CS5412 / LECTURE 13: THE <u>DANGERS</u> OF GOSSIP

Ken Birman

Spring, 2022

REMINDER: GOSSIP

When run at a steady rate, these protocols consume a fixed amount of background overhead. There can be load surges if participants are Byzantine, or if they use techniques like the Bimodal Multicast idea

In normal use, costs are quite low, like "on average, one message sent and one received per process, per second". Message sizes are generally small.

Moreover, information spreads in time r * O(log N), where r is the gossip rate. For many purposes this actually very reasonable.

SO WHY NOT USE GOSSIP "EVERYWHERE"?

There are many tasks where the fit seems quite good, like blockchain.

In a cloud datacenter, gossip is appealing for checking to see if systems have hung processes, monitoring loads, or tracking storage capacity.

The underlying values change slowly, so even a "slow" tracker will still be pretty accurate.

BUT THERE ARE SOME CAUTIONARY TALES

For example, gossip once caused all of Amazon S3 to crash!

This nearly resulted in a congressional inquiry! When S3 crashes, a great many companies also freeze up – any company that depends on the cloud depends on the S3 file system storage solution.

So… what is S3 and how does it use gossip?

AMAZON S3: THE "SIMPLE STORAGE SERVER"

S3 is a huge pool of storage nodes.

Plus, a "meta-data" server that keeps track of file names and where they can be found.

To store data, an application asks the meta-data service to allocate space, then sends the data to the appropriate storage servers.

WHY DO WE USE THE TERM "META-DATA"?

When you think about a file, you tend to think of the file name and the file contents. Like a key and a value.

But in fact files also have owners, permissions, create time, last access time, length (and perhaps, size on disk, which can be much smaller), etc.

We associate this data with the file. In Linux the inode plays this role. In S3 and other big-data systems, the meta-data service does it.

HOW DOES THE S3 META-DATA SERVICE TRACK SPACE AVAILABLE ON STORAGE UNITS?

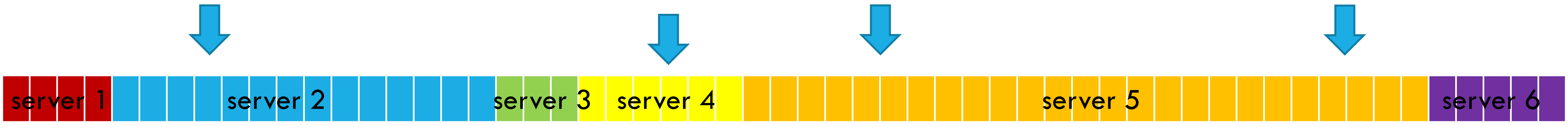
You might expect this to be easy, because the meta-data service does the allocations.

But in fact the meta-data service itself is sharded, so any single shard within it only knows (for sure) about files it is responsible for. Additionally, sometimes a server needs to take some storage offline.

To know the full state of the full S3 deployment we would need to sum across all meta-data services.

LOAD BALANCING

For each server, use gossip to track an estimate of the current amount of free space. Line them up on a “space available” line.



For a new allocation, pick a random spot in this line. This spreads the incoming load around but will be biased to favor servers with more space.

GOSSIP IS USED FOR TRACKING STORAGE

Amazon used a gossip protocol in this role, specialized to S3 meta-data.

The basic idea is to use gossip to keep track of how much space each S3 storage node is reporting that it has available.

This is inexpensive and because each storage unit holds hundreds of gigabytes, the values don’t change rapidly. A good match for gossip.

… UNTIL IT WENT WRONG!

Once upon a time, when S3 was working perfectly well, a storage server needed to take some storage offline.

Because of doing this, it suddenly went from having 10% excess capacity to being slightly over-full. This was not a bug – the servers actually have a tiny bit of reserve space, so "available capacity" **could become slightly negative.** The idea was that meta-data managers would omit that server from the line.

To support this, space available used **signed** 32 bit integers. But the S3 meta-data service declared the field to be a 32 bit **unsigned** integer.

SIGNED-TO-UNSIGNED CONVERSION IS A BUG

In older C programs and some other weakly typed languages, storing a signed value into an unsigned variable isn't flagged as an error.

