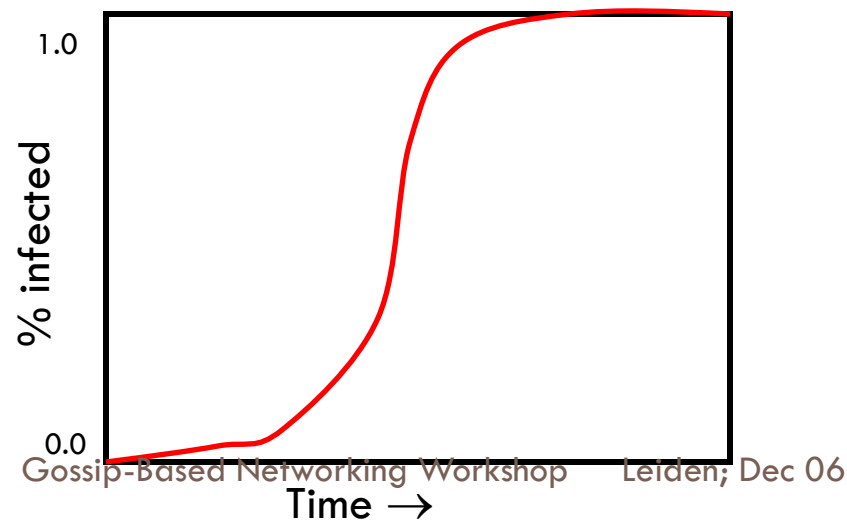


# CS541 2: BIMODAL MULTICAST ASTROLABE

# Gossip 201

2

- Recall from early in the semester that gossip spreads in  $\log(\text{system size})$  time
- But is this actually “fast”?



# Gossip in distributed systems

3

- Log(N) can be a very big number!
  - ▣ With  $N=100,000$ ,  $\log(N)$  would be 12
  - ▣ So with one gossip round per five seconds, information needs *one minute* to spread in a large system!
- Some gossip protocols combine pure gossip with an accelerator
  - ▣ A good way to get the word out quickly

# Bimodal Multicast

4

- To send a message, this protocol uses IP multicast
- We just transmit it without delay and we don't expect any form of responses
  - ▣ Not reliable, no acks
  - ▣ No flow control (this can be an issue)
  - ▣ In data centers that lack IP multicast, can simulate by sending UDP packets 1:1 without acks

# What's the cost of an IP multicast?

5

- In principle, each Bimodal Multicast packet traverses the relevant data center links and routers just once per message
- So this is extremely cheap... but how do we deal with systems that didn't receive the multicast?

# Making Bimodal Multicast reliable

6

- We can use gossip!
- Every node tracks the membership of the target group (using gossip, just like with Kelips, the DHT we studied early in the semester)
  - ▣ Bootstrap by learning “some node addresses” from some kind of a server or web page
  - ▣ But then exchange of gossip used to improve accuracy

# Making Bimodal Multicast reliable

7

- Now, layer in a gossip mechanism that gossips about multicasts each node knows about
  - ▣ Rather than sending the multicasts themselves, the gossip messages just talk about “digests”, which are lists
    - Node A might send node B
      - I have messages 1-18 from sender X
      - I have message 11 from sender Y
      - I have messages 14, 16 and 22-71 from sender Z
    - Compactly represented...
  - ▣ This is a form of “push” gossip

# Making Bimodal Multicast reliable

8

- On receiving such a gossip message, the recipient checks to see which messages it has that the gossip sender lacks, and vice versa
  
- Then it responds
  - ▣ I have copies of messages  $M$ ,  $M'$  and  $M''$  that you seem to lack
  - ▣ I would like a copy of messages  $N$ ,  $N'$  and  $N''$  please
  
- An exchange of the actual messages follows



# Optimizations

- Bimodal Multicast resends using IP multicast if there is “evidence” that a few nodes may be missing the same thing
  - ▣ E.g. if two nodes ask for the same retransmission
  - ▣ Or if a retransmission shows up from a very remote node (IP multicast doesn't always work in WANs)
- It also prioritizes recent messages over old ones
- Reliability has a “bimodal” probability curve: either nobody gets a message or nearly everyone does

# lpcast variation

10

- In this variation on Bimodal Multicast instead of gossiping with every node in a system, we modify the Bimodal Multicast protocol
  - ▣ It maintains a “peer overlay”: each member only gossips with a smaller set of peers picked to be reachable with low round-trip times, plus a second small set of remote peers picked to ensure that the graph is very highly connected and has a small diameter
  - ▣ Called a “small worlds” structure by Jon Kleinberg
- Lpcast is often faster, but equally reliable!

