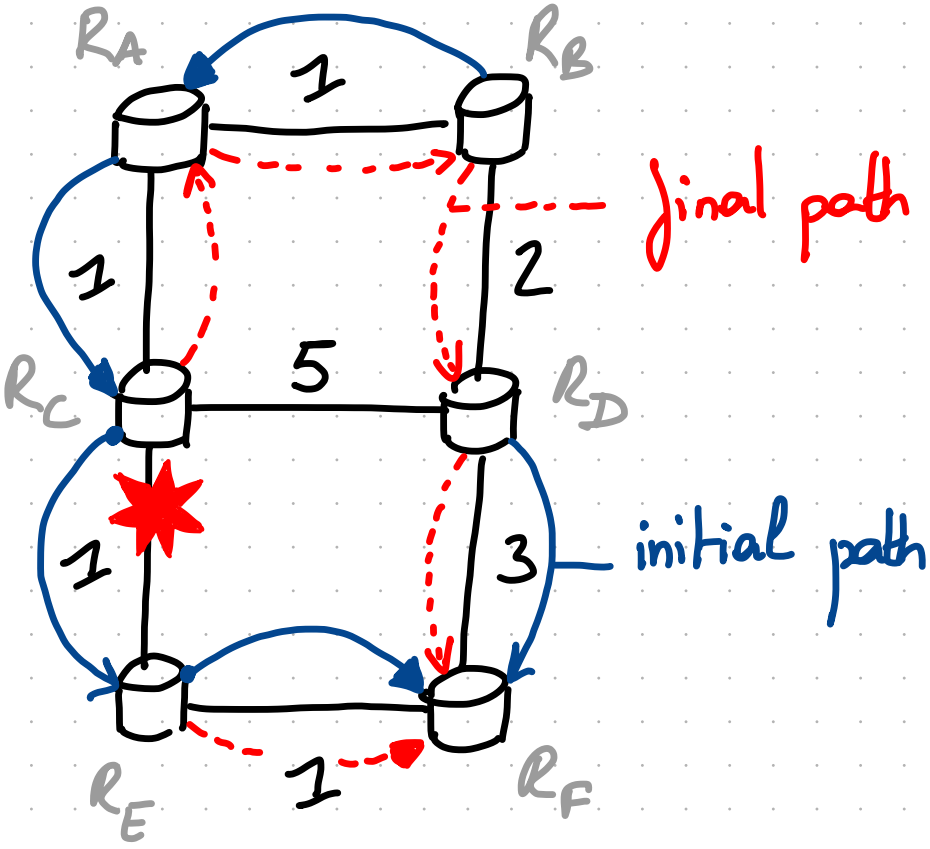
Advanced Topics in Communication Networks

L8: Fast Convergence / 10.11.2020

Prof. Laurent VANBEVER - nsg.ee.ethz.ch



How do we retrieve connectivity as fast as possible after a link or a node failure?

↳ goal: <50 ms

Lecture Outline:

1. Introduction / Motivation:
   - 1.1. What do we mean by convergence?
   - 1.2. Why should we care?
   - 1.3. What constitutes the convergence time?
2. Fast Convergence in IP networks:
   - 2.1. Fast detection
   - 2.2 Fast propagation
   - 2.3 Fast computation
   - 2.4 Fast updates
     - 2.4.1. Loop-Free Alternates
     - 2.4.2. Prefix-Independent Convergence
3. Fast Convergence in MPLS networks (next lecture)

1.1. What do mean by network convergence?

Definition: Routing convergence is the transition from a routing and forwarding state to another state.

Convergence is typically triggered by a change in the topology which is:
either sudden, in the case of a failure;
or planned, in the case of maintenance.

Convergence is a distributed process during which routers might have an inconsistent view of the network.

