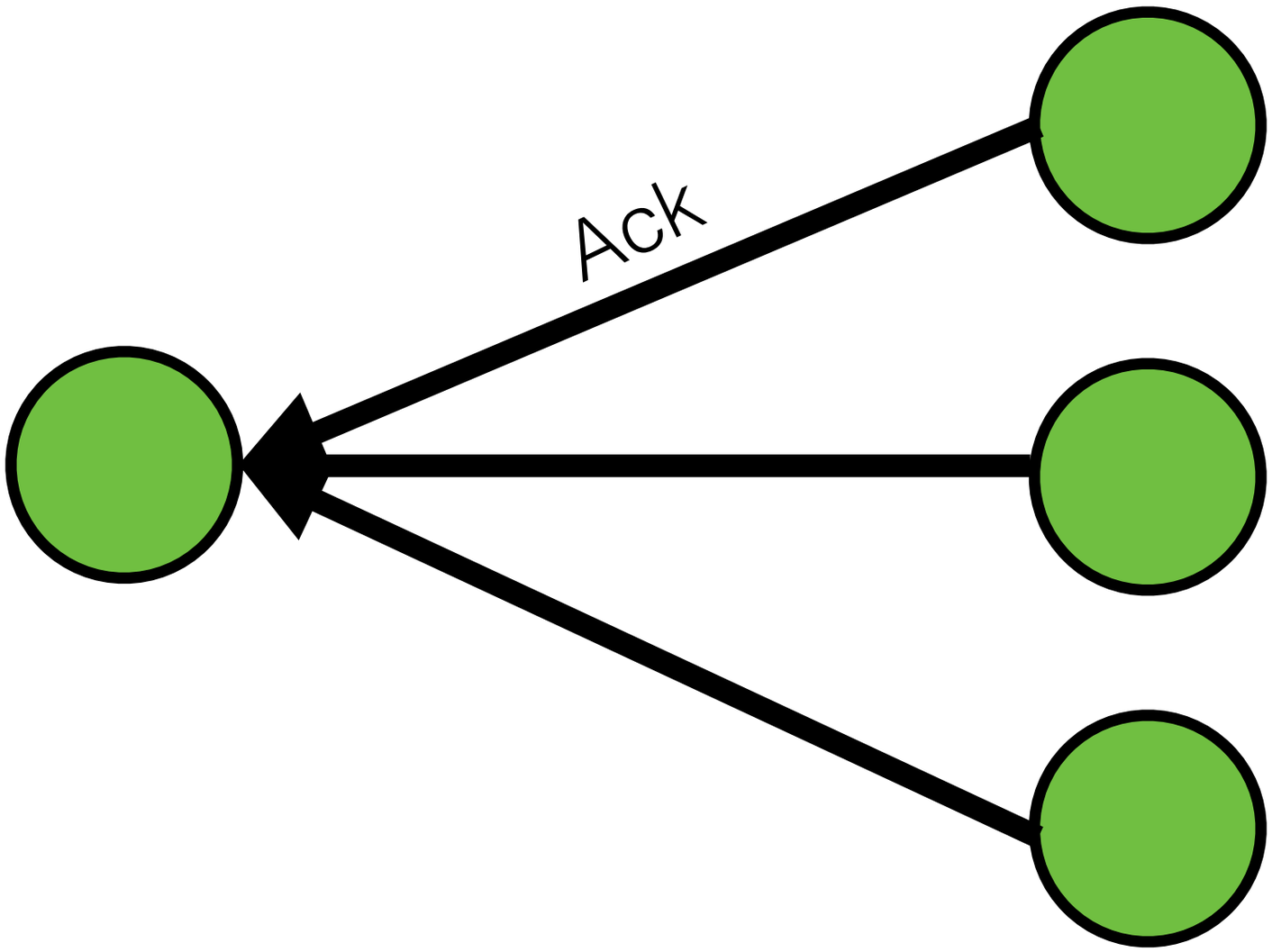
Naïve Approach

Central Service.

