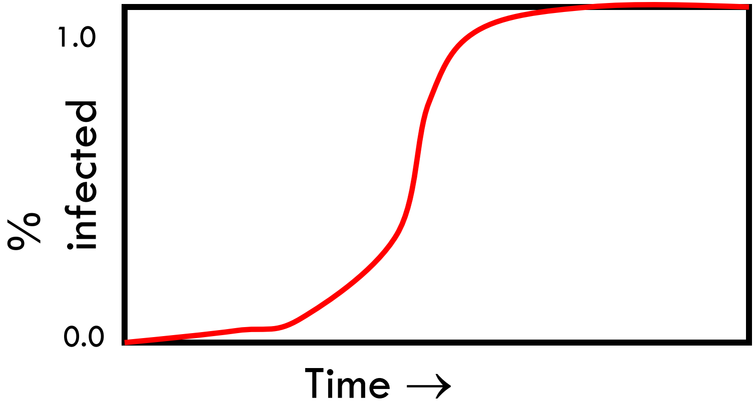
BIMODAL MULTICAST ASTROLABE

CS6410

Ken Birman

Gossip 201

- Recall from early in the semester that gossip spreads in log(system size) time
- But is this actually "fast"?



Gossip in distributed systems

- 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

- 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?

- 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

- We can use gossip!
- Every node tracks the membership of the target group (using gossip, just like with Dr. Multicast)
  - 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

- 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

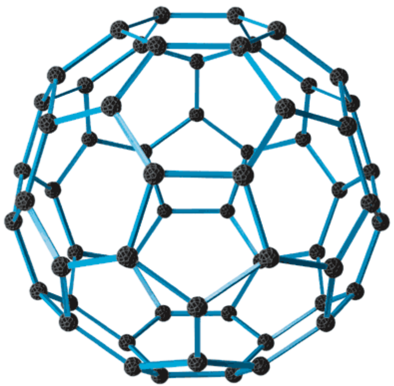
- On receiving such a gossip message, the recipient checks to see which messages it has that the gossip sender lacks, and vice versa

- 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

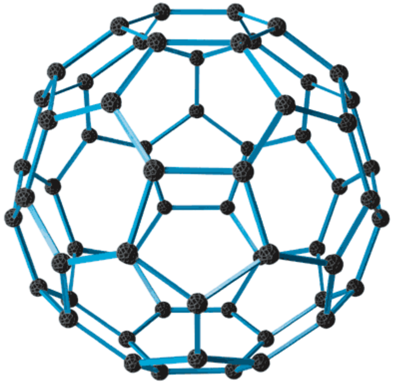
- An exchange of the actual messages follows

Two data centers… WAN link

- Interesting case that requires further optimizations



Half the traffic will
run on the WAN link!

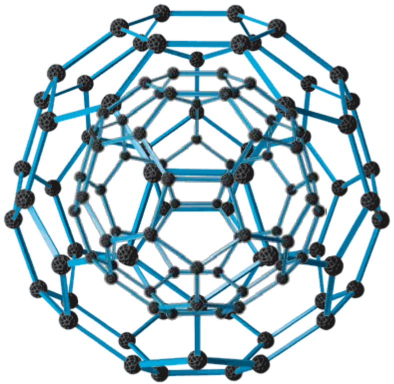


N/2 nodes

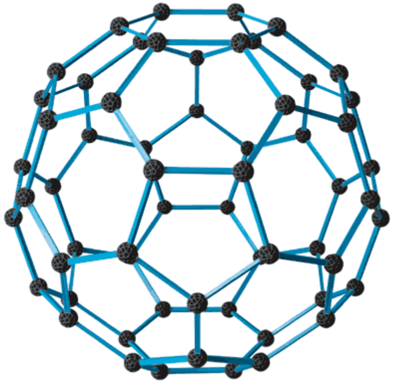
N/2 nodes

Two data centers… WAN link

- Interesting case that requires further optimizations



Now only 1/3 of A's gossip has a target in B, but 2/3's of B's gossips will have a target in A



2N/3 nodes

N/3 nodes

Why is this a problem?

- Think of a Europe/US scenario
- Local gossip is probably very fast (100us latency) but WAN gossip will be slow (125ms latency…)
- Same data will be offered again and again on the WAN connection, and while waiting for it could be pulled again and again too!

