NETWORK MANAGEMENT · PAPER 2

Pingmesh

A Large-Scale System for Data Center Network Latency Measurement and Analysis

Chuanxiong Guo, Lihua Yuan, Dong Xiang, et al. · Microsoft

ACM SIGCOMM 2015

Presented by Cheng Peng, Vincent

MOTIVATION

Three questions network management should answer

Detection

Is something broken and is it the network?

Many failures look like network problems: a P99 latency spike, throughput falling from 20 MB/s to under 5 MB/s, a component reachable only sometimes.

Localization

If it is the network, where exactly?

Hundreds of thousands of servers and switches, millions of cables. Finding the broken piece during a live incident is hard.

SLA

Is the network good enough for each service?

A Search query waits for the slowest of thousands of servers. Each service uses different servers, so each needs its own SLA.

~50% of problems reported as “network” problems were not caused by the network , but without data, the network team couldn’t prove it.

BEFORE PINGMESH

Why existing tools could not answer them

Ask the devices

SNMP counters, syslog, Cisco IP SLA on switches

Detection fails: a switch can drop packets while its counters say everything is fine.

SLA fails: measures the switch, not what each service actually feels.

Run tools on demand

traceroute, TcpPing, after someone complains

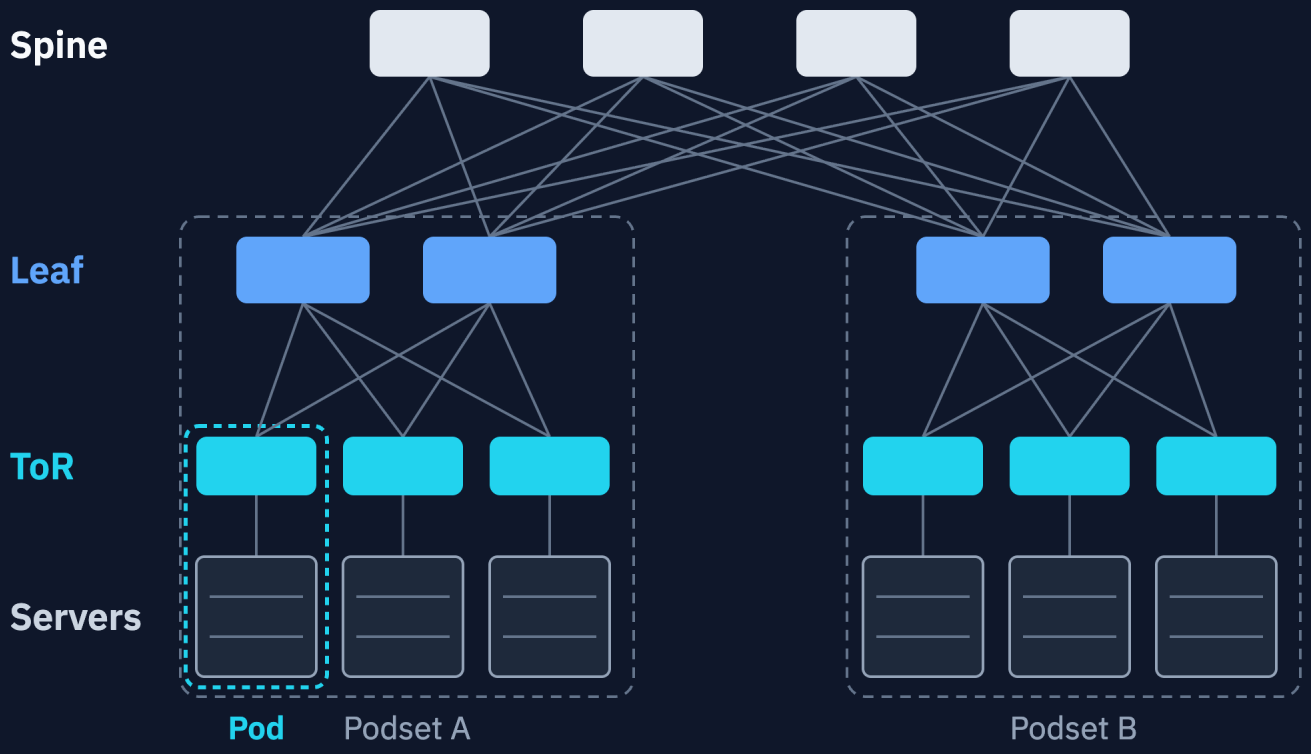
Detection fails: not always on, and transient problems are gone before you run them.