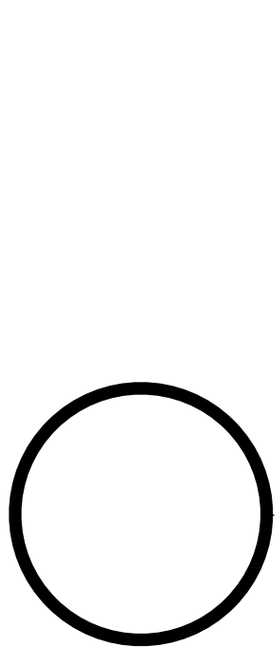
Central Service

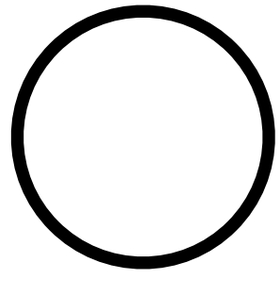


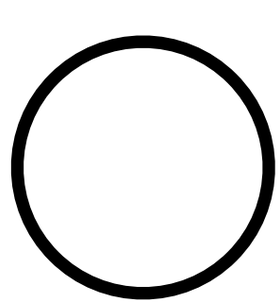
Central Service

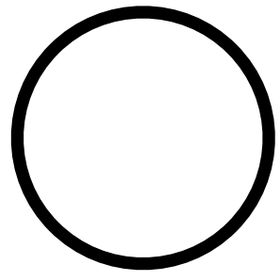


Central Service

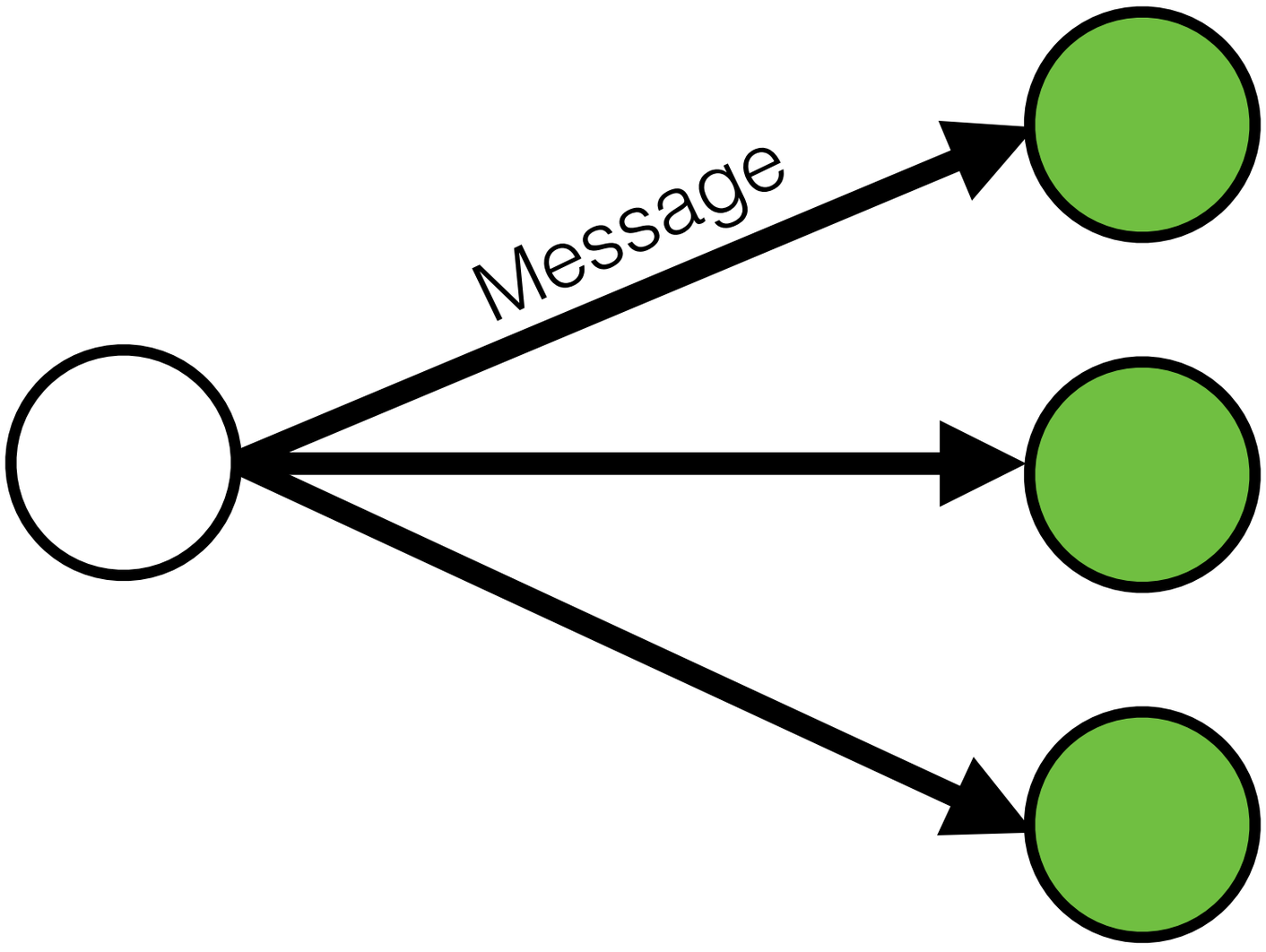




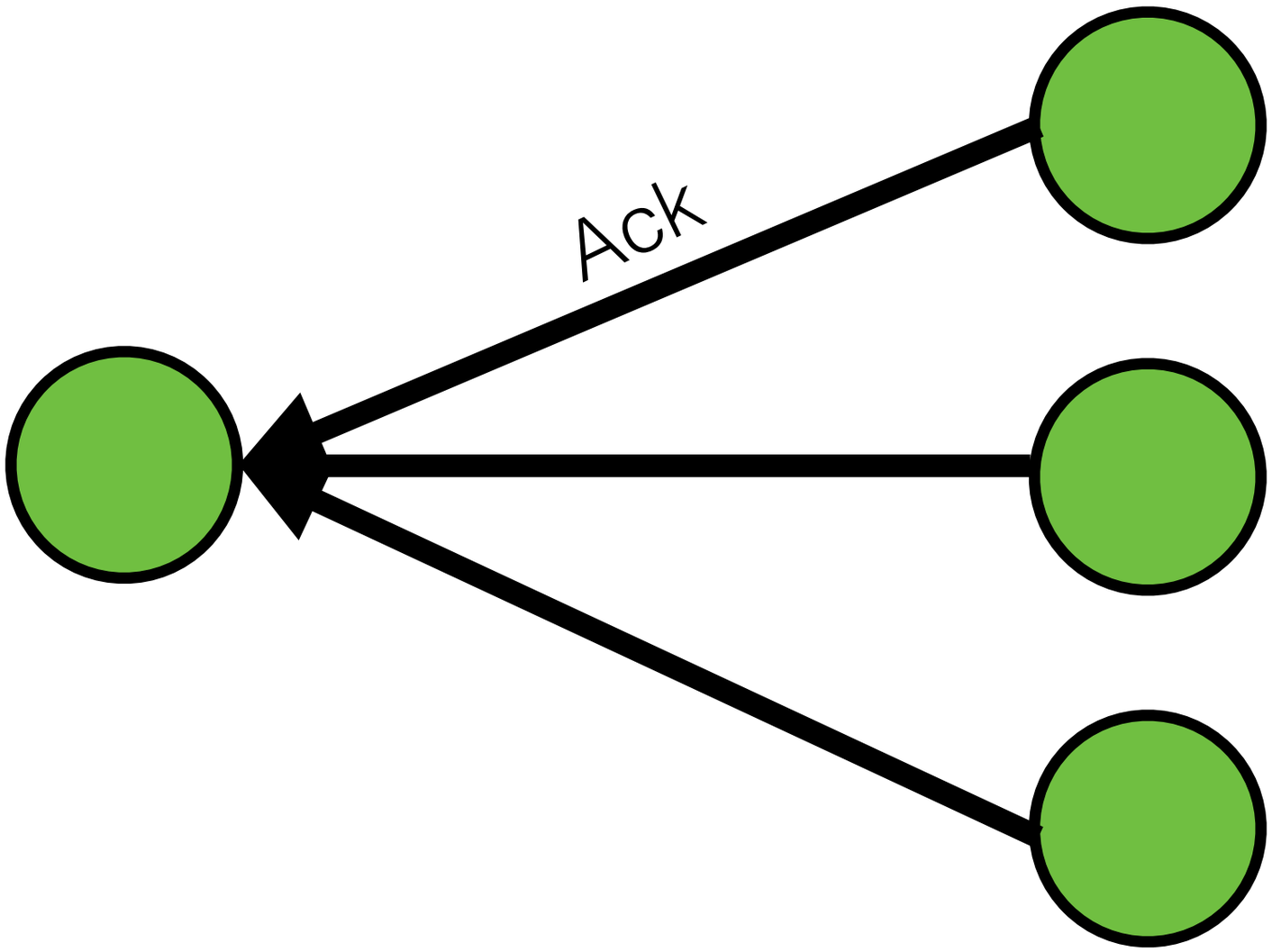




Central Service



Central Service



Central Service

Works

Problems

- Unreliable
- High Latency

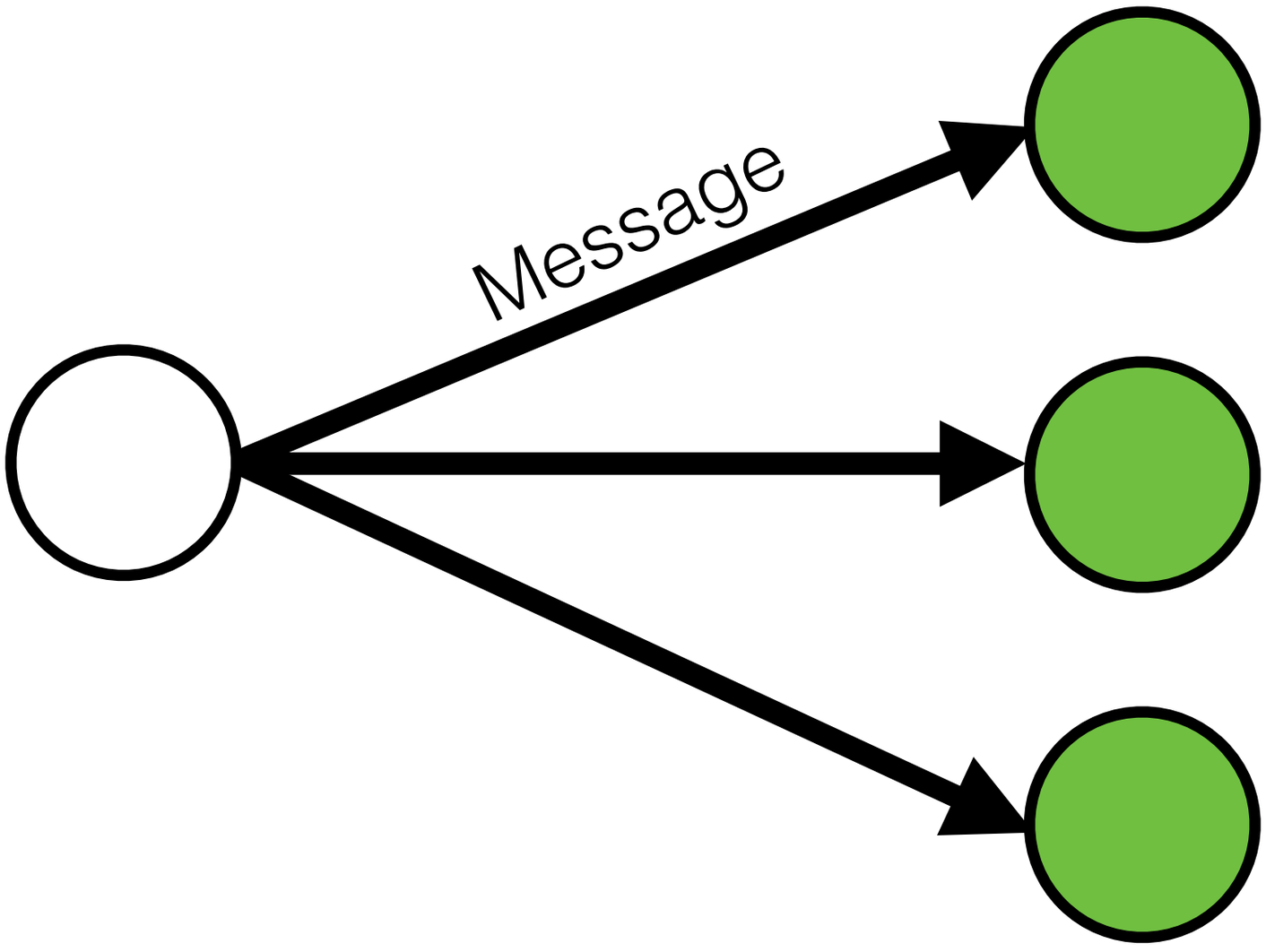
Another Idea

Cache servers can send purges themselves

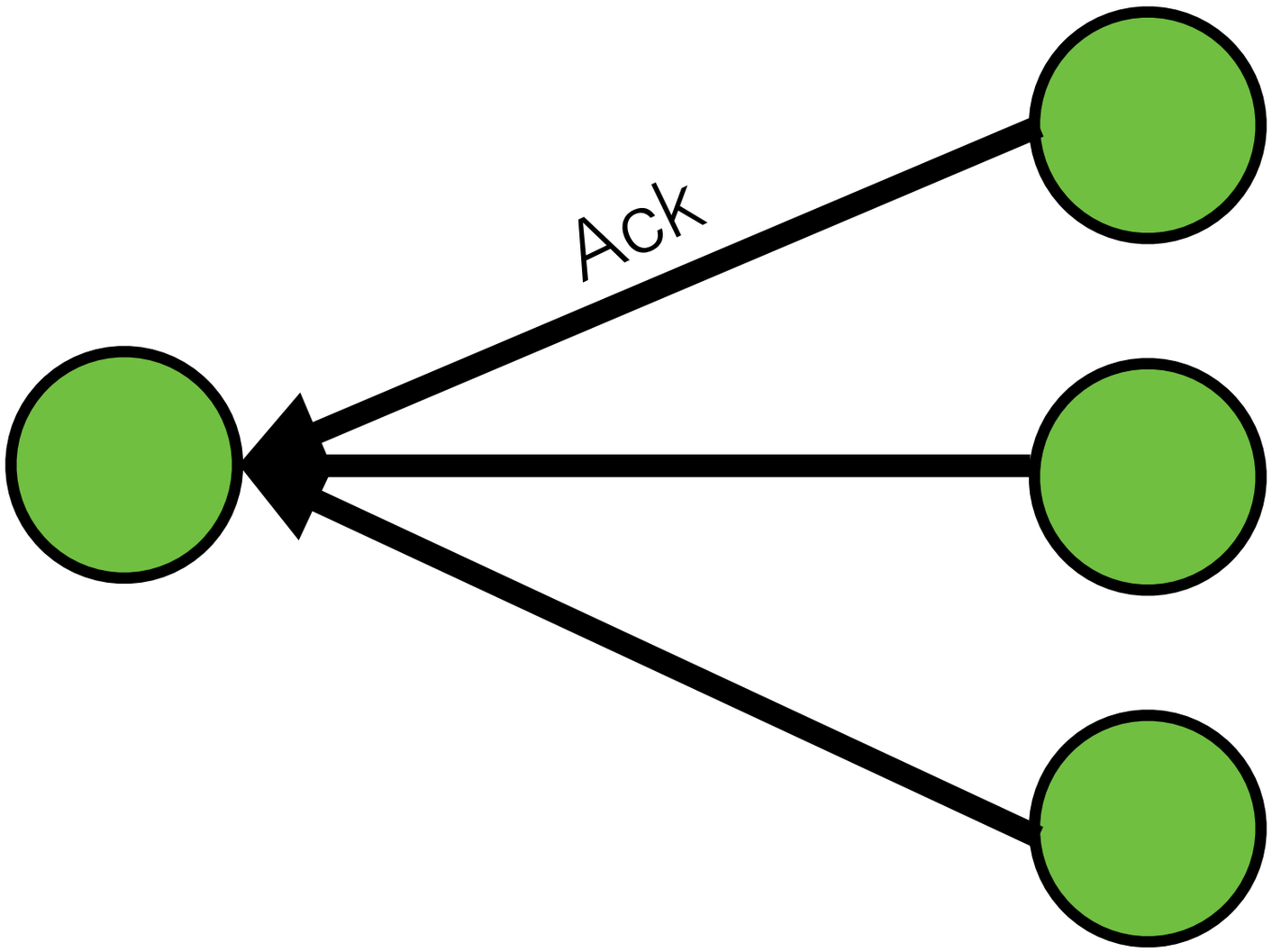
Fixes

- More Reliable
- Lower Latency

Another Idea



Another Idea



Problem

Every server sends an ack to the sender

Conclusion

This is hard.