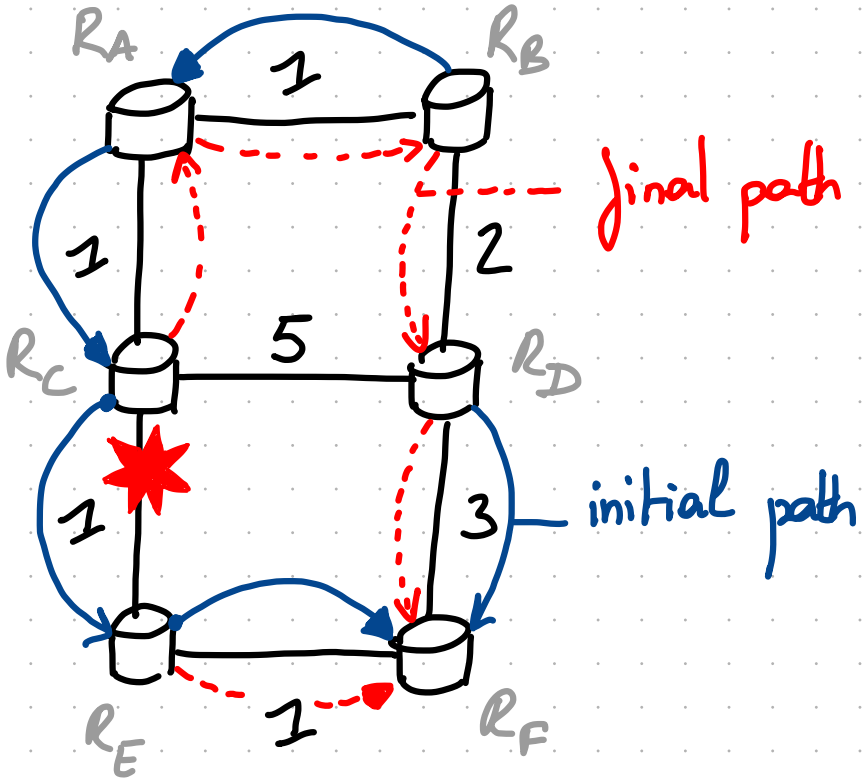
1.2. Why should we care about convergence?

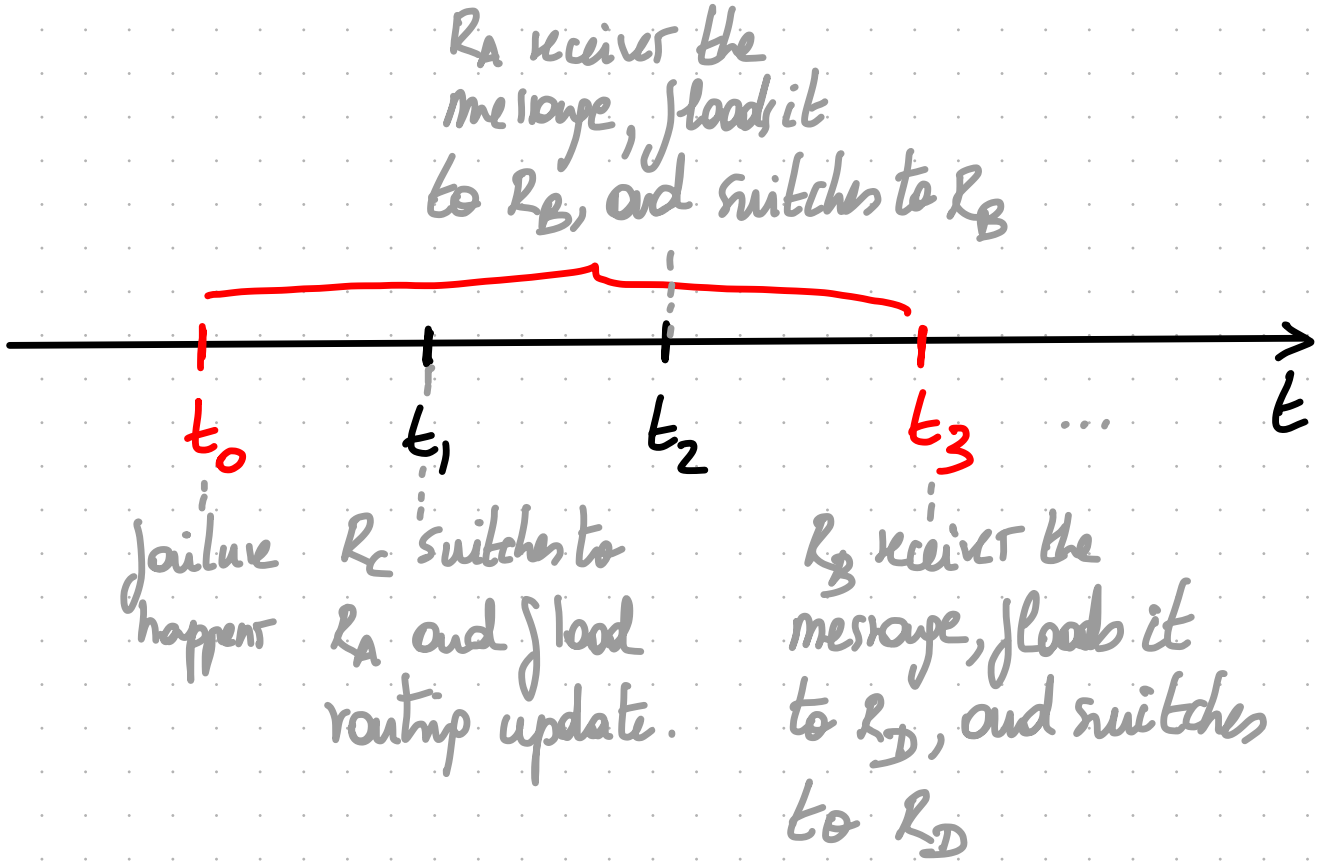
Problem: Transiently inconsistent routing and forwarding state can lead to traffic losses. These losses are due:

1. because routers do not detect changes immediately and therefore forward traffic in a "blackhole". — annoying...
2. because the forwarding paths contain forwarding loops. — VERY ANNOYING!

Note that not all transient states lead to traffic losses. In practice, operators are mainly concerned about retrieving connectivity, not avoiding transient states.

Example:





Duration of the connectivity loss : $t_3 - t_0$

1.3. What composes the convergence time?

There are 4 main sources of convergence delay:

local	1	Detect the failure.	t_1
global	2	Communicate the failure.	t_2
	3	Recompute forwarding state.	t_3
	4	Update forwarding state.	t_4

$\sum_i t_i < 50$ ms (goal)

$t_1 \approx t_2 \approx t_3$: Θ(millisecond)

t_4 : Θ(#prefixes) → can be very slow still (100s of sec)

2. Fast convergence in IP Networks.

How do we achieve our goal of 50 ms?

2.1 Fast detection

2.2 Fast propagation

2.3 Fast computation

2.4. Fast forwarding table update.

2.1 How do we detect link/node failures?

Goal:
* Fast detection
* High accuracy
* Low overhead

Mechanisms: 1. Rely on physical/link layer signals;
2. Rely on "hello"/beacons.

1/ Rely on physical/link_layer:

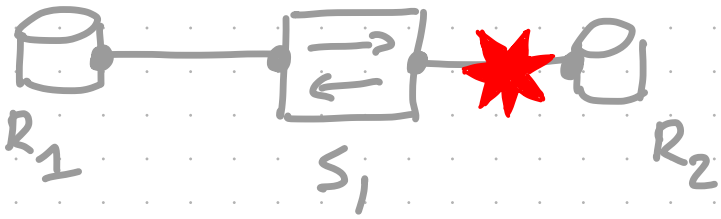
Some physical/link layers, such as optical layers, can detect failures through the loss of light or carrier signal.

Example: Synchronous Optical Networking (SONET).
Synchronous Digital Hierarchy (SDH).
Dense Wavelength Division Multiplexing (DWDM).

Pros: * As fast as you can get (few mr!)
* (Virtually) no overhead

Cons: * Only works for some {physical / link layers} (e.g. not available in Ethernet).
* Does not detect certain kind of failures.

Example:



The link (R_1, R_2) is dead even though the link layer on R_1 is still up.

2/ Rely on "hellos" / "keepalives" / "beacons"...

The idea here is to have adjacent routers regularly exchange "hellos" and signal a failure whenever k of them are missed in a row.

By default, each routing protocol comes with its own hello-based protocol. Each protocol allows the operator to configure:

- the frequency at which hellos are generated.
  typically by defining the period between two subsequent hellos (hello* interval).
- the interval after which not receiving a single hello that the protocol declares the peer dead (dead* interval).

Default...	OSPF	BGP	LDP	...
hello interval	5s	60s	15s	...
dead interval	40s	180s	45s	..

* Note that different protocols may use different names for these values...

Pros:
* Works on any router platform.
* Detects a wider range of failures because it actually tests the forwarding path.

