N

Enhanced (*and open source*) IX Fabric Monitoring

Ben Cartwright-Cox

Ben Cartwright-Cox (He/Him) Verify now

bgp.tools Guy / Systems Engineer for contract hire

London, England, United Kingdom · **Contact info**

500+ connections

Port 179 Ltd

Ben Cartwright-Cox (He/Him)

Verify now

bgp.tools Guy / Systems Engineer for contract hire

London, England, United Kingdom · Contact info

500+ connections

Port 179 Ltd

Needed better monitoring

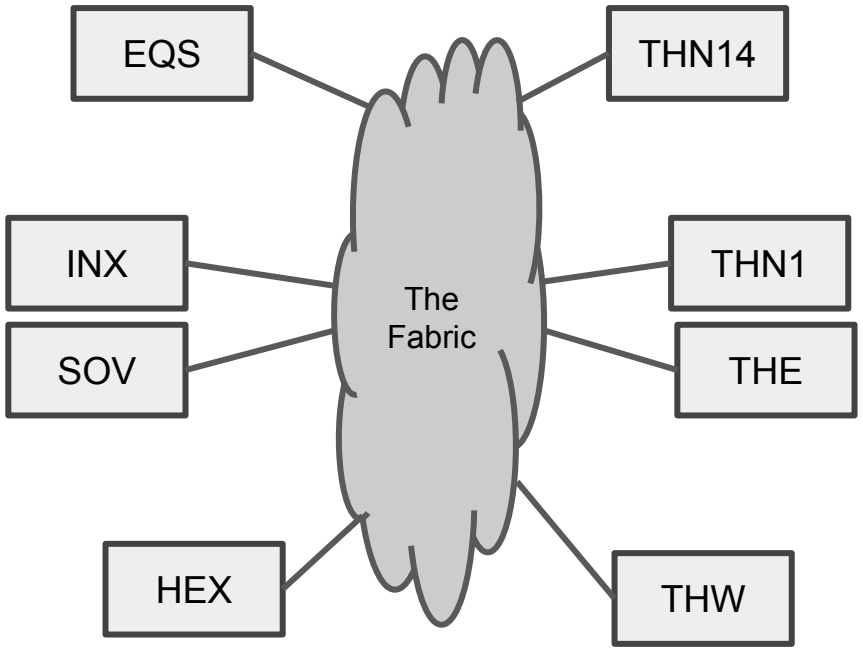
This is not a novel problem

- Many IX's have some form of fabric monitoring (to enforce a SLA)
- Most solutions are a turn key appliance box
- This is not actually that hard of a problem
- Ideally metrics are visible to all members
  - Quickly answers the "*Am I broken or is the IX broken?*" question, maybe even without a support@ email!

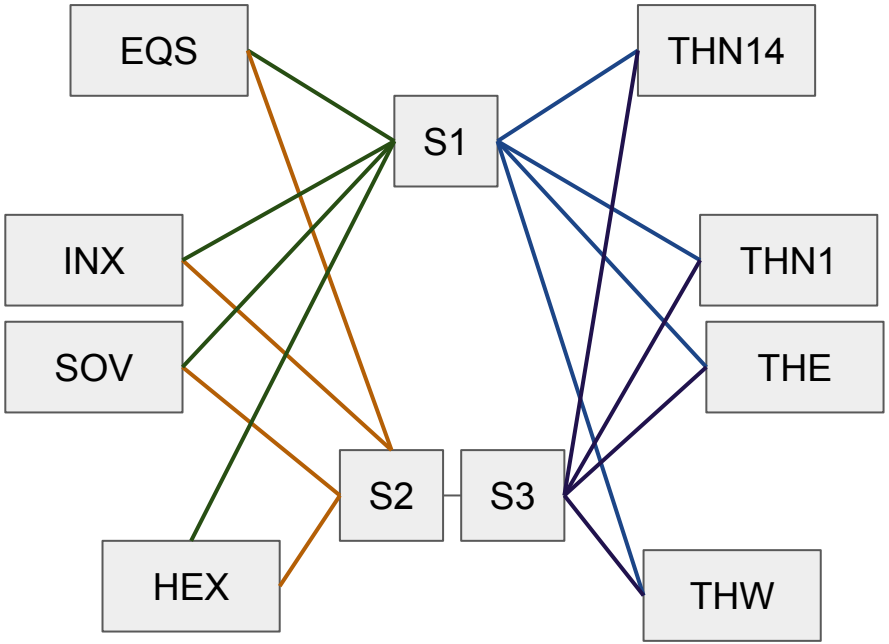
"Standard" Monitoring

- You ICMP Ping a thing
- You see if something comes back and how long it takes
- If it doesn't arrive, or takes a long time, you note that down and/or alert

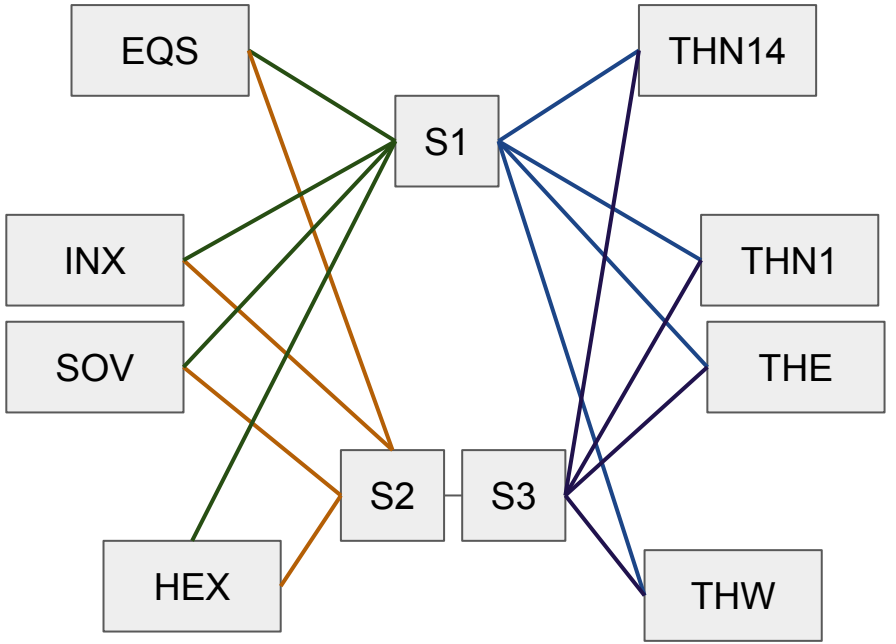
- Problem is that doing this can hide the complexity of modern fabrics

