



CS5412/LECTURE 12

GOSSIP PROTOCOLS

Ken Birman
CS5412 Spring 2019

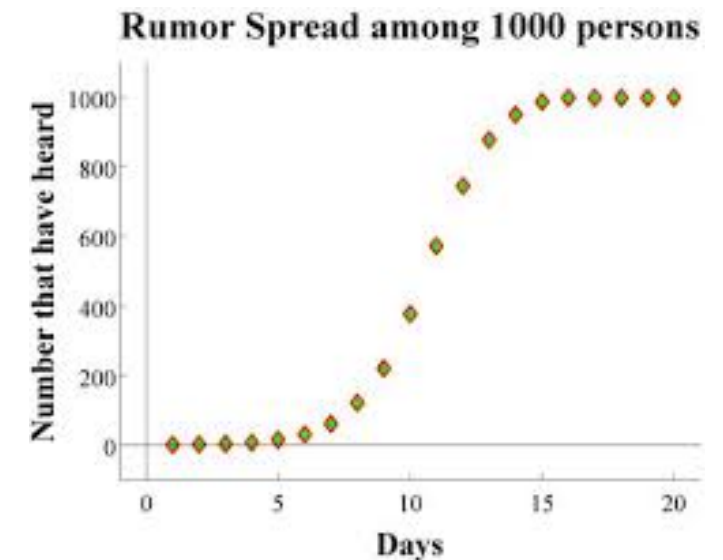
GOSSIP 101



Gossip protocols: Ones in which information is spread node-to-node at random, like a Zombie virus.

At first, the rate of spread doubles on each round of gossip.

Eventually, a lot of “already infected” events slow the spread down.



KEY ASPECTS TO THE CONCEPT

Participants have a membership list, or some random subset of it.

They pick some other participant at random, once every T time units.

Then the two interact to share data: {

- Push: A “tells” B some rumors*
- Pull: A “asks” B for news*
- Push-Pull: Both*

The messages are of fixed maximum size.

NOTICE THAT GOSSIP HAS FIXED PEAK LOAD!

Every process sends and receives at the same fixed rate (due to random peer selection, some processes might receive 2 messages in time period T , but very few receive 3 or more... the “birthday paradox”)

And at most, we fill those messages to the limit with rumors, but then they max out and nothing more can be added.

So gossip is very predictable. System managers like this aspect.

GOSSIP SPREADS SLOWLY AT FIRST, THEN FASTER

$\log(N)$ tells us how many rounds (each taking T time units) to anticipate

- With $N=100,000$, $\log(N)$ would be 12
- So with one gossip round per five seconds, information would need 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

EASY TO WORK WITH

A recent Cornell student created a framework for Gossip applications, called the [MICA system](#) (Microprotocol Composition Architecture)

You take a skeleton, add a few lines of logic to tell it how to merge states (incoming gossip), and MICA runs the resulting application for you. Plus, it supports a modular, “compositional” coding style.

Use cases were mostly focused on large-scale system management.

BIMODAL MULTICAST

This uses gossip to send a message from one source to many receivers. It combines gossip with a feature called IP multicast: an unreliable 1-to-many UDP option available on optical Ethernet

In Bimodal Multicast, the first step is to send a message using IP multicast.

- Not reliable, and we don't add acks or retransmissions
- No flow control (but it does support a rate limiting feature)
- In data centers that lack IP multicast, can simulate by sending UDP packets 1:1. Again, these use UDP 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! The “rumors” will be the IP multicast messages!

Every node tracks the membership of the target group (using gossip)

Then after doing the IP multicast, “fill in the holes” (missed messages).

MAKING BIMODAL MULTICAST RELIABLE

So, 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 of messages received, perhaps in a compressed format
- Node A might send node B
 1. I have messages 1-18 from sender X
 2. I have message 11 from sender Y
 3. I have messages 14, 16 and 22-71 from sender Z
- 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'' (which you seem to lack)
- I would like a copy of messages N , N' and N''

An exchange of the actual messages follows

THIS MAKES IT “BIMODAL”

Count of nodes
reached after
this delay



There is a first wave of message delivery from the IP multicast, which takes a few milliseconds to reach every node in the whole data center.

But a few miss the message.

