Localizing packet loss

In a large complex network

Leo Huang
lhuang@google.com

Arya Reais-Parsi
growly@google.com

Traditional network monitoring: White box

White box monitoring is basically asking the device to monitor its own vital parameters.

Unfortunately, this is far from being good enough - too often do the devices either 'lie' or **fail to give you the whole picture**.
A classic example is having packet corruption reported on the egress line card, even though the real cause is a fault on the ingress-to-fabric connection.

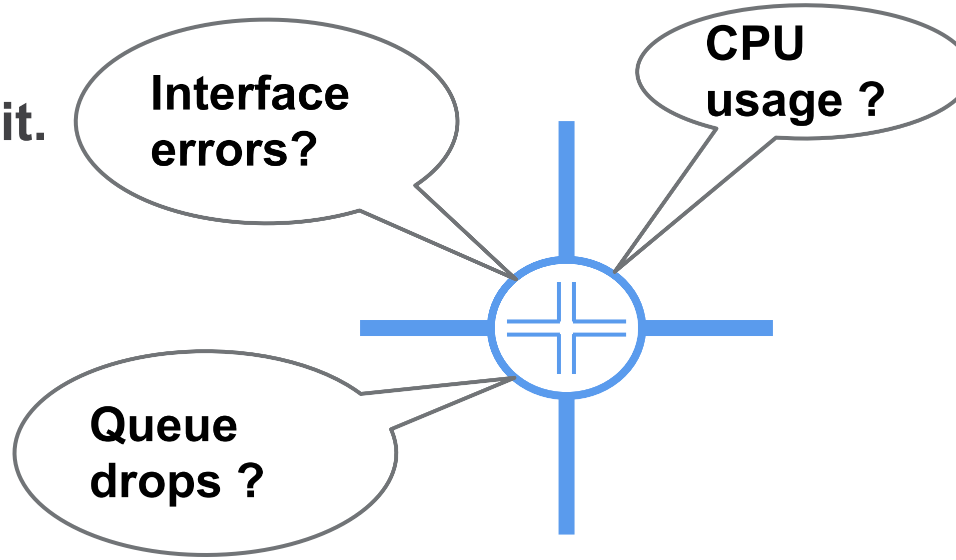
If we can't trust it, we need to test it.

Traditional network monitoring: White box

White box monitoring is basically asking the device to monitor its own vital parameters.

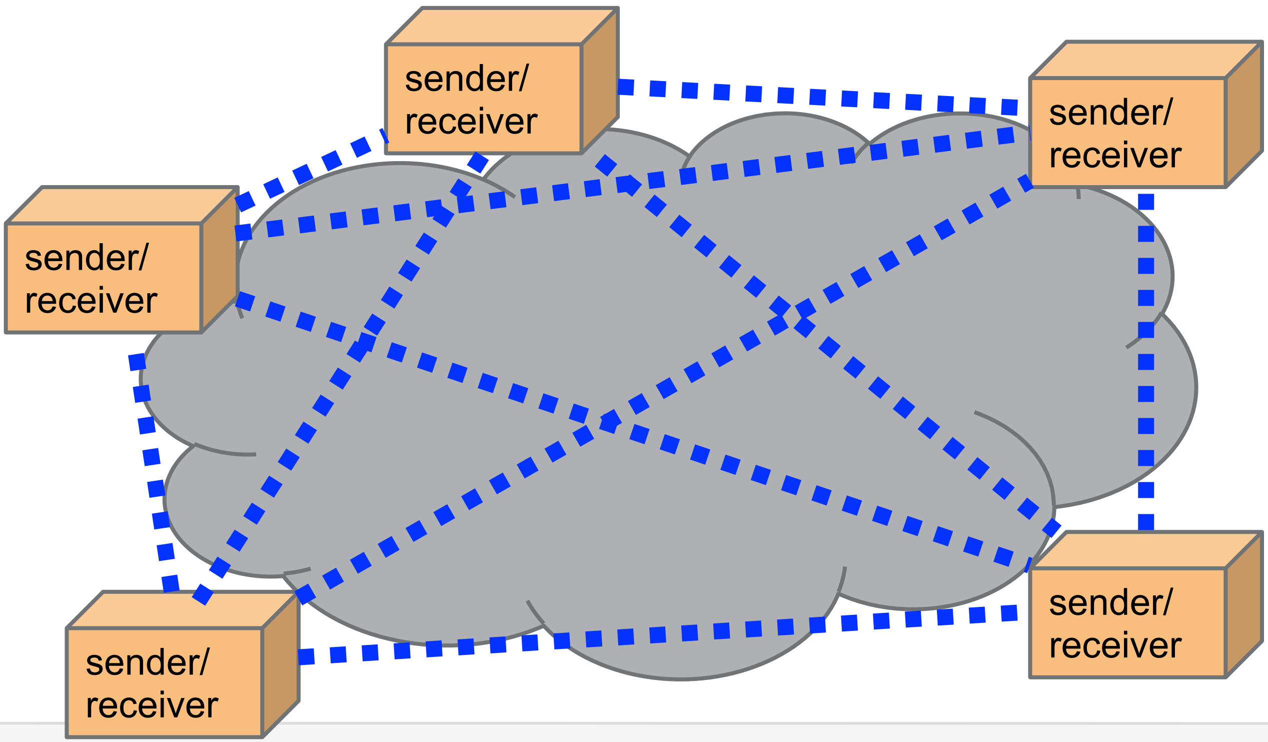
Unfortunately, this is far from being good enough - too often do the devices either 'lie' or **fail to give you the whole picture**.
A classic example is having packet corruption reported on the egress line card, even though the real cause is a fault on the ingress-to-fabric connection.

If we can't trust it, we need to test it.



Traditional network monitoring: Black box

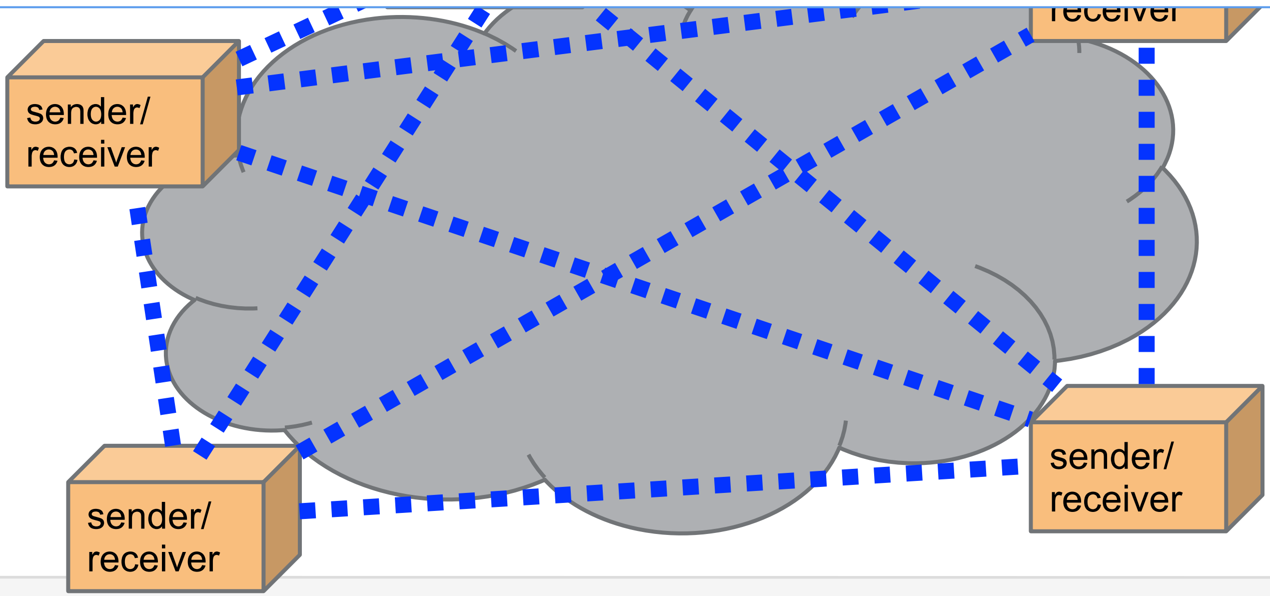
Black box monitoring consists in sending synthetic traffic that mimics production traffic and analyse characteristics such as packet loss, latency, jitter, packet corruption, CoS misclassification,...



