# Localizing packet loss

## In a large complex network

Nicolas Guilbaud nguilbaud@google.com
Ross Cartlidge rossc@google.com

Google

# Traditional network monitoring: White box

White box monitoring, it's basically asking the device and monitor its vital parameters.

Unfortunately, this is far from being good enough, too many times the device either 'lie' or **does not tell you the whole picture**.

A classical example is packets corruptions being reported on the egress line card while being caused by a fault on the ingress to fabric connection.

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

Google

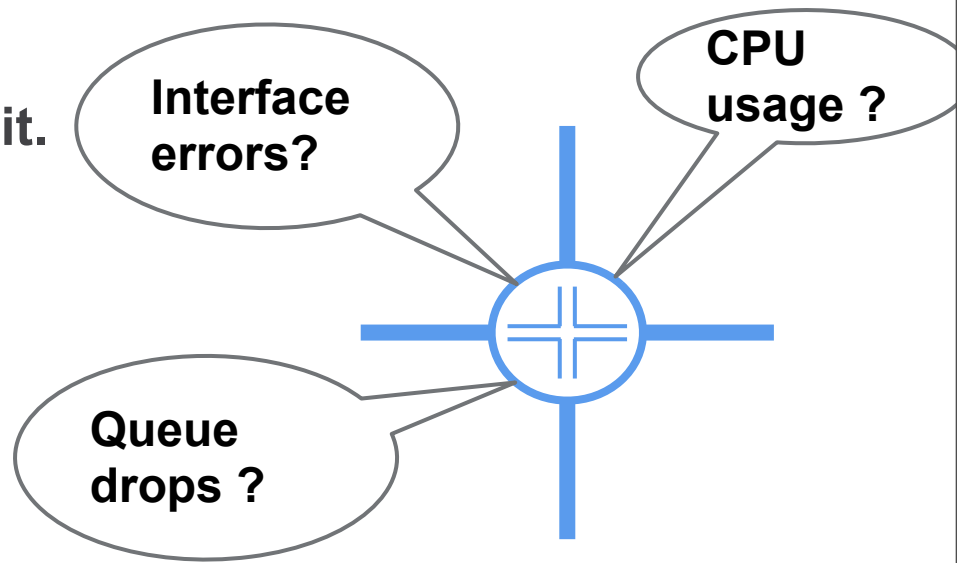
# Traditional network monitoring: White box

White box monitoring, it's basically asking the device and monitor its vital parameters.

Unfortunately, this is far from being good enough, too many times the device either 'lie' or **does not tell you the whole picture**.

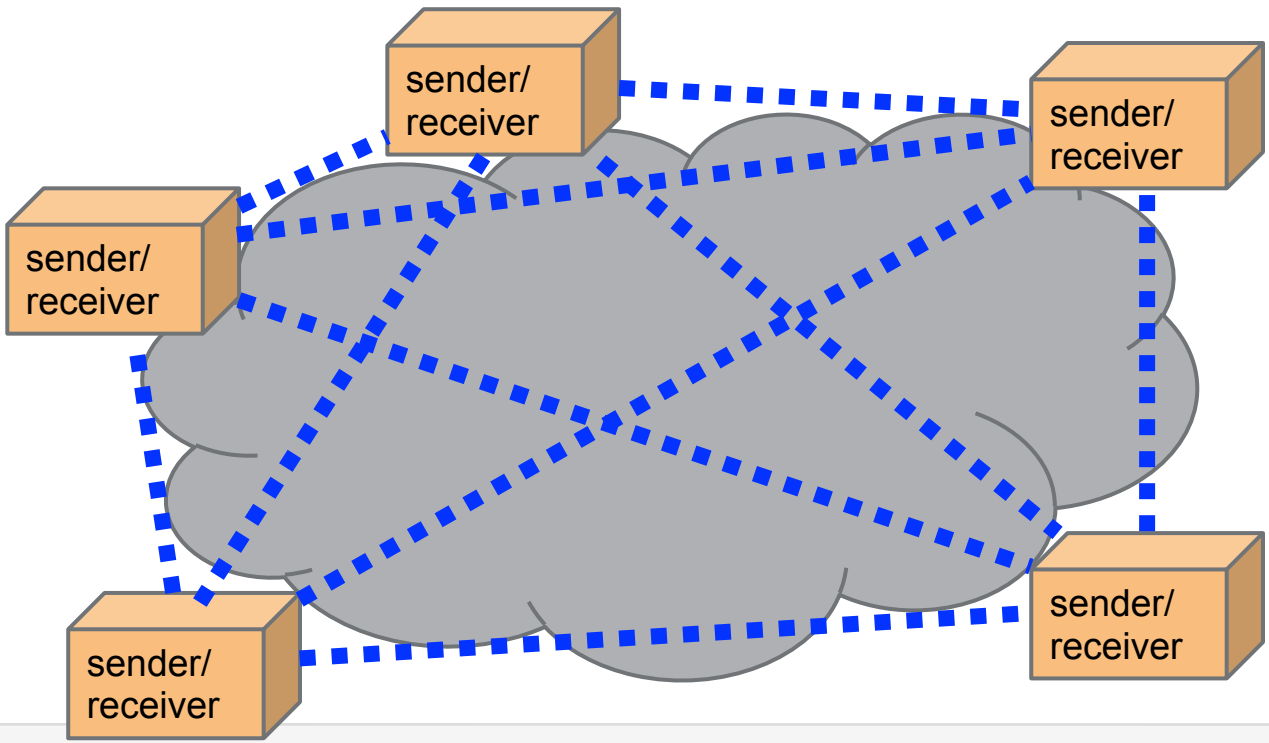
A classical example is packets corruptions being reported on the egress line card while being caused by a fault on the ingress to fabric connection.

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



# Traditional network monitoring: Black box

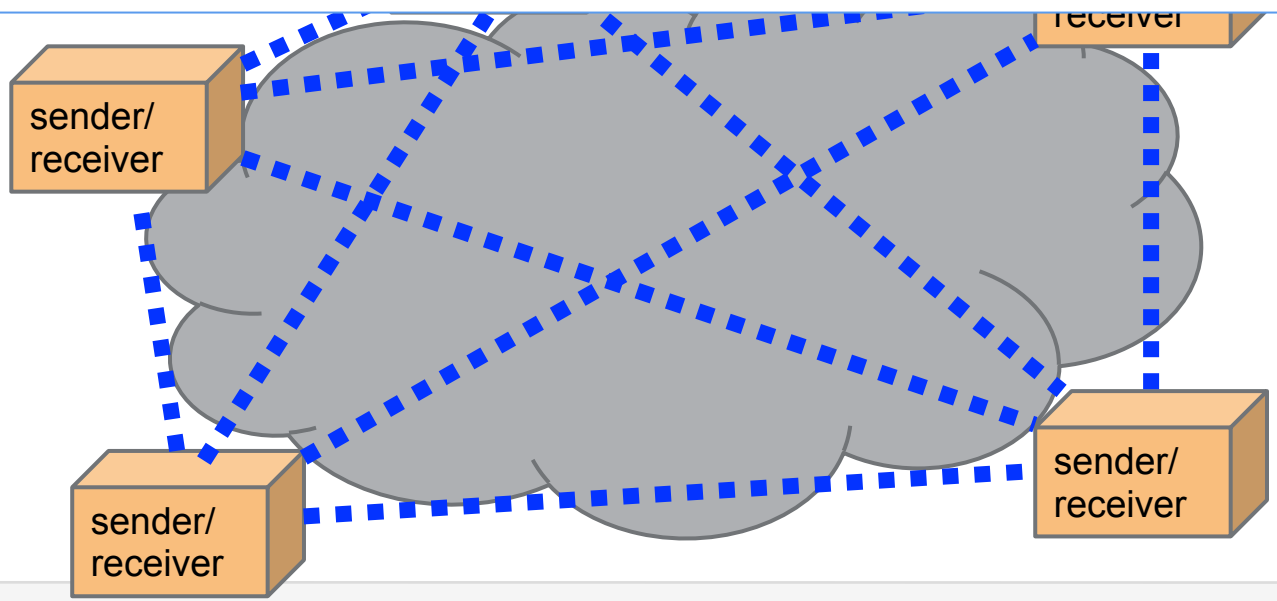
Black box monitoring consists in sending synthetic traffic that mimics production traffic and analyse characteristics such as packet loss, latency, jitter, packet corruption, CoS misclassification,...