Then a second wave of gossip follows, filling in the gaps, but this takes a few rounds, so we see a delay of $T*2$ or $T*3$ while this plays out.

EXPERIMENTAL FINDINGS

Bimodal multicasts works best if the initial IP multicast reaches almost every process, and “usually” this is so.

But “sometimes” a lot of loss occurs. In those cases, N (the number of receivers missing the message) is much larger.

Then the second “mode” (second bump in the curve) is large and slow.

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

With these changes, “almost all” receivers will get the message via IP multicast, so N is small and gossip fills gaps within just 2 or 3 rounds.

LPBCAST VARIATION (KERMARREC, GUERRAOUI)

In this variation on Bimodal Multicast instead of gossiping with every node in a system, the protocol:

- Maintains a “peer overlay”: each member tracks two sets of neighbors.
- First set: peers picked to be reachable with low round-trip times.
- Second set: peers picked to ensure that the graph is an “expander” graph.
- 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
- Even the IP multicast acceleration idea just adds an unacknowledged IP multicast message or two, per Bimodal Multicast sent.
- This is appealing at large scales

How can we generalize this concept?

ASTROLABE

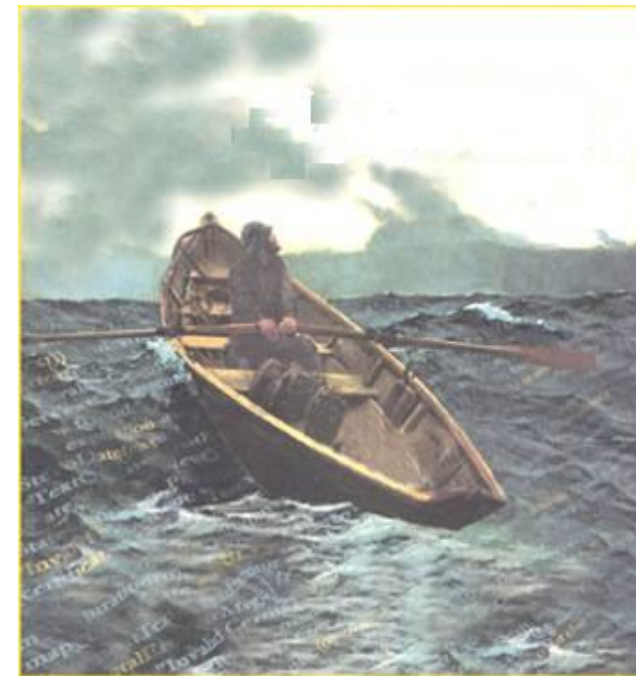
Help for applications adrift in a sea of information

Structure emerges from a randomized gossip protocol

This approach is robust and scalable even under stress that cripples traditional systems

Initially developed by a team led by Robbert van Renesse.

Technology was adopted at Amazon.com (but they rebuild it over time)