C++ and Rust are examples of languages that DO complain about this.

Amazon was using C at that time. The compiler didn't complain. And in fact their servers had so much capacity that for a long time, none ever actually went into "overload" in any case.

"I HAVE -3 GB OF FREE CAPACITY"

But then one day, we did overloaded. What happens if a signed integer becomes negative, and then we interpret it as an unsigned integer?

The sign bit will be set. 2^31 is a large number!

In effect, small negative numbers will suddenly be interpreted as big positive numbers. Our server suddenly reports: *"I have 2147483645 gigabytes of free capacity!"*

SUDDENLY LOTS OF NEW FILES WERE SENT TO THIS STORAGE SERVER!

Since it was full, it refused the requests.

S3 did have logic to handle that situation. But it became a bottleneck.

S3 became … extremely . . . slow

IMAGINE THE SITUATION FOR S3 PRODUCT OWNERS AT AMAZON

One evening you are home with your family for Thanksgiving (pre-covid)

You get a call… its Jeff Bezos…

The AWS system is broken! Could you please go figure out why and fix it? So while everyone else is carving the turkey you log in… and see *millions* of errors being logged per second from 75 subsystems

IT TOOK AMAZON NEARLY A DAY TO FIGURE THIS OUT

S3 was actually working! It did store new files. But it was weirdly slow.

Higher level applications that depend on S3 began to have request timeouts, causing a cascade of failures. *Every AWS product was broken.*

This issue of one failure triggering other failures is a major problem see in the cloud and causes a whole series of outages all to happen at once.

FROM BAD… TO WORSE?

They eventually found the issue, and came up with a great idea!

They shut down the bad server. But nothing happened… Gossip is very slow to spread the word.

So then they noticed that meta-data server md1 was gossiping that server 53 had infinite space. They killed it. Suddenly, md2 took over and started gossiping that 53 had infinite space…

ISSUES YOU SEE IN THIS STORY

With gossip, fresher data might not always spread faster than stale data

Gossip is very robust to servers being down, which means that just rebooting a single node won't fix anything.

Pushing an urgent patch didn't help either: many computers were in a thrashing state and some of those would wake up after a random amount of time. They were still gossiping these huge "free space" numbers.

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 a service that actively tracks the nodes, or the load, etc?

… 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 run in time O(log N) and generally give probabilistic solutions

LESSON LEARNED?

In retrospect, many mistakes were made!

- Use of a weakly typed language, C
- Poor communication about a feature (using a negative number to report that a server is over capacity), so some people didn't know about it
- Poor testing of the combined elements (this bug should have been seen before it was put into service)
- It isn't even completely obvious that this design was the best way to solve their actual S3 storage balancing task

NOW… AN ISSUE WITH ASTROLABE

It creates a virtual tree of nodes.

At the leaf level, the tree tracks status for individual machines.

At the inner levels (these are “virtual” tables) aggregation queries are computed from the lower levels and shared. Lightly loaded leaf nodes run the inner-level gossip protocol