# Traditional network monitoring: Black box

Two major drawbacks:

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



Google

# But that's not good enough

We want more, we want:

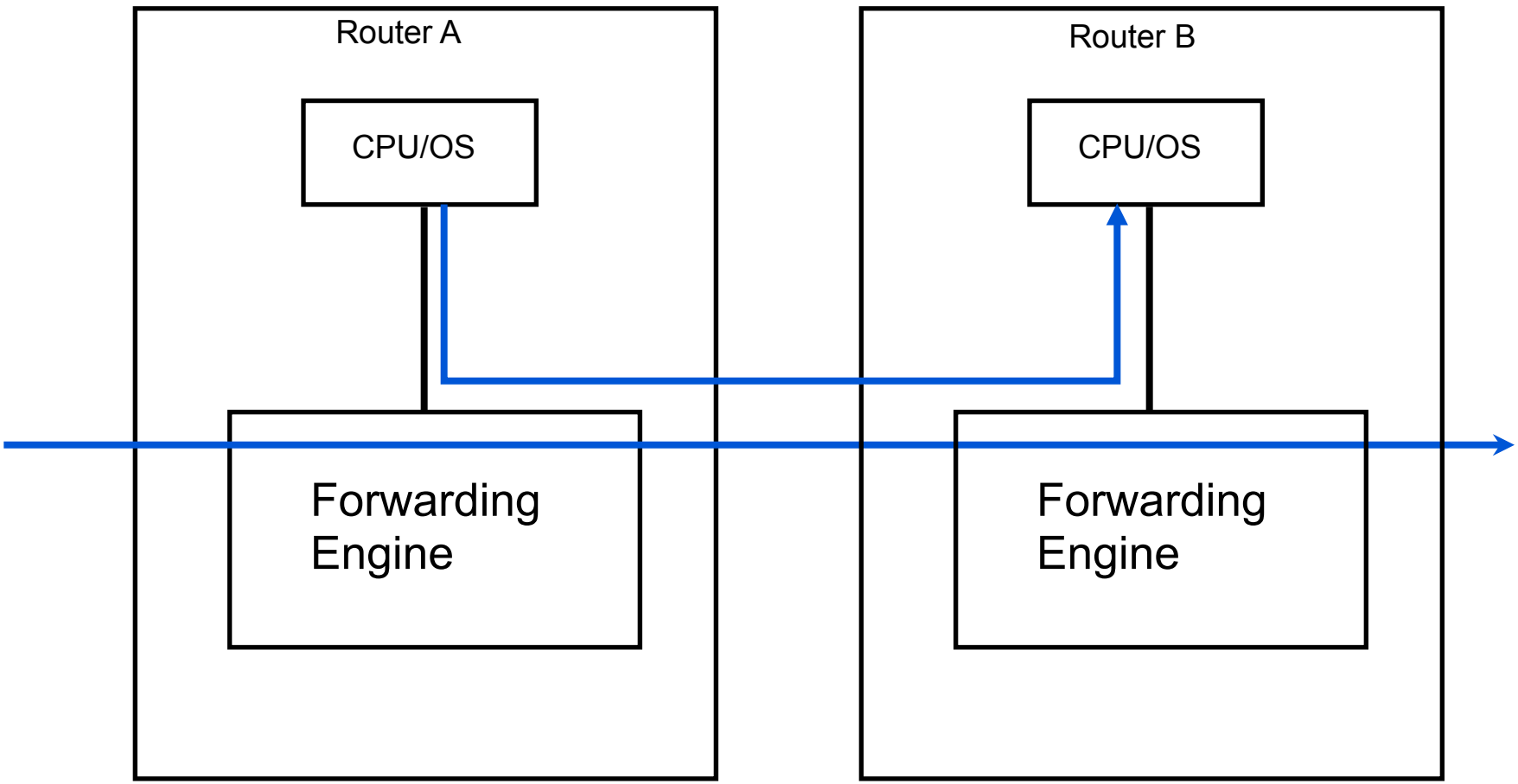
- We complete coverage. We want to test every single path in the network. Not only the best paths.
- We want to localize the fault in near real-time (within a minute from the event)
- We want to test the ability for a router to forward traffic, even when it's not part of the protocol topology.