Localization fails: needs a known source, destination pair, which you often don’t have.

> To answer all three, they needed latency data for every server, all the time, measured the way applications see the network.

BACKGROUND

How a data center network is wired



A three-level Clos network

About 40 servers connect to one top-of-rack (ToR) switch, and together they form a pod. About 20 ToRs connect to 2 to 8 Leaf switches, and this group is called a podset. Tens to hundreds of Spine switches connect all the podsets. Between two servers in different podsets, there are many paths of equal length. Each switch picks one by hashing the packet's addresses and ports (ECMP), so a server cannot tell which Spine its packets went through. This makes a broken Spine switch hard to find.

BACKGROUND

What “network latency” means here

Round-trip time (RTT): how long a message takes to go to another server and come back. The time is spent in three places:

1. Inside the servers

The app handles the message, the operating system’s network code runs, and the network card passes it along.

2. On the cables

Putting the bits onto the cable (bigger packet = longer), then the signal traveling along it (longer distance = longer).

3. Inside switches

When a switch is busy, packets wait in line before being sent on (queuing).

Why count the server part too?

Users just see “it’s slow” and blame the network. So measure the whole trip, then find which part is slow.

What the paper measures

The time to open a TCP connection between two servers, with no data attached.

DESIGN

The key idea: every server probes, all the time

Probe = one test connection. A server opens a TCP connection to another server and times how long the answer takes.

1. Every server

All servers take part. If we only pick a few, we might miss the part of the network that is broken.

2. Always on

Keep probing all the time, not only after something breaks. When a problem happens, the data is already there.

3. Use the same protocols as apps

Most apps use TCP and HTTP, so the probes use TCP and HTTP too, not ICMP ping. Then we measure what apps really see.

DESIGN

How Pingmesh works: the big picture

1. Controller

Makes the list

Decides who probes whom. Gives each server a pinglist: the servers it should probe. Servers come and pick up their own list.

2. Agent

Does the probing

A small program on every server. Probes everyone on its list, records the times, and sends the results up. Also answers other servers' probes.

3. Data analysis

Turns data into answers

Collects results from all servers, computes drop rates and latency, and makes dashboards and alerts.

DESIGN · PART 1: CONTROLLER

Who probes whom?

Every server probing every other server = far too many probes (there are hundreds of thousands of servers). And there's no need: servers in the same rack share one way out.

Level 1

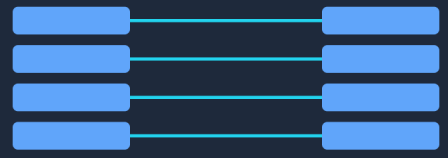
Inside a rack

The ~40 servers in the same rack all probe each other. It's a small group, so every pair gets tested.

Level 2

Between racks

Every pair of racks gets tested. Server #1 in rack A probes server #1 in rack B, #2 probes #2, and so on. So every server helps out.



Level 3

Between data centers

A few servers from each data center probe the other data centers.

Result: each server probes about 2,000–5,000 others. Not hundreds of thousands.

DESIGN · PART 2: AGENT

An agent on every server: safe and cheap?

Safe

If anything goes wrong, the agent stops itself (fail-closed).

- Uses too much memory → it gets shut down
- Hard limits on how often and how big its probes can be
- Can’t reach the controller → it stops probing

Cheap

The agent uses only:

0.26% of CPU on average

<45 MB of memory

a tiny amount of network traffic

DESIGN · PART 3: DATA ANALYSIS

Turning results into alerts

1

Collect

Gather probe results from every server.

2

Compute

Two key numbers: the drop rate, and P99 latency (how slow the slowest 1% are).

3

Act

Over the limit → alert: “network problem.” Also draw maps (heatmap) of the whole network.

Runs at two speeds: a fast path (~5 min) for quick alerts, and a detailed path (~20 min) for deeper analysis.

SLA · STEP 1 OF 3: NORMAL LATENCY

What does normal latency look like?

Pingmesh watches two numbers: **latency** and **packet drops.** To spot a problem, we first need to know what normal looks like.

Normal #1

Most probes are very fast.

In a busy data center, the median probe takes **0.22 ms** inside one rack and **0.27 ms** between racks. Crossing the upper-level switches adds only about 0.05 ms, so queuing in the network is small.