Traditional network monitoring: Black box

Two major drawbacks:

- Only the best paths between the senders/ receivers are monitored
- It's hard to isolate a faulty element



That is not good enough

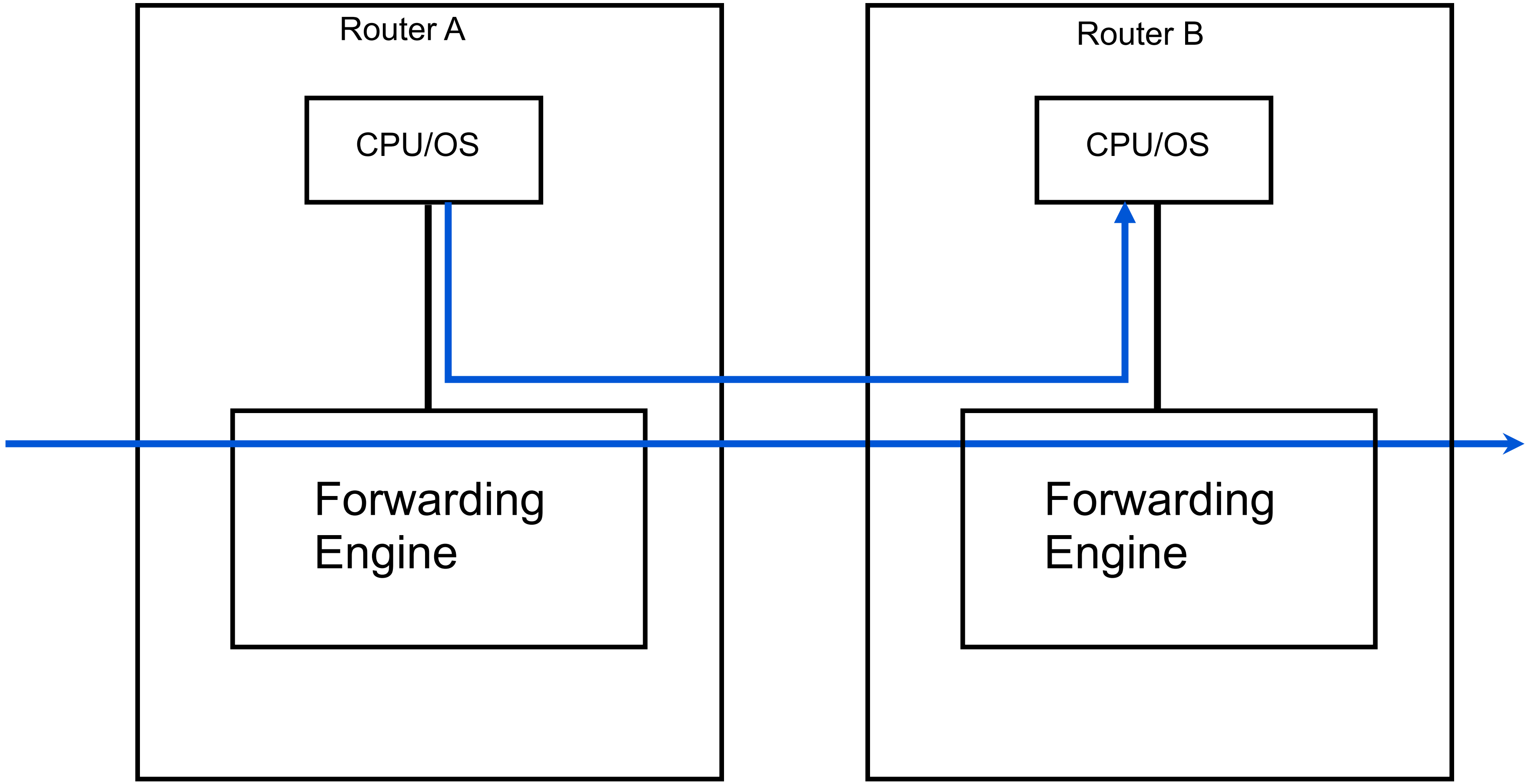
We want:

- Complete coverage. We want to test every single path in the network. Not only the best paths.
- To localize the fault in near real-time (within a minute from the event).
- To test the ability for a router to forward traffic, even when it's not part of the protocol topology.

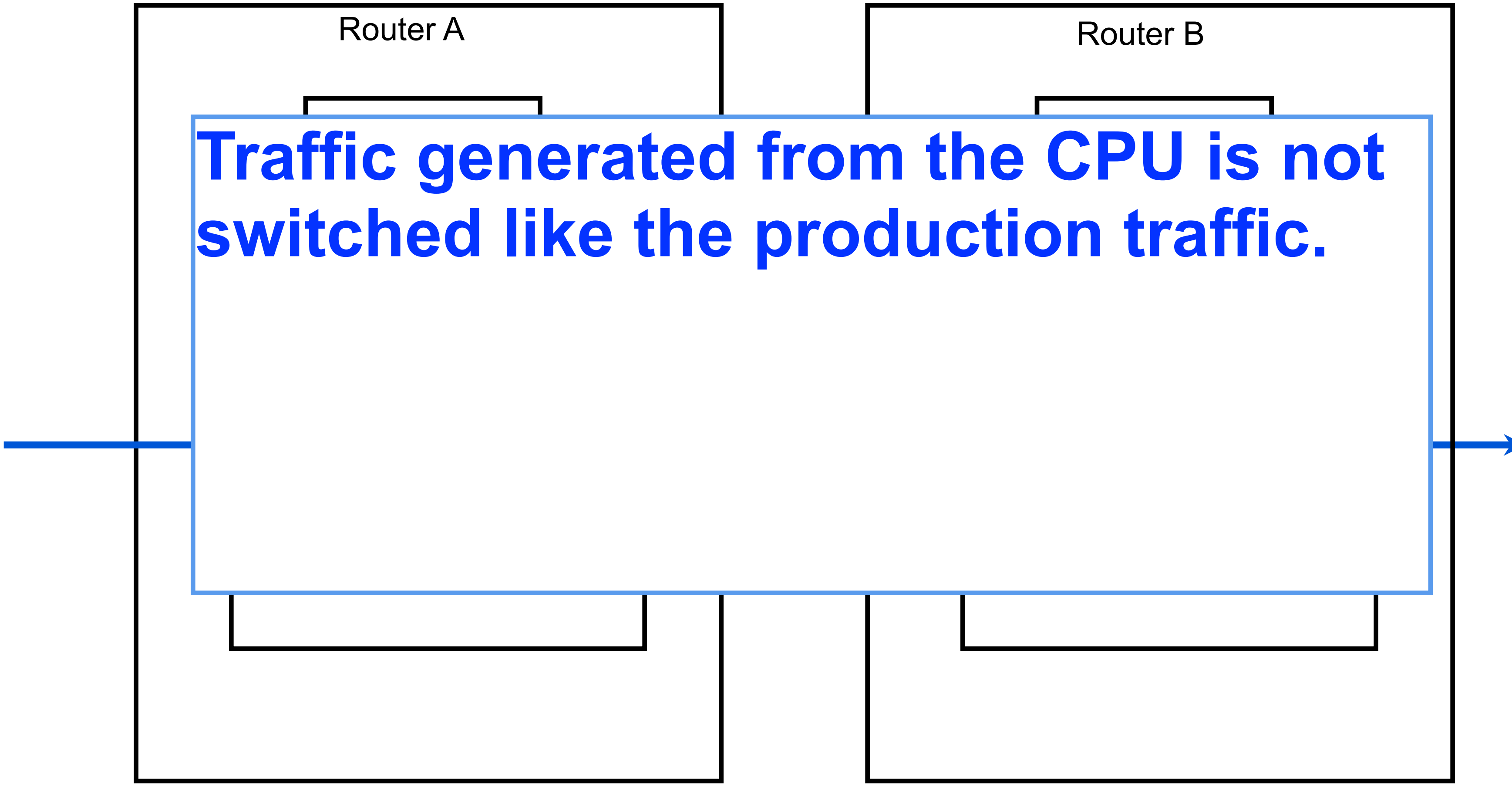
How do we cover every component ?