# Speculation... about speed

11

- When we combine IP multicast with gossip we try to match the tool we're using with the need
- Try to get the messages through fast... but if loss occurs, try to have a very predictable recovery cost
  - ▣ Gossip has a totally predictable worst-case load
  - ▣ This is appealing at large scales
- How can we generalize this concept?

# A thought question

12

- What's the best way to
  - ▣ Count the number of nodes in a system?
  - ▣ Compute the average load, or find the most loaded nodes, or least loaded nodes?
  
- Options to consider
  - ▣ Pure gossip solution
  - ▣ Construct an overlay tree (via “flooding”, like in our consistent snapshot algorithm), then count nodes in the tree, or pull the answer from the leaves to the root...

# ... and the answer is

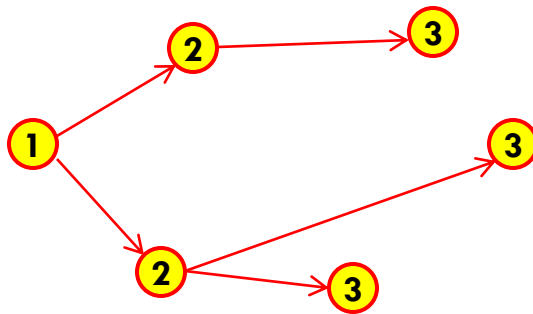
13

- Gossip isn't very good for some of these tasks!
  - ▣ There are gossip solutions for counting nodes, but they give approximate answers and run slowly
  - ▣ Tricky to compute something like an average because of “re-counting” effect, (best algorithm: Kempe *et al*)
- On the other hand, gossip works well for finding the  $c$  most loaded or least loaded nodes (constant  $c$ )
- Gossip solutions will usually run in time  $O(\log N)$  and generally give probabilistic solutions

# Yet with flooding... easy!

14

- Recall how flooding works



*Labels: distance of the node from the root*

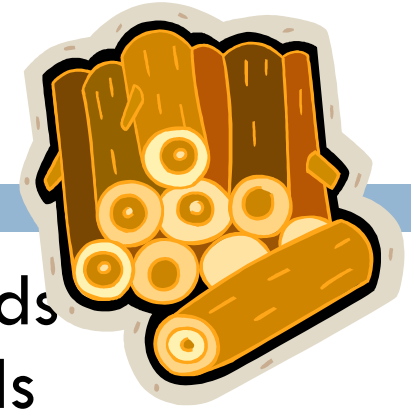
- Basically: we construct a tree by pushing data towards the leaves and linking a node to its parent when that node first learns of the flood
- Can do this with a fixed topology or in a gossip style by picking random next hops

# This is a “spanning tree”

15

- Once we have a spanning tree
  - ▣ To count the nodes, just have leaves report 1 to their parents and inner nodes count the values from their children
  - ▣ To compute an average, have the leaves report their value and the parent compute the sum, then divide by the count of nodes
  - ▣ To find the least or most loaded node, inner nodes compute a min or max...
- Tree should have roughly  $\log(N)$  depth, but once we build it, we can reuse it for a while

# Not all logs are identical!



16

- When we say that a gossip protocol needs time  $\log(N)$  to run, we mean  $\log(N)$  rounds
  - ▣ And a gossip protocol usually sends one message every five seconds or so, hence with 100,000 nodes, 60 secs
- But our spanning tree protocol is constructed using a flooding algorithm that runs in a hurry
  - ▣  $\log(N)$  depth, but each “hop” takes perhaps a millisecond.
  - ▣ So with 100,000 nodes we have our tree in 12 ms and answers in 24ms!