Reliable Broadcast

Basic Problem

Send a message to a set of servers

Applications

- Stock Exchanges
- Control Systems
- Configuration
- Cache Invalidation

Atomic Broadcast

- Paxos
- ZooKeeper Atomic Broadcast
- etc…

Strong Guarantees

- Guaranteed Delivery

- Total Order

Too Many Guarantees

- Don’t need Ordering
- Don’t need Atomicity

“Best-Effort” Broadcast

“Try very hard” to deliver a message

Algorithms

- Scalable Reliable Multicast
- Bimodal Multicast
- Plumtree
- Sprinkler
- etc…

Bimodal Multicast

KENNETH P. BIRMAN
Cornell University
MARK HAYDEN
Digital Equipment Corporation/Compaq
OZNUR OZKASAP and ZHEN XIAO
Cornell University
MIHAI BUDIU
Carnegie Mellon University
and
YARON MINSKY
Cornell University

There are many methods for making a multicast protocol "reliable." At one end of the spectrum, a reliable multicast protocol might offer atomicity guarantees, such as all-or-nothing delivery, delivery ordering, and perhaps additional properties such as virtually synchronous addressing. At the other are protocols that use local repair to overcome transient packet loss in the network, offering "best effort" reliability. Yet none of this prior work has treated stability of multicast delivery as a basic reliability property, such as might be needed in an internet radio, television, or conferencing application. This article looks at reliability with a new goal: development of a multicast protocol which is reliable in a sense that can be rigorously quantified and includes throughput stability guarantees. We characterize this new protocol as a "bimodal multicast" in reference to its reliability model, which corresponds to a family of bimodal probability distributions. Here, we introduce the protocol, provide a theoretical analysis of its behavior, review experimental results, and discuss some candidate applications. These confirm that bimodal multicast is reliable, scalable, and that the protocol provides remarkably stable delivery throughput.

Goals

- “Best-Effort” Delivery

- Predictable Performance

Algorithm

- Dissemination

- Anti-Entropy (Gossip)

Dissemination

- Unreliably broadcast a message to all other servers

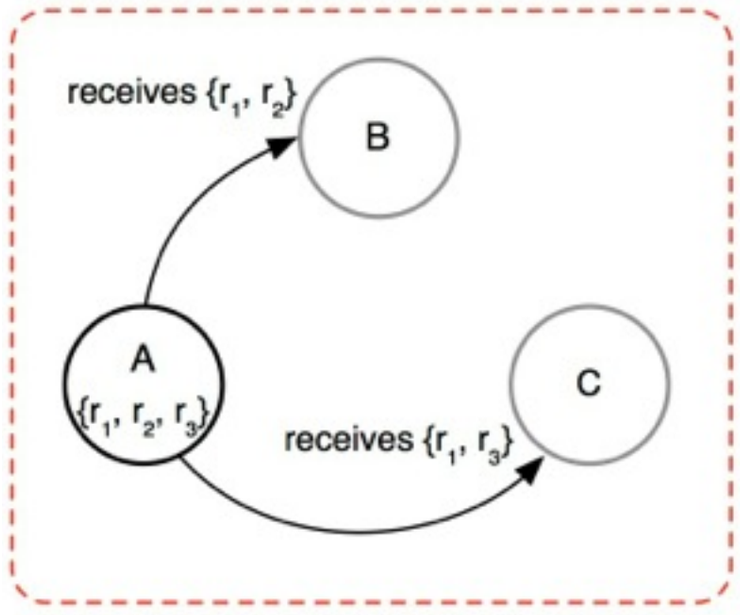
- IP multicast, UDP in a for loop, etc…

Anti-Entropy

- Each server sends a random server a digest of the messages it knows about

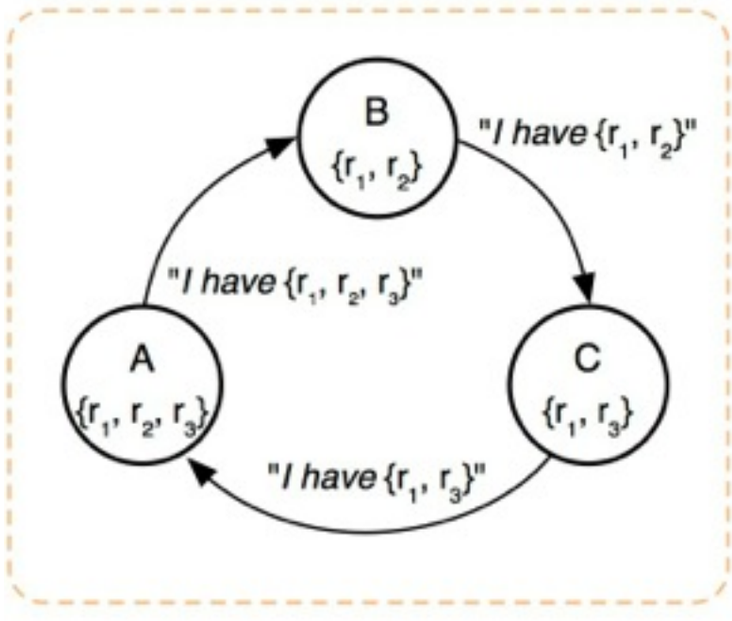
- If the other server hasn't received some of those messages, it requests that they be retransmitted