(Not just the best paths)

We could run an “IPSLA”-like process on each node...



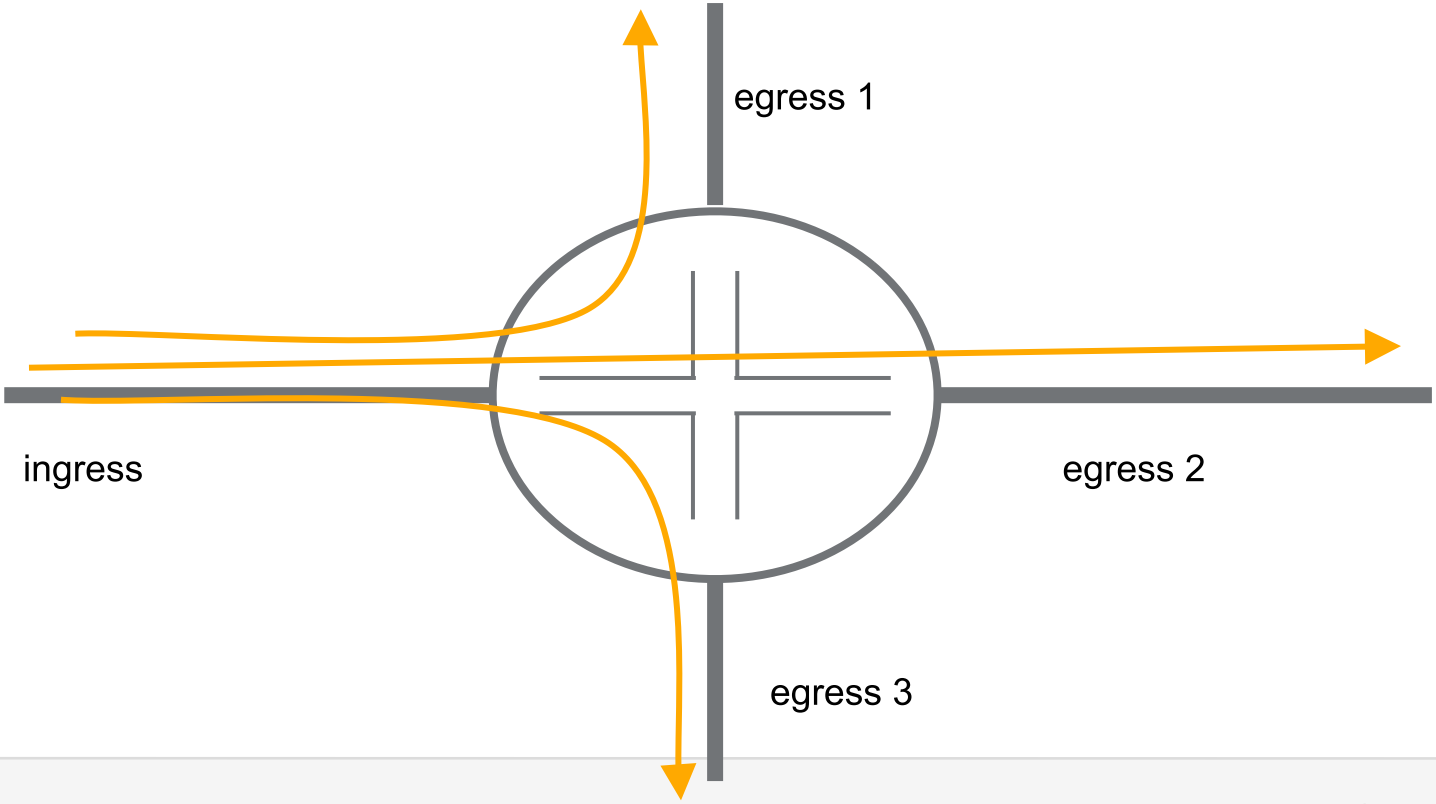
We could run an “IPSLA”-like process on each node...



Traffic generated from the CPU is not switched like the production traffic.

Exhaustive coverage

More importantly, instead of just testing interfaces and nodes, we test the ability for a node to forward a packet each ingress interface to each other egress interface.



Exhaustive coverage

We can't just rely on destination based routing, otherwise only the best paths between two locations would get tested.

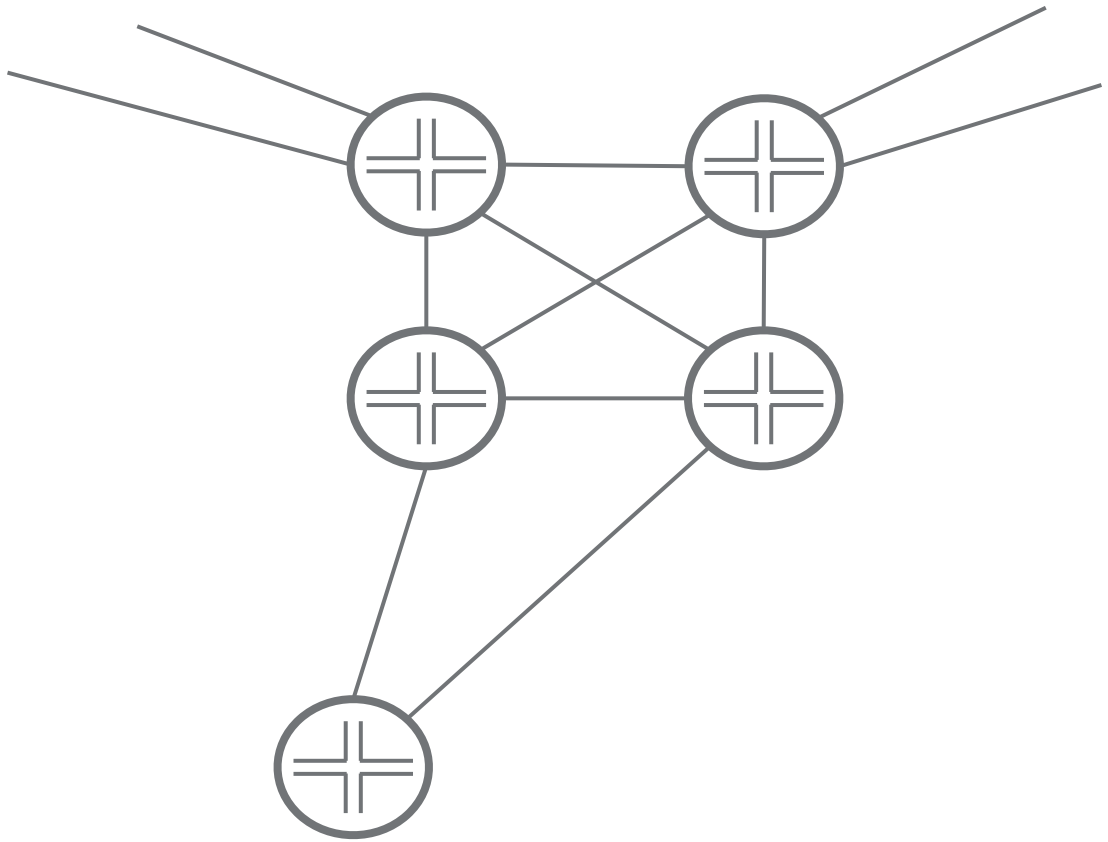
We source route the test packets instead.

With source routing, we can target what gets monitored and ensure full layer-3 coverage.