Normal #2

But a few are always slow.

On a day with no network incidents, the slowest 0.01% of probes took **1.4 seconds or more** in the busy data center, and **0.1 seconds or more** in a lighter one.

> Most probes are fast, but even on a normal day the slowest few can take over a second. So one slow probe does not mean the network is broken. Instead, Pingmesh watches P99, the slowest 1%. It is stable on a normal day, so when it becomes much slower than usual, something is wrong.

SLA · STEP 2 OF 3: NORMAL PACKET DROPS

What is the normal drop rate?

Drop rate = out of all the packets sent, how many get lost on the way.

1. How Pingmesh measures it

Pingmesh can't see lost packets directly. But if a probe's first packet is lost, TCP waits 3 seconds and sends it again. So a probe that takes about 3 seconds tells us a packet was lost.

2. What is normal?

1 to 10

packets, roughly, are lost out of every 100,000 sent. This range held in five data centers, tracked daily, unless there was an incident.

3. Where are packets lost?

Packets that travel between racks are typically lost several times more often than packets that stay inside one rack. This suggests most drops happen in the network, not in the servers.

> So the drop rate has a stable normal range. A big jump means something is wrong.

DETECTION · STEP 3 OF 3: NOT NORMAL → ALERT

Is it a network issue?

When is it “not normal”?

Pingmesh raises an alert if either of these happens:

- More than 100 packets are lost out of every 100,000.
- The slowest 1% of probes (P99) take longer than 5 ms.

Before Pingmesh

An engineer had to ask which servers were affected, log in, run tools, and try again and again. It was slow and done by hand.

With Pingmesh

The data is already there for every server. If the numbers look normal, it’s not the network. If not, Pingmesh shows which servers are affected.

> This answers the first question from the start: is it really the network?

LOCALIZATION · CASE 1 OF 2: BLACK HOLES

Where is the network issue?

Pingmesh found switches that drop packets silently: they lose packets, but their own records say everything is fine. There are two kinds: black hole and random drops

What is a black hole?

A black hole is a switch that drops every packet for certain source–destination pairs. For example, server A can't reach server B at all, but A can still reach C and D. All four servers are healthy, so the problem is a switch in the middle. A common cause is a corrupted forwarding-table entry in the switch chip. Reloading the switch fixes it.

How Pingmesh finds the bad switch

1. For each rack, compute the fraction of its servers that show this symptom.
2. If that fraction is high in only some racks of a group, those racks' switches are reloaded automatically (at most 20 per day).
3. If every rack in a group shows it, the fault is likely in a higher-level switch, so engineers are notified.

> Result: at first, about 20 switches a day were flagged. Over time this dropped to only a few per day, and black-hole complaints stopped.

LOCALIZATION · CASE 2 OF 2: RANDOM DROPS

Random drops

A random drop is when a switch loses a small share of its packets, no matter where they come from or where they go.

Here is a real case. Every user in one data center suddenly saw higher latency. Pingmesh showed that packets were lost much more often than normal: about 200 out of every 100,000, instead of the usual 1 to 10. Only this one data center had the problem.

So the problem was likely in the top-level switches, because a lower-level switch connects too few servers to slow down everyone. The heatmap pointed there too. But those switches' own records showed no errors.

Next, Pingmesh found pairs of servers losing 1 to 2% of their packets. The engineers ran TCP traceroute on these pairs, which shows the switches a packet passes through. This led them to one top-level switch. Once it was taken out of service, the drops stopped. The cause was a hardware part inside the switch that was flipping bits.

> Unlike a black hole, reloading the switch does not fix this. The broken hardware has to be replaced.

LOCALIZATION · READING THE MAP

How to read a heatmap

Pingmesh collects more than 200 billion measurements every day. No one can read them one by one, so Pingmesh draws them as a heatmap. Here is a small example with 6 racks in 3 podsets:



Rows: who sends. Columns: who receives.