Example



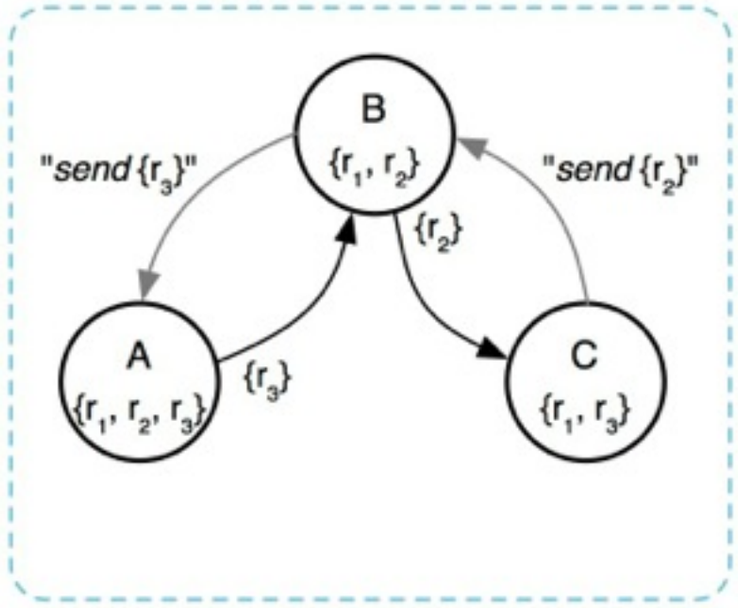
Broadcast

Example

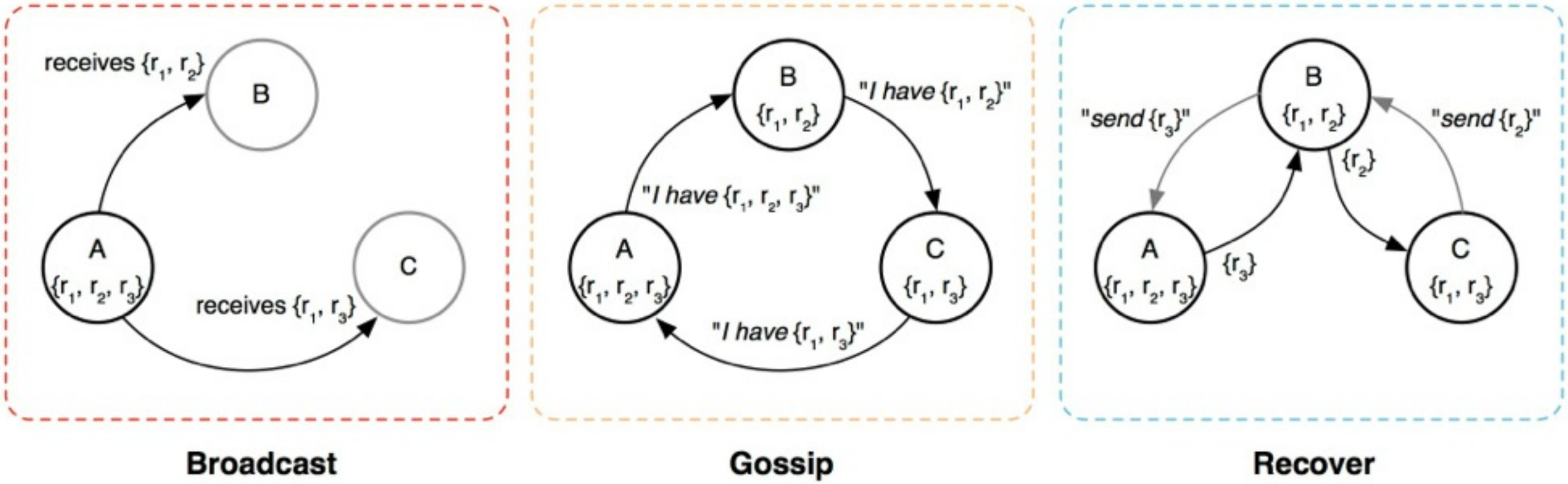


Gossip

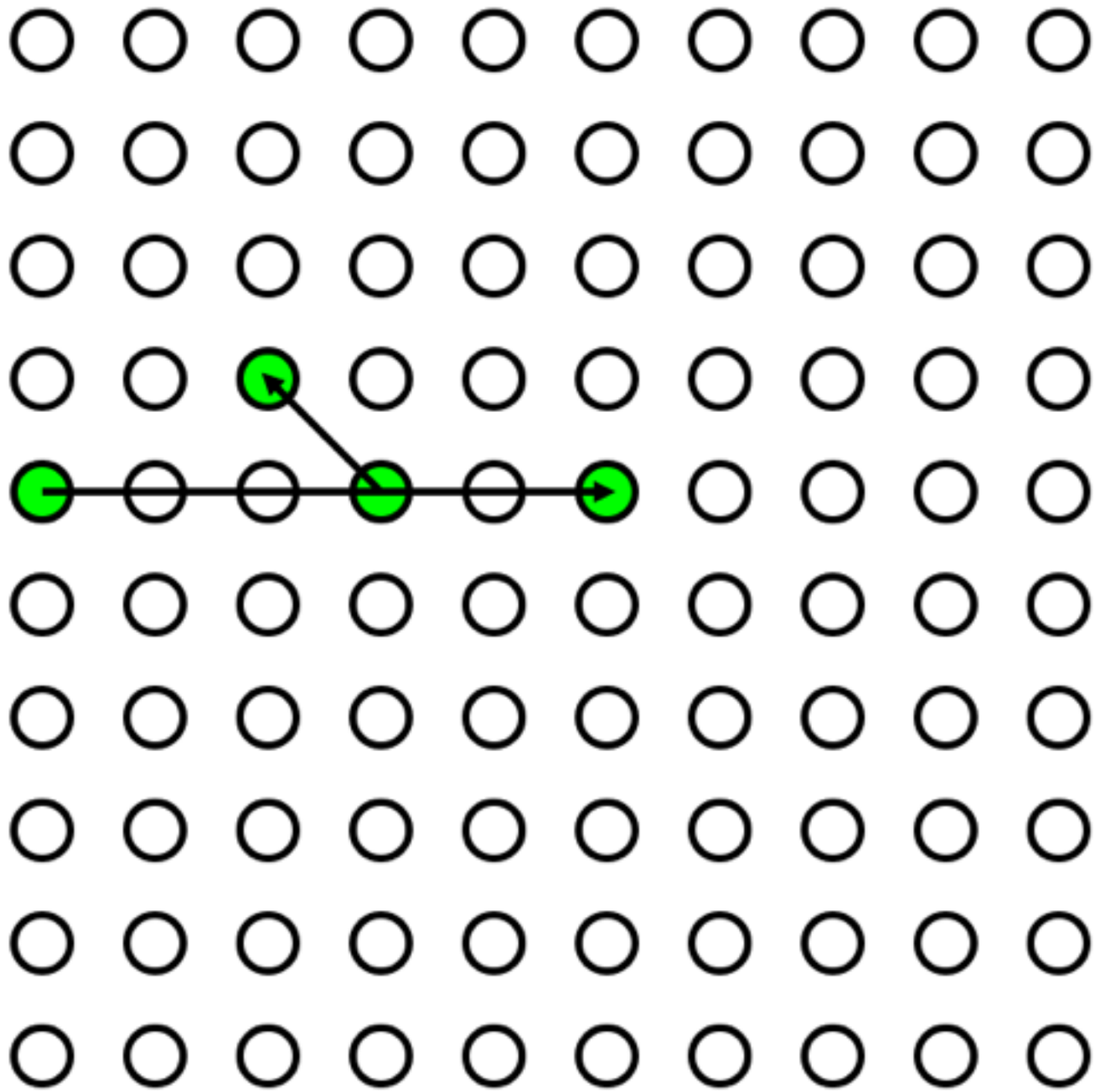
Example



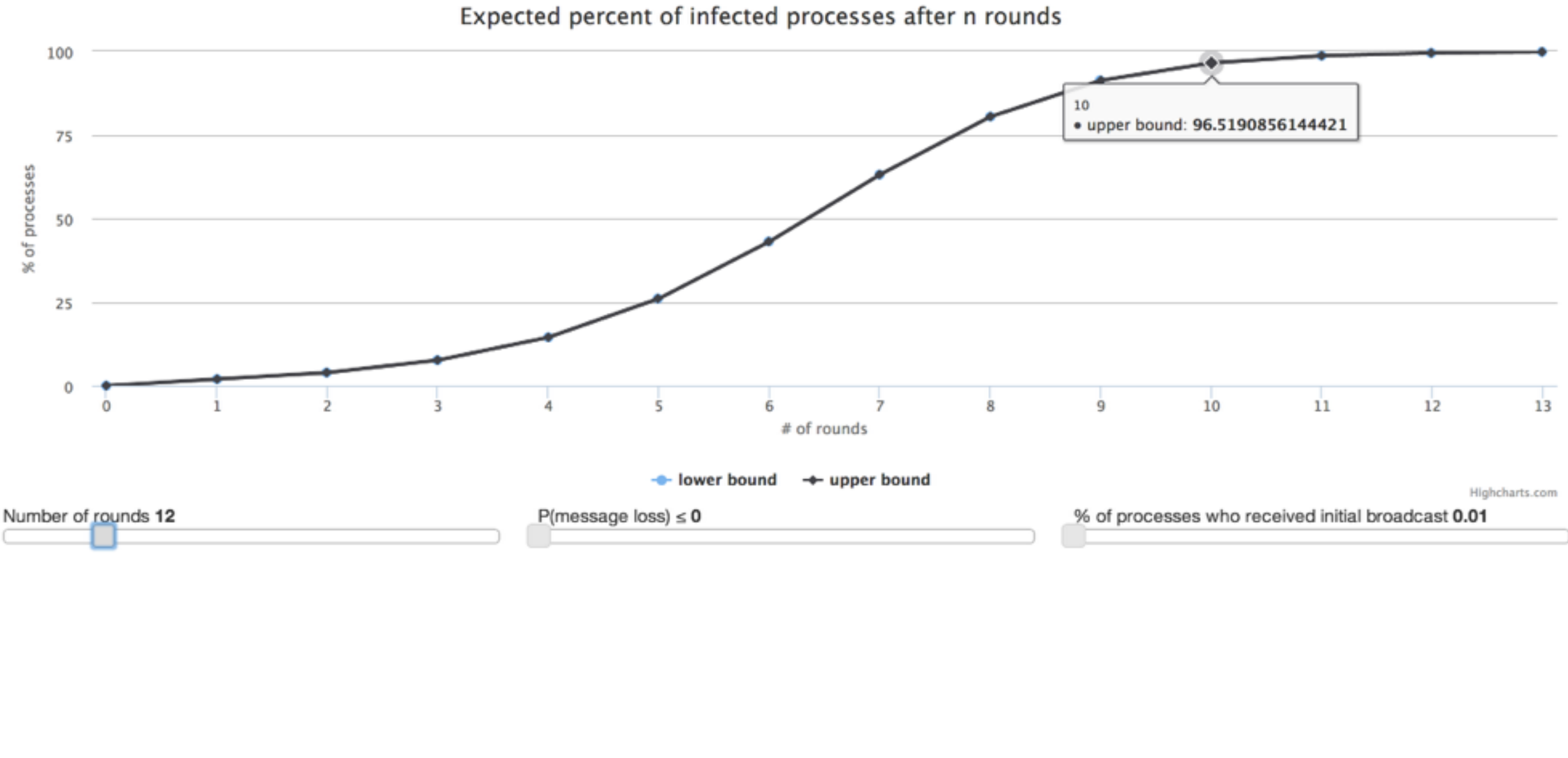
Recover


receives {r1, r2}
B
A
{r1, r2, r3}
C
receives {r1, r3}
Broadcast
B
{r1, r2}
"I have {r1, r2}"
"I have {r1, r2, r3}"
A
{r1, r2, r3}
C
{r1, r3}
"I have {r1, r3}"
Gossip
B
{r1, r2}
"send {r3}"
"send {r2}"
{r2}
A
{r1, r2, r3}
{r3}
C
{r1, r3}
Recover