Google

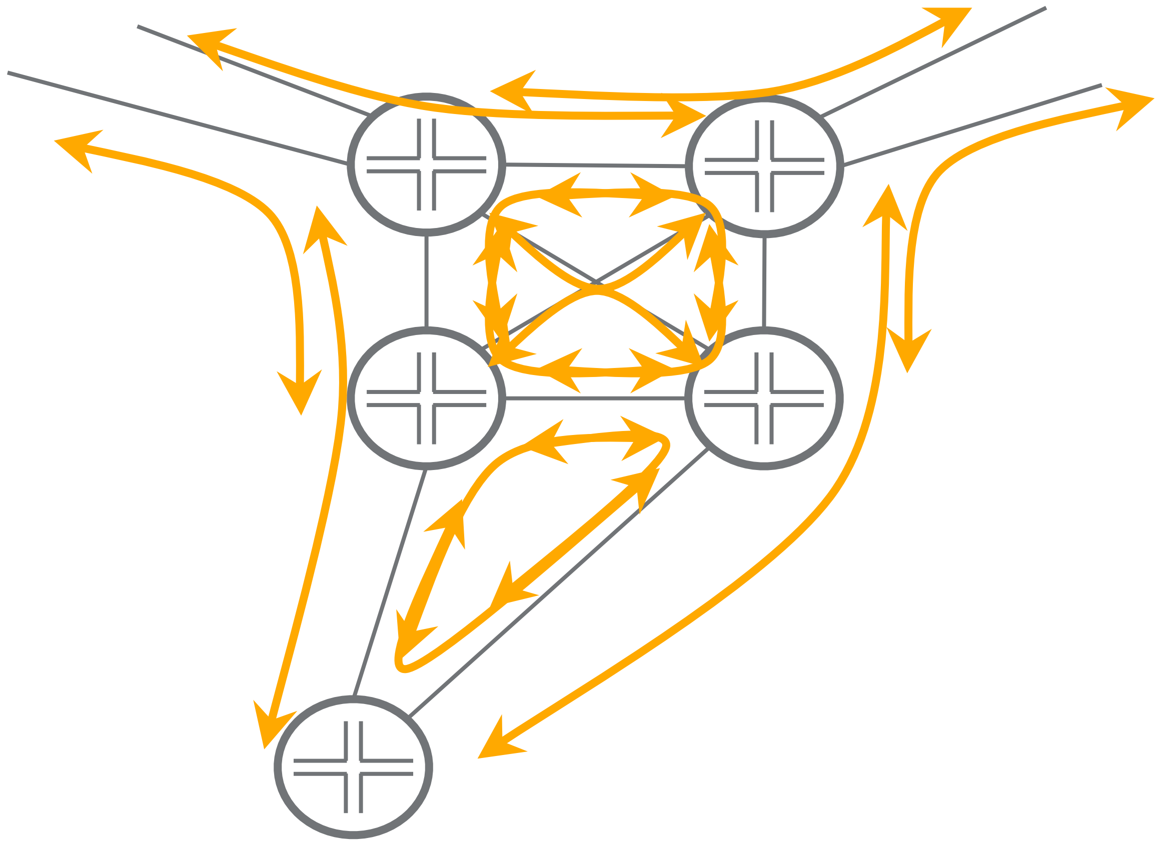
Exhaustive coverage: Testing every forwarding path

Quick illustration of what the coverage of a typical Core, Distribution, Access topology would look like.



Exhaustive coverage: Testing every forwarding path

Quick illustration of what the coverage of a typical Core, Distribution, Access topology would look like.



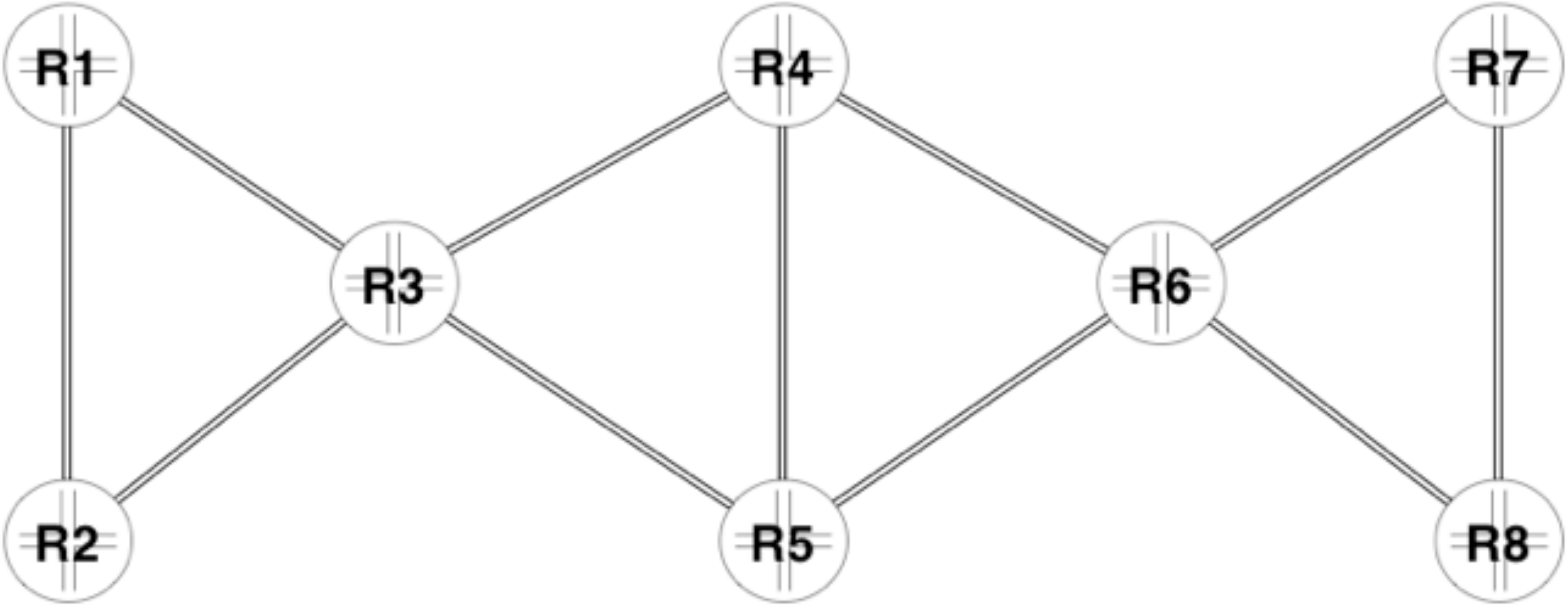
How do we localize faulty components, automatically?

Google

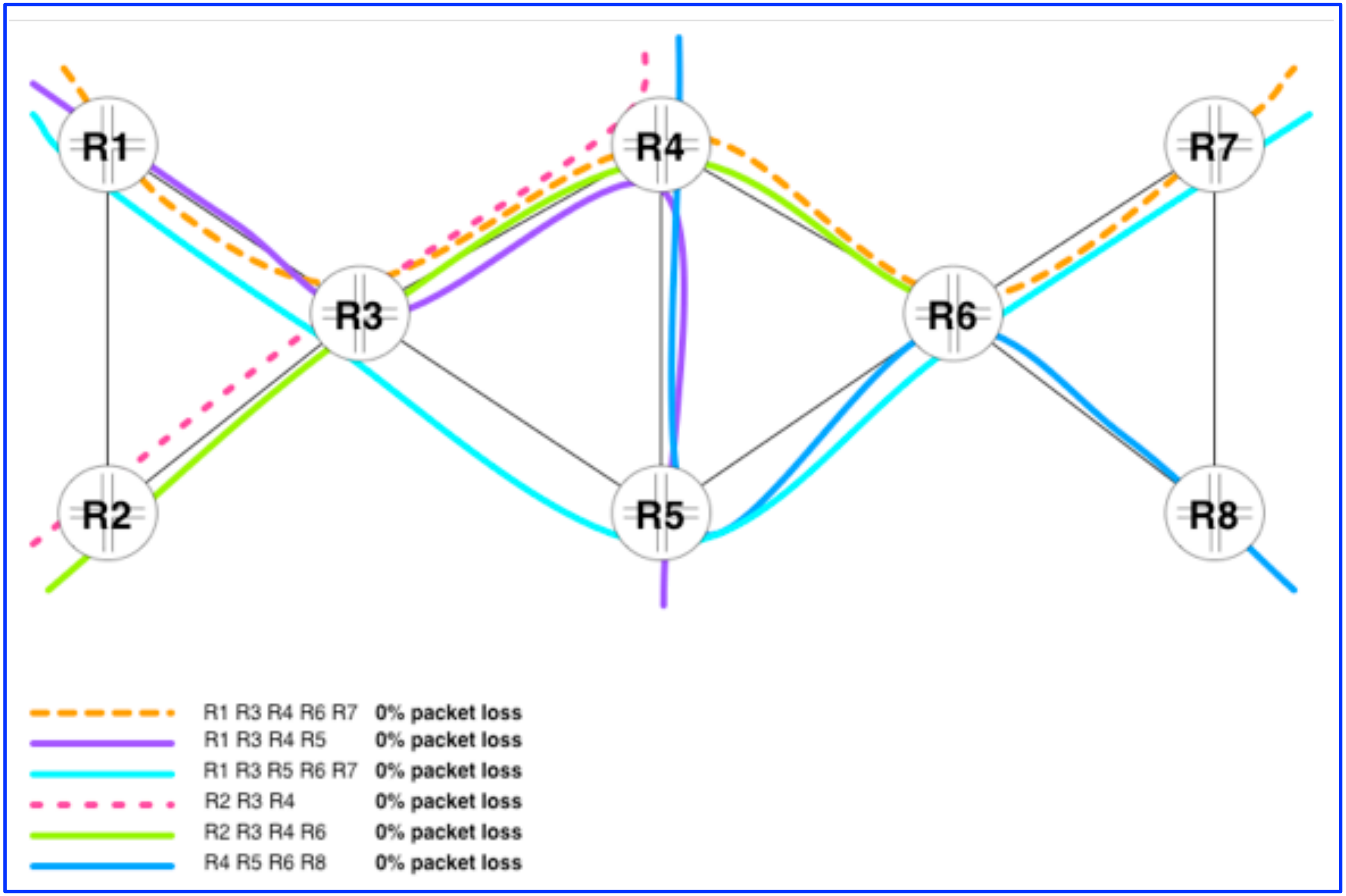
We have a correlator. It must:

- find the faulty links;
- have very few false positives or negatives - crying wolf means alerts will be ignored;
- calculate a magnitude for the fault; and
- log test results over time.

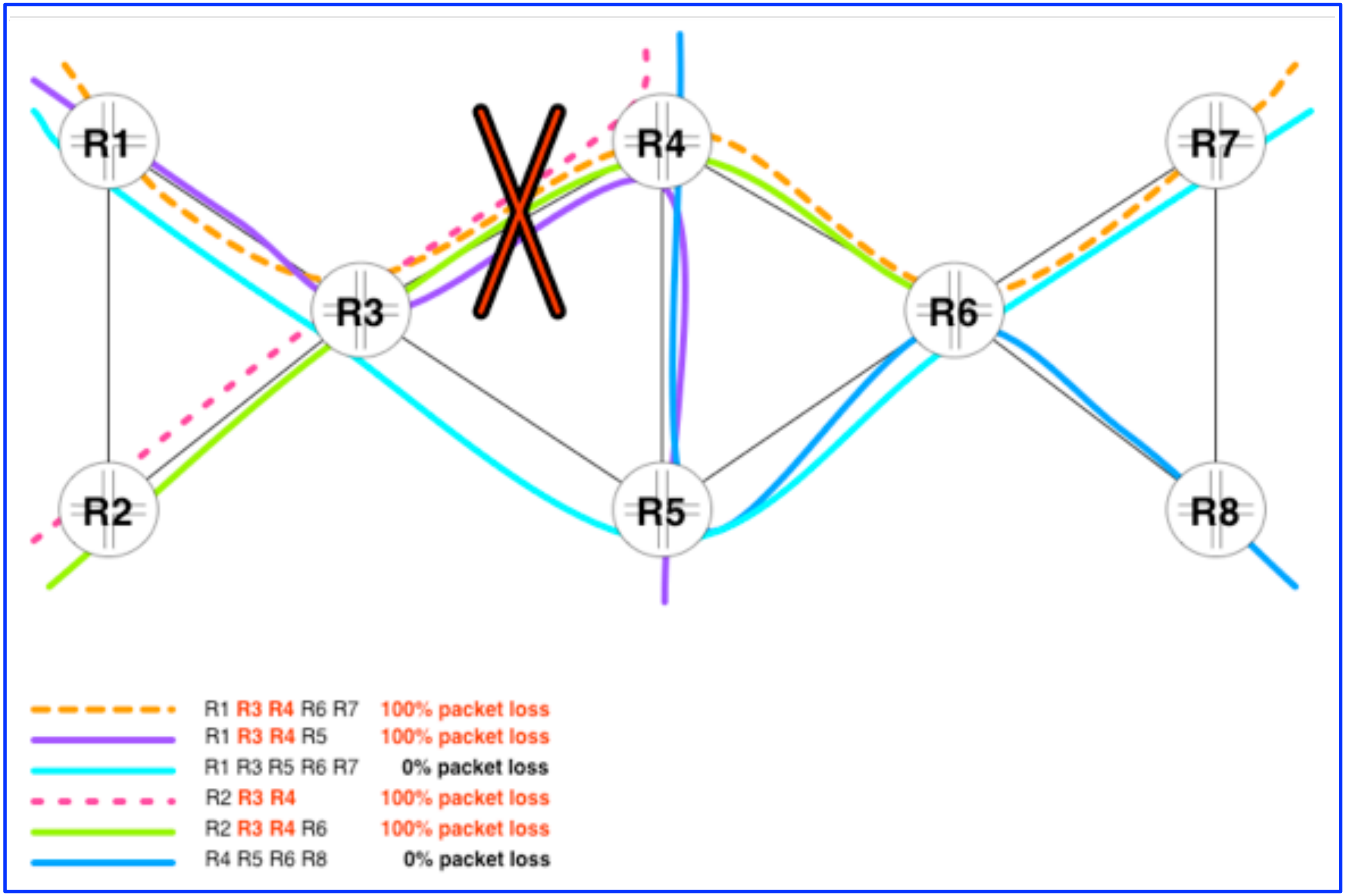
Simple network example



Simple problem



Simple problem

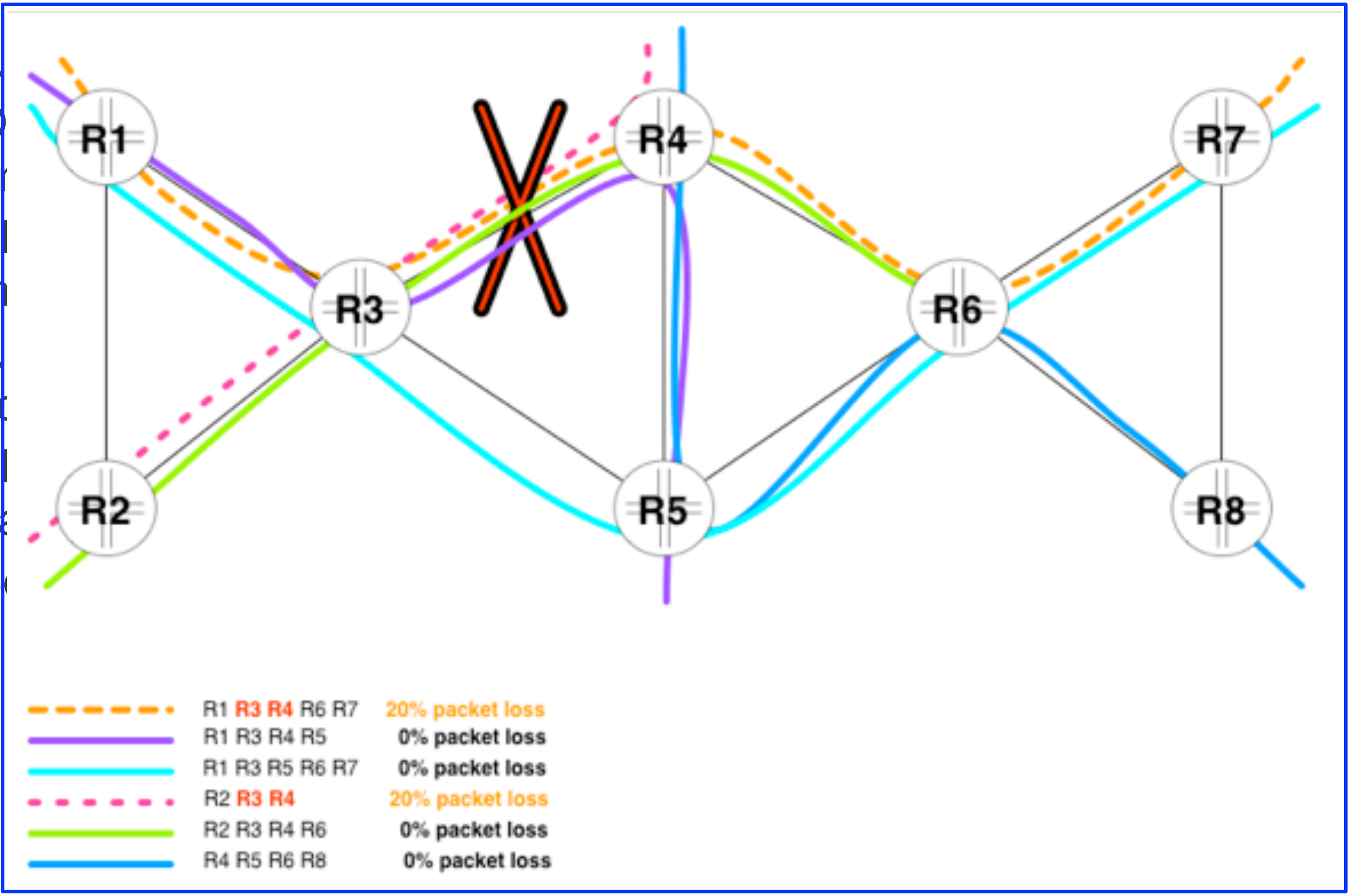


Not so simple...

- A logical link could be a bundle of many physical links
- One path through a link could go through a good physical link while another could go through a bad link
- The hash is not known - and hash salting to eliminate polarisation makes it nearly impossible to predict
- A link may have a low packet loss so many paths through the link could be clear, while others could have loss
- The simple correlator would (and does) fail in these cases - it can't handle a link being both good and bad
- So how do we handle this duality?

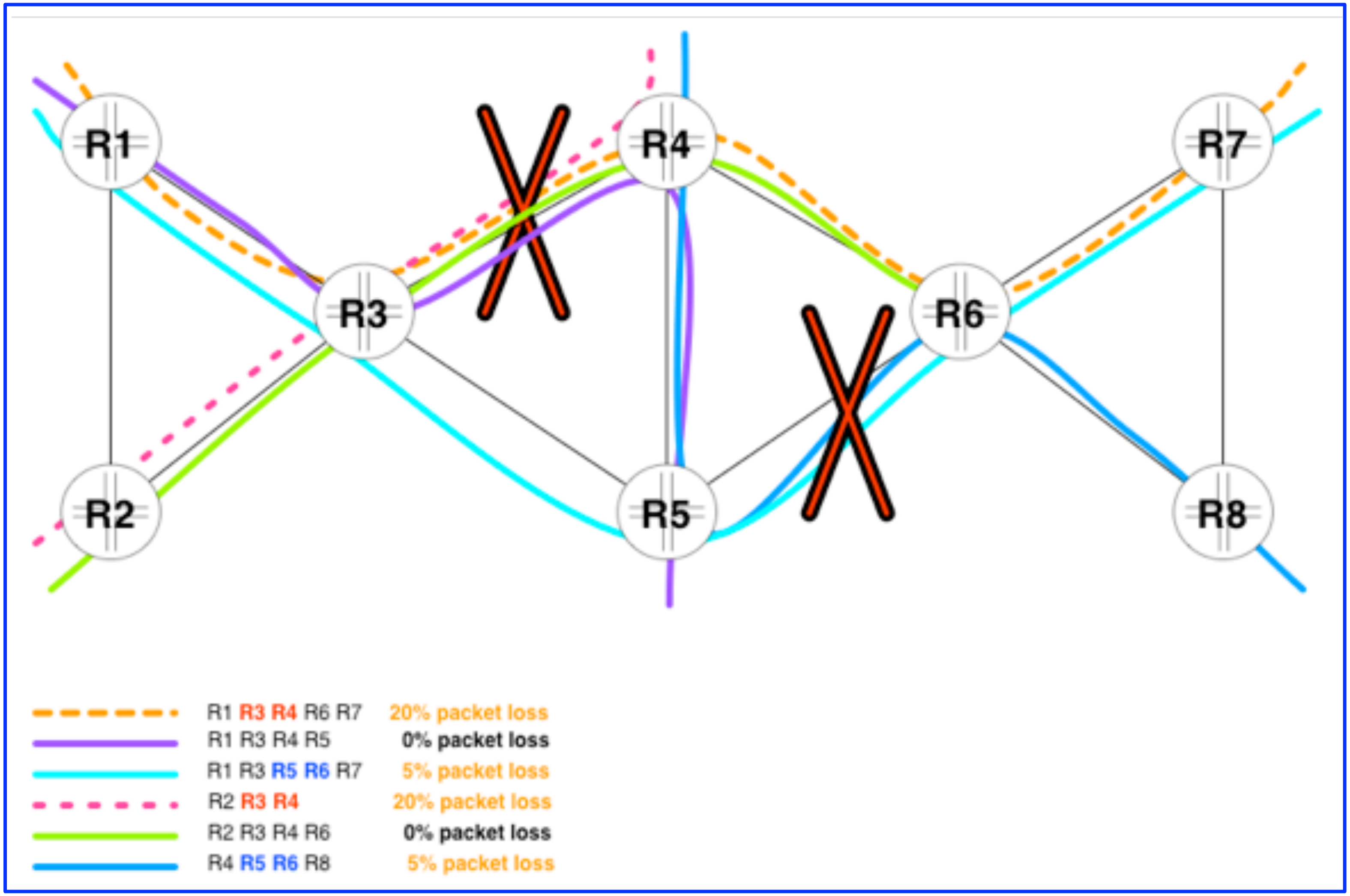
Not so simple...

- A
- O
  ar
- Tl
  m
- A
  cc
- Tl
  hi
- S



Google

Multiple faults in the network



What we tried first

- Average the loss and get a loss value for each link
- Bad links spread their loss to good links via paths they shared
- You could graphically see the places where loss was occurring, but this was way too fuzzy for NOC alerts
- We tried to sharpen with multi-passes. Remove the down links, recalculate the losses, remove the worst, ... repeat...
- Screamed "optimisation problem"

Framing the problem as an optimisation problem

- We have results for each path, the packet loss (0 to 100%)
- We can classify each path as either good or faulty
- A faulty path must be caused by a faulty link in the path

So the problem can be framed as:

"Find the set of links which, if faulty, best explains all the faulty paths."

- Traditional **cover** problem; just need to define this

Finding the best list of faulty links

- Much experimentation; lots of trial and error