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

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

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

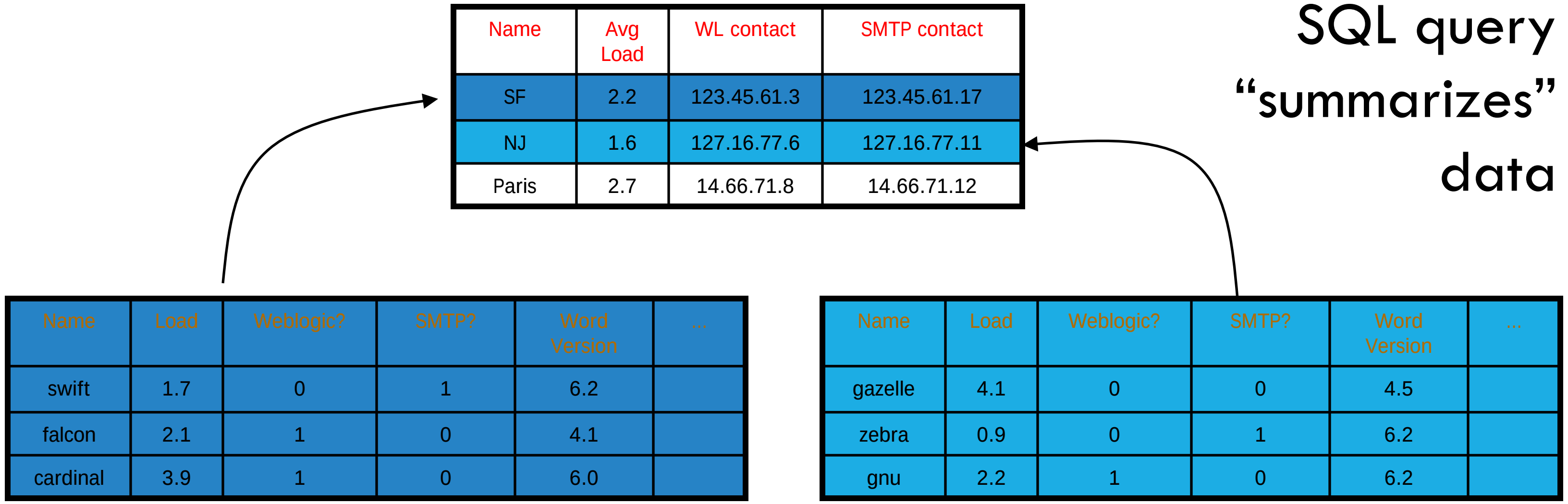
cardinal.cs.cornell.edu

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

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

Dynamically changing query output is visible system-wide

SQL query "summarizes" data



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

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

ANOTHER REALLY BAD STORY…

A company experimented with using Astrolabe

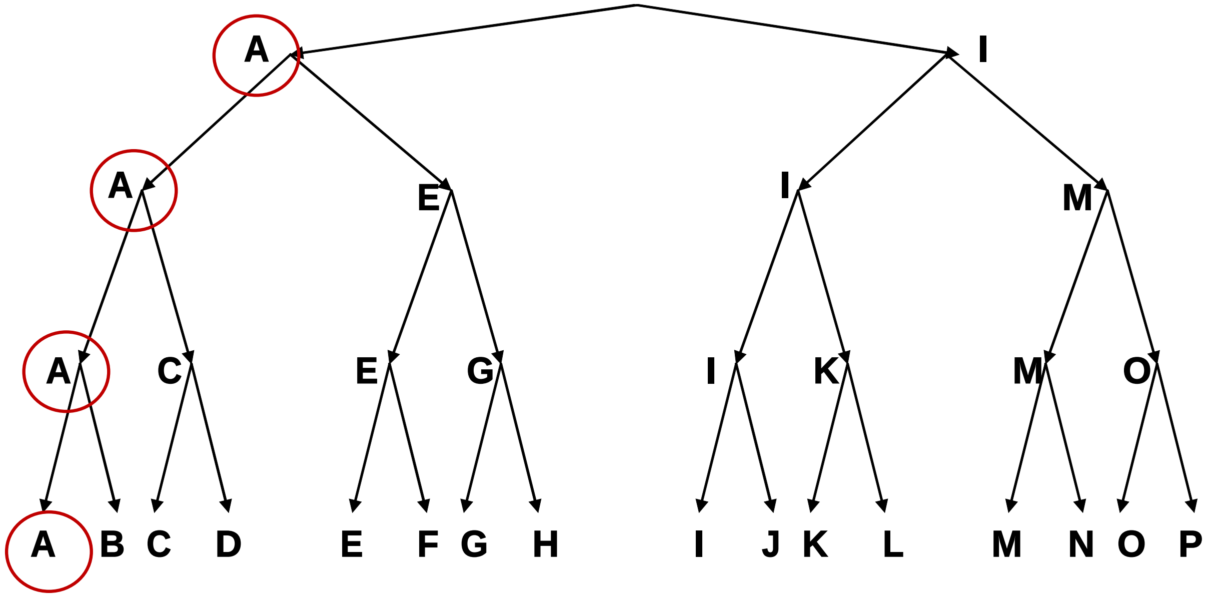
In their experiment, which was never deployed in practice, they had the idea that instead of the least loaded leaf nodes playing the inner gossip role, every node would have an equal role.