# How do we cover every component ?

not only the best paths

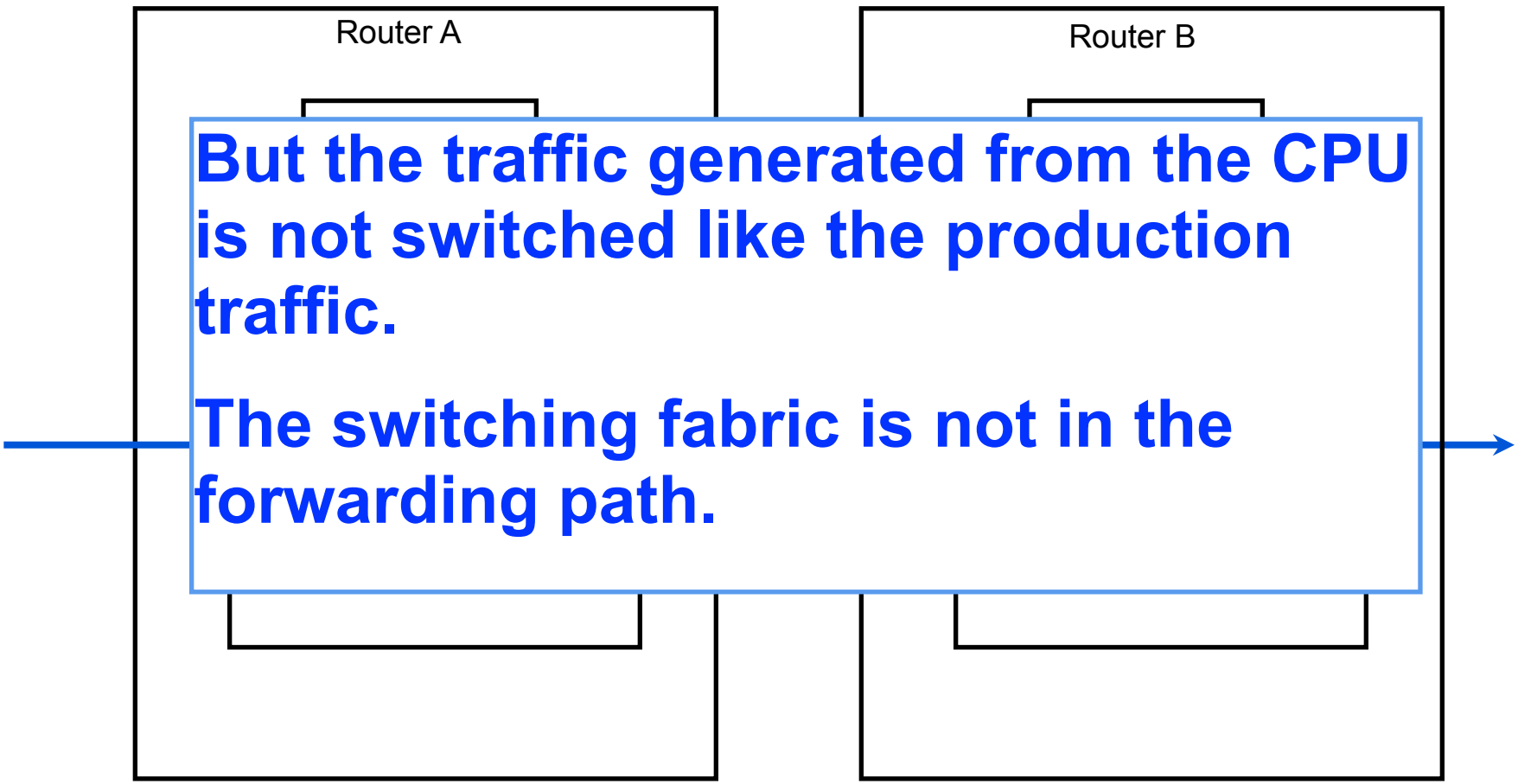
Google

# We could be running “IPSLA” like process on each node..



Google

# We could be running “IPSLA” like process on each node..



# 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 layer3 coverage.

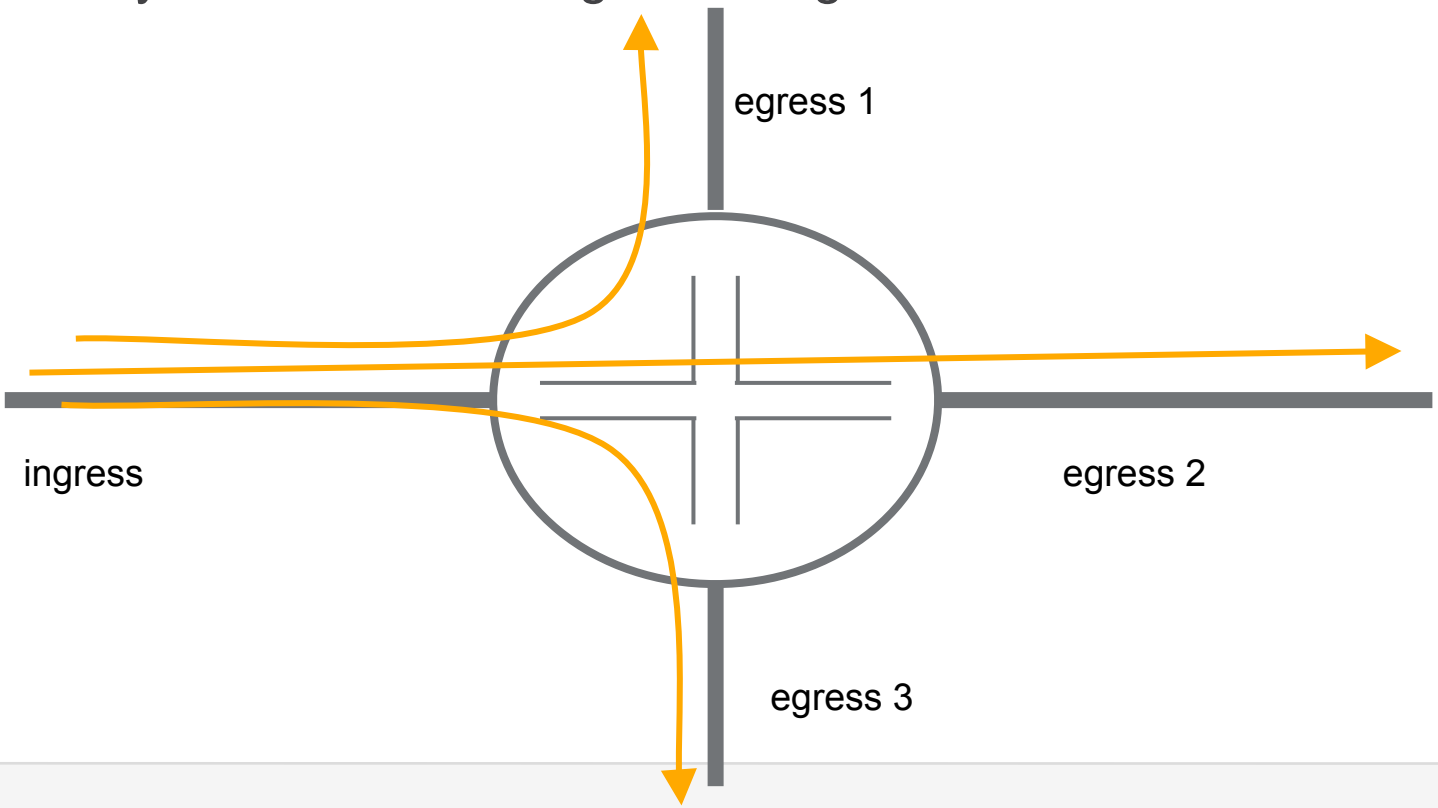
For layer3 paths composed with link aggregation groups, we cover them by creating *n* distinct flows per individual link (today we use *n*=4). The flows need to match the hashing algorithm configured.

Google

# Exhaustive coverage.

More importantly, instead of testing simply interfaces and nodes, we test the ability for a node to forward a packet from every ingress to every egress interfaces.

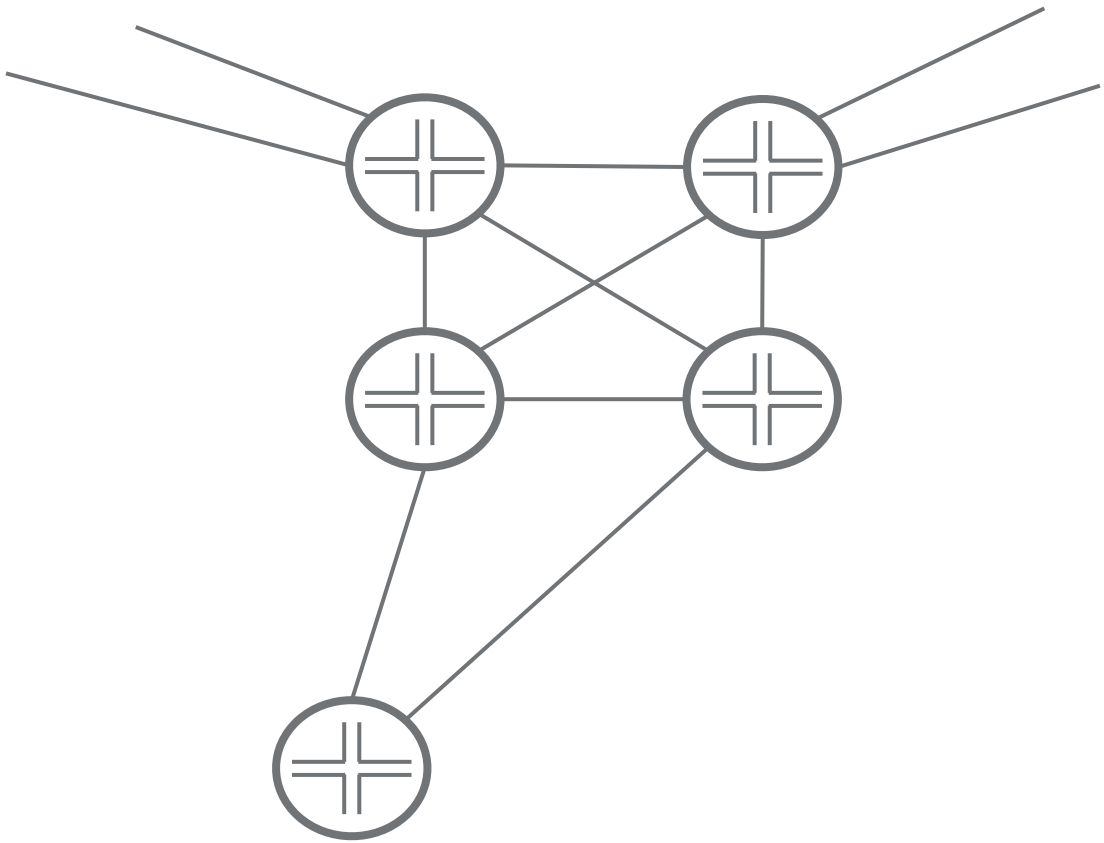
We test every combination of ingress to egress for each node.