Gossip



Convergence

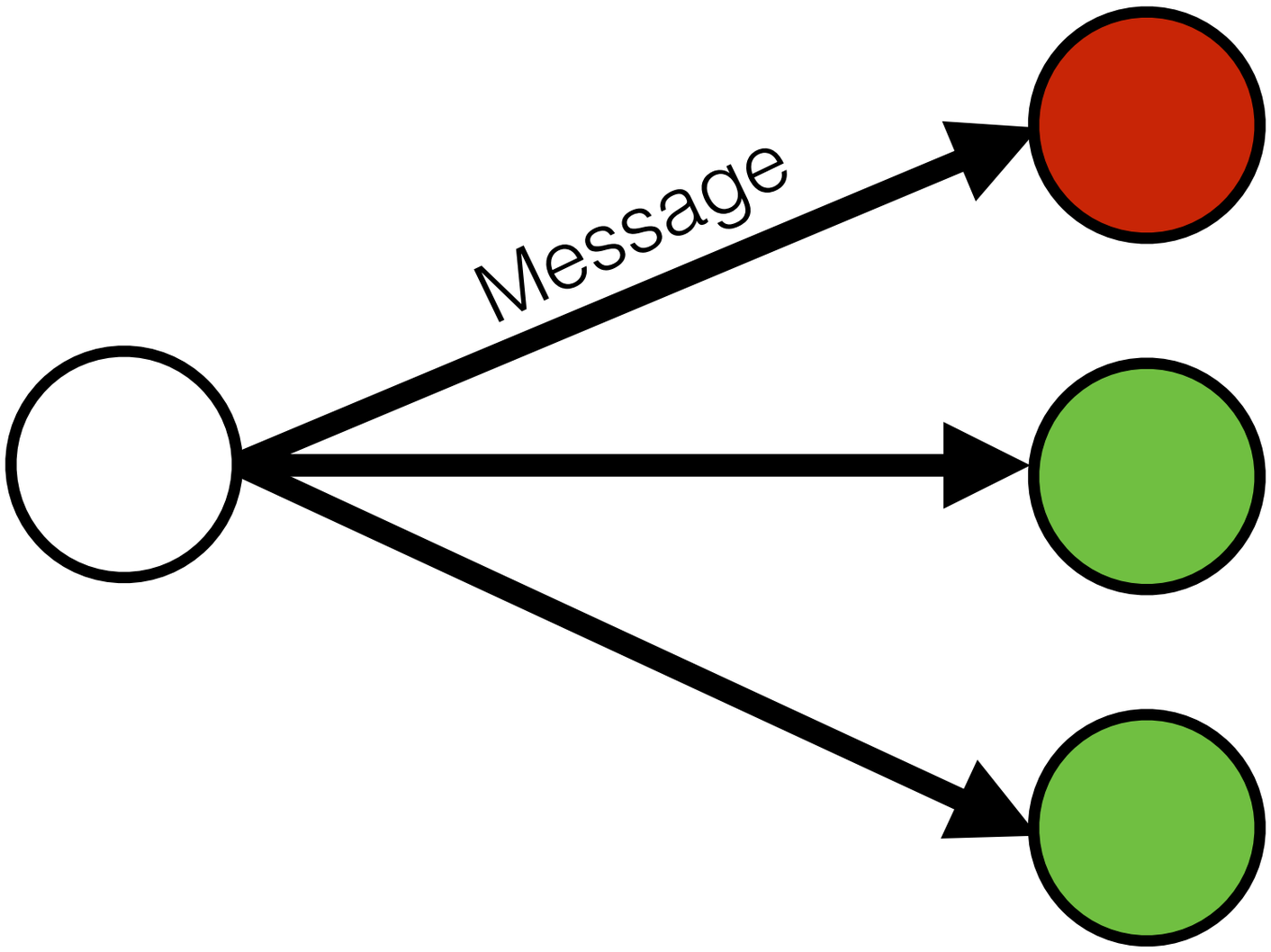


Goals

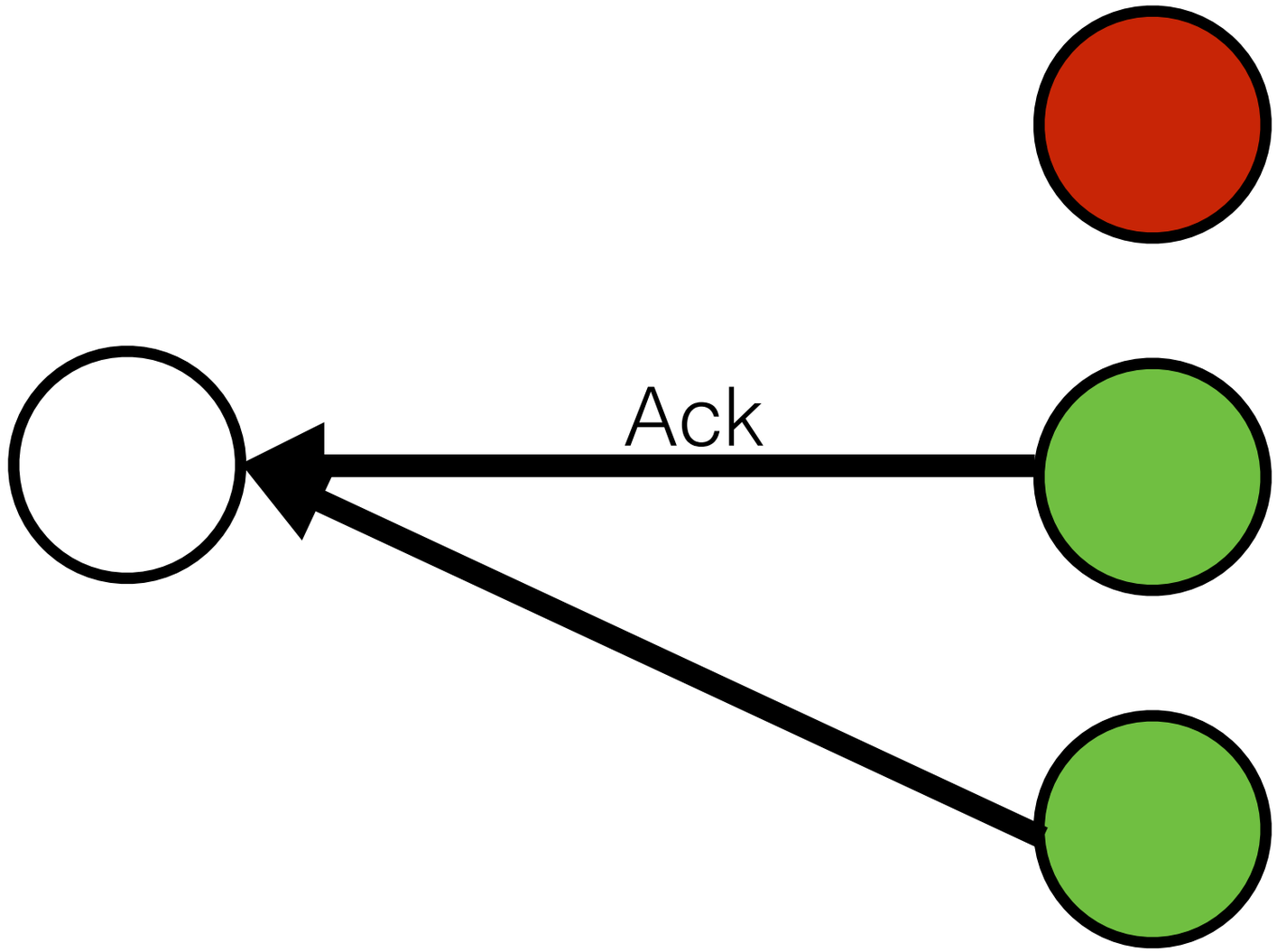
- “Best-Effort” Delivery

- Predictable Performance

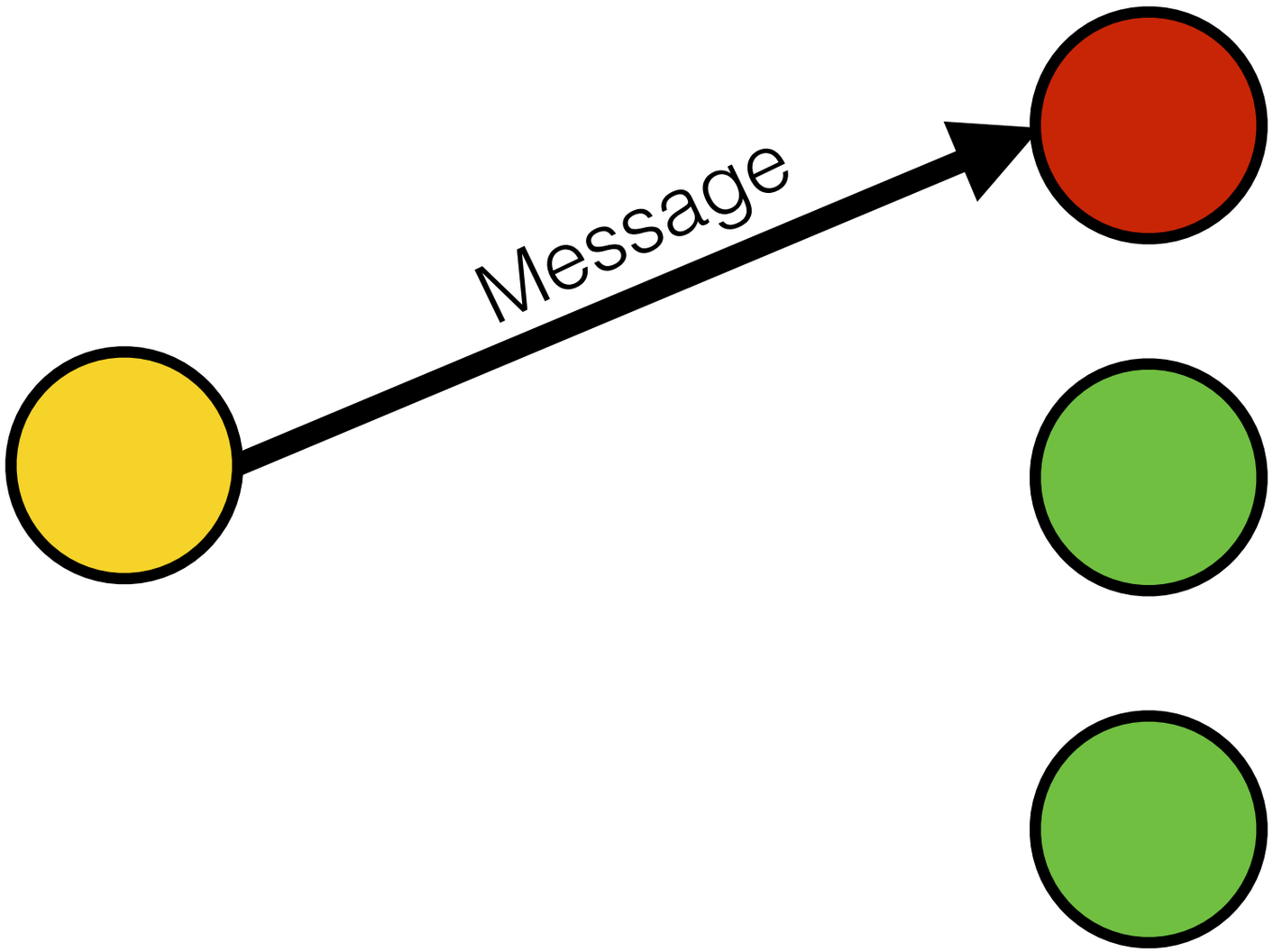
Stable Throughput



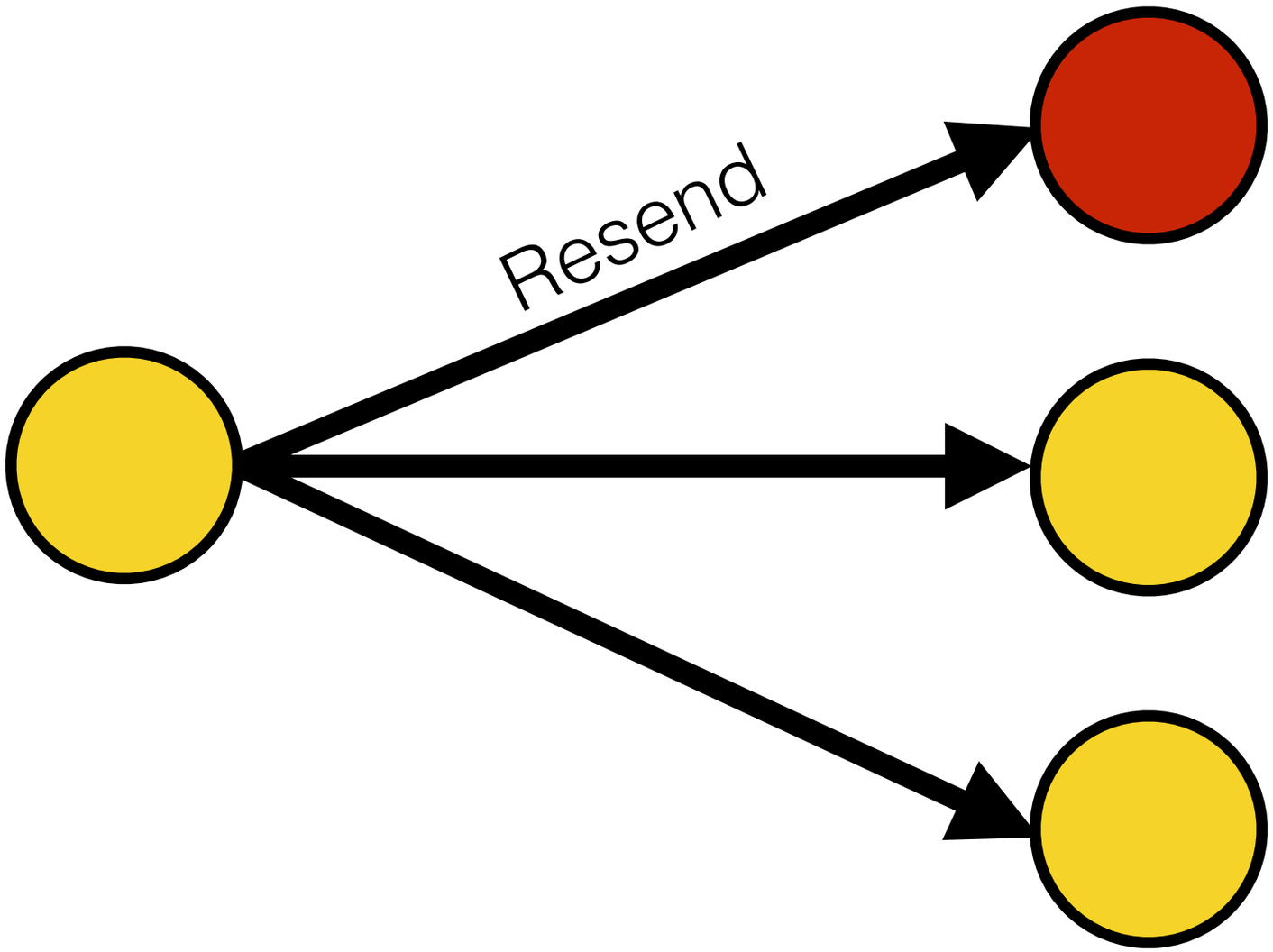
Stable Throughput



Stable Throughput



Stable Throughput



Independence

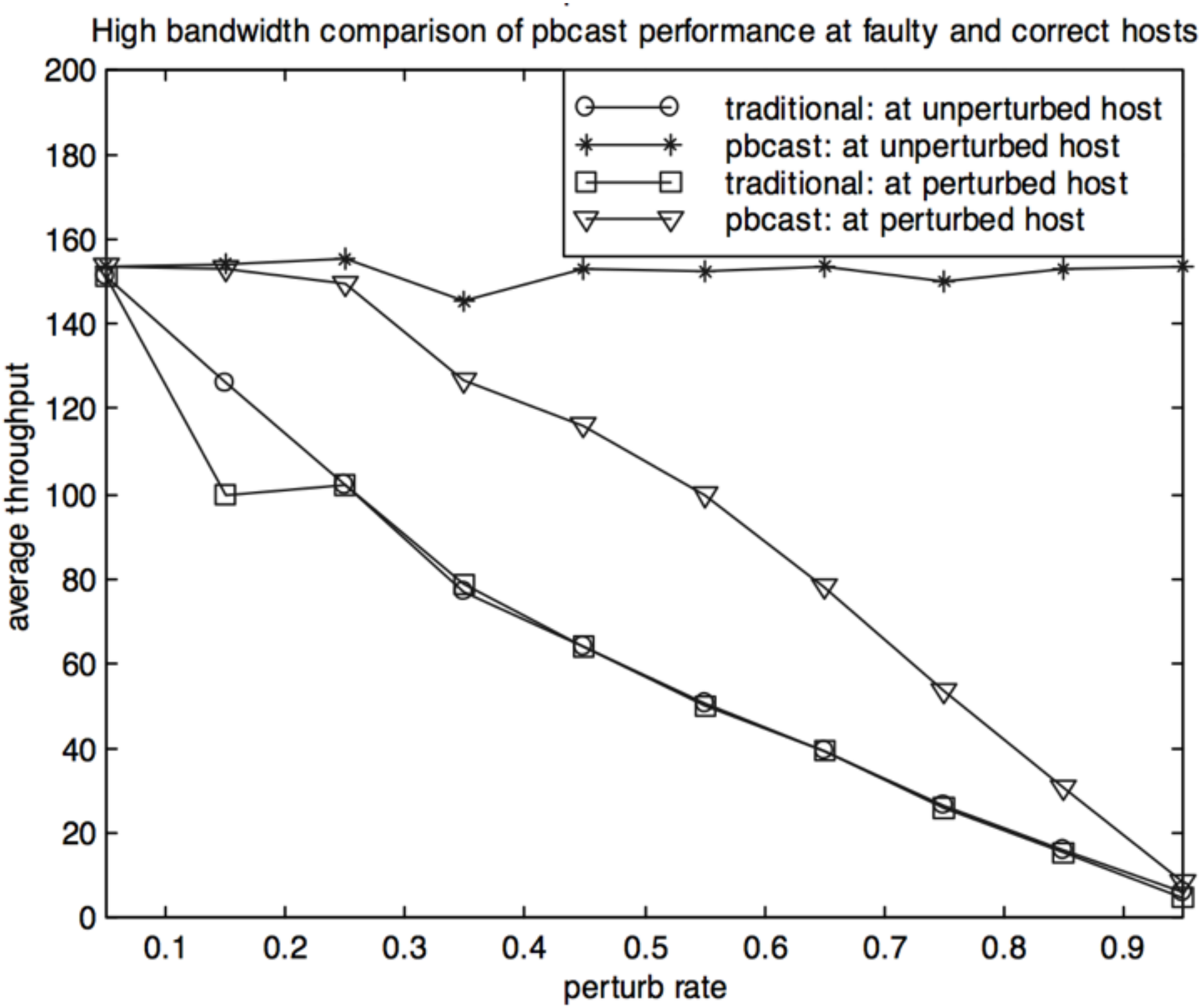
A server that’s behind will recover from many servers in the cluster.

Retransmission Limits

Don’t DDoS servers that are trying to recover.

Soft Failure Detection

Ignore servers that are running behind.


High bandwidth comparison of pbcast performance at faulty and correct hosts
traditional: at unperturbed host