So they came up with a new kind of Astrolabe tree

A NORMAL AGGREGATION TREE

In this tree, the lowest level has fanout of 2, whereas Astrolabe used 100. But this is still fine.

Notice that node A is elected to gossip at several levels of the hierarchy



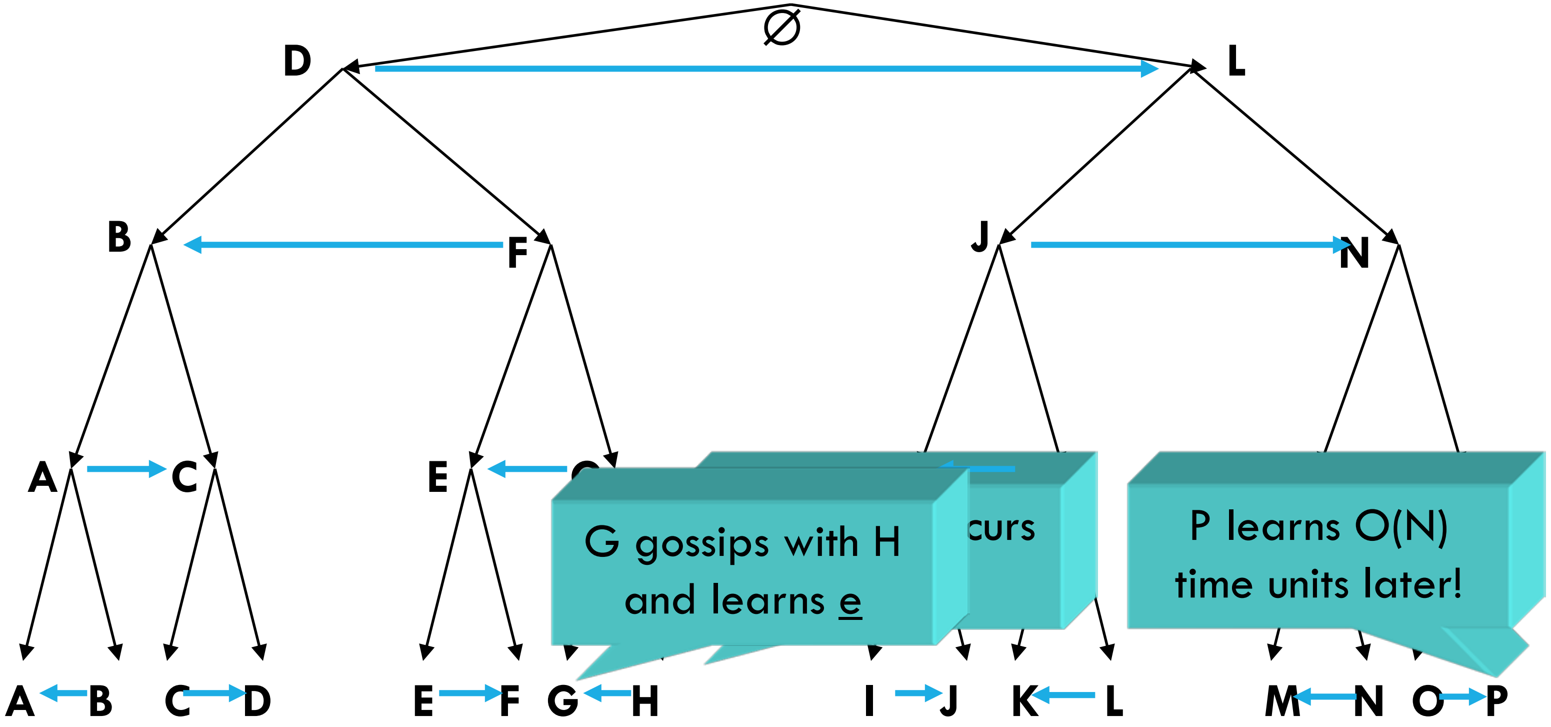
DEPLOYMENT TEAM ASKS… IS THIS “FAIR”?

When a company acquires a technology they often redesign some aspects

In this particular scenario, the new owners new that Astrolabe was kind of slow (due to gossip) but had the idea that maybe a more even gossip role sharing would help.