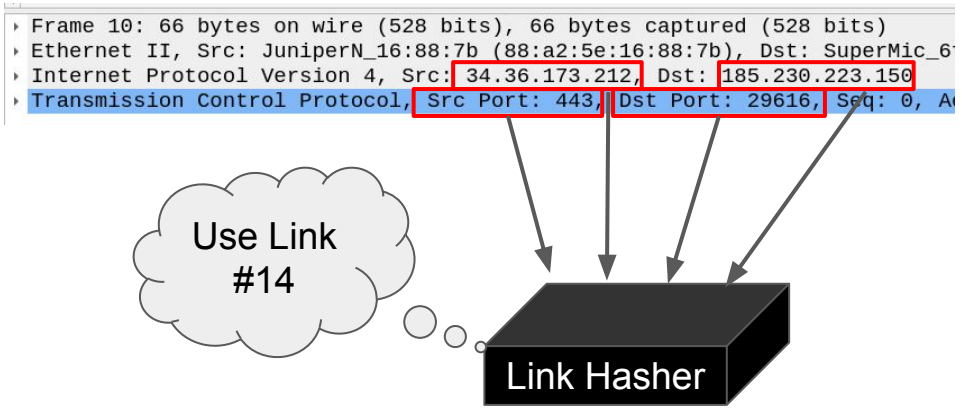


Actual Fabric reality

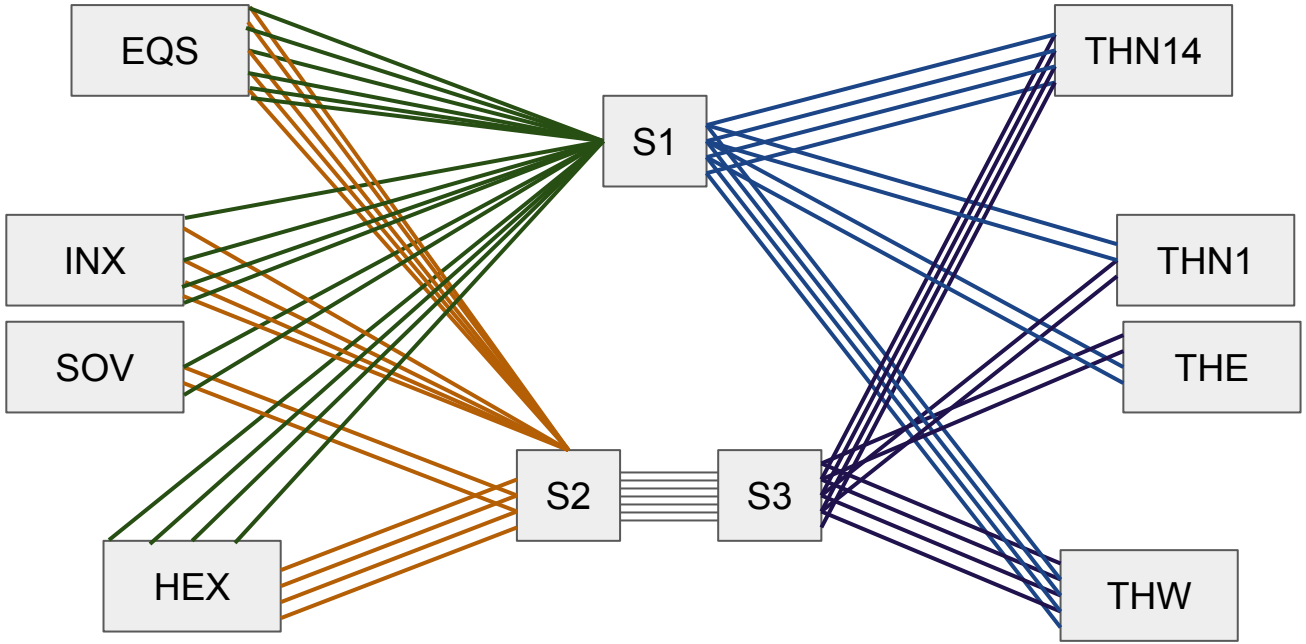


Actual Fabric reality

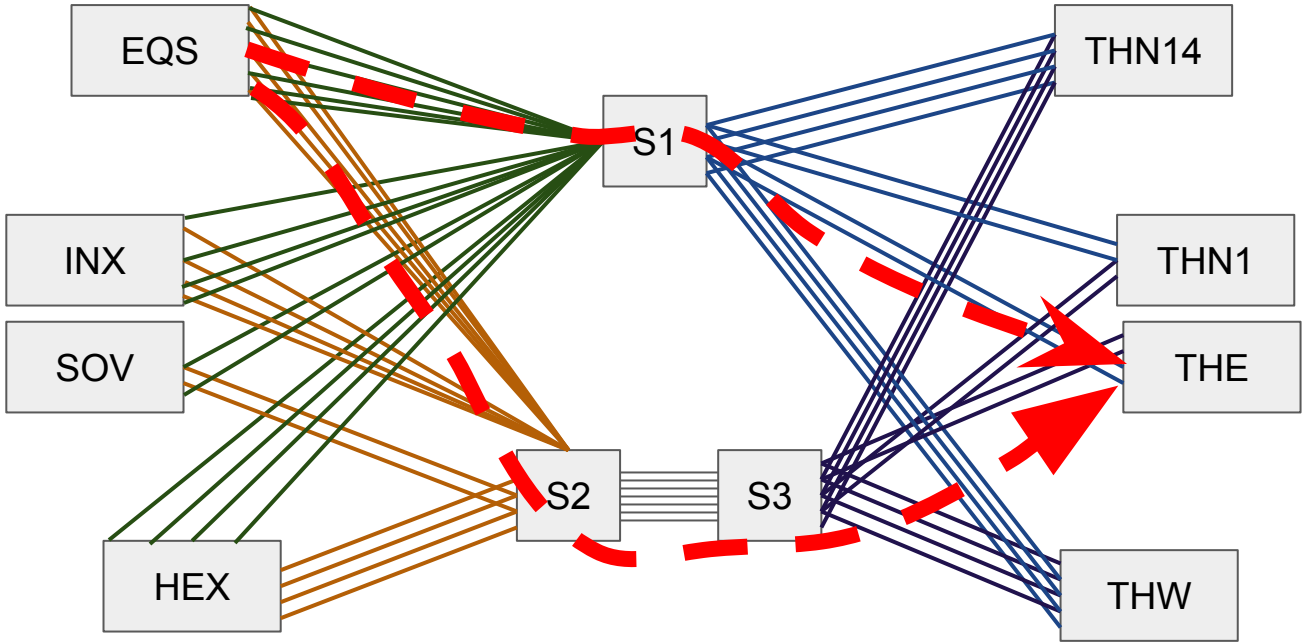




Actual Actual Fabric reality

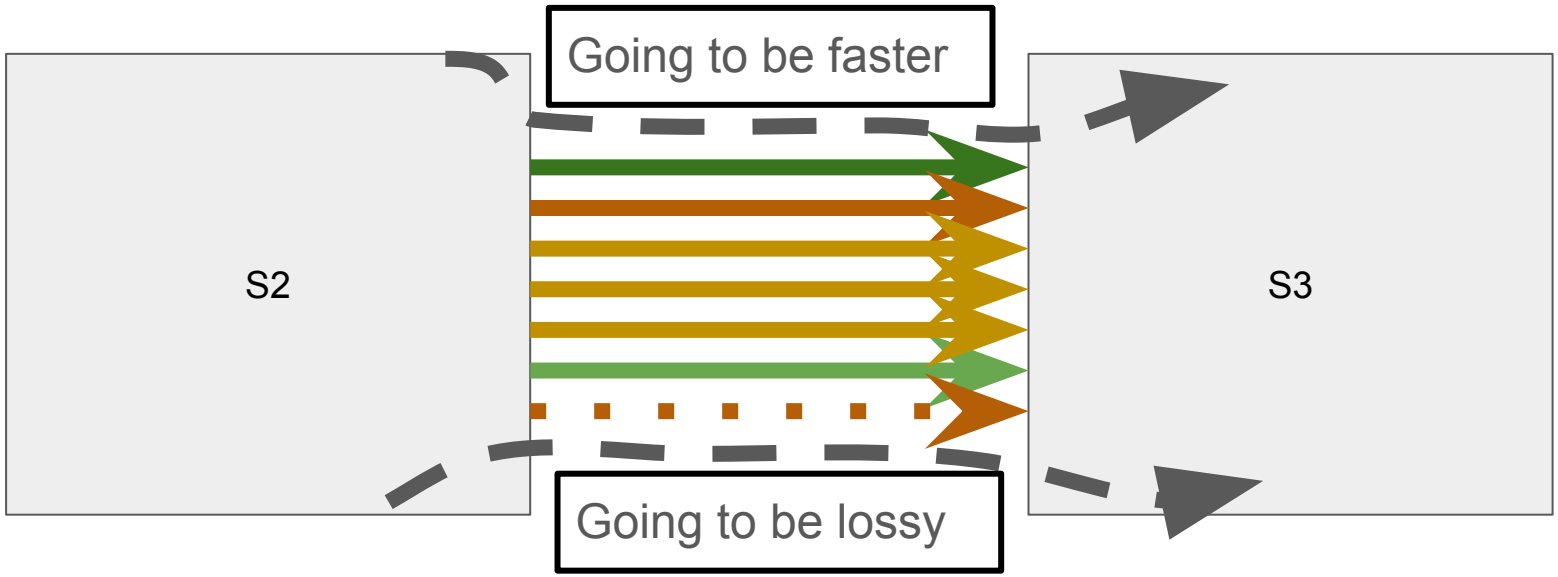


Actual Actual Fabric reality



One of these is going to be faster

***Actual* Actual** Actual Fabric reality

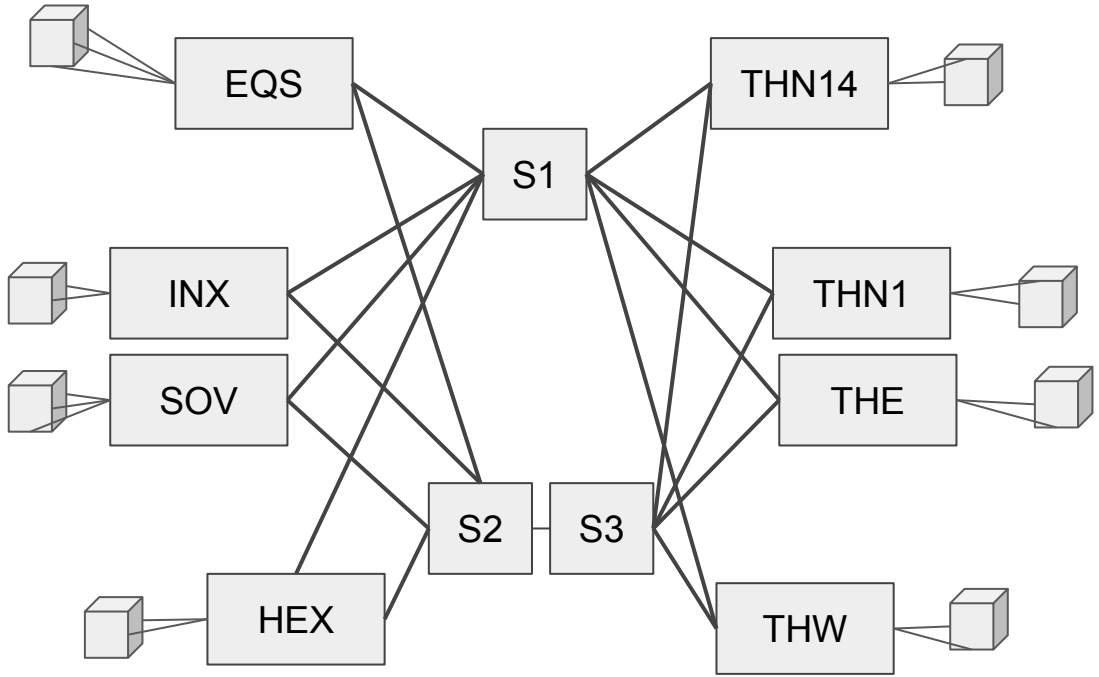


The software - ixp-xping

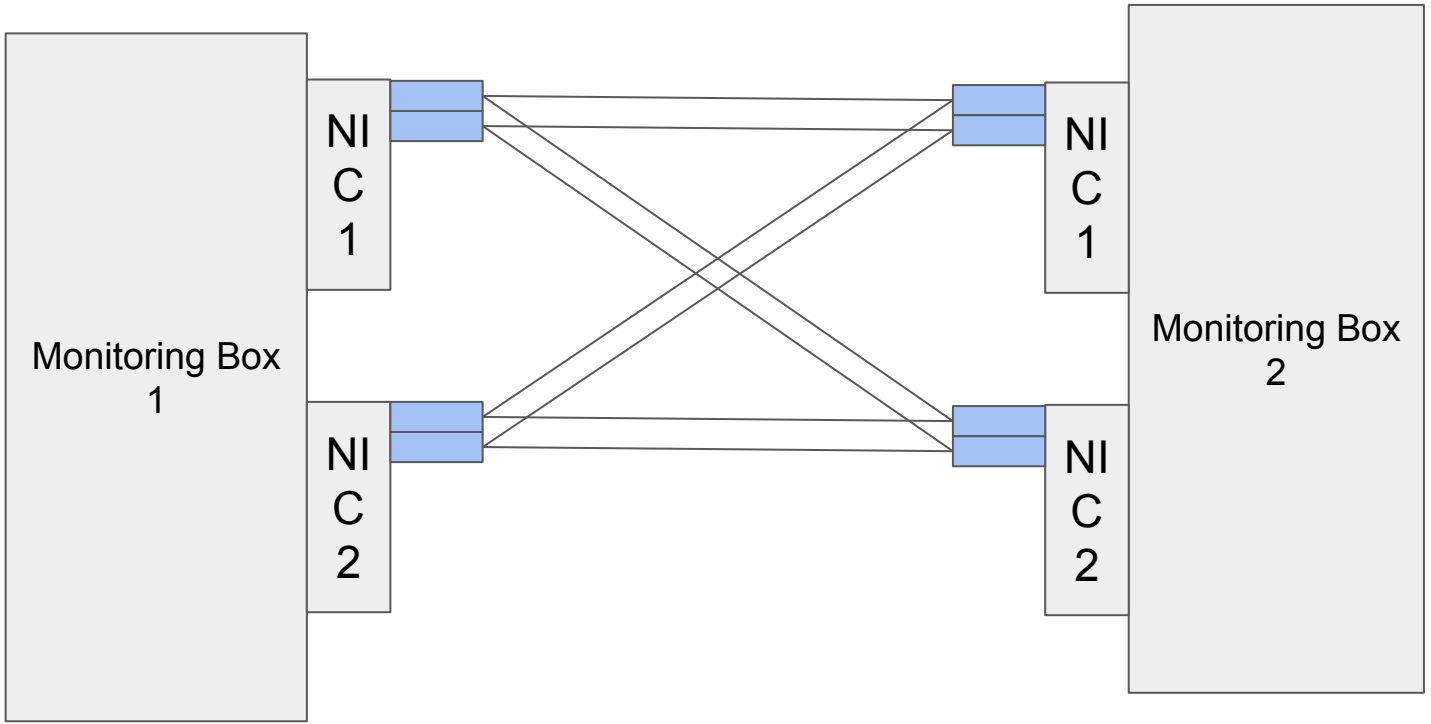
- Software that is capable to do this sorta exists
  - Automattic Pingo (https://github.com/Automattic/pingo)
    - Extremely good at detecting this condition, but requires Policy Based Routing to be setup