# Insight?

17

- Gossip has time complexity  $O(\log N)$  but the “constant” can be rather big (5000 times larger in our example)
- Spanning tree had same time complexity but a tiny constant in front
  
- But network load for spanning tree was much higher
  - ▣ In the last step, we may have reached roughly half the nodes in the system
  - ▣ So 50,000 messages were sent all at the same time!

# Gossip vs “Urgent”?

18

- With gossip, we have a slow but steady story
  - ▣ We know the speed and the cost, and both are low
  - ▣ A constant, low-key, background cost
  - ▣ And gossip is also very robust
  
- Urgent protocols (like our flooding protocol, or 2PC, or reliable virtually synchronous multicast)
  - ▣ Are way faster
  - ▣ But produce load spikes
  - ▣ And may be fragile, prone to broadcast storms, etc

# Introducing hierarchy

19

- One issue with gossip is that the messages fill up
  - With constant sized messages...
  - ... and constant rate of communication
  - ... we'll inevitably reach the limit!
  
- Can we introduce hierarchy into gossip systems?

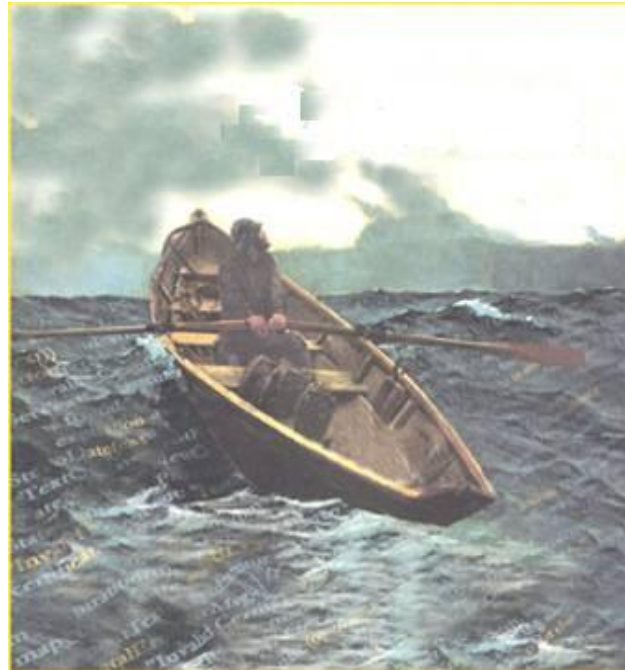
# Astrolabe

20

- **Intended as help for applications adrift in a sea of information**
- **Structure emerges from a randomized gossip protocol**
- **This approach is robust and scalable even under stress that cripples traditional systems**

**Developed at RNS, Cornell**

- **By Robbert van Renesse, with many others helping...**
- **Today used extensively within Amazon.com**



# Astrolabe is a flexible monitoring overlay

21



**swift.cs.cornell.edu**

| Name     | Time | Load | Weblogic? | SMTP? | Word Version |
|----------|------|------|-----------|-------|--------------|
| swift    | 2271 | 1.8  | 0         | 1     | 6.2          |
| falcon   | 1971 | 1.5  | 1         | 0     | 4.1          |
| cardinal | 2004 | 4.5  | 1         | 0     | 6.0          |

**Periodically, pull data from monitored systems**



**cardinal.cs.cornell.edu**

| Name     | Time | Load | Weblogic ? | SMTP? | Word Version |
|----------|------|------|------------|-------|--------------|
| swift    | 2003 | .67  | 0          | 1     | 6.2          |
| falcon   | 1976 | 2.7  | 1          | 0     | 4.1          |
| cardinal | 2231 | 1.7  | 1          | 1     | 6.0          |

# Astrolabe in a single domain

22

- Each node owns a single tuple, like the management information base (MIB)
- Nodes discover one-another through a simple broadcast scheme (“anyone out there?”) and gossip about membership
  - ▣ Nodes also keep replicas of one-another’s rows
  - ▣ Periodically (uniformly at random) merge your state with some else...