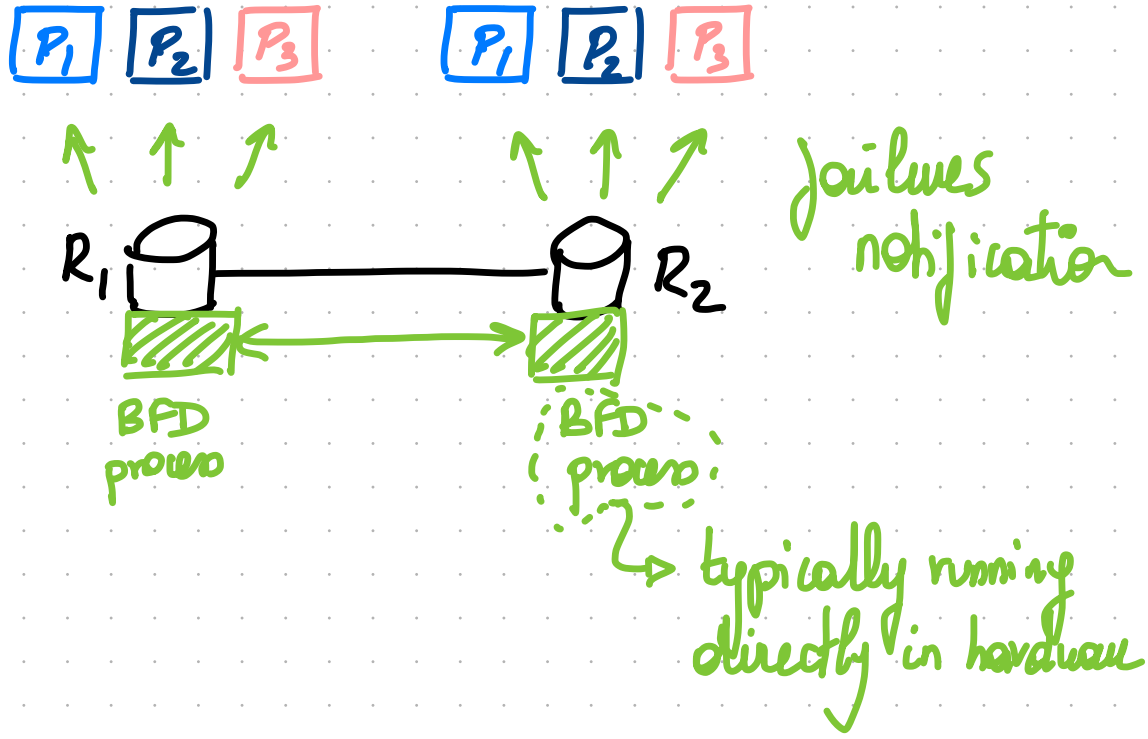
Cons:
* SLOW
* HUGE overhead

To decrease the detection time, one has to stress the control plane a lot.

This is incredibly wasteful (i.e high overhead) as each protocol are basically doing the exact same thing without exchanging any info...

The modern way to solve these two problems is to rely on one protocol-agnostic hello-based service that can further directly run in the hardware. This service is known as

Bidirectional Forwarding Detection (BFD)



The hello interval can be as low as 50 ms, with dead interval around 150 ms.

→ MUCH faster than protocol-based detection.

Pros:

* Fast detection
* Low overhead

} especially when BFD runs in hardware.

* High coverage → Because BFD tests the actual forwarding path (not just the interface liveness).

Cons:

* Not all routers can run BFD in the hardware.

Conclusion:

1. Use link-layer mechanisms whenever available.
2. Complement these mechanisms with a hardware-based BFD service, if available.
3. Fallback on protocol-based detection in last resort.

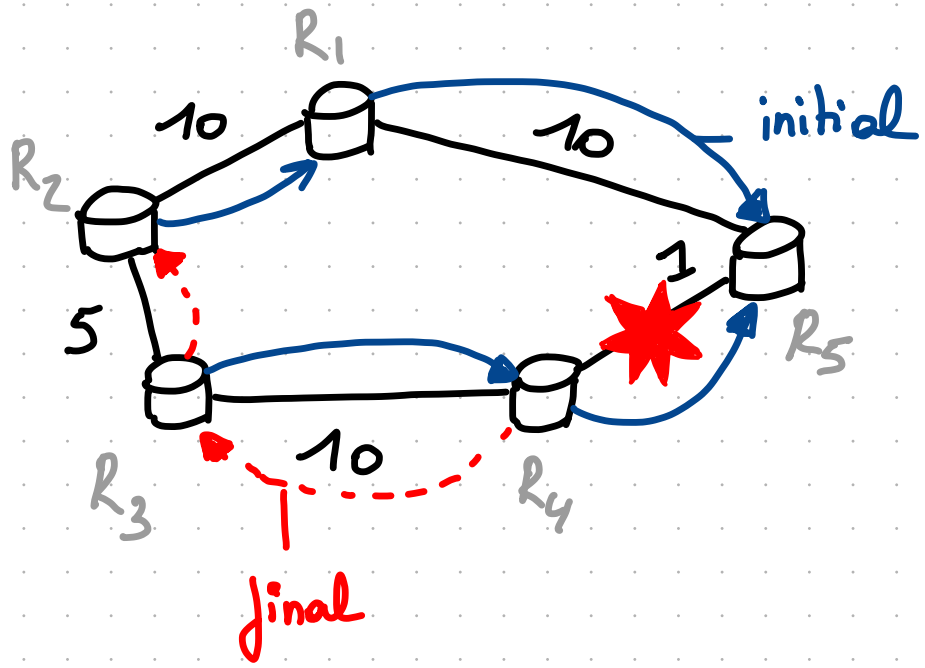
2.2 How do we quickly communicate the failure network wide?