    - Not ideal for this use case
  - There is TWAMP based stuff as well
    - But that feels overly complex, It's like using a grenade launcher against squirrels
- Ideally we need software that can handle multiple network interfaces
  - So we don't need to any fancy container/namespace/isolation, or use more than one server per PoP
- The final code comes in at around ~650 lines of Go
  - It could be more fancy, but more fancy has some downsides and this is "good enough"

Monitoring all paths in practise

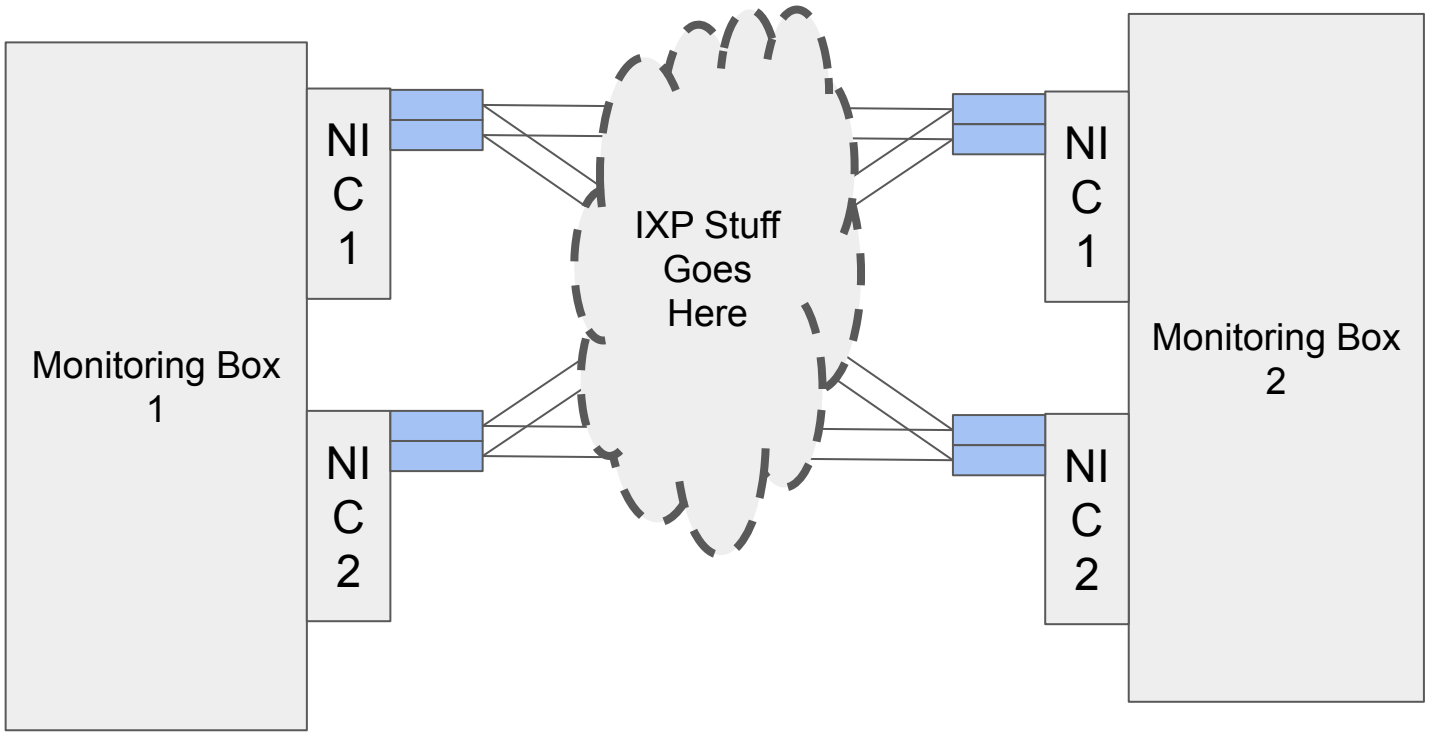
- CPU E5-2630 v4
- 64G RAM
- 10G Intel X520
- 40G ~~Mellanox~~ Nvidia CX-3
- Cheap HPE Kit