Google

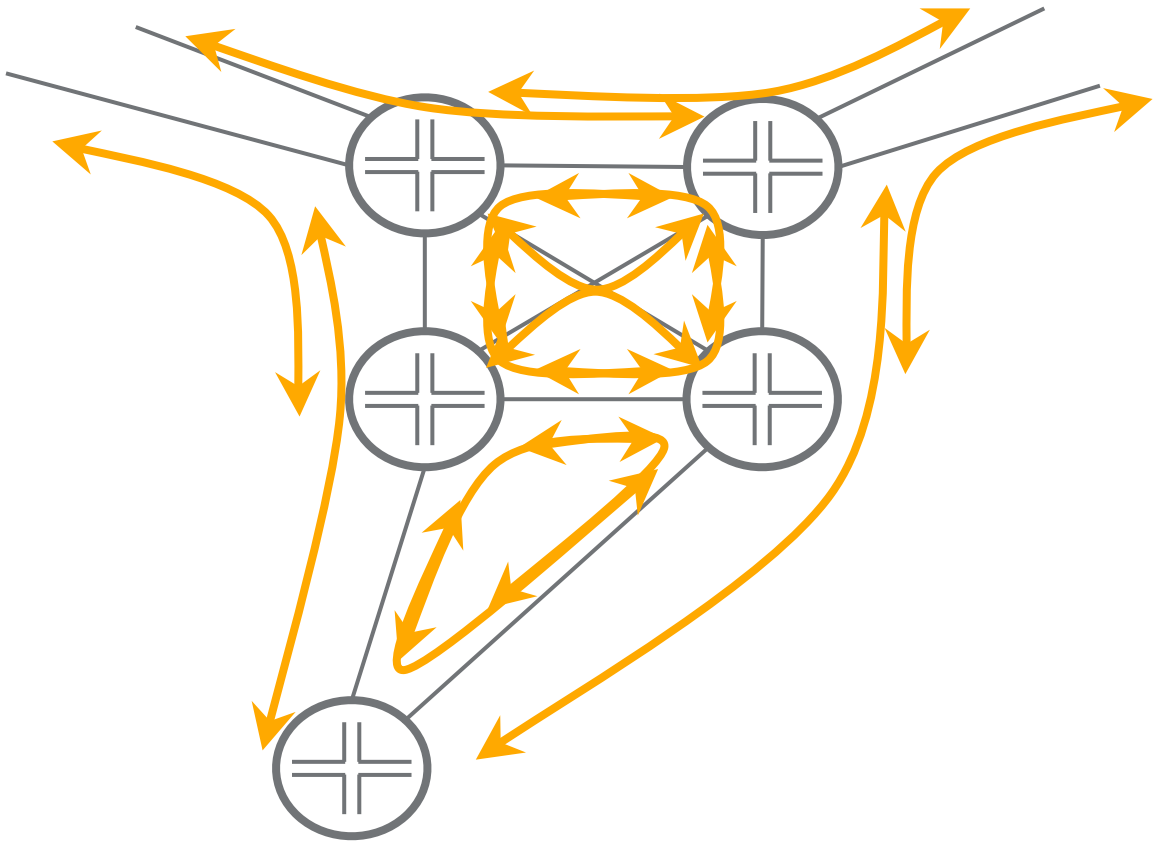
# Exhaustive coverage: Testing every forwarding path

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



# Exhaustive coverage: Testing every forwarding path

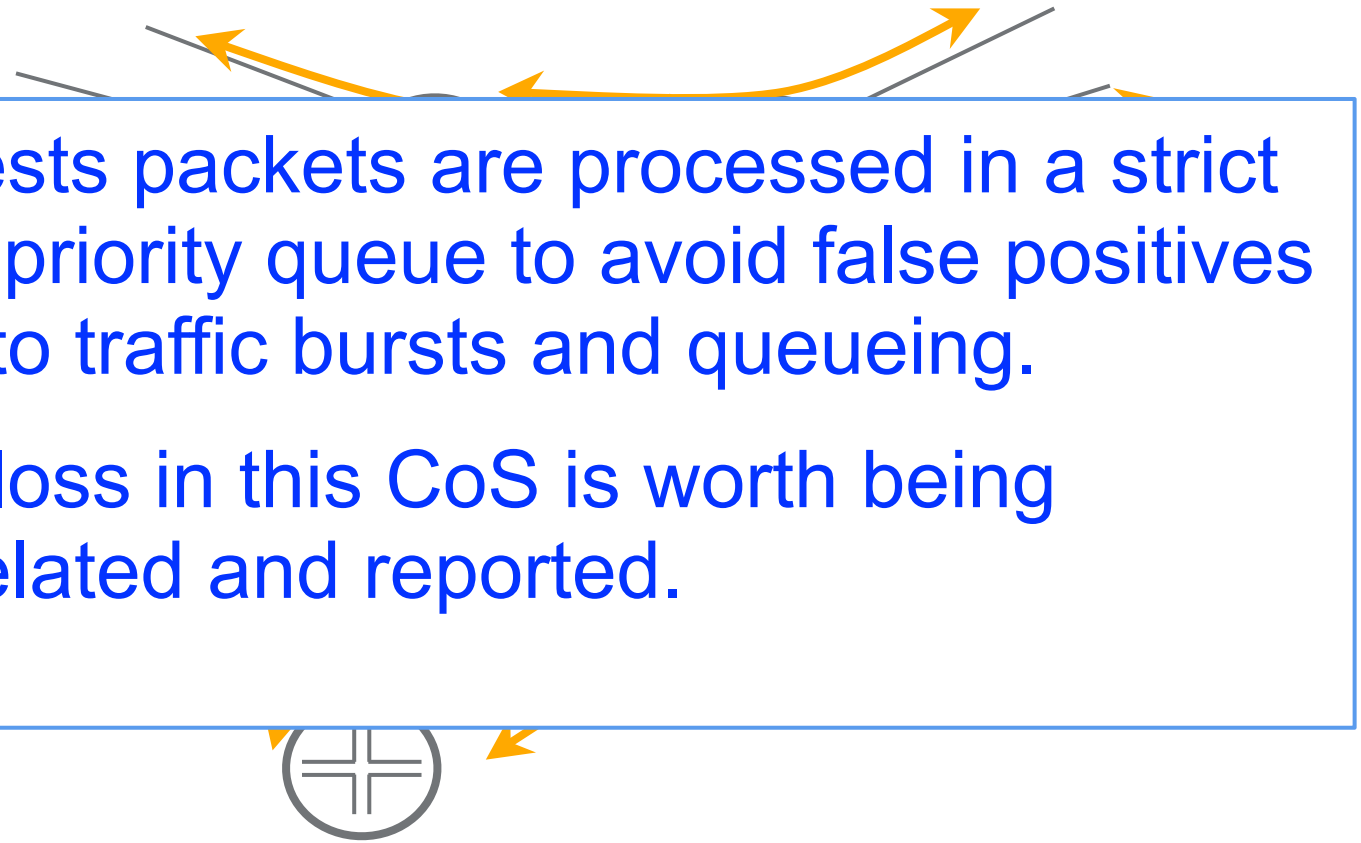
Quick illustration of what the coverage of a typical Core, Distribution, Access topology would look like.



Google

# Exhaustive coverage: Testing every forwarding path

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



All tests packets are processed in a strict high priority queue to avoid false positives due to traffic bursts and queueing.

Any loss in this CoS is worth being correlated and reported.

# How do we localize faulty components ?

Without manual troubleshooting.