This one is pretty easy to solve. Essentially, you want to ensure two things:

1. That the flooding of the failure notification happens immediately after the detection.
2. That the flooded packets are sent with absolute priority over the rest of the traffic.

Note that for convergence to happen, not necessarily all the network need to hear about the failure.

Let's look at an example...

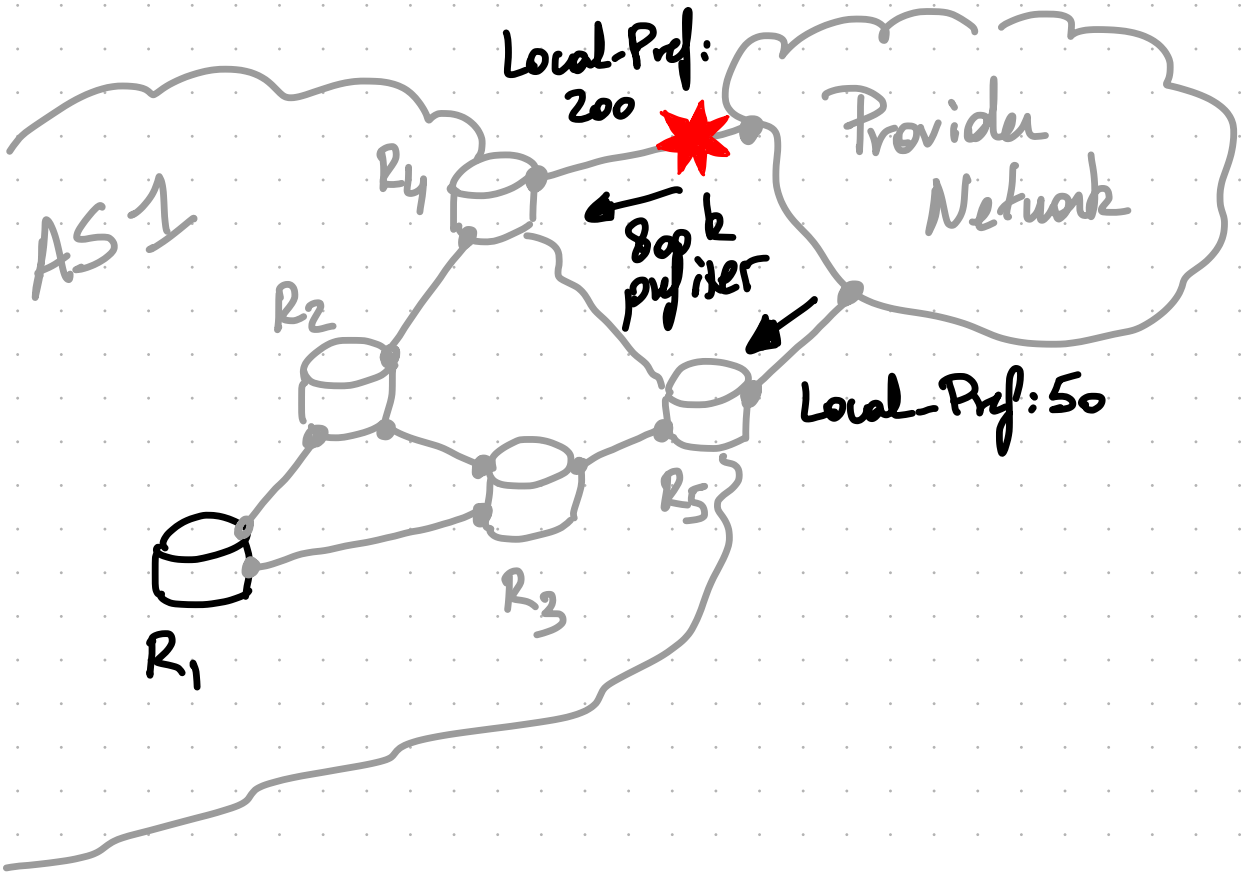


In the example above, "only" R_3 and R_4 need to be notified about the failure to retrieve connectivity towards R_5.