# State Merge: Core of Astrolabe epidemic

23



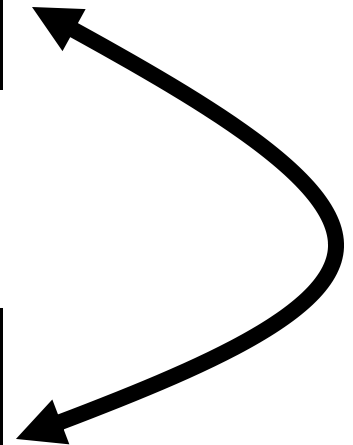
**swift.cs.cornell.edu**

| Name     | Time | Load | Weblogic? | SMTP? | Word Version |
|----------|------|------|-----------|-------|--------------|
| swift    | 2011 | 2.0  | 0         | 1     | 6.2          |
| falcon   | 1971 | 1.5  | 1         | 0     | 4.1          |
| cardinal | 2004 | 4.5  | 1         | 0     | 6.0          |



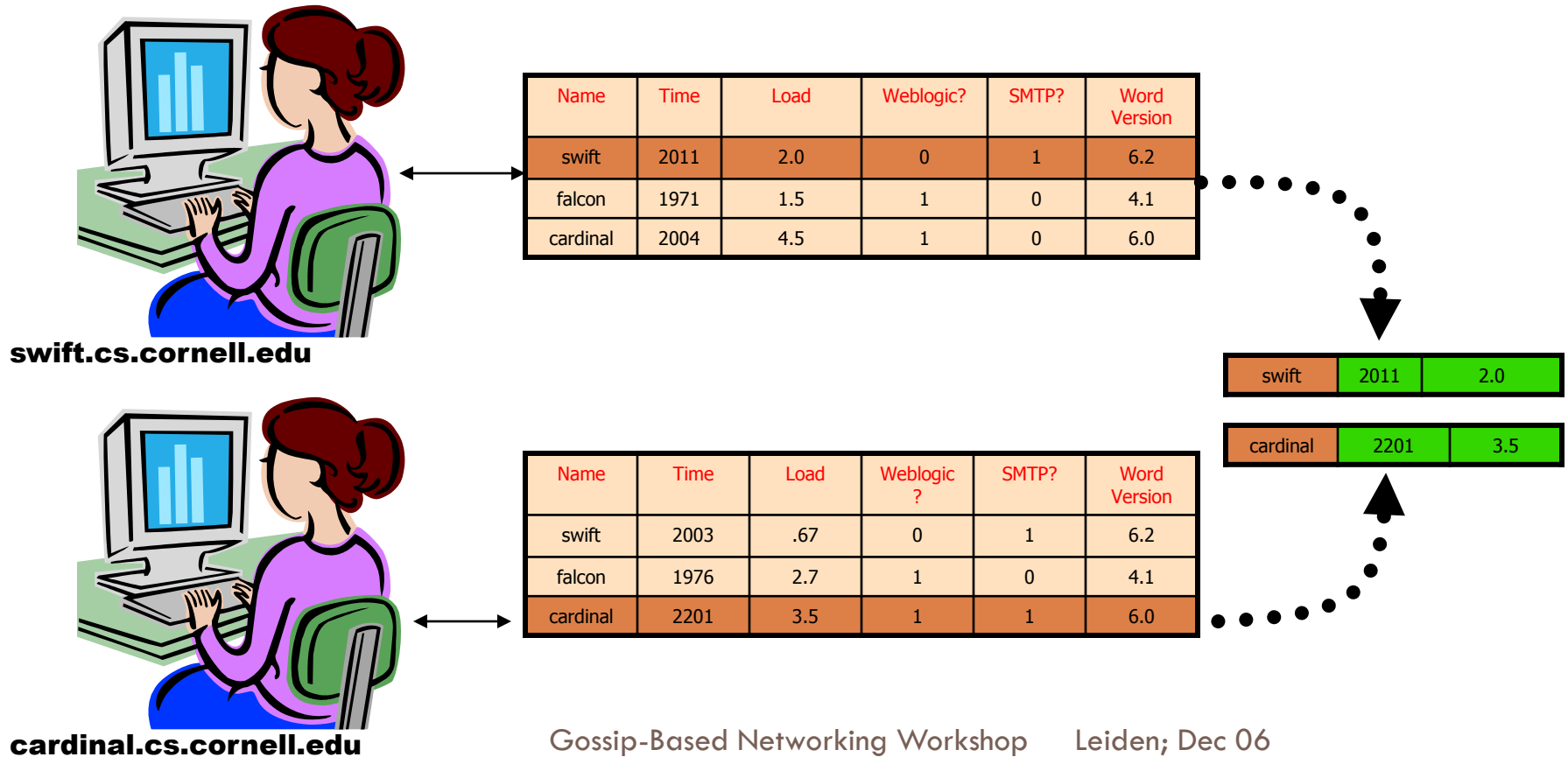
**cardinal.cs.cornell.edu**

| Name     | Time | Load | Weblogic ? | SMTP? | Word Version |
|----------|------|------|------------|-------|--------------|
| swift    | 2003 | .67  | 0          | 1     | 6.2          |
| falcon   | 1976 | 2.7  | 1          | 0     | 4.1          |
| cardinal | 2201 | 3.5  | 1          | 1     | 6.0          |



# State Merge: Core of Astrolabe epidemic

24





# State Merge: Core of Astrolabe epidemic

25



**swift.cs.cornell.edu**

| Name     | Time | Load | Weblogic? | SMTP? | Word Version |
|----------|------|------|-----------|-------|--------------|
| swift    | 2011 | 2.0  | 0         | 1     | 6.2          |