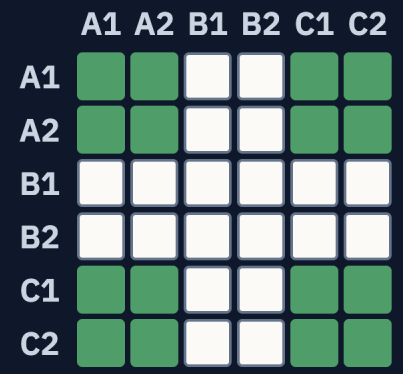
- Each row is a rack that sends. Each column is a rack that receives.
- Each square shows how fast that pair is. For example, row A1, column B2 shows how fast A1 talks to B2.
- Racks with the same letter are in the same podset: A1 and A2 are in podset A, B1 and B2 in podset B, and C1 and C2 in podset C.
- Green means normal, red means slow, and white means no data.

> Here everything is green, so the network is healthy. A real heatmap has thousands of racks, but you read it the same way.

LOCALIZATION · READING THE MAP

Which level of switches is broken?

Different problems make different shapes. Same 6 racks, three kinds of failures:



1. Podset B is down

Podset B sends and receives nothing, typically because of a power loss. Its rows and columns have no data, making a white cross.



2. Podset B is failing

Traffic to or from podset B is slow, for example because one of its switches drops packets. This makes a red cross.



3. A top-level switch is broken

Inside each podset is still green. Between podsets is red, because that traffic goes through the top-level switches.

> Shape 3 is exactly what they saw in the random drop case. That's how they knew it was a top-level switch.
> A1 to A1 means two different servers inside the same rack. Their traffic only goes through the rack's own ToR switch

LIMITATIONS

What Pingmesh can’t do

Localization gap

It finds the level, not the exact switch.

There are tens to hundreds of top-level switches. Pingmesh can say “one of them is broken,” but not which one. The last step still needs traceroute.

Detection gap

It only times one quick “hello.”

Some problems only show up when sending lots of data. Once, a bug made TCP send fewer packets at a time, so long transfers got hundreds of milliseconds slower. Pingmesh didn’t notice, because one “hello” still took the same time.

Coverage gap

It needs a program on every server.

That works on Microsoft’s own servers. But you can’t install it inside customers’ virtual machines in the cloud. There, you have to watch real traffic instead.

> These three gaps are the open problems that later work tries to address.

LOOKING FORWARD · CLOSING THE LOCALIZATION GAP

ClusterMapper: choose the path

Pingmesh-style probing has two problems. Because of ECMP, you are not sure which path a probe took. And every probe involves two hosts, so a lost probe may just mean the other host is down.

ClusterMapper keeps Pingmesh's core idea: it sends synthetic probes, measures end-to-end loss, and does not depend on router telemetry. But it makes two changes. First, it uses SRv6 to send each probe along a path it chooses. Second, a node probes itself: the probe leaves through one NIC and comes back through another NIC on the same node.

So the path is known, and only one host is involved. There is no doubt about which link caused a loss, and repeating this across different paths covers the network.

Example path: **NIC A** → **Leaf A** → **Spine 1** → **Leaf C** → **NIC B** (same node)

Source: Araujo et al., "Resilient AI Supercomputer Networking using MRC and SRv6," arXiv, 2026 (presented in lecture by Jitu Padhye)

LOOKING FORWARD · RESILIENCY TODAY

The same question in large-scale AI training

Meta's Llama 3 training run on 16,384 GPUs had 419 unexpected interruptions in 54 days, about one every three hours.

From Meta, "The Llama 3 Herd of Models," 2024

	Pingmesh (2015)	AI training (today)
The question	Is it a network issue?	Is it a network issue, or a slow GPU or server?
Why the slowest one matters	A search request waits for the slowest of thousands of servers.	In expert parallelism, each rank exchanges data with every other rank in its group (all-to-all) and must wait for all of them. So one slow rank, called a straggler, holds up all the others.
Why it is hard to localize	Many equal-cost paths (ECMP): it finds the level, not the exact switch.	When one rank is slow, every rank looks slow, so it is hard to tell which one is the real cause. This is still an valuable open research question.

DISCUSSION

Questions for us

1. Pingmesh can only tell which level of switches is broken. ClusterMapper sends each probe along a path it chooses, so it can point to the exact broken link. What do we give up by choosing the path ourselves?

2. Pingmesh uses fixed limits to decide what is abnormal. In distributed training, we could compare each rank with its peers instead. But in MoE models, some ranks get more tokens than others. How can we tell a rank that is broken from one that is just busy?

3. Pingmesh was built for data centers in 2015. Which of its design choices still work for today's large-scale AI training clusters and which need to change?

Thank you!