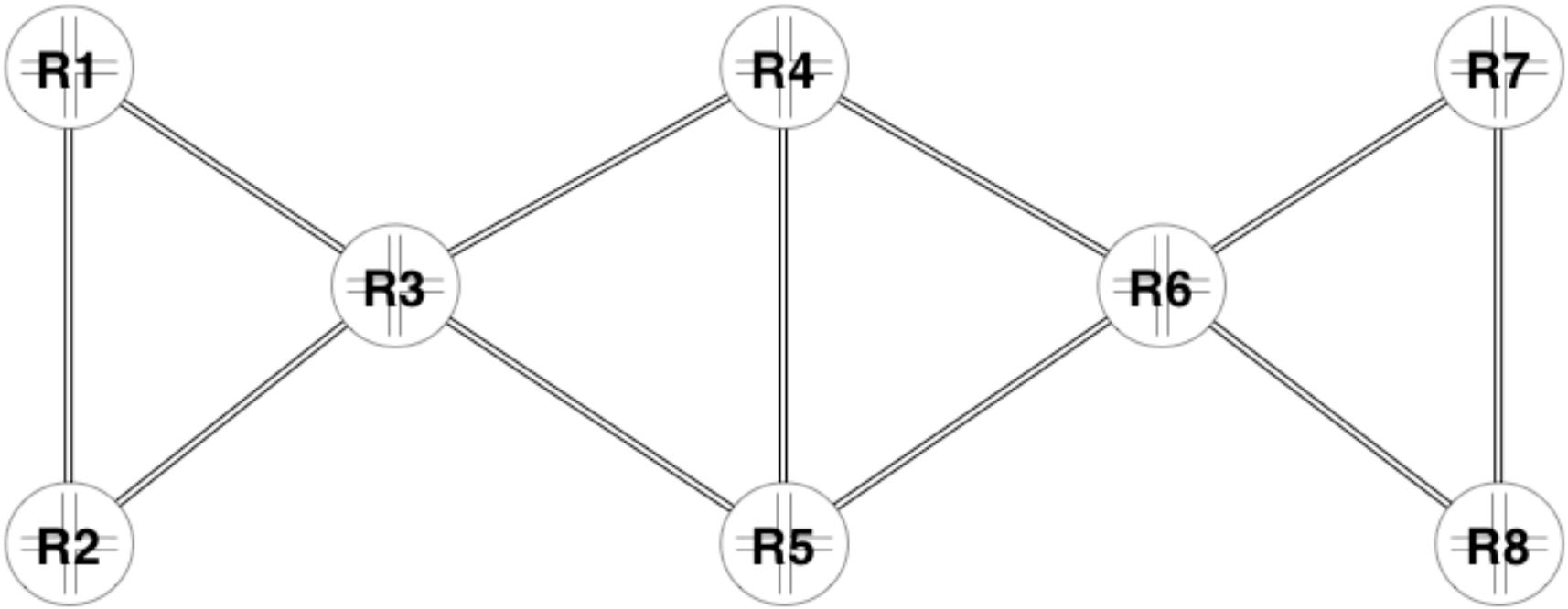
Google

# The role of the correlator.

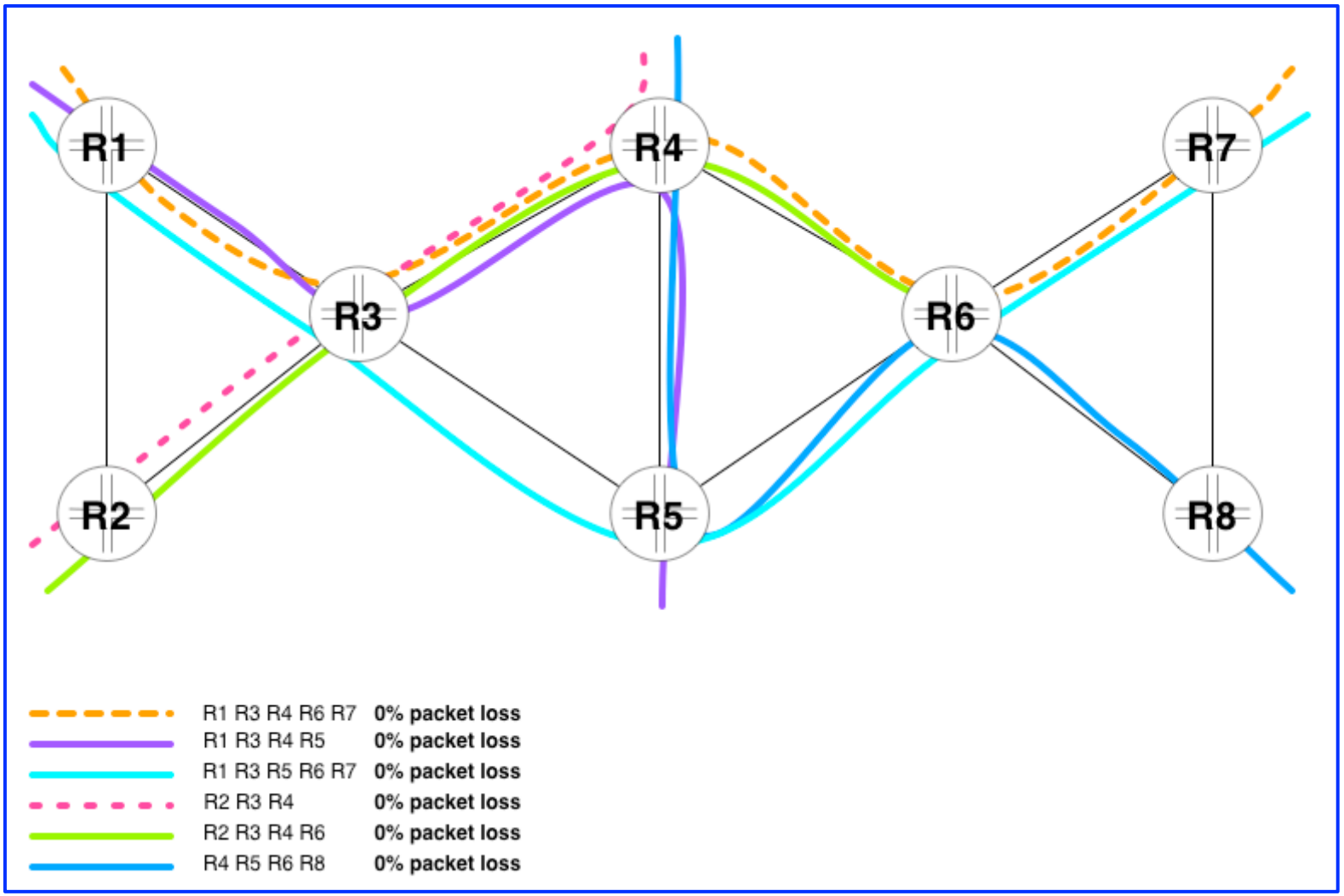
- It must find the faulty links
- Have very few false positives or negatives. Crying wolf means alerts will be ignored.
- Calculate a magnitude for the fault.
- Create time series.