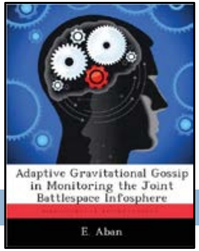
WAN optimizations…

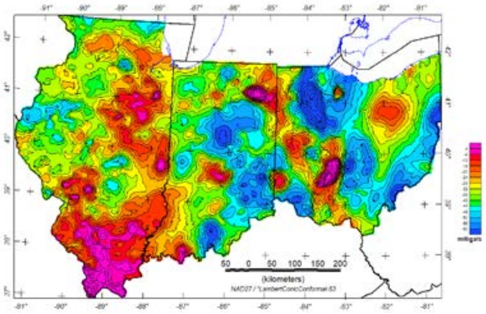
- A should treat all of B as a single destination, and B treats all of A as a single destination
- Less gossip will cross the WAN link, but this is good because the naïve approach is very redundant

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

Gravitational Gossip





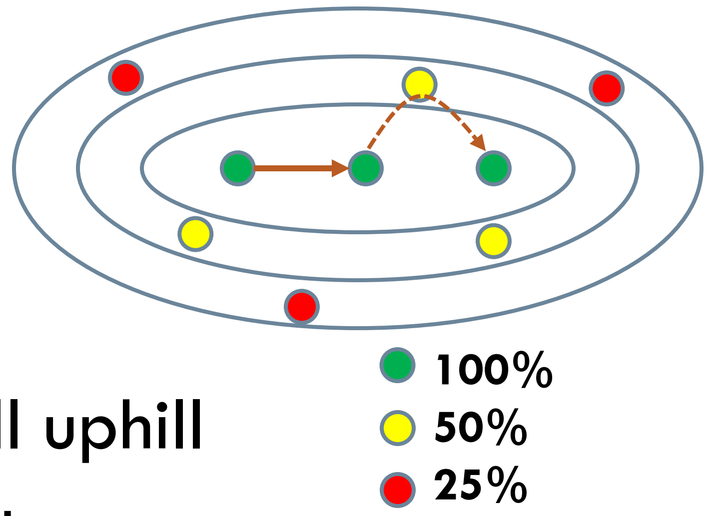
- Key idea was due to Kate Jenkins
- Assumes a large system, participants gossip on topics
  - Each subscribes with some sort of "level of interest"

    E.g. I want to see all status reports for the local power grid but a 5% sample from the neighboring power grid in Ohio
  - Trick: use gossip but shape the contents of messages to conform a "gravity well" model. Very robust and much more efficient than having one epidemic per topic. But sampling is random.

Adaptive Gravitational Gossip: A Gossip-Based Communication Protocol with User-Selectable Rates. Ken Hopkinson, Kate Jenkins, et. al. IEEE TPDS 20:12, Feb 2009.

Gravity well trick: Single topic example

- Think of each gossip item as a marble rolling in a dish
- The marble is eager to roll along the bottom of the valley or downhill into it, but not to roll uphill
- Turns out that this model gives the desired behavior (marble ≡ gossip msg)



lpbcast variation

- 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
- Lpbcast is often faster, but equally reliable!

Speculation... about speed

- 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

- 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…

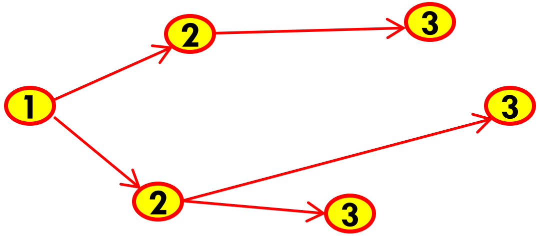
… and the answer is

- 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!

- Recall how flooding works



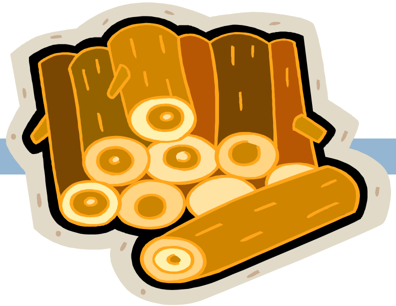
Labels: distance of the node from the root

- 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"

- 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!



- 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?

- 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"?

- 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

- 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 inroduce hierarchy into gossip systems?