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



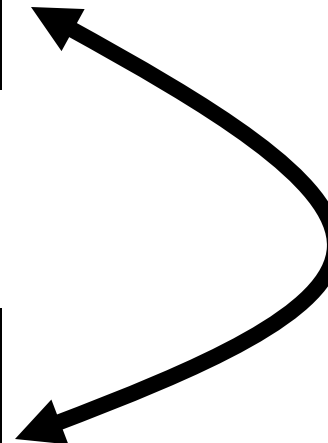
swift.cs.cornell.edu

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

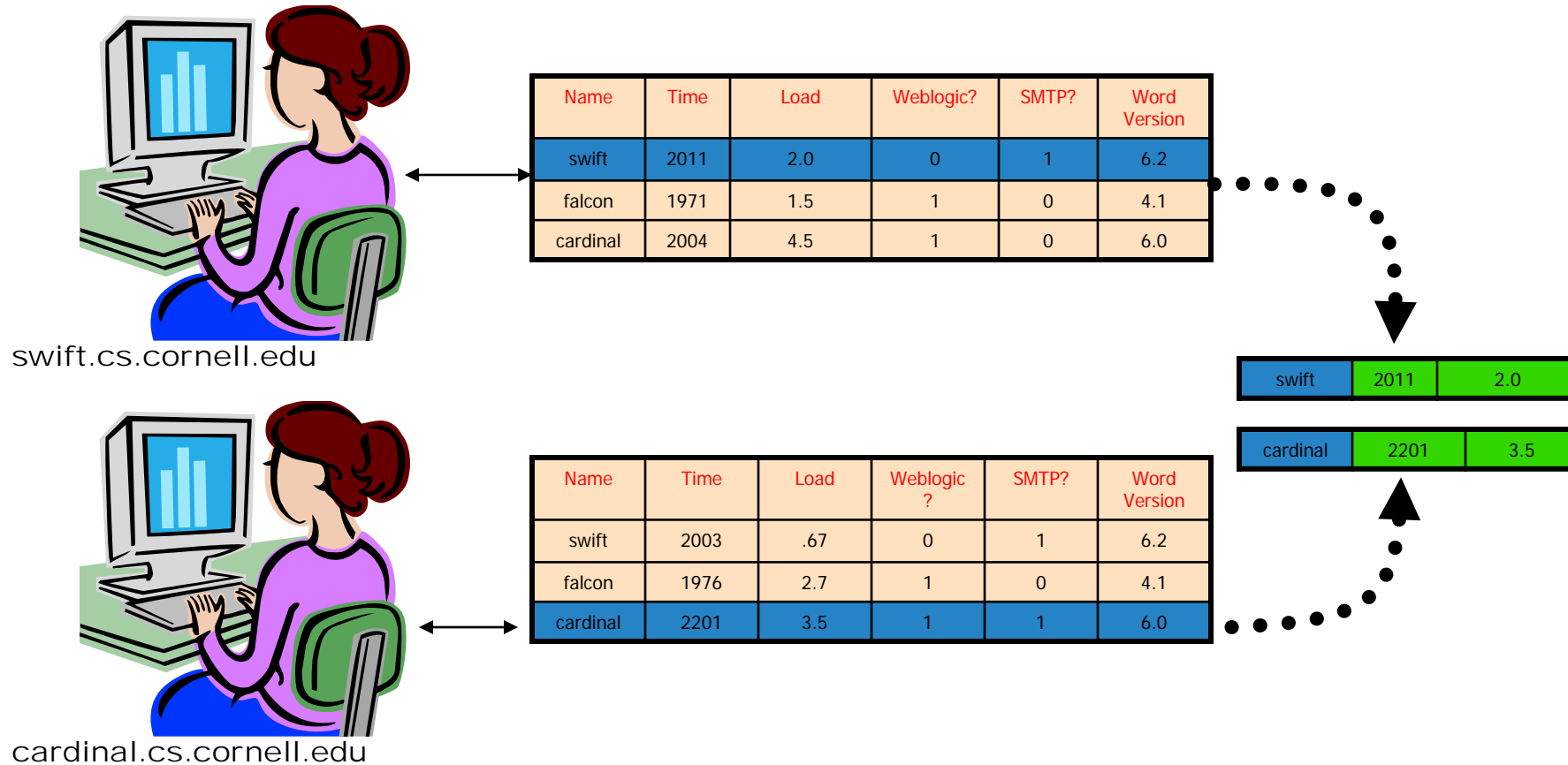


cardinal.cs.cornell.edu

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



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	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

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...

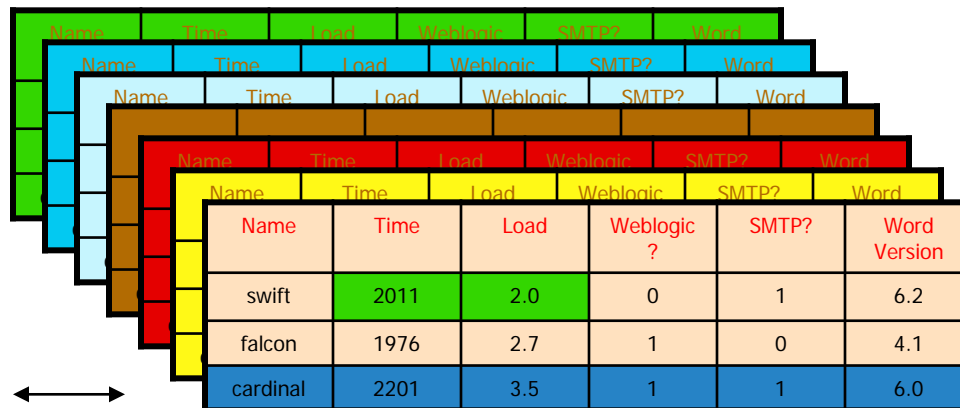
With a stack of domains, we don't want every system to "see" every domain