Google

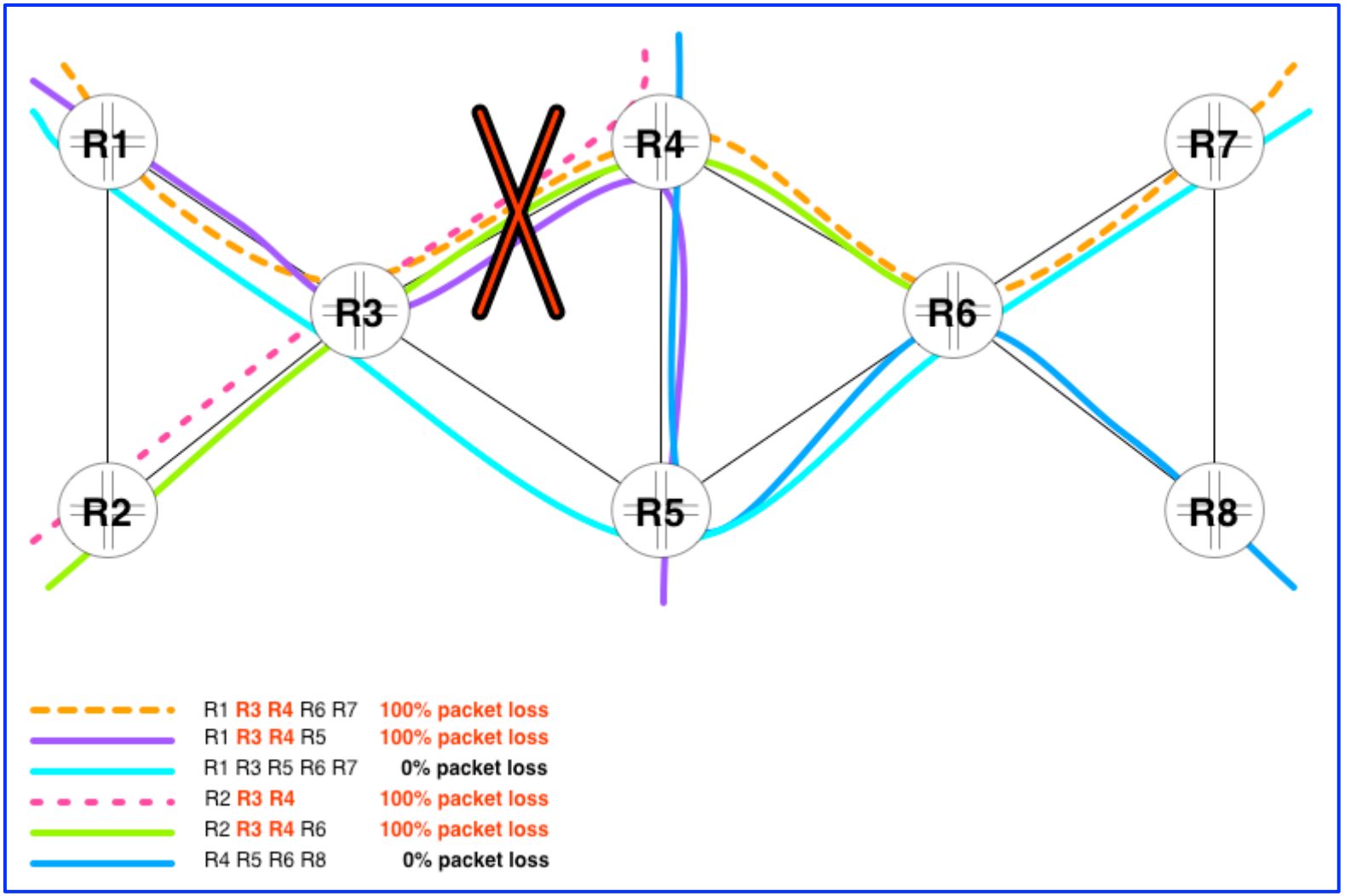
# Simple Network Example



# Simple problem



# Simple problem



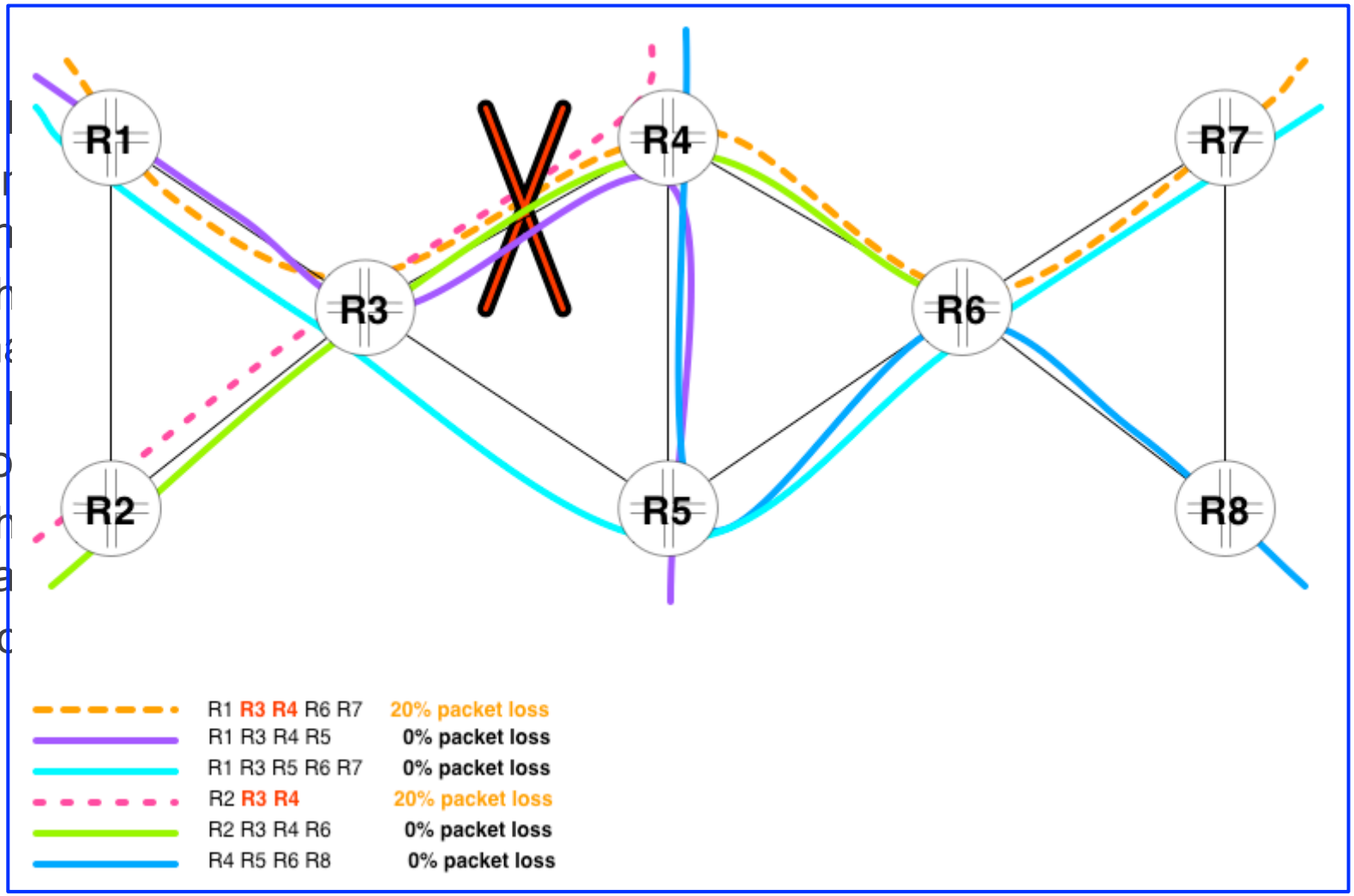
Google

# Not so simple :-)

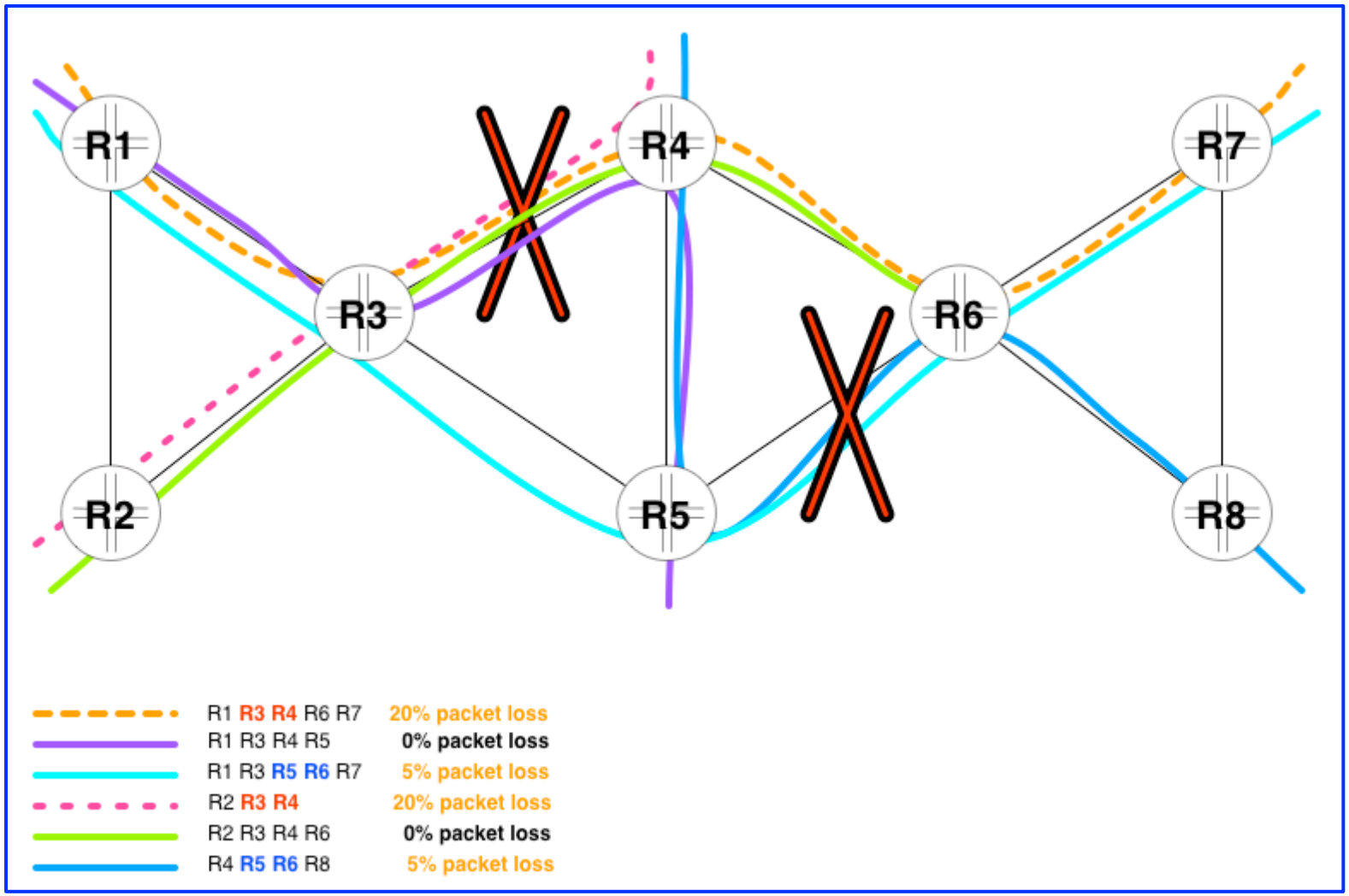
- A logical link could be a bundle of 8 physical links
- One path through a link could go through a good physical link while another could go through a bad link.
- The hash is not known - and hash salting to eliminate polarisation makes it nearly impossible.
- A link may have a low packet loss so many 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 l
- Or
  an
- Th
  m
- A l
  co
- Th
  ha
- Sc