pbcast: at unperturbed host
traditional: at perturbed host
pbcast: at perturbed host
average throughput
perturb rate
200
180
160
140
120
100
80
60
40
20
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

“Best effort” is ok!

- Messages are delivered
- Predicable Performance

Powderhorn

Bimodal Multicast in the Wild

Development

All the logic is in failure handling.

Jepsen

- Five nodes
- Simulated partitions, packet loss
- http://github.com/aphyr/jepsen

“Scale” Testing

- EC2
- 200 m1.mediums

Avalanche

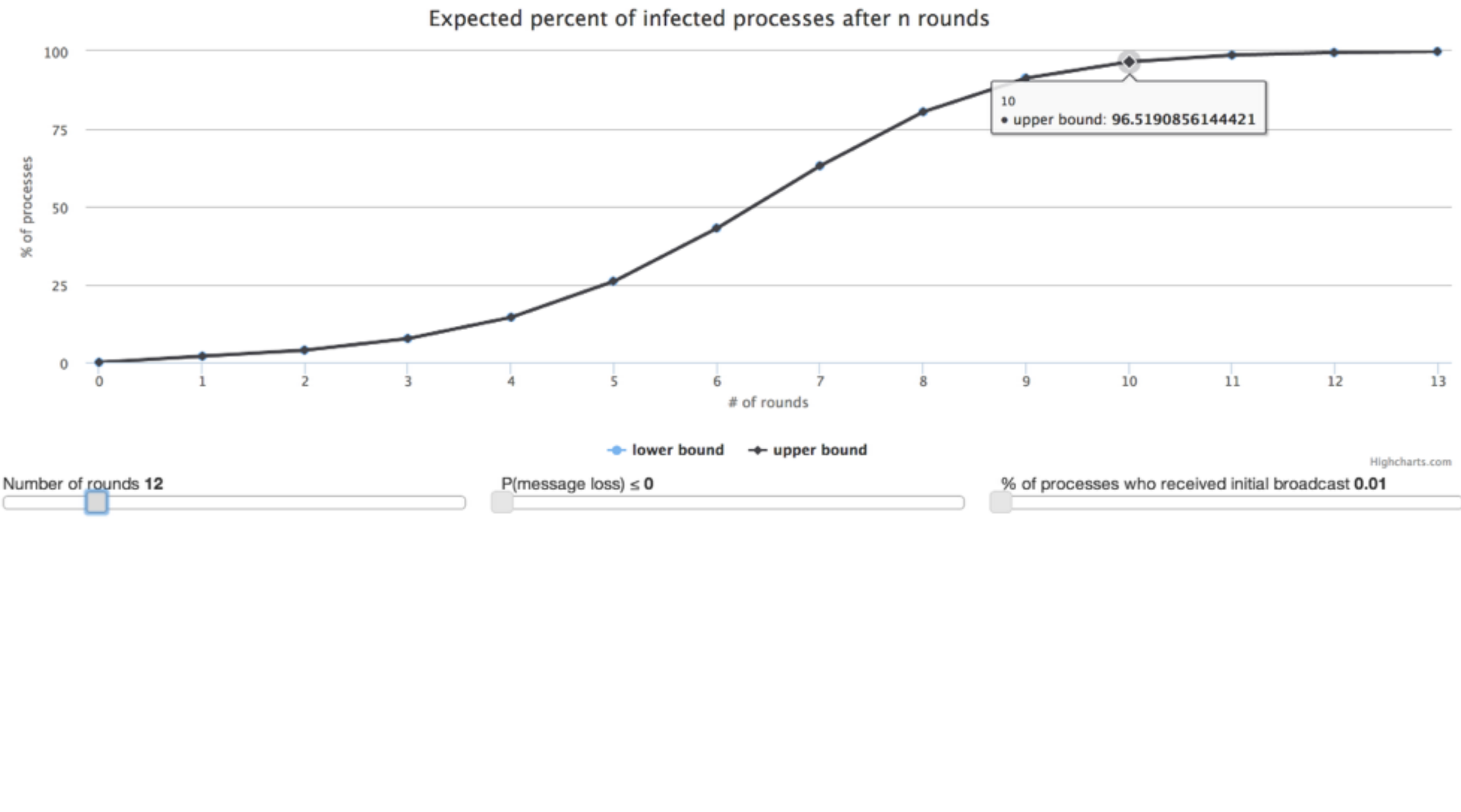
- Random, repeatable network fault injection

- https://github.com/fastly/avalanche

Garbage Collection

“I have messages {1,2,3,4,5,6,7,8,9,…}”