So they went with a different approach

A DIFFERENT AGGREGATION TREE


∅
D
L
B
F
J
N
A
C
E
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
G gossips with H and learns e
curs
P learns O(N) time units later!

WAIT! P LEARNS N TIME-STEPS LATER?

Wasn't Astrolabe supposed to run in O(log N) time?

Why is it suddenly running in time O(N)?

WHAT WENT WRONG?

In this horrendous tree, each node has equal “work to do” but the information-space diameter is larger!

Astrolabe was actually benefitting from “instant” knowledge because the epidemic at each level is run by someone elected from the level below

INSIGHT: TWO KINDS OF SHAPE

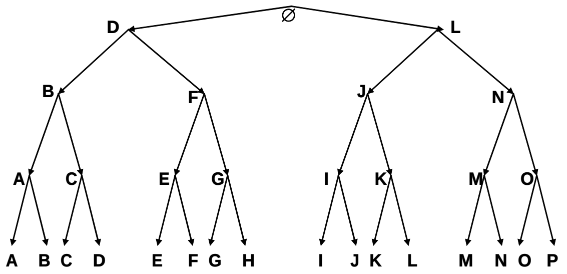
We’ve focused on the aggregation tree

But in fact should also think about the information flow tree

Our example was showing how an information flow tree can be slow.

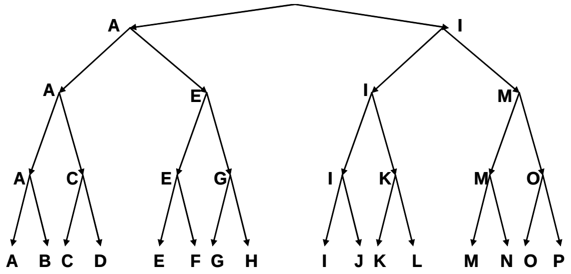
INFORMATION SPACE PERSPECTIVE

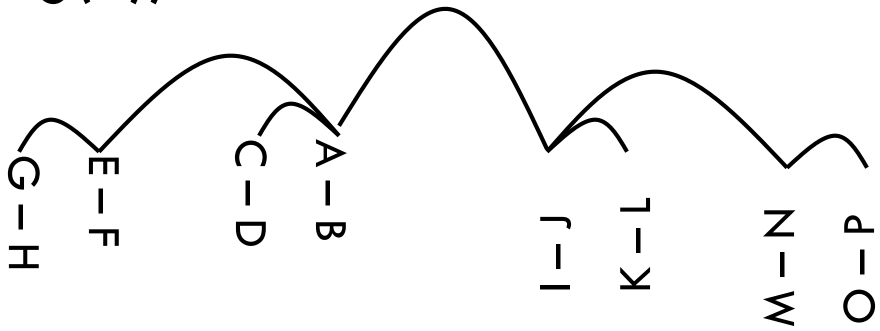
Bad aggregation graph: diameter O(n)



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

Astrolabe version: diameter O(log(n))





SO… WE FIXED THAT

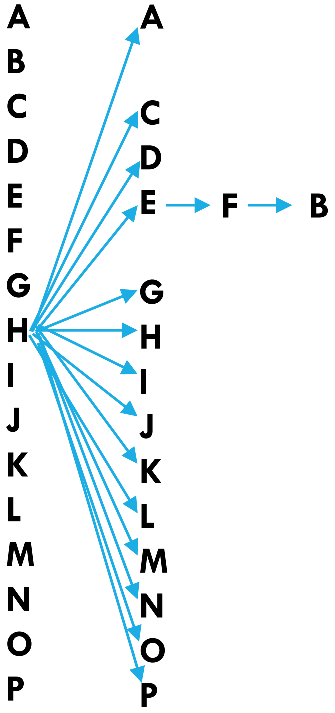
But then they had another idea.

Recall how UDP multicast was used to speed up urgent notifications with Bimodal Multicast.

Could something like that be used to speed up Astrolabe?

INFORMATION SPACE PERSPECTIVE

UDP multicast causes a fast "all to most" exchange. Then a few stragglers need to catch up in the next gossip round or two:



In this UDP-multicast accelerated graph, we get a very accelerated covergence

WE WON’T ANSWER THAT QUESTION

We asked “could UDP multicast speed up Astrolabe” but in fact we don’t have time today to explore this (open) question.