| falcon   | 1971 | 1.5  | 1         | 0     | 4.1          |
| cardinal | 2201 | 3.5  | 1         | 0     | 6.0          |



**cardinal.cs.cornell.edu**

| Name     | Time | Load | Weblogic ? | SMTP? | Word Version |
|----------|------|------|------------|-------|--------------|
| swift    | 2011 | 2.0  | 0          | 1     | 6.2          |
| falcon   | 1976 | 2.7  | 1          | 0     | 4.1          |
| cardinal | 2201 | 3.5  | 1          | 1     | 6.0          |

# Observations

26

- Merge protocol has constant cost
  - ▣ One message sent, received (on avg) per unit time.
  - ▣ The data changes slowly, so no need to run it quickly – we usually run it every five seconds or so
  - ▣ Information spreads in  $O(\log N)$  time
- But this assumes bounded region size
  - ▣ In Astrolabe, we limit them to 50-100 rows

# Big systems...

27

- A big system could have *many* regions
  - ▣ Looks like a pile of spreadsheets
  - ▣ A node only replicates data from its neighbors within its own region

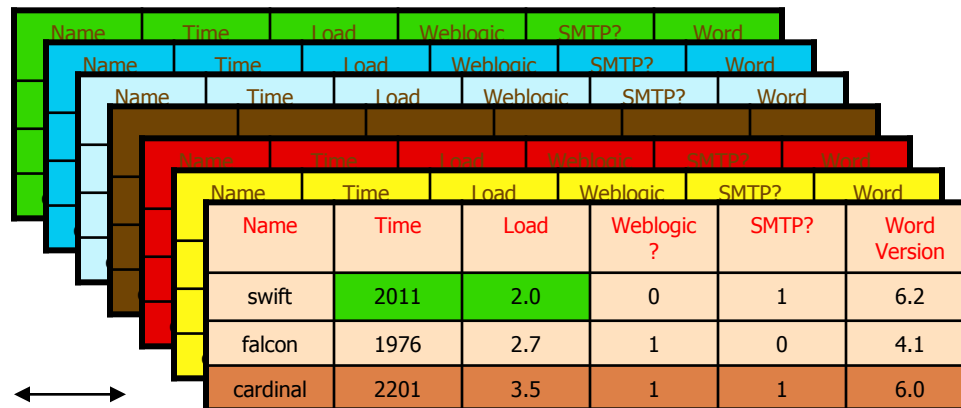
# Scaling up... and up...