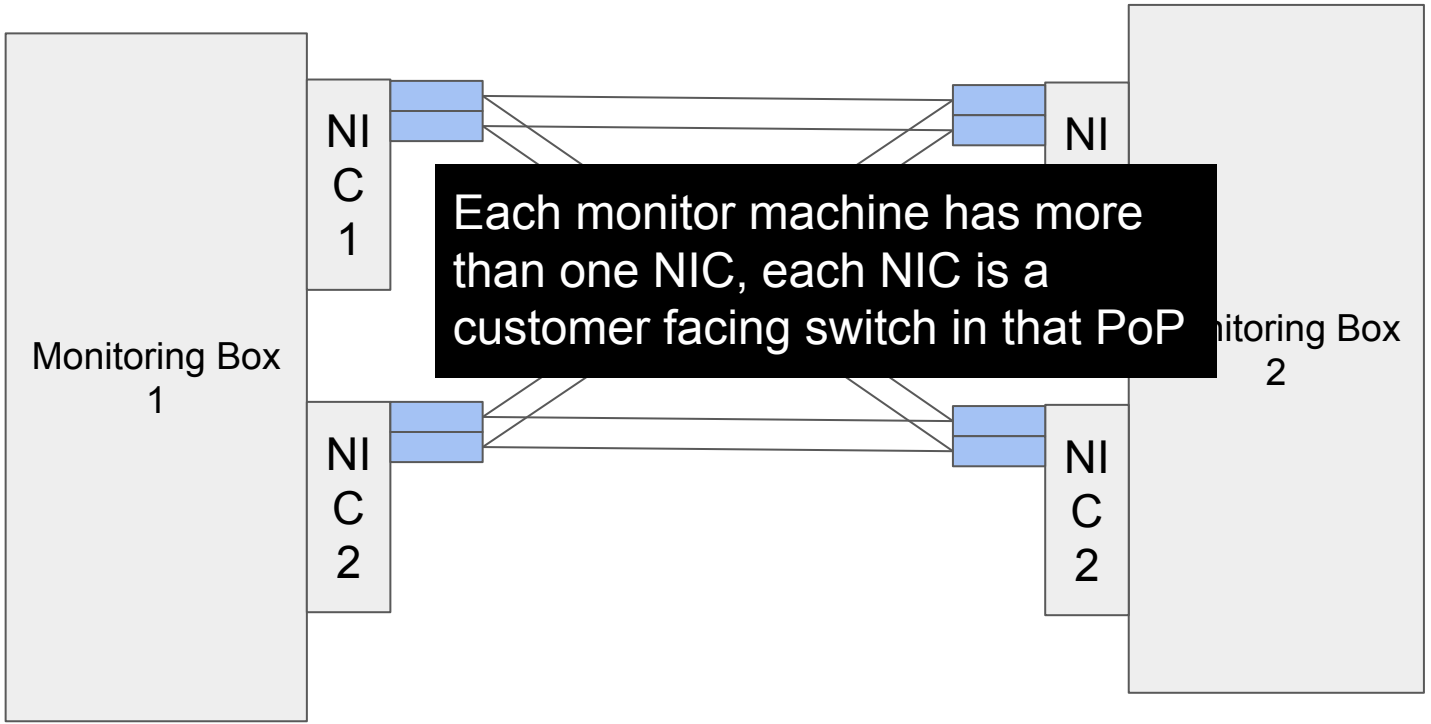
The basic setup



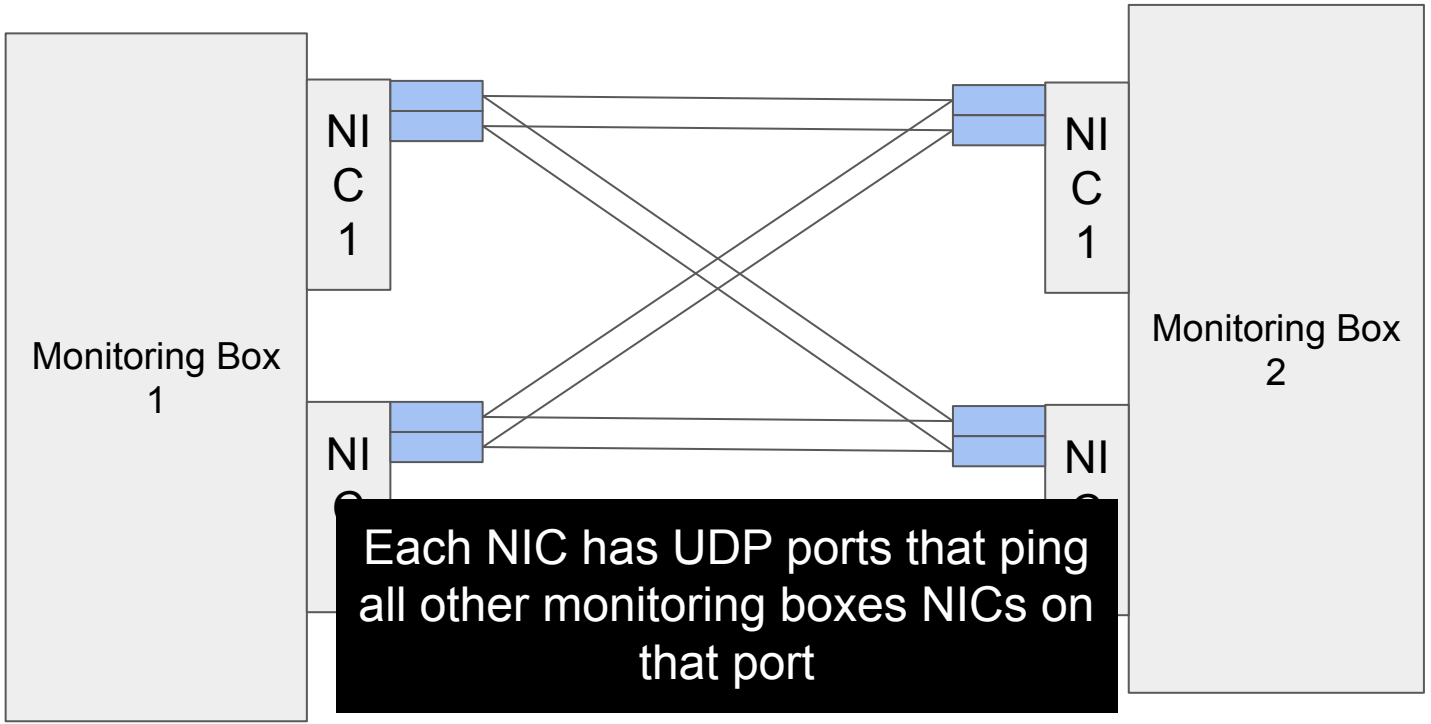
The basic setup



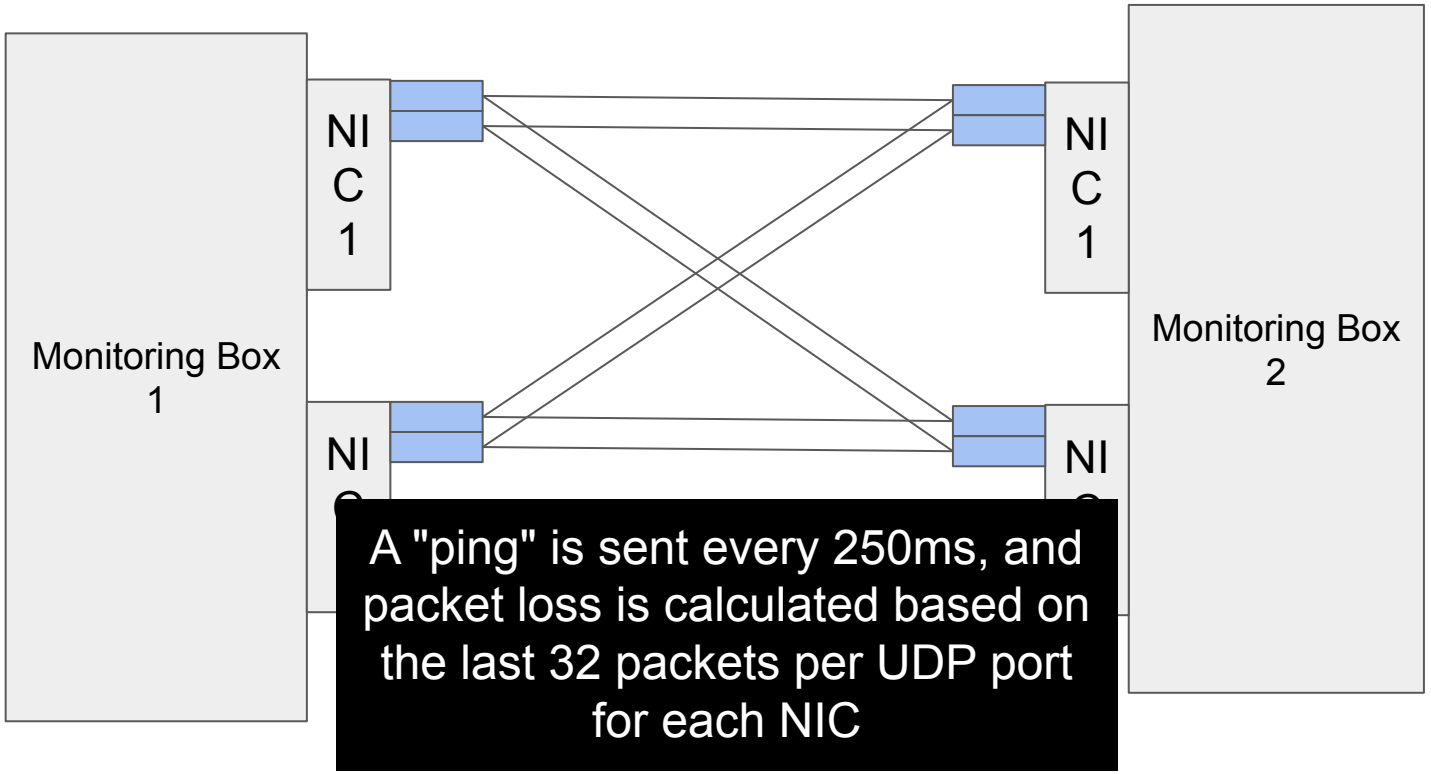
The basic setup



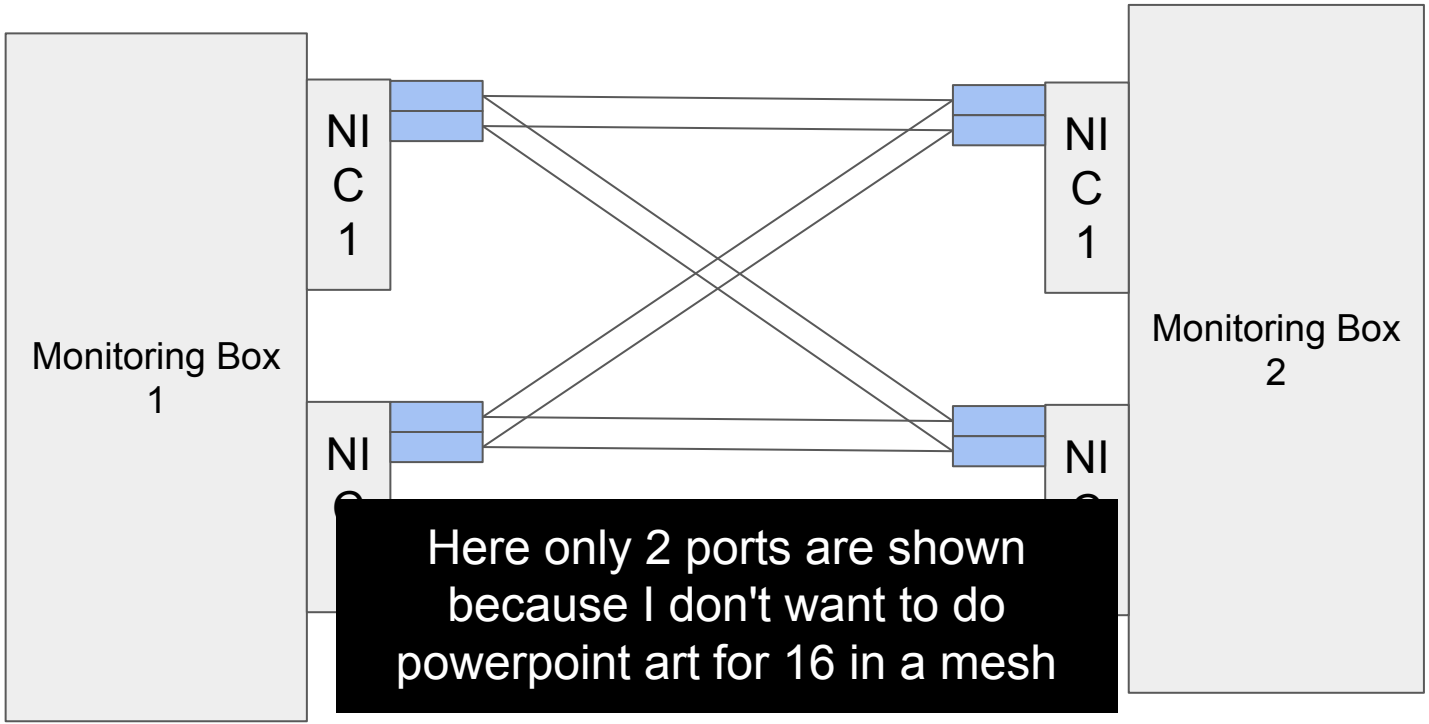
The basic setup



The basic setup



The basic setup



A note on latency

- Golang has a garbage collector
  - Every garbage collector does at some point have to "stop the world", even for a very short amount of time
  - This still happens in ixp-xping
- The network latency paths are sensitive to GC latency since we are measuring things in the 10's of μ seconds!
  - We can have the kernel timestamp stuff for us, so even if we were GC-ing while a packet came in, we can just use the kernel provided timestamp
  - This does not work for the TX path, but jitter in this path is quite rare (It does happen though)
  - To solve that is a lot of extra work, and ideally requires you to make your packets look like Precision Time Protocol so the kernel can automatically "fill in" the timestamps in the packets