But we do have time to understand UDP multicast in more detail, and to hear about an issue of its very own

BUILDING BLOCKS

Infrastructure tools designers think about technology as building blocks.

They focus on modular components and match the properties of the resulting infrastructure tool to the available building blocks.

But each new combination can bring unexpected problems caused by interactions between elements that work perfectly well “on their own”!

HOW UDP MULTICAST REALLY WORKS

First, the IP system reserves a class of IP addresses for use in UDP multicast. They are "class D" addresses, and we can think of each one as a unique id plus a unique port number *shared by a set of receivers.*

For example, "Ken's Magic Message Bus" might reserve IP address D:224.10.20.30 port number 7890. Every server process in the KMMB service has this hard-wired in.

ROLES OF HARDWARE

In UDP multicast, the hardware itself is supposed to route packets only to where they are wanted.

For KMMB, this will be “nodes subscribing to the topic”. Each (ip,port) pair corresponds to a topic, and we want our packets to go only to subscribers

So the network becomes active, and *filters* traffic

THE BASIC MECHANISMS

When a machine boots, the KMMB server instance launches. It creates a socket and *binds* this standard IP address and port to it.

This causes the NIC to begin to watch for messages that match. In addition, the top of rack switch and datacenter routers are informed that there is a new multicast listener on this segment of the network.

The routers use this knowledge to filter on each forwarding link.

CONCEPT: A BLOOM FILTER: A WAY TO TRACK SET MEMBERSHIP CHEAPLY (O(1) INCLUSION COST)

A Bloom filter is a set of (usually) 3 bit-vectors of some length (usually) 1K

To "remember" X, the filter computes hash(X), hash(X+1), hash(X+2) and sets the corresponding bit in vector 0, 1 and 2.

Later to answer the question "does this filter include X" we repeat but this time check the bits. Answer yes if all 3 bits are set, no if not.

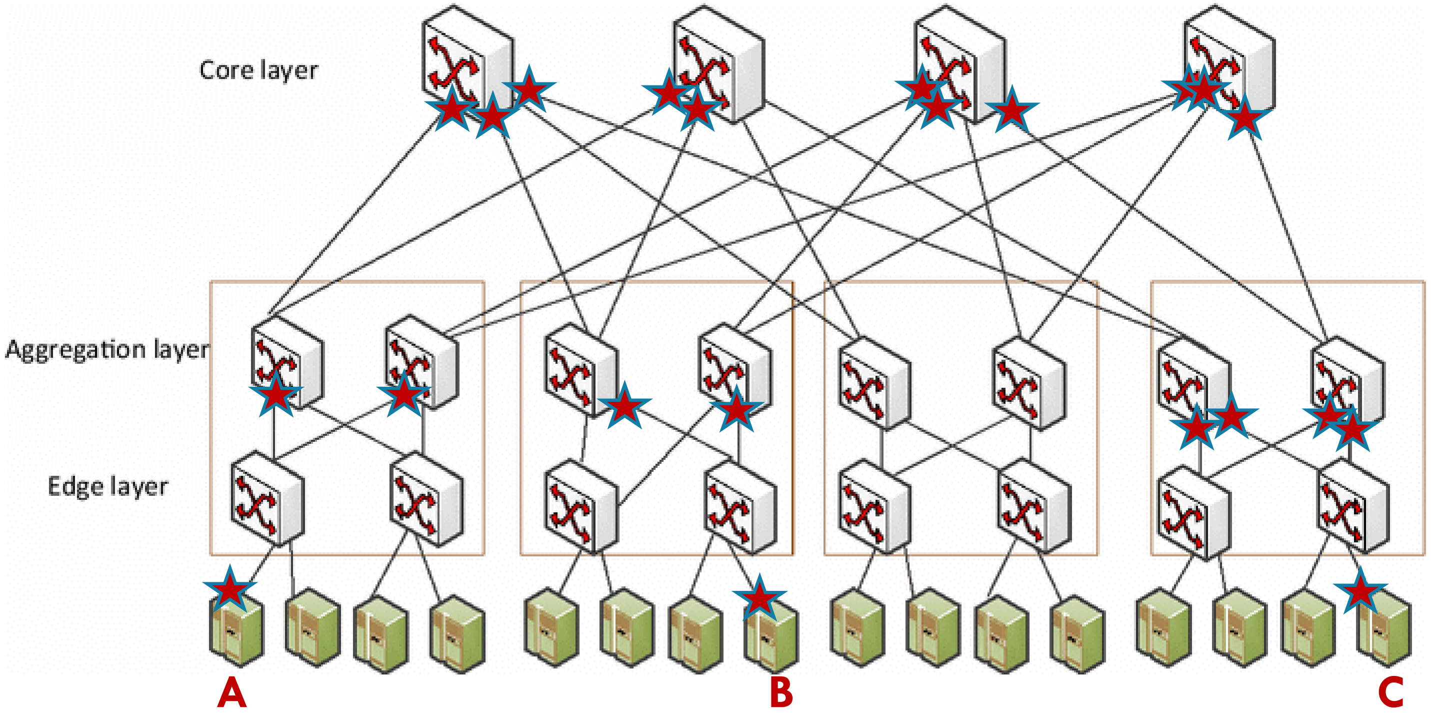
USE OF THESE FILTERS?