- From testing on real problems we settled on a combination of:
    a. minimising the number of links creating our known faults; and
    b. biasing away from too simple solutions
- Framed as a linear programming problem
- Reliable and focussed results
- Then combined data from known-bad and known-good links, quantify fault severity
- Log all data vs. time, have dashboard for proactive monitoring
- Good enough to raise alerts to network operations - no false positives
- Network Operations trust the signal - we kept alerts to production problems

Detecting low level, intermittent loss

- Inspired by Radio Astronomy long exposures
- Add new timeseries thats reports highest loss on path over longer time (say 5m, 30m, ...)
- Correlate on this variable so if 5 different paths, say, have loss in 5 mins, but only 1 in any one 15s poll, it can still correlate unambiguously
- Trade off temporal accuracy for loss sensitivity
- We can have a hierarchy of alerts from high level, short term to low level long term
- In production now: found to be a very useful signal, particularly for detecting traditionally hidden low-level and intermittent losses

Performance vis-a-vis traditional monitoring

- Can precisely pinpoint a problem much better that with blackbox: we now know exact location of faults, not just the existence of a fault
- Very low pps is required for very accurate results:
  - To monitor *n* paths with the ability to detect loss lower than .01% we are only sending 20*n* pps
  - Orders of magnitude lower that our existing blackbox or whitebox monitoring
- We correlate our results to the existing blackbox and whitebox systems
- As we test what a device does rather than what it says it does, we get a more reliable indication of performance
- No need to craft whitebox monitoring for each element
- We get the accuracy of whitebox with the simplicity of blackbox for a lower overhead than both

How do we build a good set of paths?