- Cost would be huge

So instead, we'll see a summary



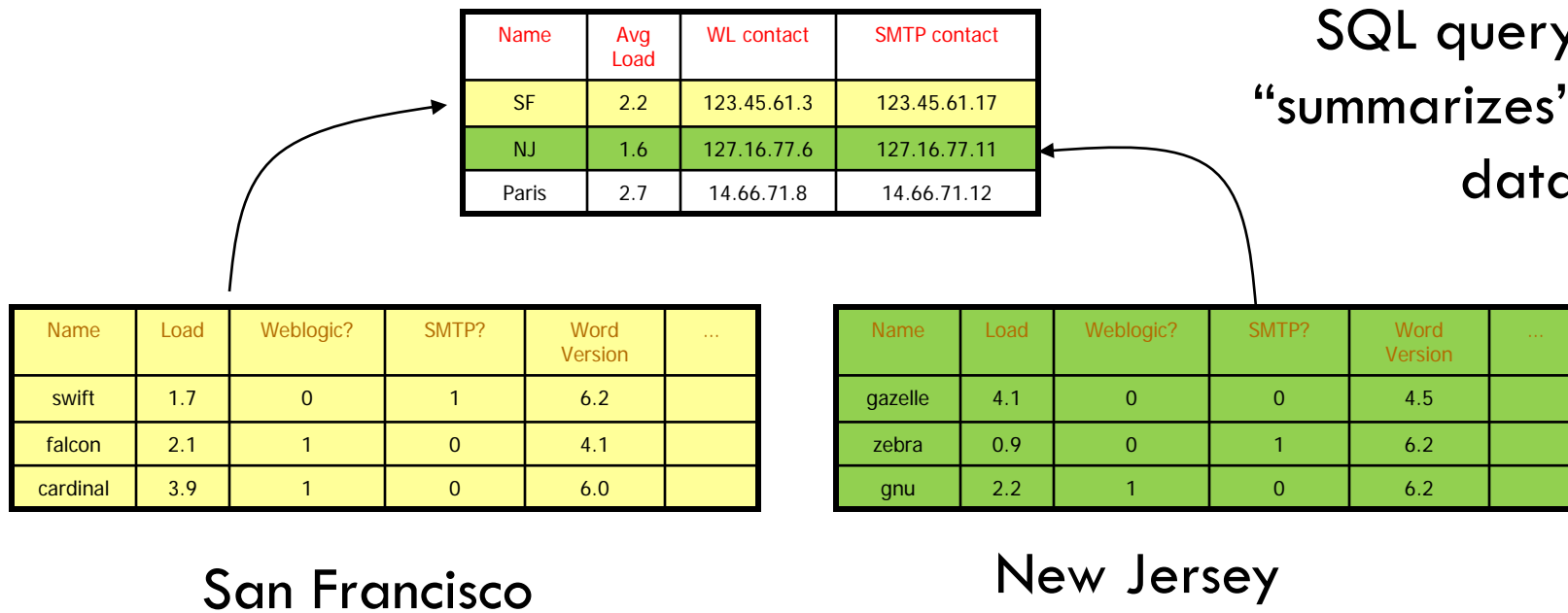
cardinal.cs.cornell.edu



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

ASTROLABE BUILDS A HIERARCHY USING A P2P PROTOCOL THAT “ASSEMBLES THE PUZZLE” WITHOUT ANY SERVERS

Dynamically changing query output is visible system-wide



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 REDPLICATED

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

Gnu 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

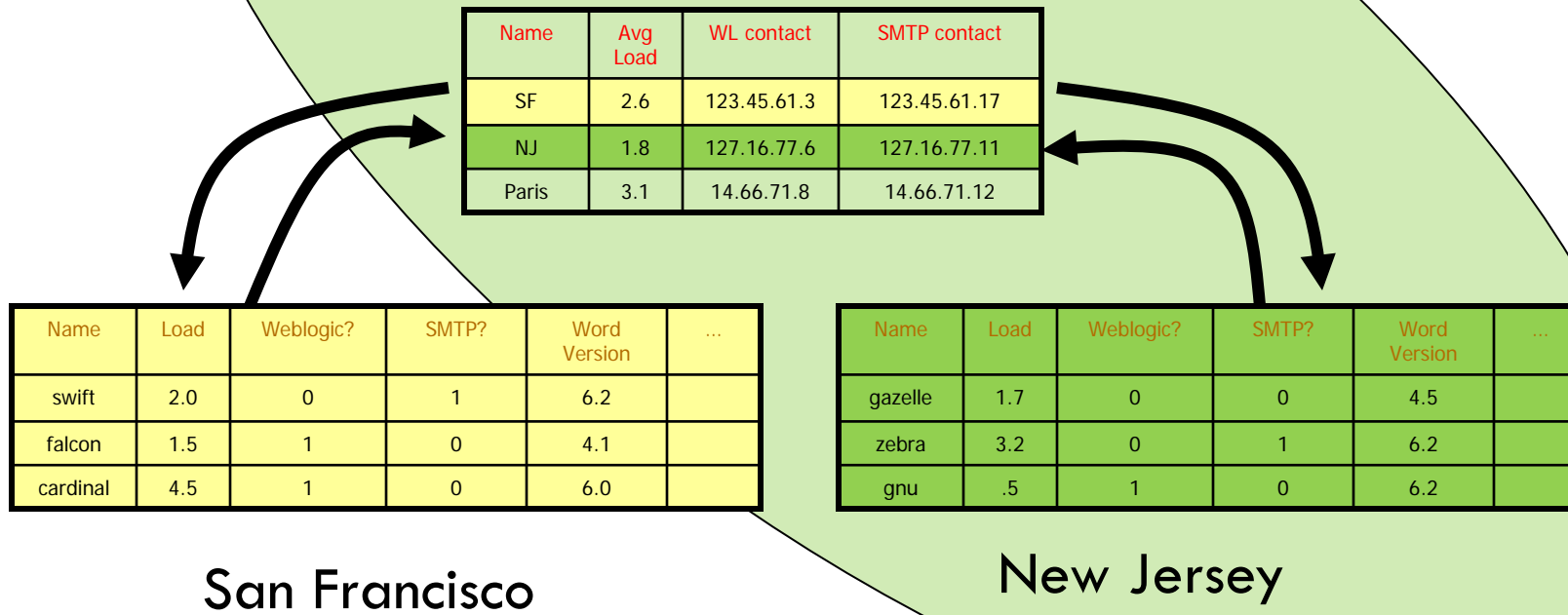
Name	Load	...	SMTP?	Word Version	...
gazelle	1.7	0	0	4.5	
zebra	3.2	0	1	6.2	
gnu	.5	1	0	6.2	

San Francisco

New Jersey

HIERARCHY IS VIRTUAL... DATA IS REDPLICATED

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



WORST-CASE LOAD?

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 5s

We limit message size so even during periods of turbulence, no message can become huge.

WHO USES ASTROLABE?

Amazon doesn't use Astrolabe in this identical form, but they built gossip-based monitoring systems based on the same ideas.

They deploy these in their big data centers!

- Astrolabe-like mechanisms track overall state to diagnose issues
- They also 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?