The NIC uses a Bloom filter to recognize incoming IP multicast packets it should accept.

The TOR switch uses a Bloom filter to track which links lead to machines listening for a particular IP multicast address.

The fat-tree of datacenter routers uses this to remember which subnetworks have a machine listening for an IP multicast address.

EXAMPLE: NODES A, B AND C ARE IN IP MULTICAST GROUP OF KMMB



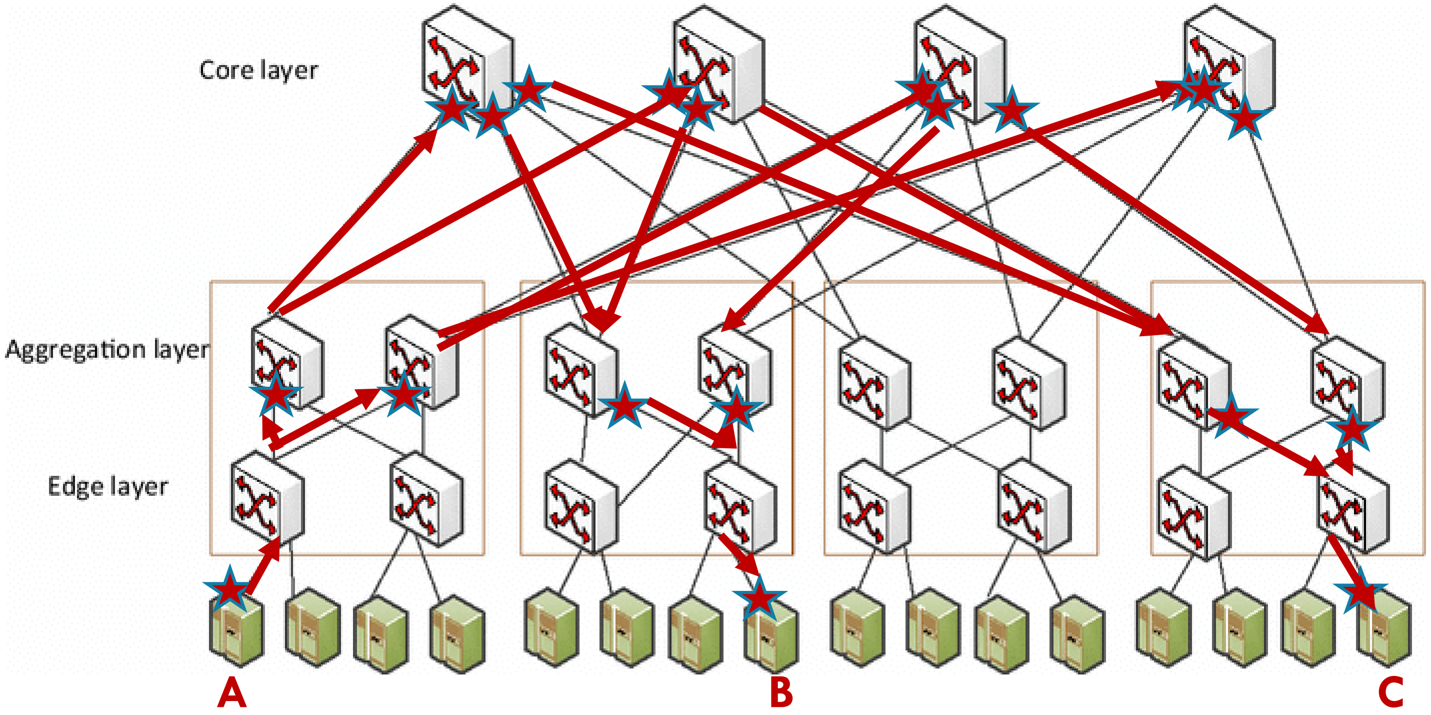
A SENDS A MULTICAST

Suppose we want to publish some event from A to the Foo “group”?

A prepares a UDP packet, puts the special address in it, and calls sendto

At each stage it will be forwarded towards any receivers

EXAMPLE: NODES A, B AND C ARE IN IP MULTICAST GROUP OF KMMB



BLOOM FILTER ROLE?

At the “line rate” of the network (75M packets/second) we have very little time to decide where to forward copies.

The Bloom filter can be implemented in hardware (the hashing policy is the expensive step) and runs fast enough to make the decision before the switch or router exceeds its available time

So we get a very clean UDP multicast that might show up multiple times per receiver, but won’t bother non-receivers…

SOME USES

Maybe KMMB is super popular. **Each user has their own instance.**

Pub/sub is fantastic for tracking debug data and network management properties. If nobody is using the debug monitor ("subscribing"), the network itself automatically discards the UDP packets!

… so perhaps we see a "linear adoption". Maybe for every 1500 machines we see one additional thing using UDP multicast.

WHAT DOES THIS TELL US?

When the datacenter was small, it worked awesomely.

500,000 / 1500 = 330. Our Bloom Filter bit vectors each have 1024 bits. Not very many are set, and filtering genuinely prevents UDP multicast packets from being forwarded unless there is a real listener down that link

WHAT DOES THIS TELL US?

When the datacenter was small, it worked awesomely.

But then the boss gave the order and we doubled the size! Plus, more and more people are using KMMB for debugging and similar tasks