# Multiple faults in the network



Google

# What we tried first

- They are not black and white so lets use grey :-)
- Look at each path through each link
- Average the loss and get a loss value for each link
- Bad links spread their loss to good links via paths they shared
- Viewed graphically you could see the places where loss was occurring but way too fuzzy for NOC alerts.
- We tried to sharpen with multi-passes. Remove the down links, recalculate the losses, Remove the worst, .... Rinse and repeat...
- Screamed of an optimisation problem :-)

Google

# Framing the problem as an optimisation problem

- We have results for each path, the packet loss from 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 best list of links that explains each faulty path"
- Traditional **cover** problem.
- Just need to define **best**

Google

# Finding the best list of faulty links

- This was the most experimental part
- 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
  b. Making the results as unambiguous as possible
  c. Biasing away from too simple solutions.
- We framed this as a linear programming problem
- Delivered very reliable and focussed results
- Good enough to raise alerts to network operations. No false positives
- Network Operations trust the signal. Kept alerts to production problems
- Also maintained a dashboard to allow pro-active monitoring

Google

# Using the good paths

- We only used the faulty paths to find the faulty links
- What can the the good paths be used for?
- Once we know where the bad links are we can then attribute all loss on a path to the bad links in each path
- So we can use the good paths to calculate the magnitude of the fault.
- For example, if a link has 10 paths traversing it each path has sent 10 packets and 1 path only receives 8 back then the loss is 1 - (10*9+8)/10*10 = 0.012 or 2%.
- Allows resolution far better than supplied by any one path.
- Create a timeseries of packet loss for each link in the network

Google

# Low level loss

- How to detect losses that doesn't happen every 15s?
- 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 as existing blackbox and whitebox monitoring is not good at very low-level and intermittent loss.

Google

# Performance vis-à-vis traditional monitoring.

- Precise pinpointing of problem is much better that blackbox. We know exact location of faults, not just the existence of a fault
- Very low pps is required for very accurate results. To monitor *n* links 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.
- 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.
- We correlate our results to the existing blackbox and whitebox systems.

# How do we build a good set of paths?

Google

# What is a good map?

- Create paths from prober to prober and from prober to devices
- Cover every link multiple times
- 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 u[date when the network changes

Google

# How the mapper works

- It is an NP complete 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 links 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

Google

# How well did it work ?

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

Google

# What's next ?

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

# What did we learn ?

Google

# It works

- It is working pretty 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 down to 0.01%
- It takes about 60 seconds with our current deployment, from fault to localization
- 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 nexthops can become a problem.
  The "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

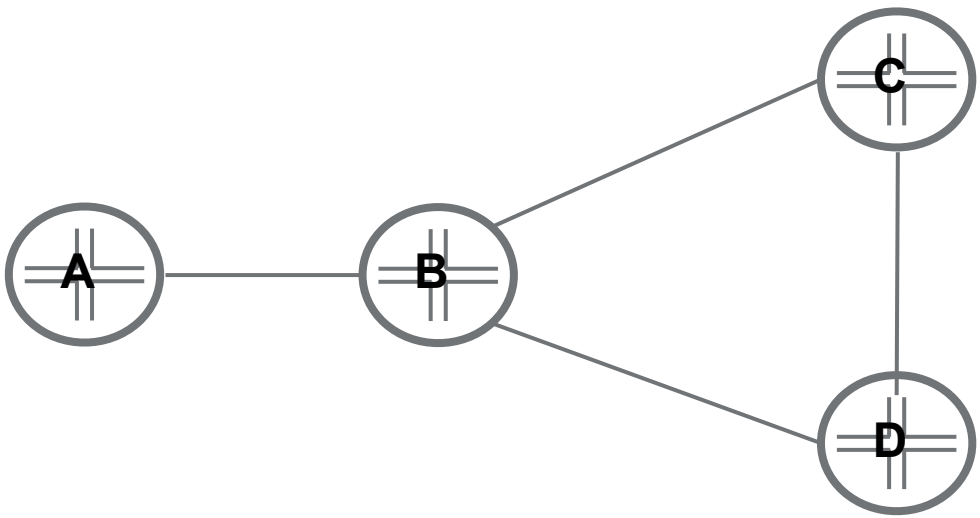
# What's next ?