28

- With a stack of domains, we don't want every system to "see" every domain
  - ▣ Cost would be huge
- So instead, we'll see a summary



cardinal.cs.cornell.edu



| Name     | Time | Load | Weblogic | SMTP? | Word |
|----------|------|------|----------|-------|------|
| swift    | 2011 | 2.0  | 0        | 1     | 6.2  |
| falcon   | 1976 | 2.7  | 1        | 0     | 4.1  |
| cardinal | 2201 | 3.5  | 1        | 1     | 6.0  |

Gossip-Based Networking Workshop Leiden; Dec 06

# Astrolabe builds a hierarchy using a P2P protocol that “assembles the puzzle” without any servers

29

Dynamically changing query output is visible system-wide



# Large scale: “fake” regions

30

- These are
  - ▣ Computed by queries that summarize a whole region as a single row
  - ▣ Gossiped in a read-only manner within a leaf region
- But who runs the gossip?
  - ▣ Each region elects “k” members to run gossip at the next level up.
  - ▣ Can play with selection criteria and “k”

# Hierarchy is virtual... data is replicated

31

Yellow leaf node "sees" its neighbors and the domains on the path to the root.

| Name | Avg Load | WL contact  | SMTP contact |
|------|----------|-------------|--------------|
| SF   | 2.6      | 123.45.61.3 | 123.45.61.17 |
| NJ   | 1.8      | 127.16.77.6 | 127.16.77.11 |

Falcon runs level 2 epidemic because it has lowest load

| Name     | Load | ... | Version |
|----------|------|-----|---------|
| swift    | 2.0  | 0   | 6.2     |
| falcon   | 1.5  | 1   | 4.1     |
| cardinal | 4.5  | 1   | 6.0     |

San Francisco

Gnu runs level 2 epidemic because it has lowest load

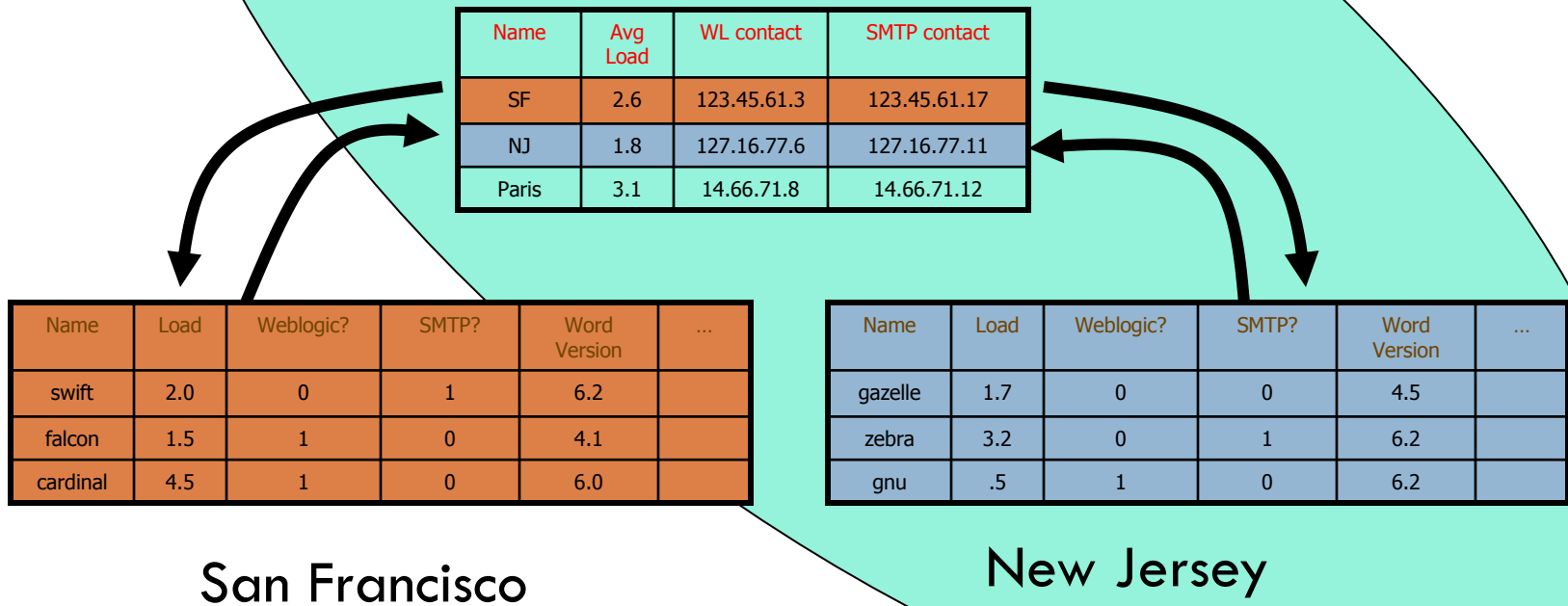
| Name    | Load | ... | SMTP? | Word Version | ... |
|---------|------|-----|-------|--------------|-----|
| gazelle | 1.7  | 0   | 0     | 4.5          |     |
| zebra   | 3.2  | 0   | 1     | 6.2          |     |
| gnu     | .5   | 1   | 0     | 6.2          |     |