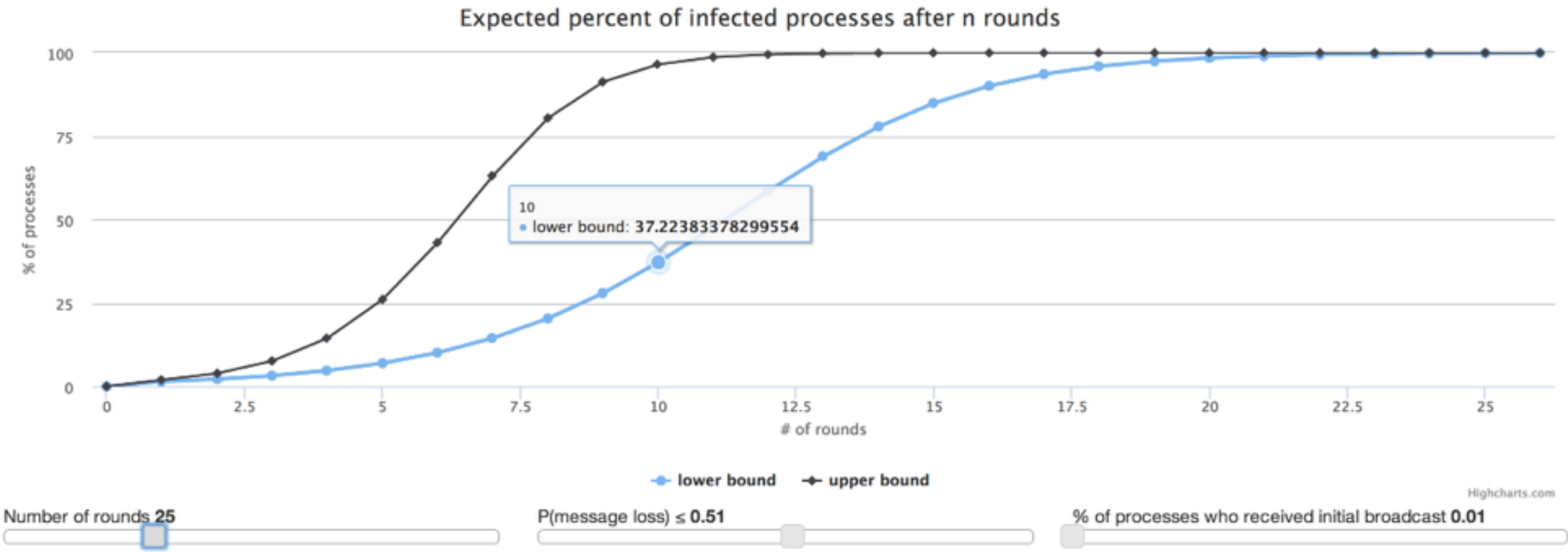
Garbage Collection



Computers are Horrible

We see high packet loss and network partitions
all the time.

Convergence



The Digest

List of Message IDs

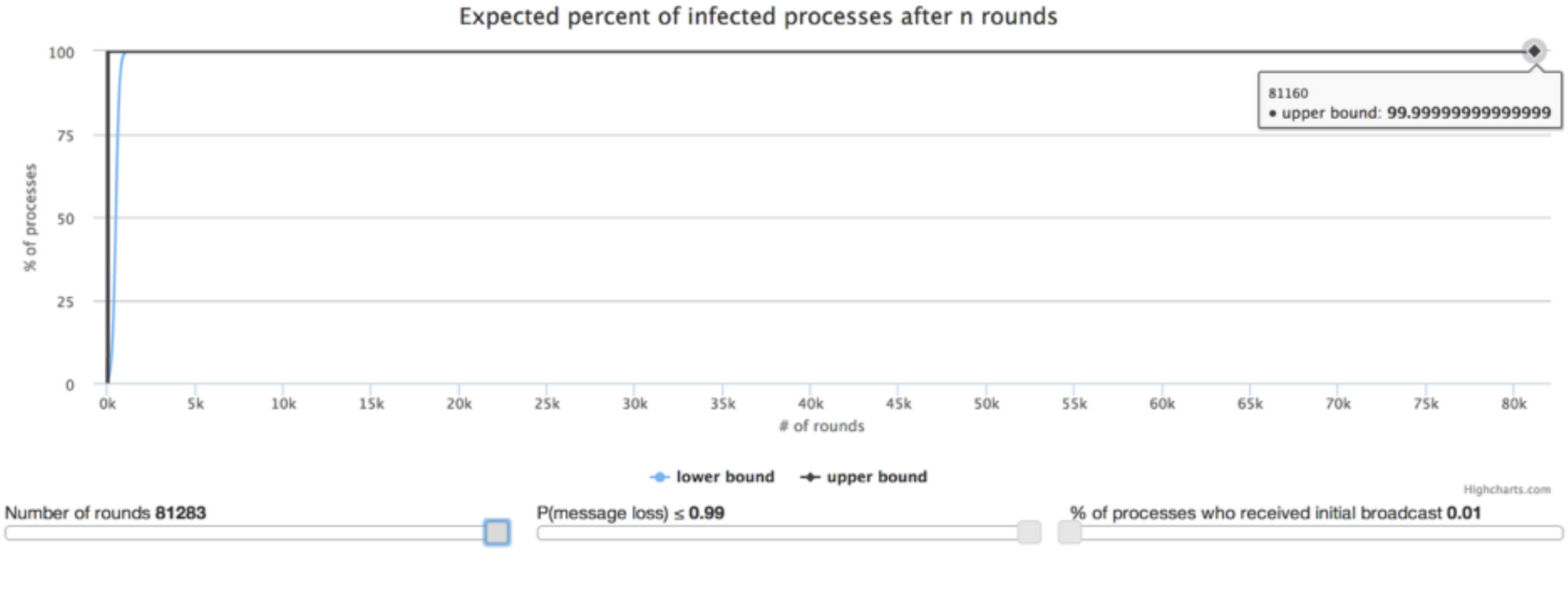
The Digest

Doesn’t Have to be a List

The Digest

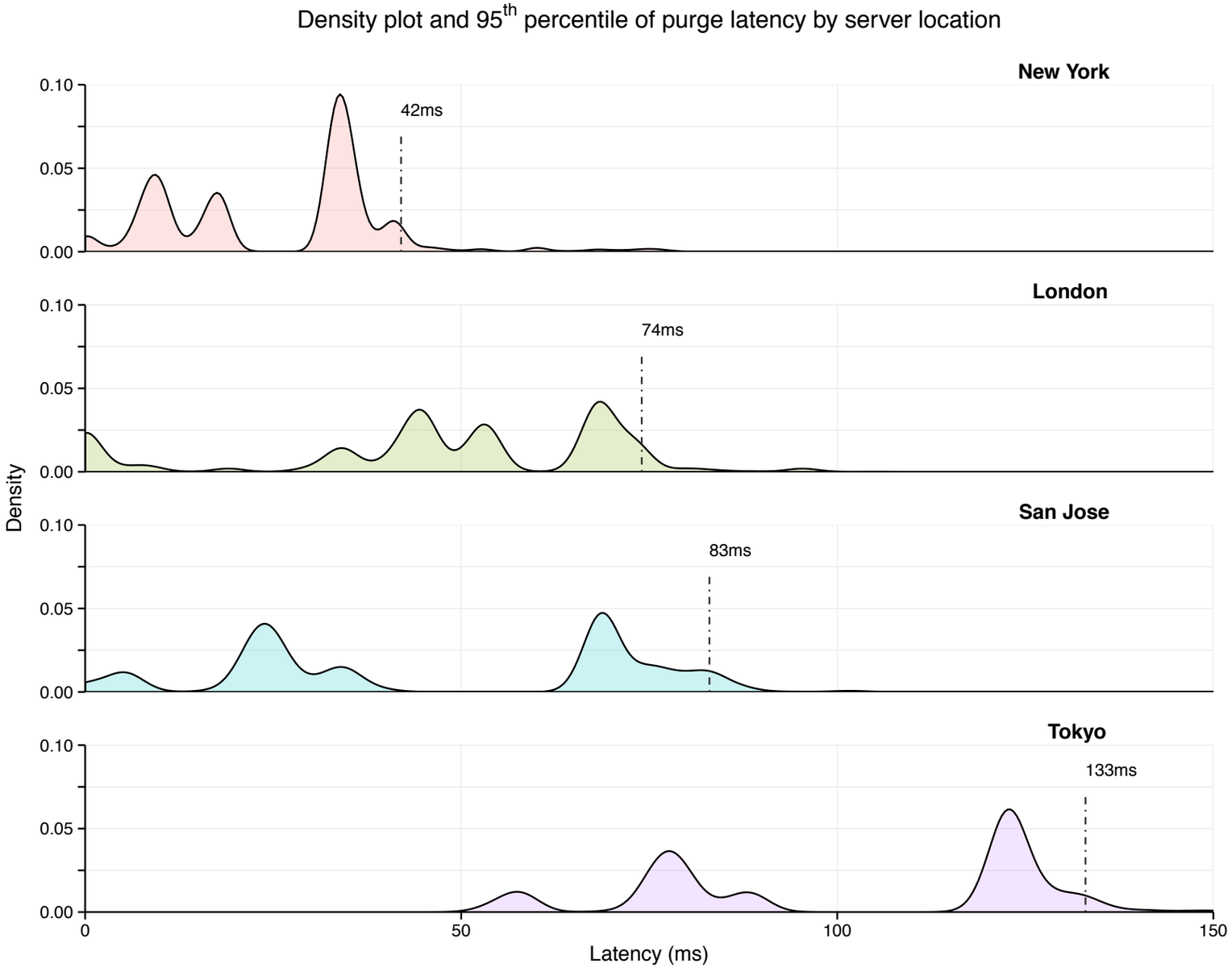
- **send ranges of ids of known messages**
- “messages 1 to 1,000,000 from server 1"
- can represent large numbers of messages