## 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

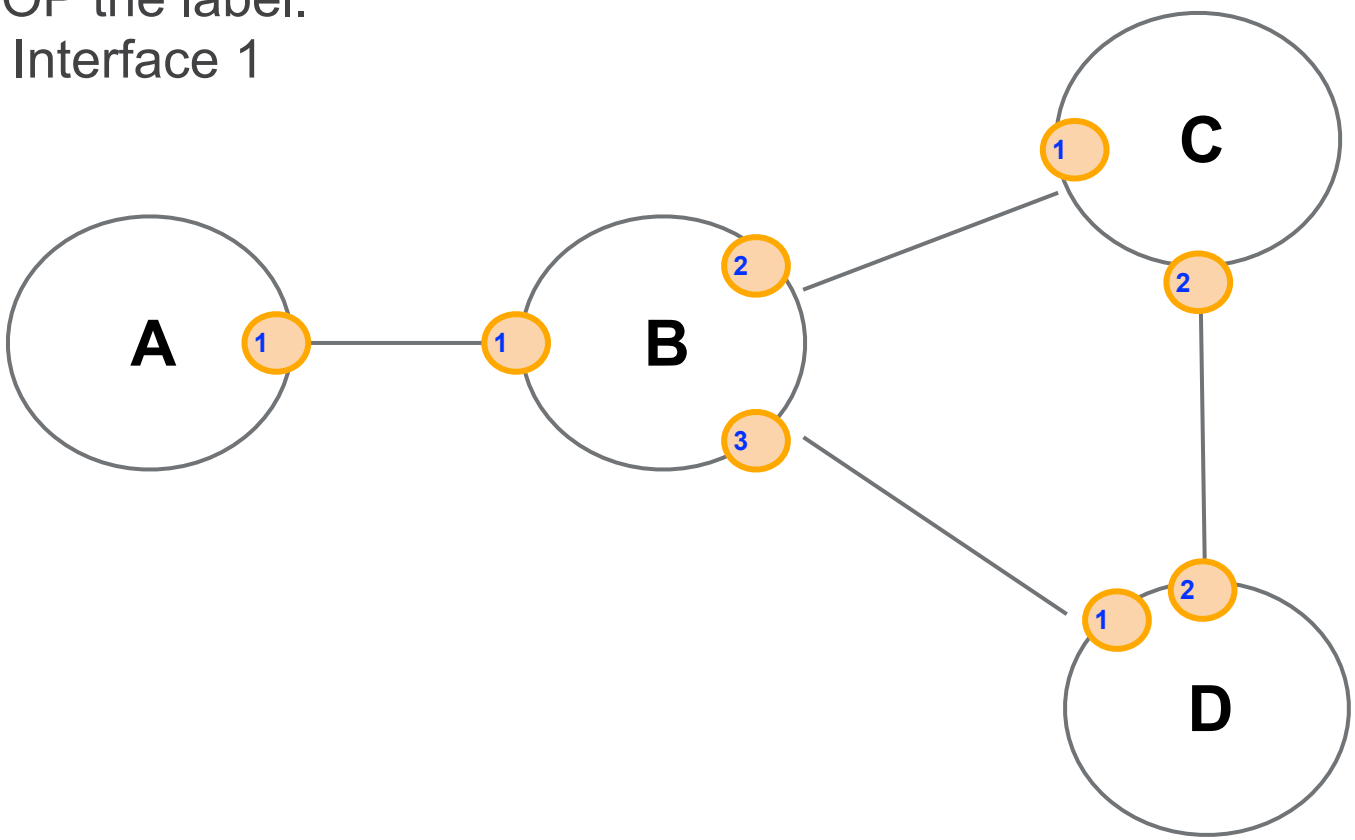


Google

# Reduce state: One hop static LSP

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

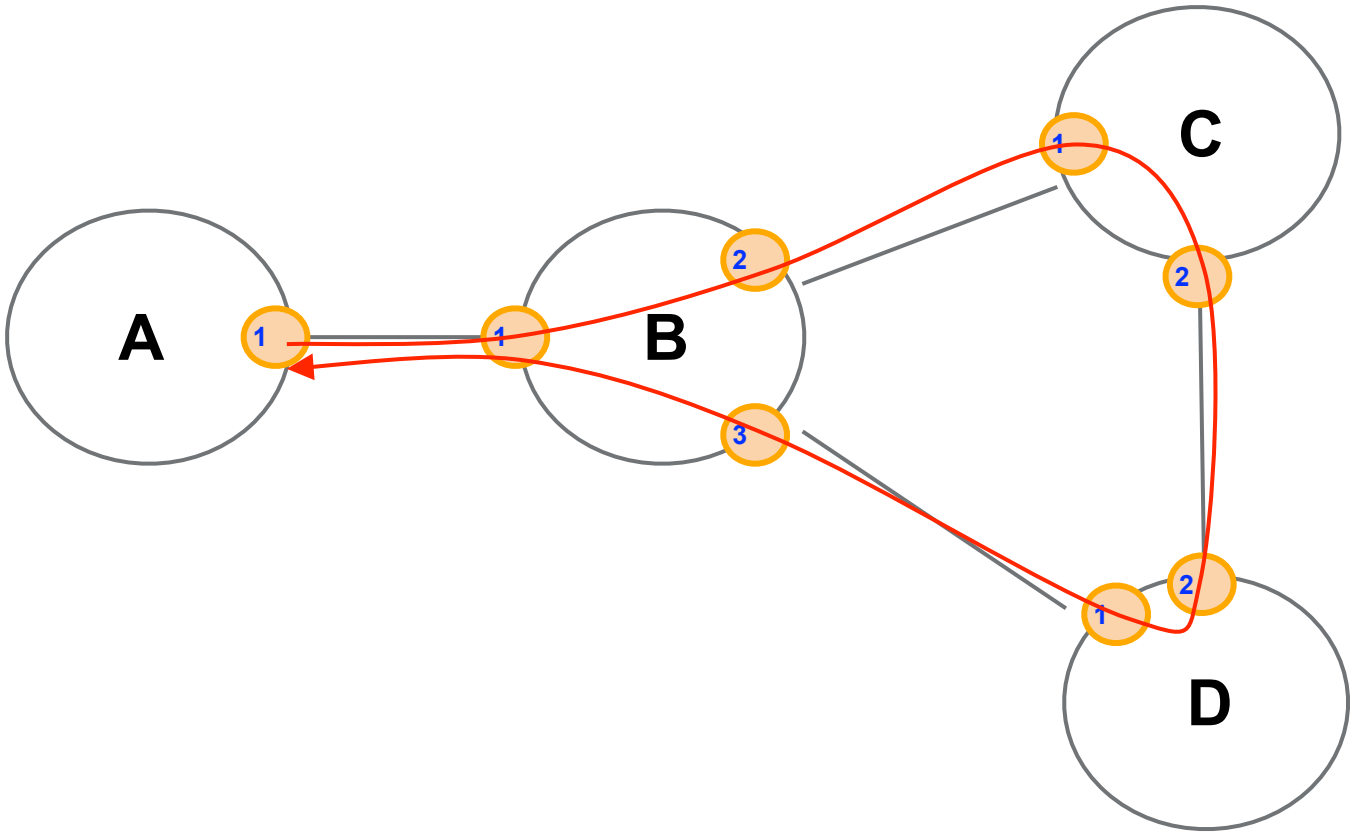
1 Interface 1



Google

# Reduce state: One hop static LSP

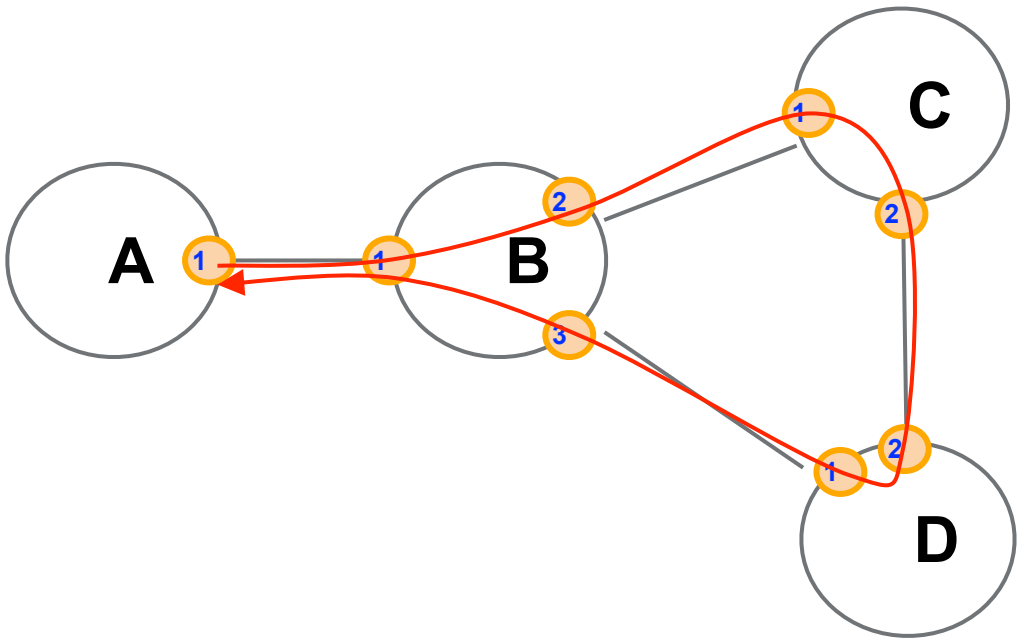
We want to send a packet through the following path:
A:1 ->B:1
B:2 ->C:1
C:2 ->D:2
D:1 ->B:2
B:1 ->A:1



# Reduce state: One hop static LSP

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

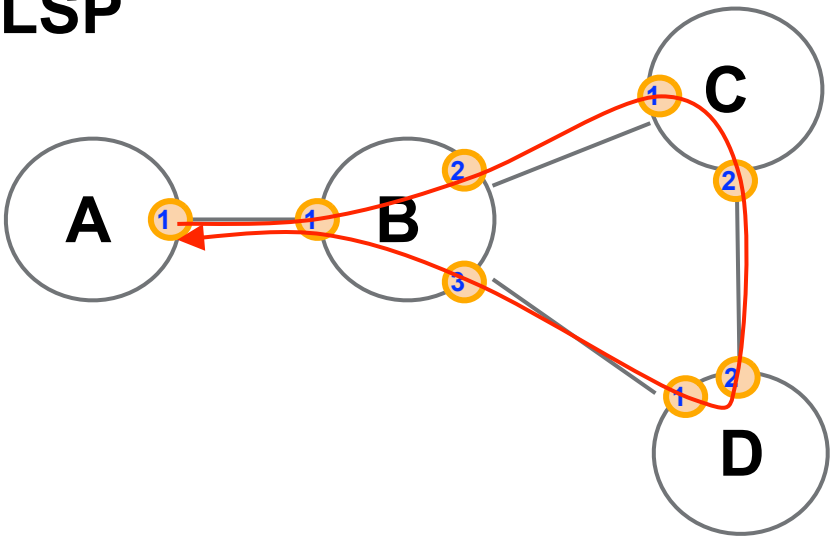
Router A has a static LSP that says:
For packets with incoming label 1, pop the label and forward to interface 1.



# 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.



Google

Questions ?

nguilbaud@google.com
rossc@google.com