Astrolabe

- 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…
- Technology was adopted at Amazon.com (but they build their own solutions rather than using it in this form)

Astrolabe is a flexible monitoring overlay

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

- 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

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

swift.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

cardinal.cs.cornell.edu

State Merge: Core of Astrolabe epidemic

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

swift	2011	2.0
cardinal	2201	3.5

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

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

swift.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

cardinal.cs.cornell.edu

Observations

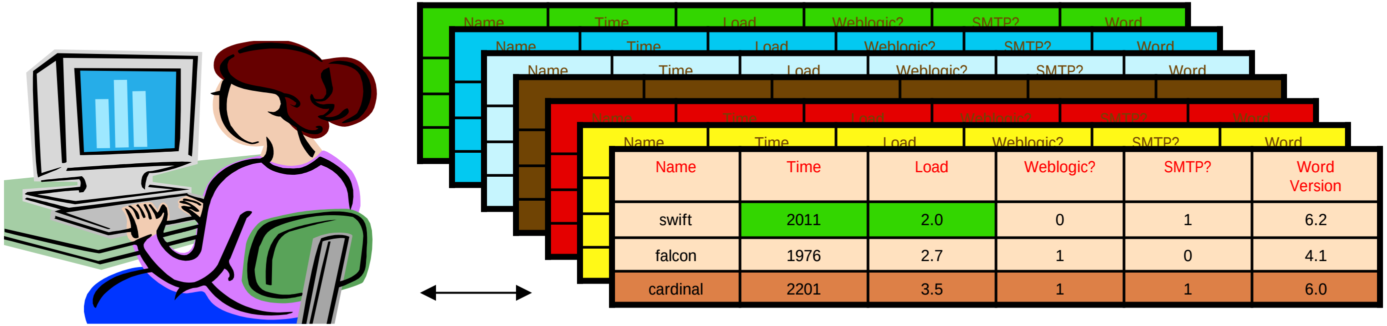
- Merge protocol has constant cost
  - One message sent, received (on avg) per unit time.
  - The data changes slowly, so no need to run it quickly – we usually run it every five seconds or so
  - Information spreads in O(log N) time
- But this assumes bounded region size
  - In Astrolabe, we limit them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

Big systems…

- 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…

- 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



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

Astrolabe builds a hierarchy using a P2P protocol that "assembles the puzzle" without any servers

Dynamically changing query output is visible system-wide

Name	Avg Load	WL contact	SMTP contact
SF	2.2	123.45.61.3	123.45.61.17
NJ	1.6	127.16.77.6	127.16.77.11
Paris	2.7	14.66.71.8	14.66.71.12

SQL query "summarizes" data

Name	Load	Weblogic?	SMTP?	Word Version	...
swift	1.7	0	1	6.2	
falcon	2.1	1	0	4.1	
cardinal	3.9	1	0	6.0	

San Francisco

Name	Load	Weblogic?	SMTP?	Word Version	...
gazelle	4.1	0	0	4.5	
zebra	0.9	0	1	6.2	
gnu	2.2	1	0	6.2	

New Jersey

Large scale: “fake” regions

- 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

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
[illegible]	[illegible]	[illegible]	[illegible]1.12

Falcon runs level 2 epidemic because it has lowest load

Gnu runs level 2 epidemic because it has lowest load

Name	Load	[illegible]	[illegible]	[illegible]	
swift	2.0	[illegible]	1	6.2	
falcon	1.5	1	0	4.1	
cardinal	4.5	1	0	6.0	

San Francisco

[illegible]	Load	[illegible]	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

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

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
Paris	3.1	14.66.71.8	14.66.71.12

Name	Load	Weblogic?	SMTP?	Word Version	…
swift	2.0	0	1	6.2	
falcon	1.5	1	0	4.1	
cardinal	4.5	1	0	6.0	

San Francisco

Name	Load	Weblogic?	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

Worst case load?

- A small number of nodes end up participating in $O(\log_{fanout} N)$ 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?