The Digest

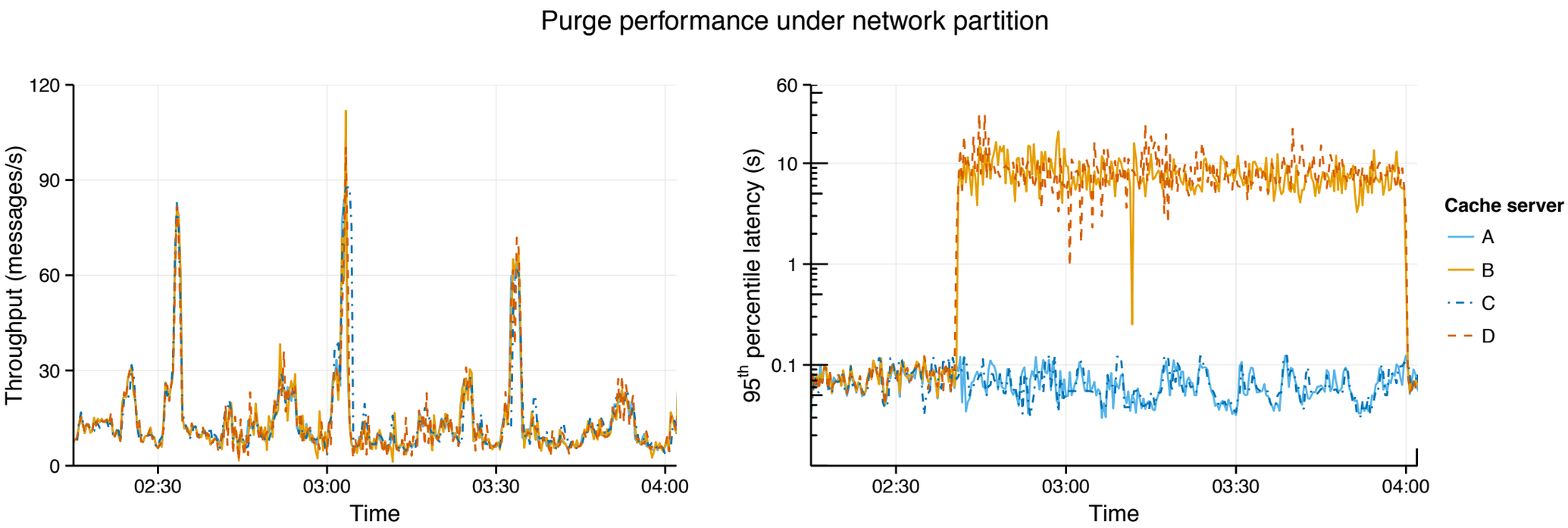

Expected percent of infected processes after n rounds
100
75
50
25
0
% of processes
0k
5k
10k
15k
20k
25k
30k
35k
40k
45k
50k
55k
60k
65k
70k
75k
80k
# of rounds
81160
• upper bound: 99.99999999999999
lower bound
upper bound
Highcharts.com
Number of rounds 81283
P(message loss) ≤ 0.99
% of processes who received initial broadcast 0.01

Behavior

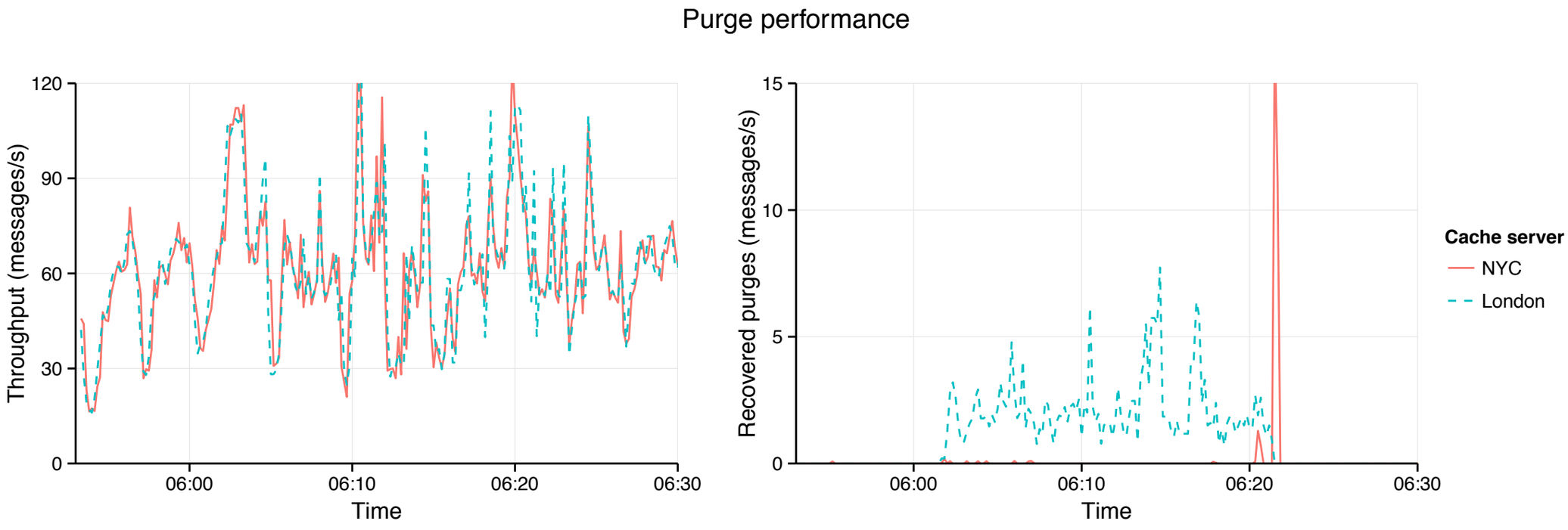
End-to-End Latency



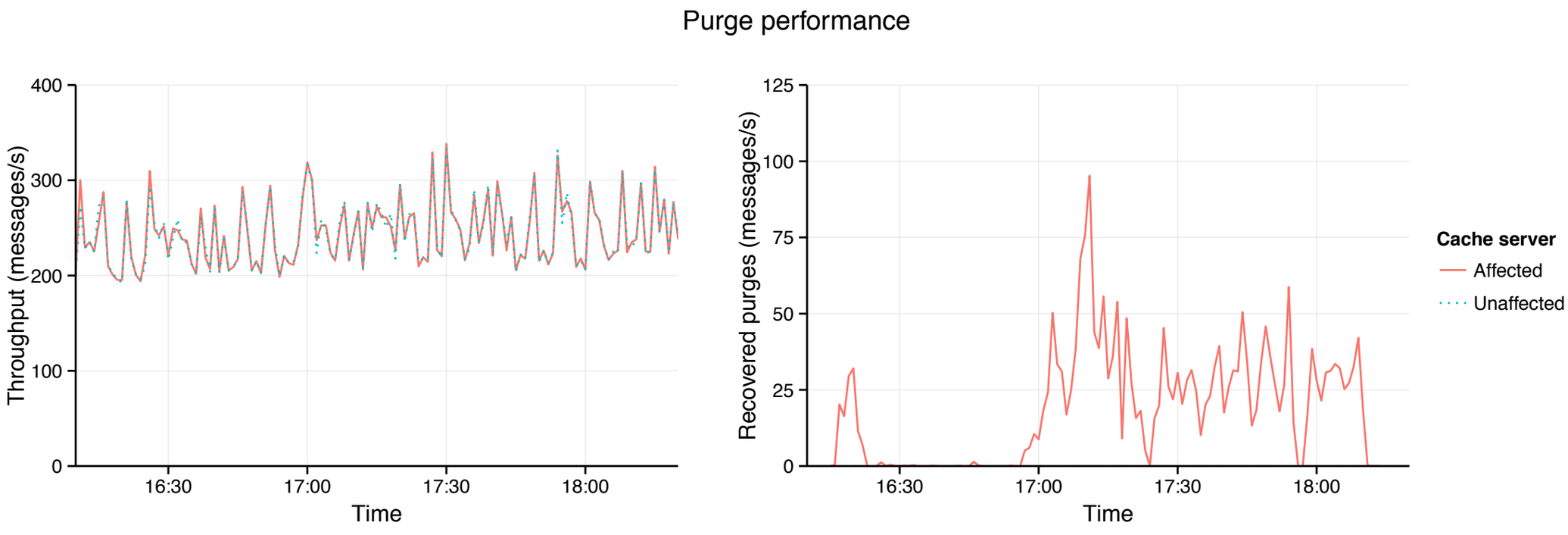
Firewall Partition



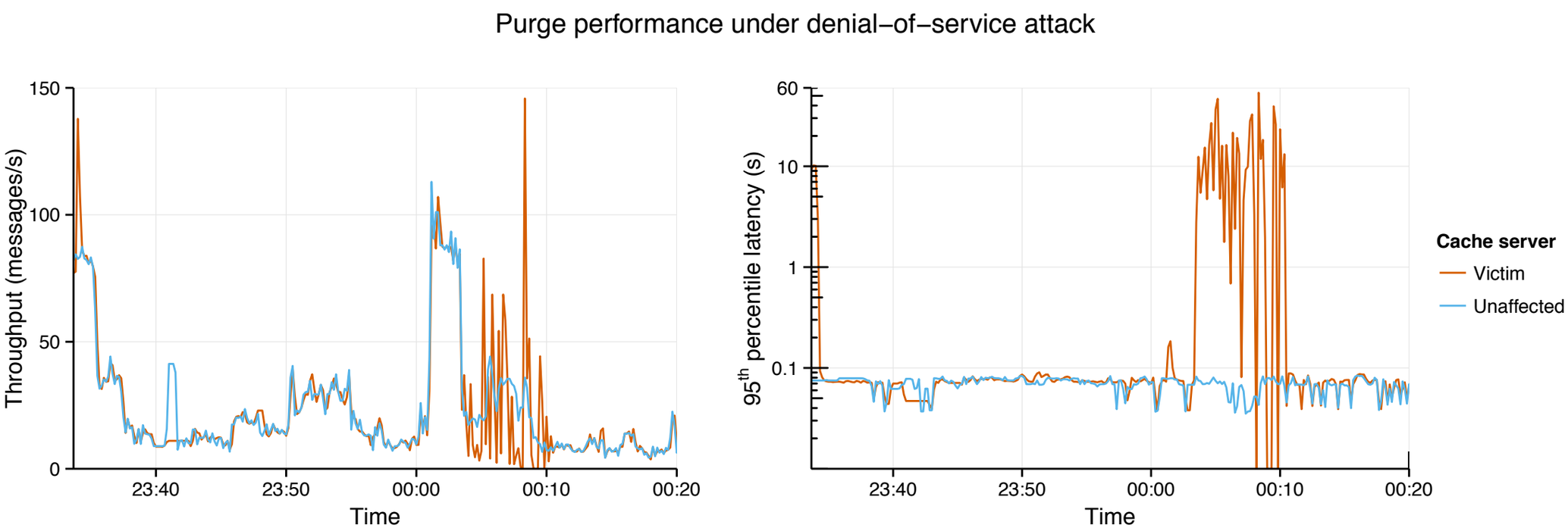
NYC to London Partition



APAC Packet Loss



DDoS


Purge performance under denial–of–service attack
Throughput (messages/s)
150
100
50
0
23:40
23:50
00:00
00:10
00:20
Time
95th percentile latency (s)
60
10
1
0.1
23:40
23:50
00:00
00:10
00:20
Time
Cache server
Victim
Unaffected

Bimodal Multicast is Great

We generally don’t have to worry about purging failing, even when the network does.

Fin.

brucespang.com/bimodal

Questions?

brucespang.com/bimodal

We're Hiring

www.fastly.com/about/jobs

Decent Hash Table