IDEA THAT ONE COMPANY HAD

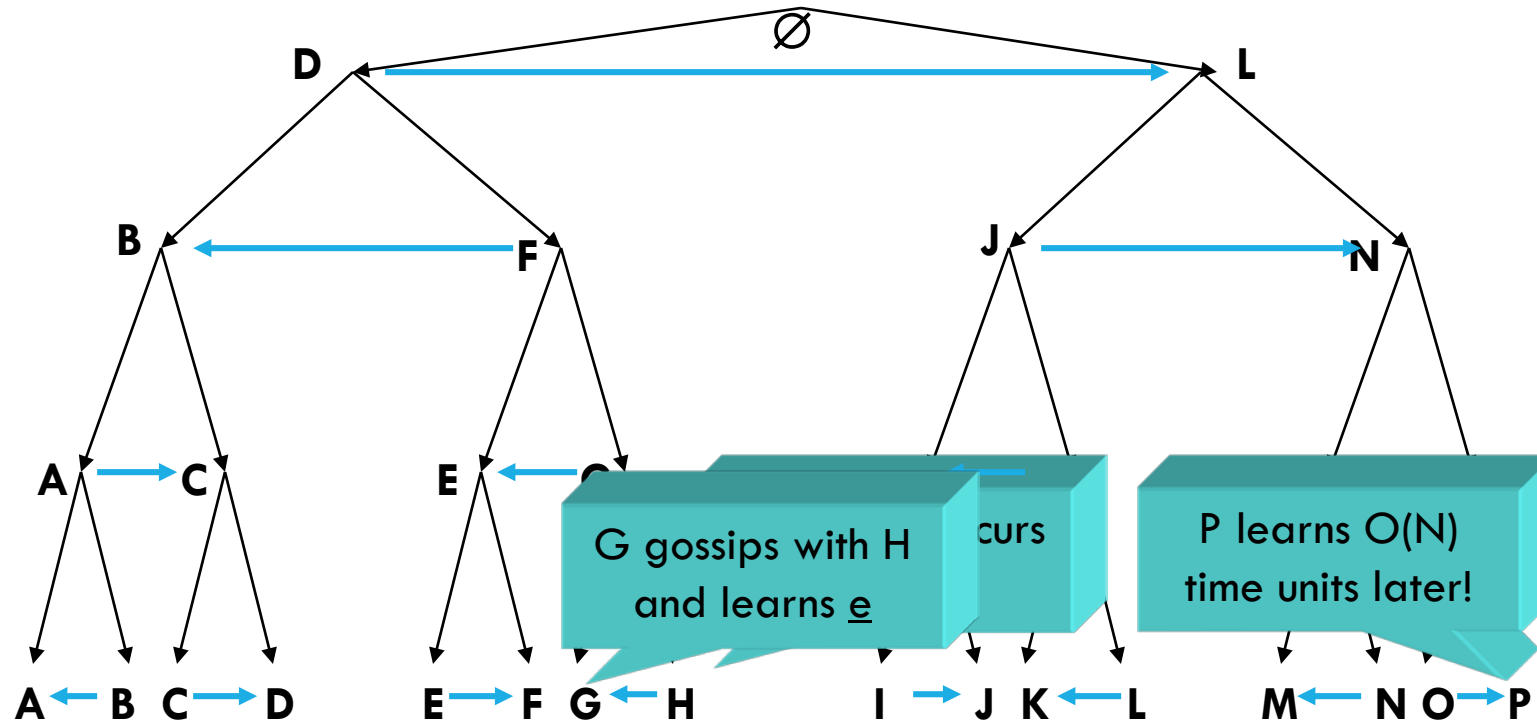
Start with the normal Astrolabe approach

But instead of electing nodes to play inner roles, assign them roles, left to right

$N-1$ inner nodes play two roles: aggregation and “be a leaf node”.

What impact will this have on Astrolabe?

WORLD'S WORST AGGREGATION TREE!



WHAT WENT WRONG?

Each node does equal “work” but information spreads very slowly – $O(N)$

In a normal configuration Astrolabe benefits from “instant” knowledge because the epidemic at each level is run by someone elected from the level below. This short-circuits the path and speeds the spread of gossip.

In the modified configuration, those short-circuit steps no longer occur.