(Informally, a "map".)

How the mapper works

What are some features of a "good" map?

- Ability to isolate multiple simultaneous faults
- Minimise degradation when faults occur
- Minimise the number of paths
- Spread the paths as even as possible across the routers
- Be easy to deploy to the network and update when the network changes
- Cover every link multiple times

How the mapper works

It's an NP hard problem, so we solve the problem heuristically:

- Enumerate all links
- Create paths to each link from multiple probers
- Follow approximate shortest path
- Add weights to used paths to push paths away from heavily used links
- Generate paths to least covered paths first
- Keep tabs of number of transit links and keep below a device specific maximum
- Stop once every link is covered by a minimum number of paths

How well did it work

- Paths with high diversity are created
- Scales near-enough linearly with number of links
- All links are covered
- Allows for incremental addition of paths after small network changes
- Handles multiple simultaneous failures very well
- Doesn't add too much state to the network

What's next

- Automatically update map as network changes
- Use feedback from the correlator to improve the path design
- Investigate other algorithms, heuristics for path generation

What did we learn ?

It works!

- It is working very well, we found problems that nobody knew about
- Silent drops are not that frequent but it does happen regularly
- The system finds low level packet loss well below 0.01%
- It currently takes about 60 seconds to localize a fault following its occurrence
- We can test components (interfaces, links, devices) **not yet in production**, because we use source based routing (in the form of RSVP-TE LSPs signaled with strict static EROs)
- We found RSVP signaling errors (bugs or shortcuts to improve convergence time)

Limitations

- The “pathing” is done at layer3, when covering aggregated links, we rely on the vendors hashing algorithm to map different flows onto different components. The mapper takes as a constraint that each “bundle” needs to be covered with a minimum amount of flows.
- It creates a lot of state. For a 16 interfaces device it creates a combination of at least 120 tests. When using RSVP-TE that results in 240 LSPs. Multiply that by hundreds for a large network and the RSVP state and amount of next-hops can become a problem. Mapping and correlation can become fairly complex to limit the amount of of state, especially on transit nodes.
- It takes a fair amount of processing power to:
  - create and optimize the mapping
  - create and send the probes on each test path
  - collect the results
  - correlate and report

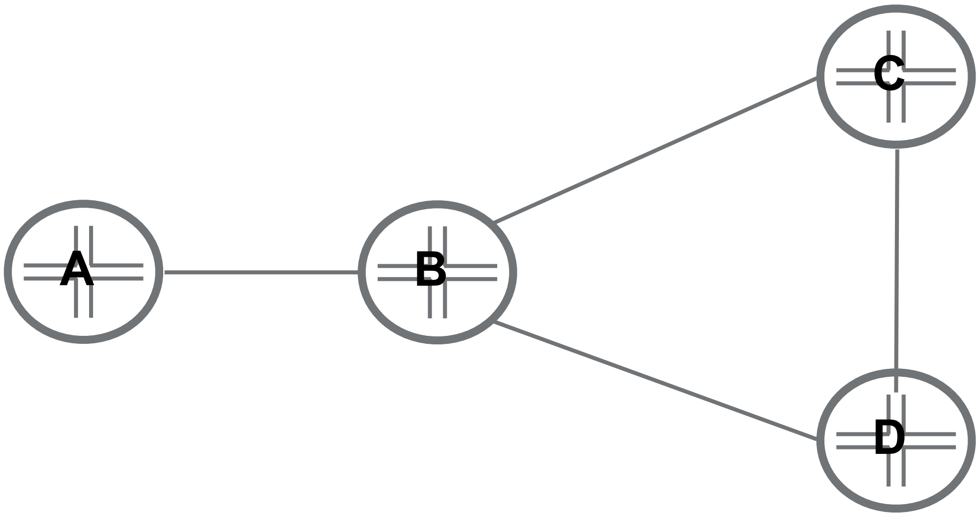
What's next ?

Reducing state

Reduce state

An alternative is **not to use RSVP** but keep the state in the test packets instead.

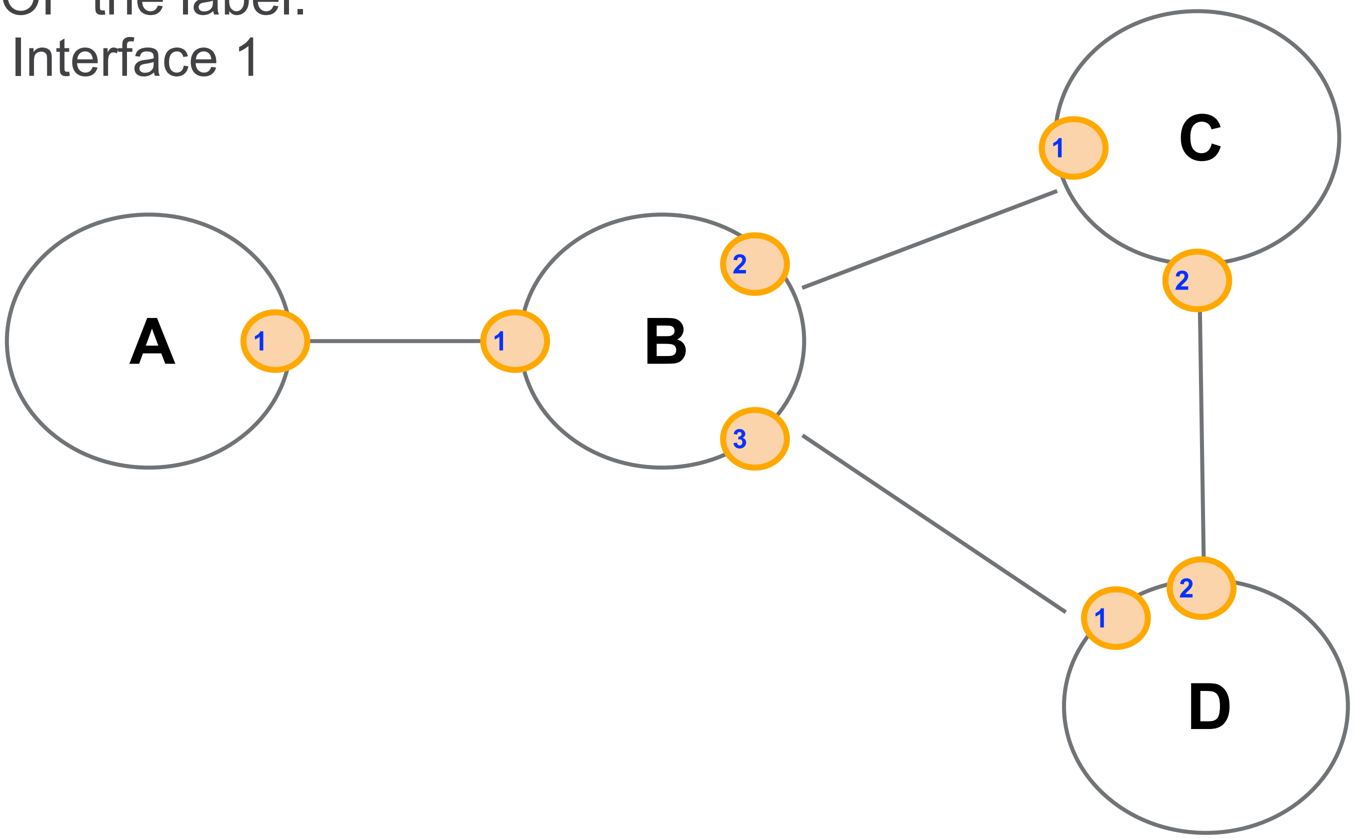
Let say we want to test:
A -> B -> C -> D -> B -> A