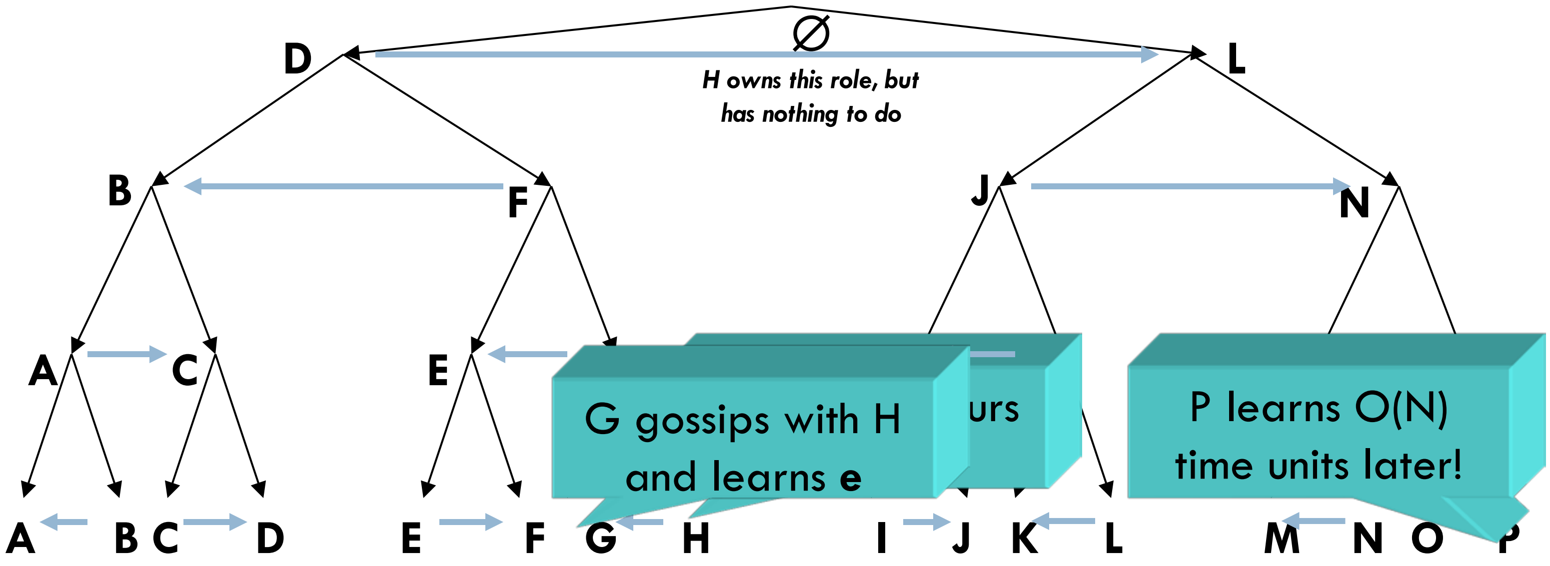
- Amazon uses Astrolabe throughout their big data centers!
  - For them, Astrolabe helps them track overall state of their system to diagnose performance issues
  - They can also use it to automate reaction to temporary overloads

Example of overload handling

- 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?

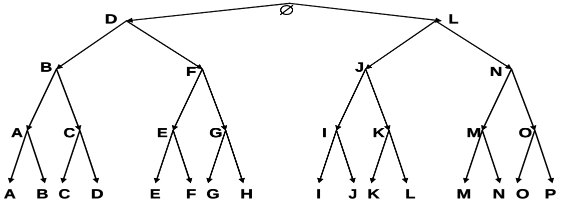
- In this horrendous tree, each node has equal "work to do" but the information-space diameter is larger!
  - In fact, Astrolabe needs an "expander graph"
  - But this tree is not an expander
- Astrolabe normally benefits from "instant" knowledge because the epidemic at each level is run by someone elected from the level below

Insight: Two kinds of shape

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

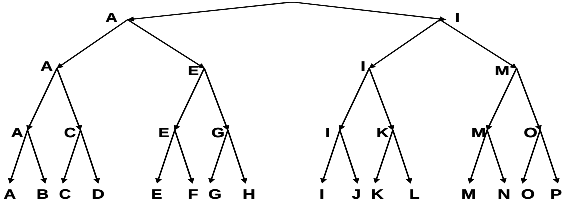
Information space perspective

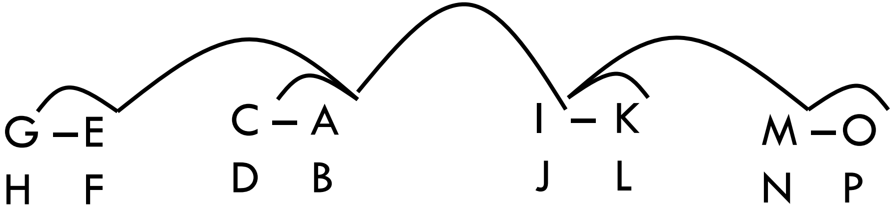
- Bad aggregation graph: Not an expander, diameter O(n)



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

- Astrolabe version: Expander, hence diameter (log(n))





Summary

- 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