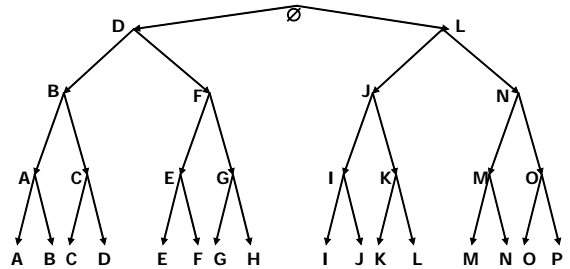
INSIGHT: TWO KINDS OF SHAPE

We've focused on the aggregation tree

But in fact should also think about the information flow tree

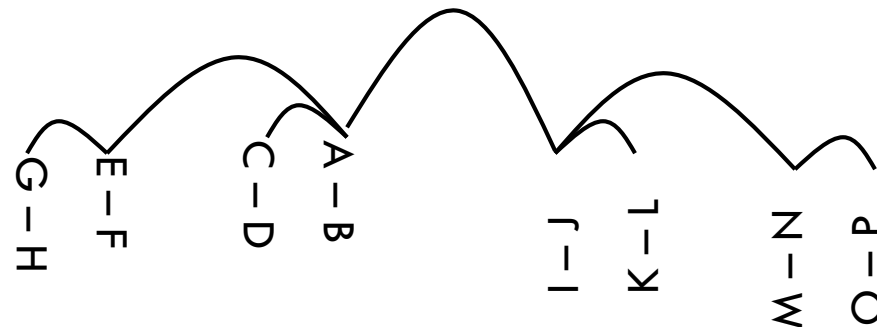
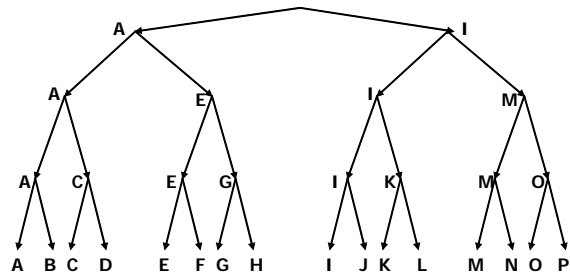
INFORMATION SPACE PERSPECTIVE

Bad aggregation graph: worst-case diameter N



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

Astrolabe version: expander graph, with diameter $\log(N)$



IS GOSSIP USED MUCH TODAY?

The basic gossip method remains very valuable for systems that need to do some form of steady-state tracking of loads, available space on servers, etc. Bimodal Multicast is widely cited and probably also used.

The predictable steady loads and the guarantee of freedom from load spikes and instabilities are valuable.

But Astrolabe's hierarchical structure is viewed as more of a cool teaching idea and is not used in real systems (as far as Ken knows).

FAMOUS TALE OF WOE

I'm full

I'm half full with
room for 26 gallons

I'm overfull with 4,294,967,093
gallons free space



Amazon's S3 system (cloud storage) uses gossip to track available space.

But one file system become “overfull” and reported -53 blocks of space. Amazon's system was using unsigned numbers for these reports.

Unfortunately, -53 is $0xFFFFFFFF35 = (\text{unsigned}) 4,294,967,093\dots$ And worse, Amazon couldn't “purge” this bad number from their gossip system!

SWARM COMPUTING



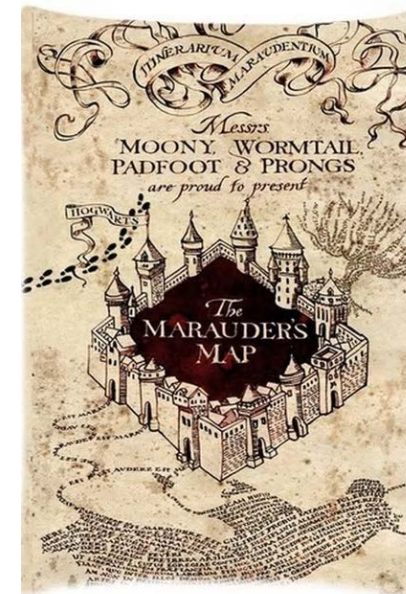
One use case that looked promising seems to have failed:

- Swarm-style computing for small devices, robotics
- Convoy-style communication for self-driving cars.

The concept and potential value should be obvious.

But they failed because we lack suitable hardware for quick connection establishment and then rapidly exchanging packets.

MILITARY AD-HOC NETWORKS



Ad-hoc networks for soldiers on a mission, or first-responders.

The exchange of gossip can populate a map showing friendly forces, hostiles, what was searched, what still needs to be searched, etc.

Downside: WiFi devices emitted signals that can be seen with night-vision scopes or similar hardware.

CONCLUSIONS?

Gossip is a mature and effective technique.

It works very well for robustly propagating system monitoring information at constant load and with guarantees that the load won't spike.

Overcomes even extreme conditions. But slow, and if misused, is as capable of malfunctioning as any other protocol!