Reduce state: One hop static LSP

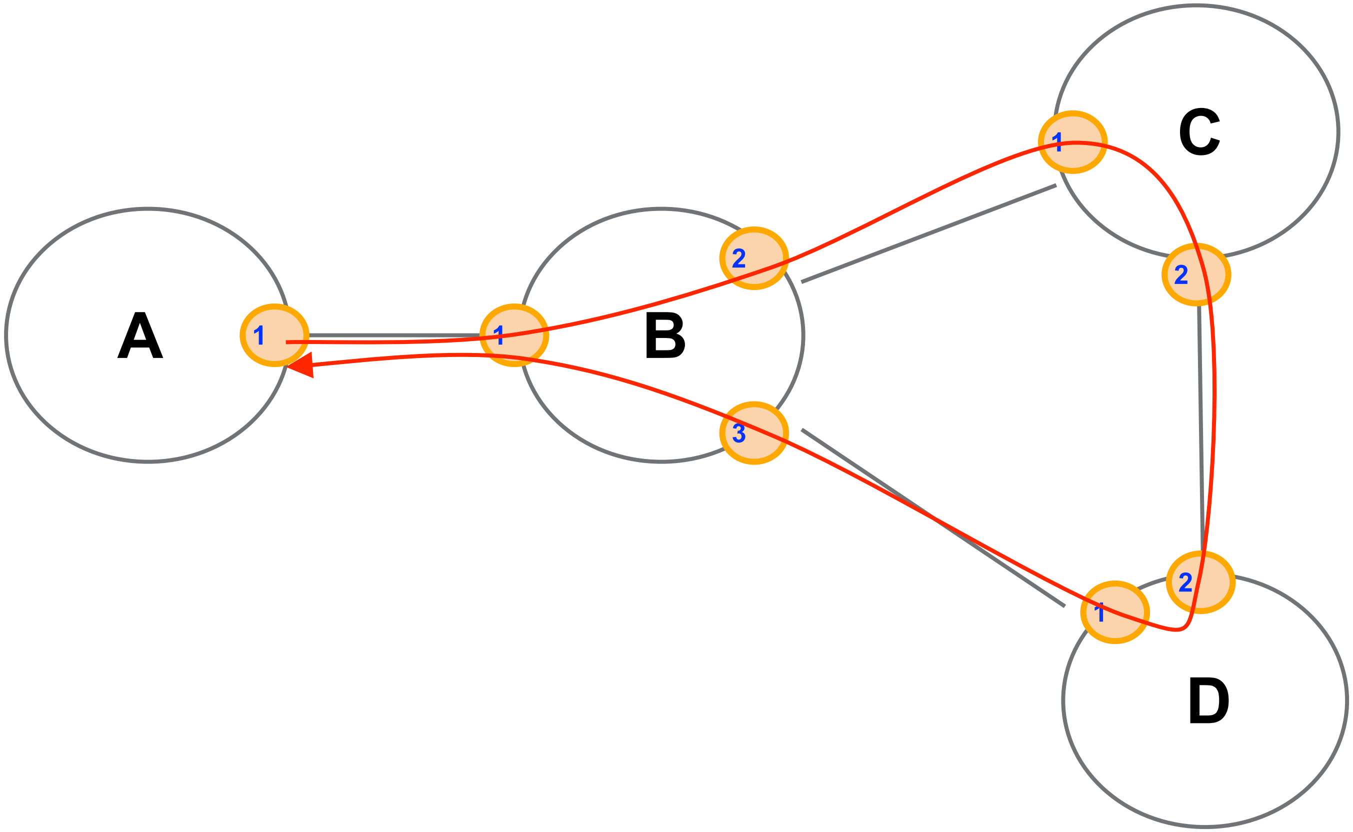
We can create static LSPs that direct traffic to a specific interface and POP the label.

1 Interface 1



Reduce state: One hop static LSP

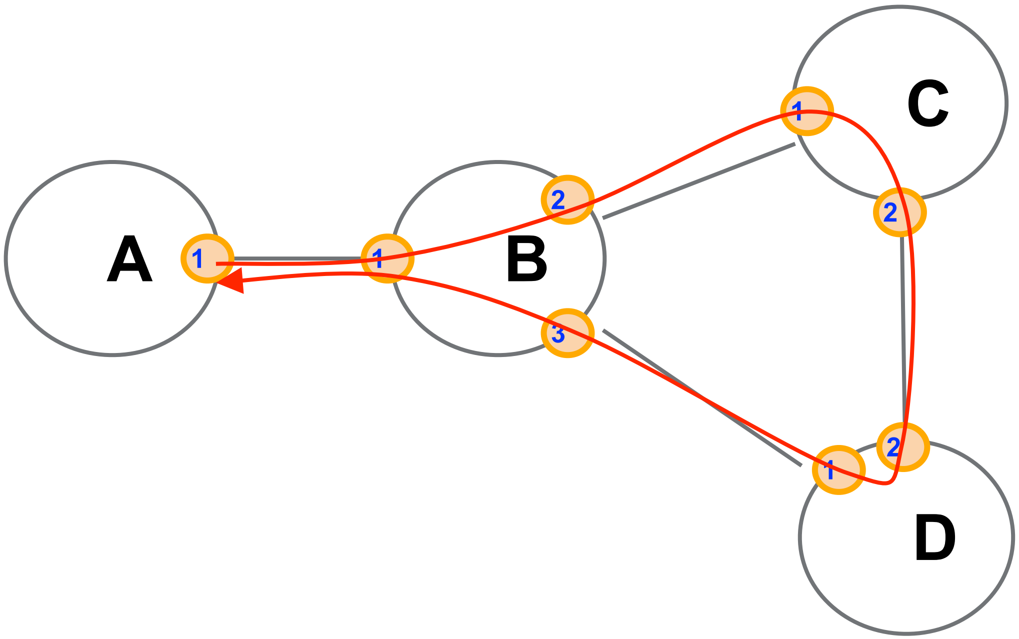
We want to send a packet through the following path:
A:1 ->B:1
B:2 ->C:1
C:2 ->D:2
D:1 ->B:3
B:1 ->A:1



Reduce state: One hop static LSP

We just build a packet with the following stacked labels:
[1, 2, 2, 1, 1(S)]

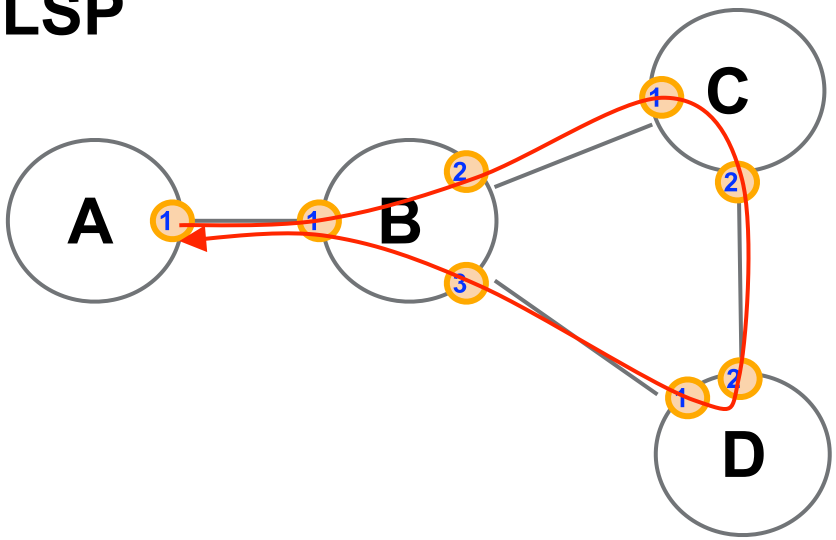
Router A has a static LSP that says:
For packets with incoming label 1, pop the label and forward to interface 1.



Reduce state: One hop static LSP

1. Router A: [1, 2, 2, 1, 1]
   =>POP label 1 and fwd to Router B
2. Router B: [2, 2, 1, 1]
   =>POP label 2 and fwd to Router C
3. Router C: [2, 1, 1]
   =>POP label 2 and fwd to Router D
4. Router D: [1, 1]
   =>POP label 1 and fwd to Router B
5. Router B: [1]
   =>POP label 1 and fwd to Router A

Router A looks up the IP dest address and sends the packet to its destination.



Questions?

lhuang@google.com
growly@google.com