Enabling RX timestamping is 2 syscalls away

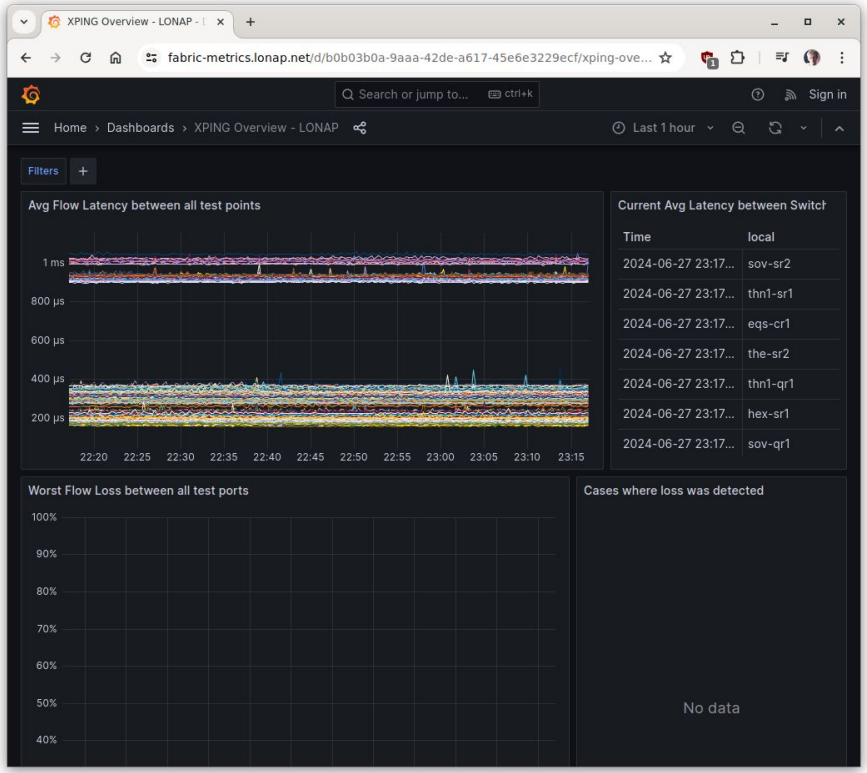
```go
func EnableRXTimestampsOnSocket(uconn *net.UDPConn) error {
    file, err := uconn.File()
    if err != nil {
        return err
    }

    fd := file.Fd()
    err = unix.SetsockoptInt(int(fd), unix.SOL_SOCKET, unix.SO_TIMESTAMP, 1)
    if err != nil {
        return err
    }

    err = unix.SetsockoptInt(int(fd), unix.SOL_SOCKET, unix.SOF_TIMESTAMPING_RX_SOFTWARE, 1)
    if err != nil {
        return err
    }

    return nil
}
```

A note on Linux ARP behaviour

- Linux has a very liberal default on how it responds to ARP
- If you have a IP on NIC1, and someone on NIC2 ARPs for NIC1's IP address, Linux will respond to ARP on NIC2 for NIC1s IPs, ***with NIC2's MAC address***!
- If you attach 3 different switch ports to the IX on the same Linux machine, this gets very """fun"""
- Sysctl's can fix this
  - `net.ipv4.conf.all.arp_announce=2`
  - `net.ipv4.conf.all.arp_ignore=2`

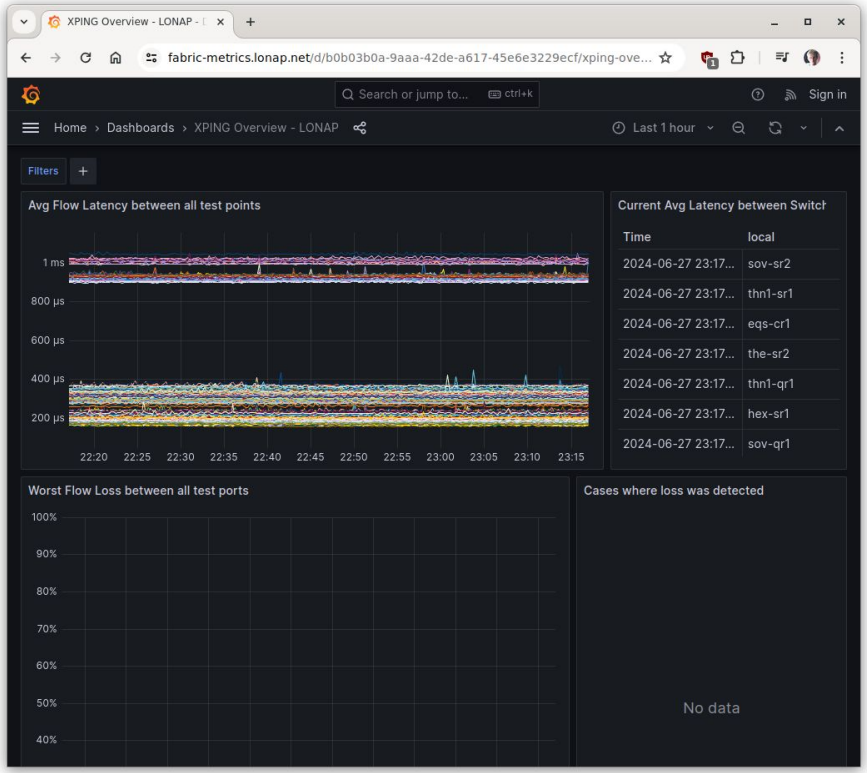
Exporting data

- At least ½ of the industry status quo is Prometheus now
- ixp-xping exports its data out to a prometheus server
  - The prometheus server has some "recording rules" to make common queries faster
- That prometheus sever goes into a publicly accessible grafana
  - Grafana also has easy ways to do this, so it's a "no brainer", most people are also now aware of grafana and have used it before

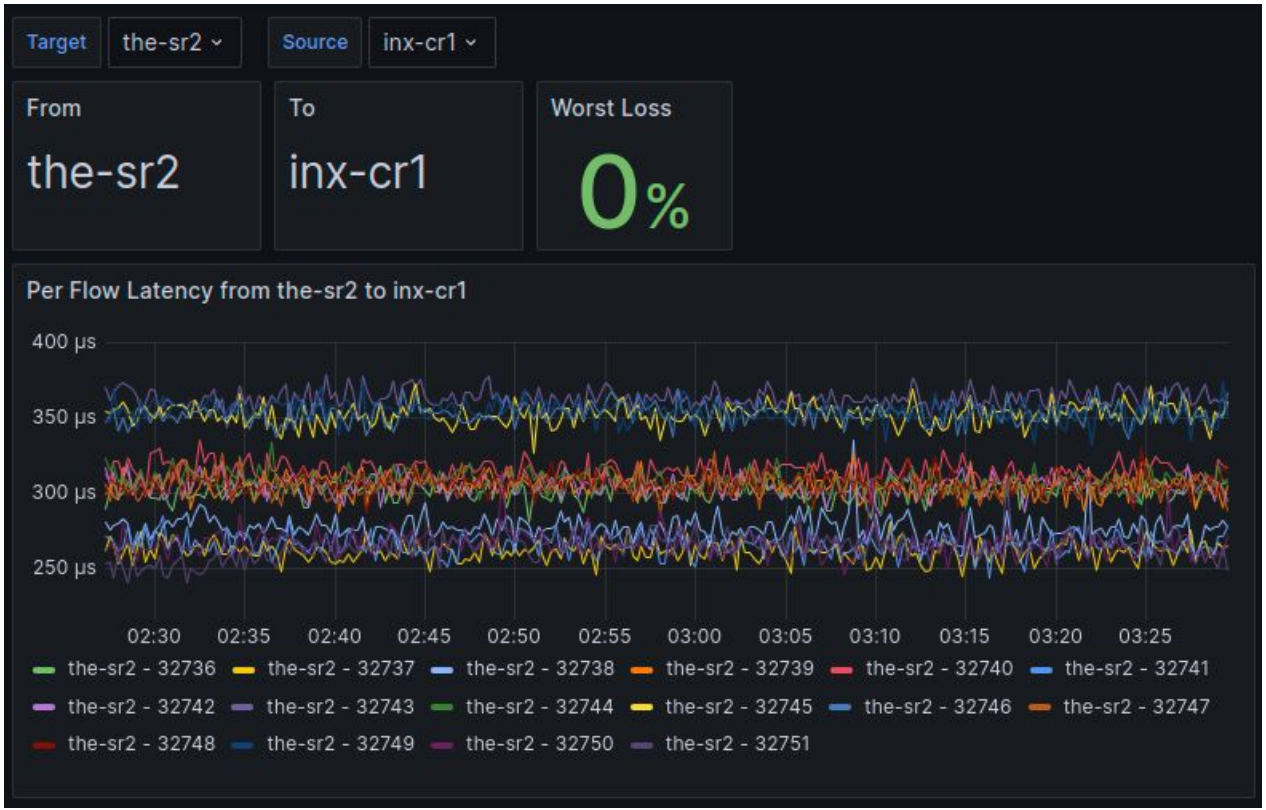


Alerting

- The critical concern with this monitoring is to detect when loss is happening
- **Especially** if it's only happening on a small number of flows
  - These are extremely hard to debug normally, and can go unnoticed for a long time
  - End user transparency and internal improved alerting is the goal here