1M / 1500 = 660. Our Bloom Filter bit vectors only had 1024 bits. So now most of them will be set. Yesterday with 500,000 machines this wasn't the case – only 330 were set, per vector.

FALSE POSITIVES

As we scale up the data center, more and more of the UDP packets are going to be forwarded to more and more machines, due to Bloom Filter matches.

In effect we go from the network filtering out "no receiver" packets to forwarding every packet, many copies each, on every link.

This overloads the network and it becomes lossy. It may also overload individual machines if some machines are listening for many IP multicast addresses

WE CALL THESE MULTICAST STORMS

The term refers to an all-to-all message pattern that overwhelms the entire data center.

Basically, a single event ended up crashing the whole datacenter within seconds!

HUGE % OF MESSAGES GET DROPPED

All the machines see a huge overload. They are receiving packets they didn't subscribe to, and must "manually" discard them, which takes time

The whole data center grinds to a halt.

Lots of other services begin to get timeouts due to unresponsive servers, causing even more errors to report

THESE PROBLEMS HAVE ACTUALLY BEEN SEEN!

One result is that most modern data centers disable UDP multicast.

Either trying to use it always gives errors or, more common, when you try to use it they automatically set up TCP connections and route your messages over TCP.

For smaller multicast groups this works… but you can't make 100,000 TCP connections from one node to 100,000 other nodes. So we can't use the UDP speedup feature in most datacenter systems.

SUMMARY

Stairway to heaven needs repairs!

Gossip is tricky! UDP multicast is tricky too! In fact everything except TCP seems to be risky!

A gossip mechanism will have constant, low overheads and very predictable delay, provided that the information sharing graph is of low diameter. This is what blockchain gossip layers assume.

But small mistakes can yield gossip solutions that actually malfunction in major ways, potentially shutting down entire datacenters!