New Jersey

# Hierarchy is virtual... data is replicated

32

Green node sees different leaf domain but has a consistent view of the inner domain





# Worst case load?

33

- A small number of nodes end up participating in  $O(\log_{\text{fanout}} N)$  epidemics
  - ▣ Here the fanout is something like 50
  - ▣ In each epidemic, a message is sent and received roughly every 5 seconds
- We limit message size so even during periods of turbulence, no message can become huge.

# Who uses Astrolabe?

34

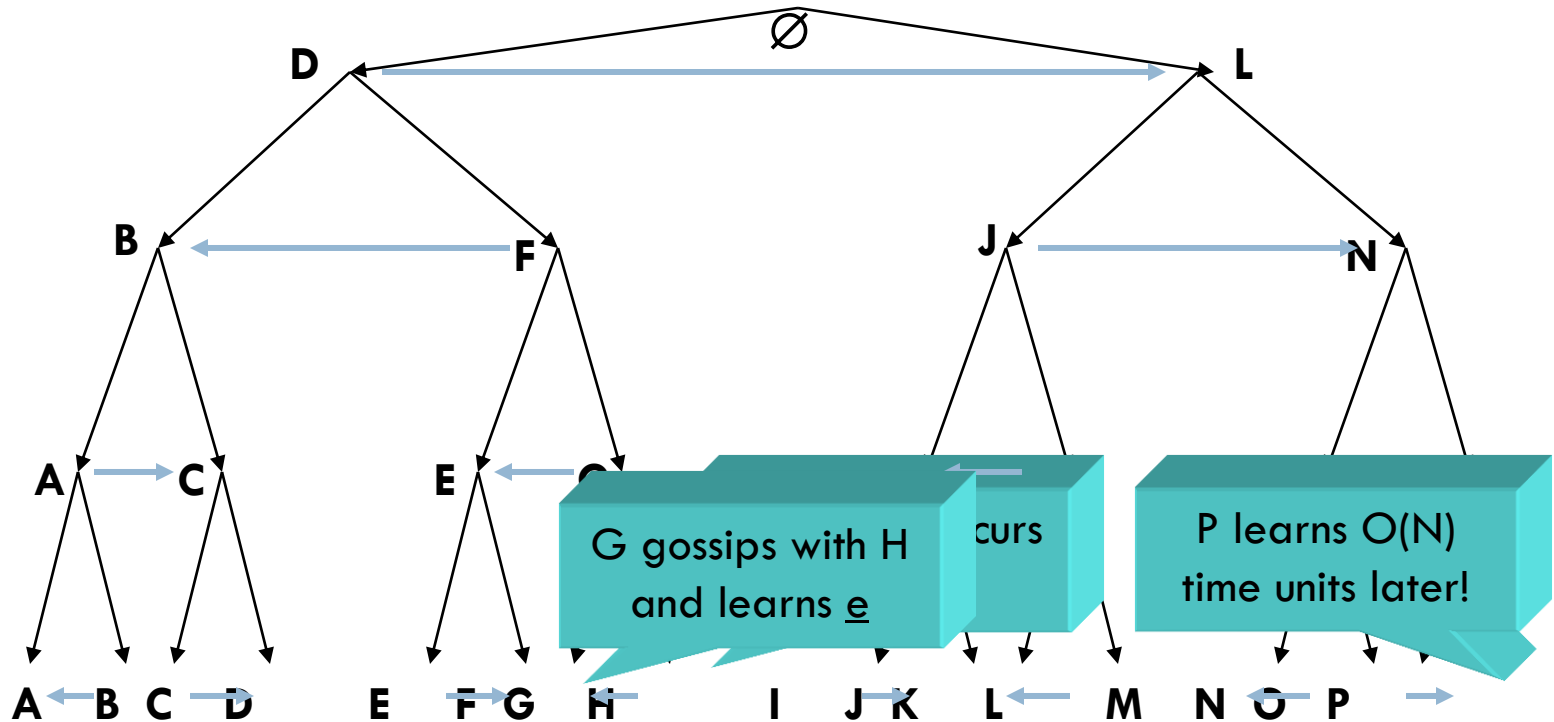
- Amazon uses Astrolabe throughout their big data centers!
  - ▣ For them, Astrolabe helps them track overall state of their system to diagnose performance issues
  - ▣ They can also use it to automate reaction to temporary overloads