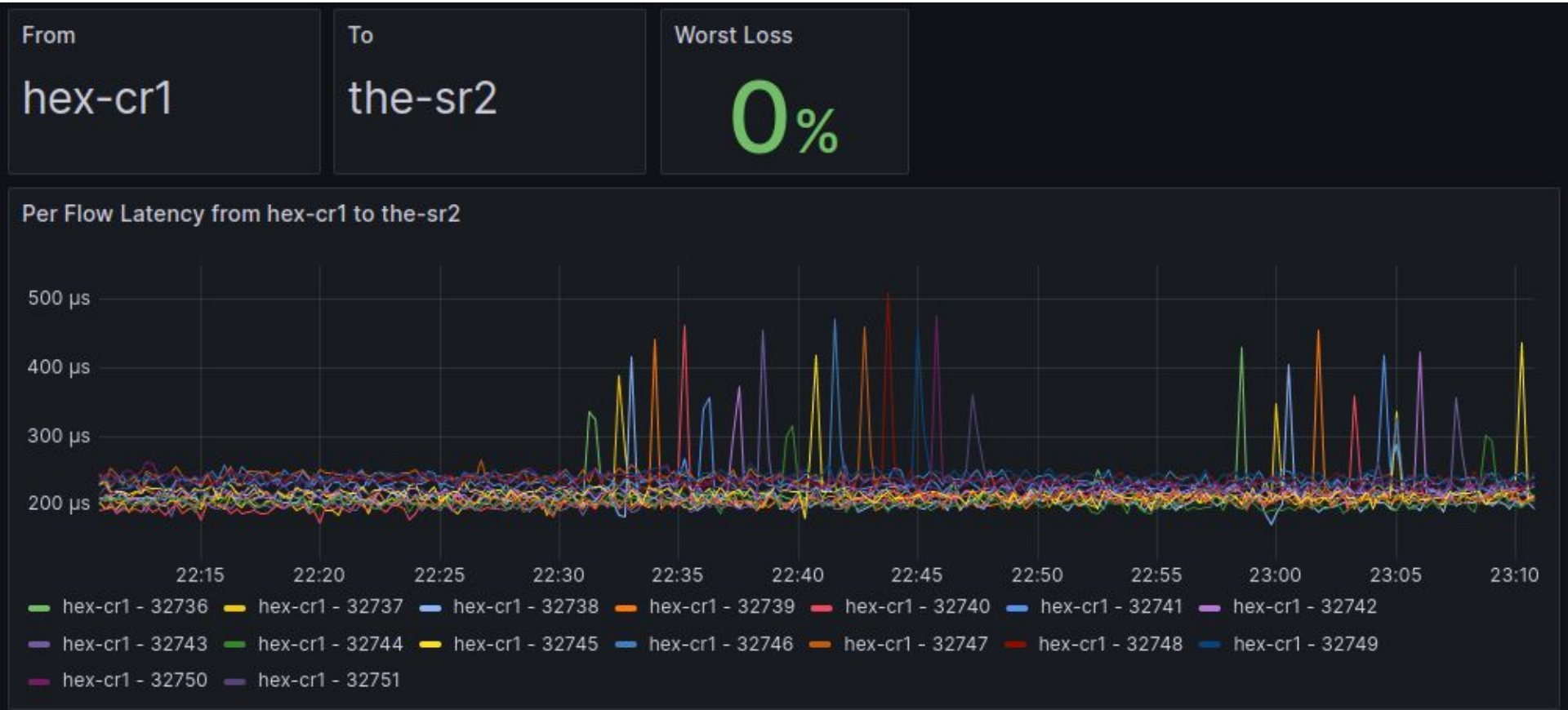
Pretty Graphs!



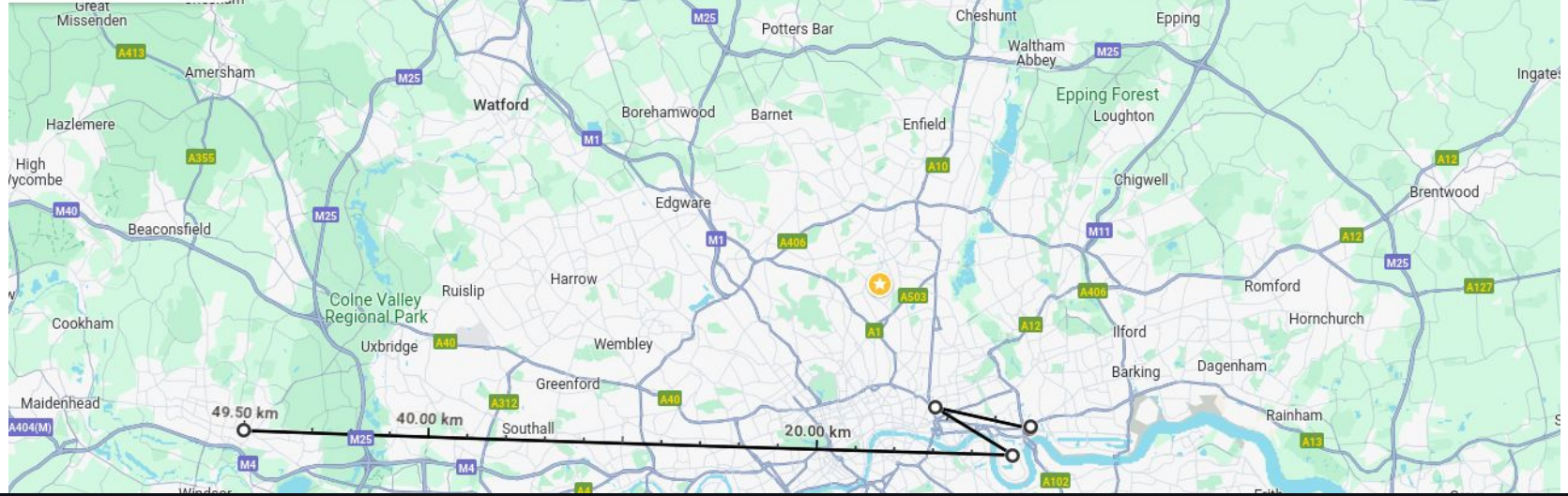
- Each testing box has 16 UDP port pairs
- They all ping in a mesh, on each UDP port, on each NIC, every 250 ms

- **You can now see/monitor individual flows and paths around the fabric**

You can still observe TX jitter from time to time

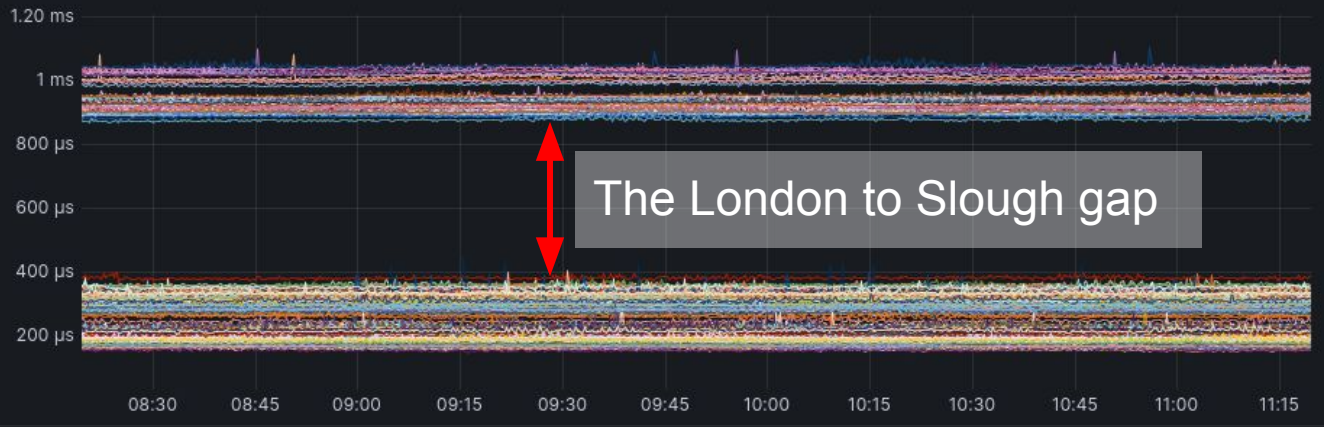


Latency for all switches to all switches



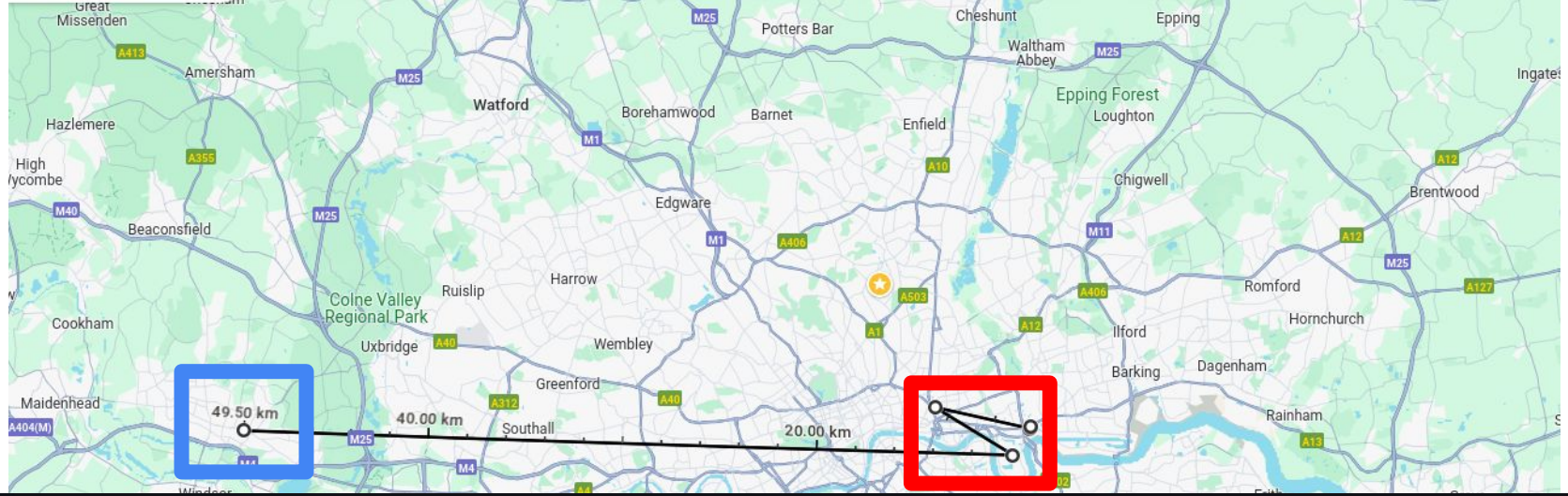
Filters +

Avg Flow Latency between all test points



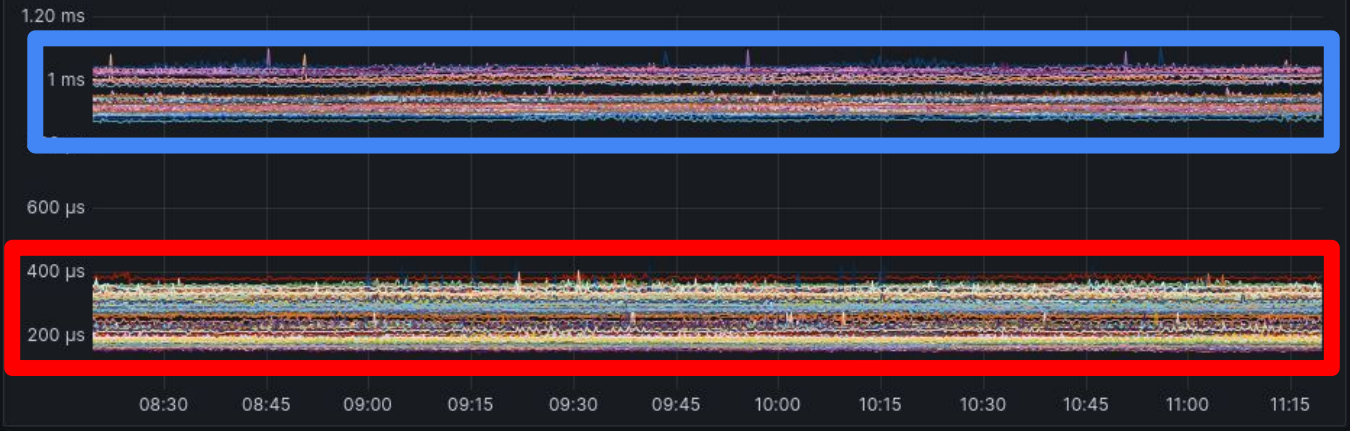
Current Avg Latency between Switches

	local	peer	Value
19:...	thn14-...	eqs-cr1	950 µs
19:...	thn14-...	eqs-qr1	974 µs
19:...	thn14-...	hex-cr1	227 µs
19:...	thn14-...	hex-qr1	233 µs
19:...	thn14-...	hex-sr1	241 µs
19:...	thn14-...	inx-cr1	354 µs
19:...	thn14-...	inx-sr1	380 µs



Filters +

Avg Flow Latency between all test points

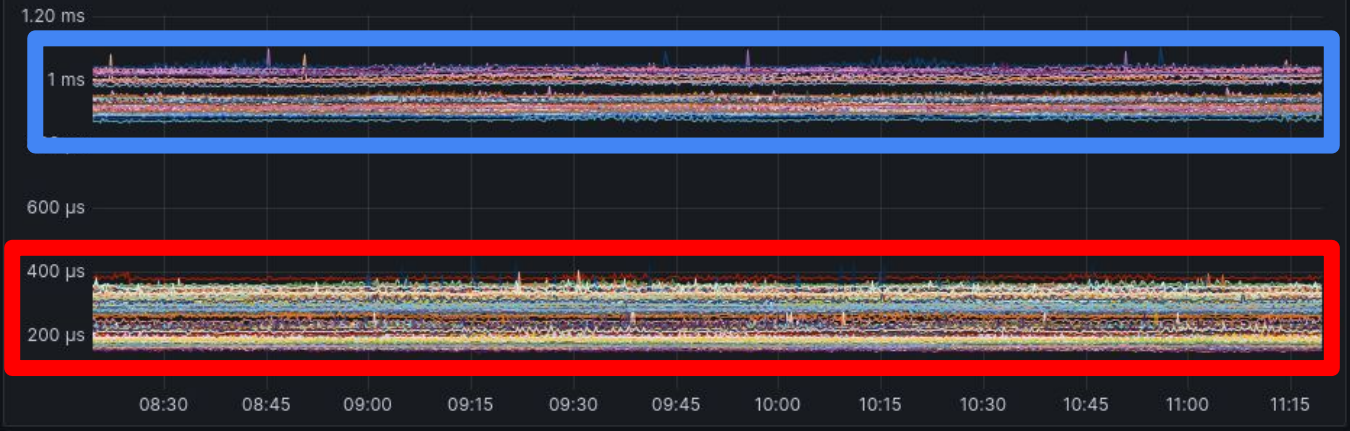


Current Avg Latency between Switches

	local	peer	Value
19:...	thn14-...	eqs-cr1	950 µs
19:...	thn14-...	eqs-qr1	974 µs
19:...	thn14-...	hex-cr1	227 µs
19:...	thn14-...	hex-qr1	233 µs
19:...	thn14-...	hex-sr1	241 µs
19:...	thn14-...	inx-cr1	354 µs
19:...	thn14-...	inx-sr1	380 µs

Filters +

Avg Flow Latency between all test points



Current Avg Latency between Switches

	local	peer	Value
19:...	thn14-...	eqs-cr1	950 µs
19:...	thn14-...	eqs-qr1	974 µs
19:...	thn14-...	hex-cr1	227 µs
19:...	thn14-...	hex-qr1	233 µs
19:...	thn14-...	hex-sr1	241 µs
19:...	thn14-...	inx-cr1	354 µs
19:...	thn14-...	inx-sr1	380 µs

Finishing up

- This is now live at https://fabric-metrics.lonap.net/

- If you want to run this yourself, you can!
  - https://github.com/lonap/ixp-xping
  - Should have everything needed to build and run it

Finishing up

- This is now live at https://fabric-metrics.lonap.net/

- If you want to run this yourself, you can!
  - https://github.com/lonap/ixp-xping
  - Should have everything needed to build and run it

- Questions?