# Example of overload handling

35

- Some service  $S$  is getting slow...
  - ▣ Astrolabe triggers a “system wide warning”
- Everyone sees the picture
  - ▣ “Oops,  $S$  is getting overloaded and slow!”
  - ▣ So everyone tries to reduce their frequency of requests against service  $S$
  
- What about overload in Astrolabe *itself*?
  - ▣ Could everyone do a fair share of inner aggregation?

# A fair (but dreadful) aggregation tree



# What went wrong?

37

- In this horrendous tree, each node has equal “work to do” but the information-space diameter is larger!
- Astrolabe benefits from “instant” knowledge because the epidemic at each level is run by someone elected from the level below

# Insight: Two kinds of shape

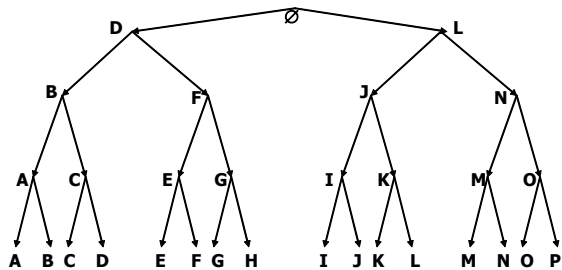
38

- We've focused on the aggregation tree
- But in fact should also think about the information flow tree

# Information space perspective

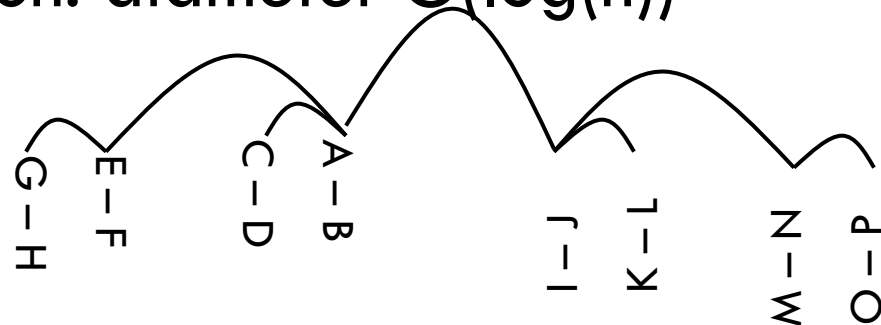
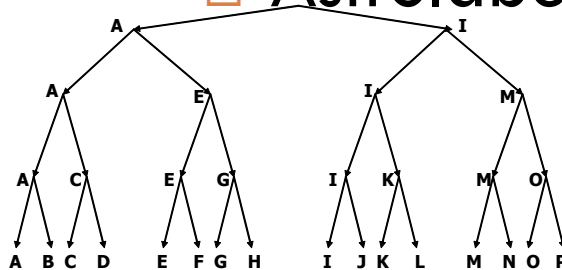
39

□ Bad aggregation graph: diameter  $O(n)$



H-G-E-F-B-A-C-D-L-K-I-J-N-M-O-P

□ Astrolabe version: diameter  $O(\log(n))$



# Summary

40

- First we saw a way of using Gossip in a reliable multicast (although the reliability is probabilistic)
- Then looked at using Gossip for aggregation
  - ▣ Pure gossip isn't ideal for this... and competes poorly with flooding and other urgent protocols
  - ▣ But Astrolabe introduces hierarchy and is an interesting option that gets used in at least one real cloud platform
- Power: make a system more robust, self-adaptive, with a technology that won't make things worse
- But performance can still be sluggish