At the same time, "only" R_1 and R_2 need to be notified about the failure to retrieve connectivity towards R_4.

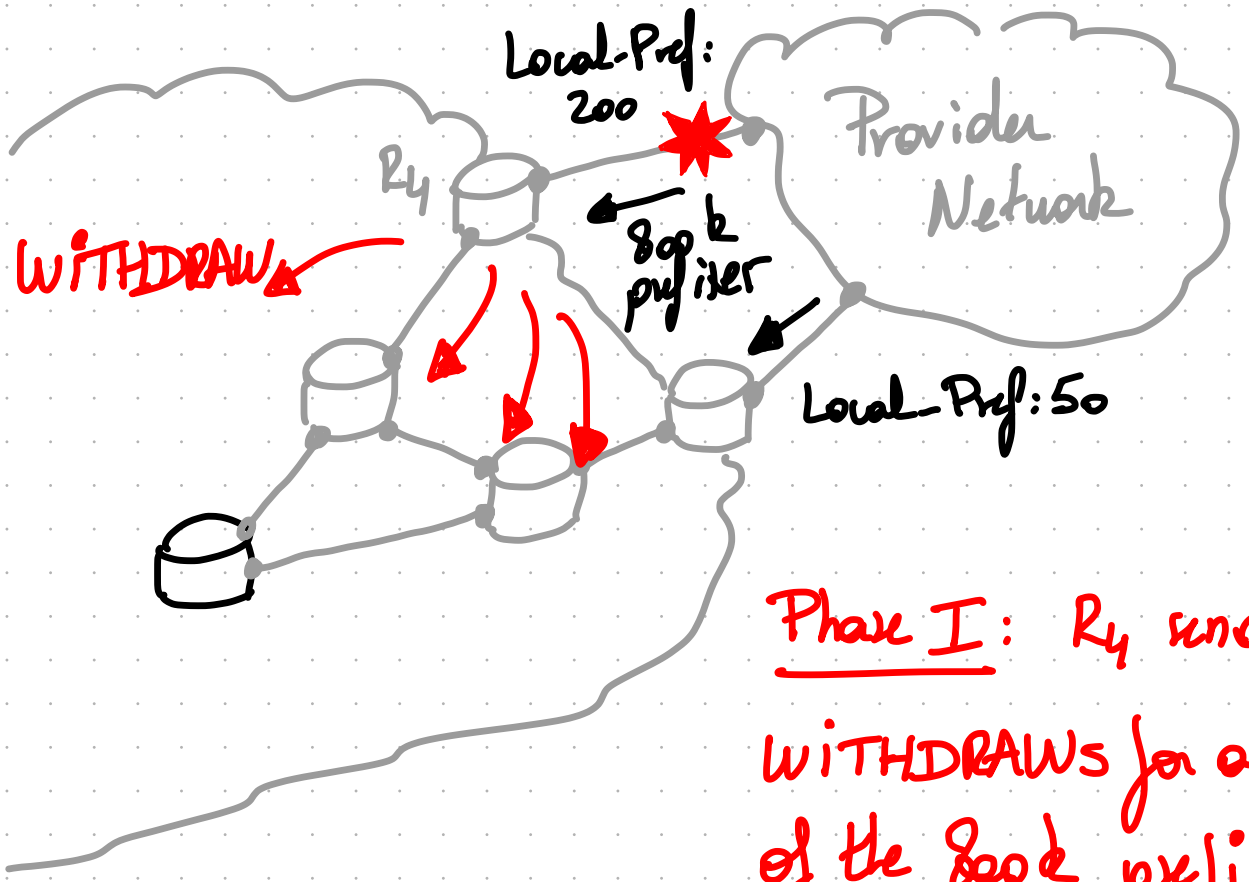
2.3 How do we quickly recompute forwarding state?

- For shortest-path-based protocols such as OSPF, IS-IS, this is not a problem anymore.
  - → Computing an entire shortest-path tree in a HUGE network (think Tier-1) only takes a few milliseconds nowadays.
  - → In addition, incremental shortest-path computation can help scaling this even further.
- For BGP, this remains a problem:
  - → the computation is done <u>per-prefix</u>.
  - → oftentimes, BGP routers do not even know an alternate path and litteraly need to "hunt" for it.

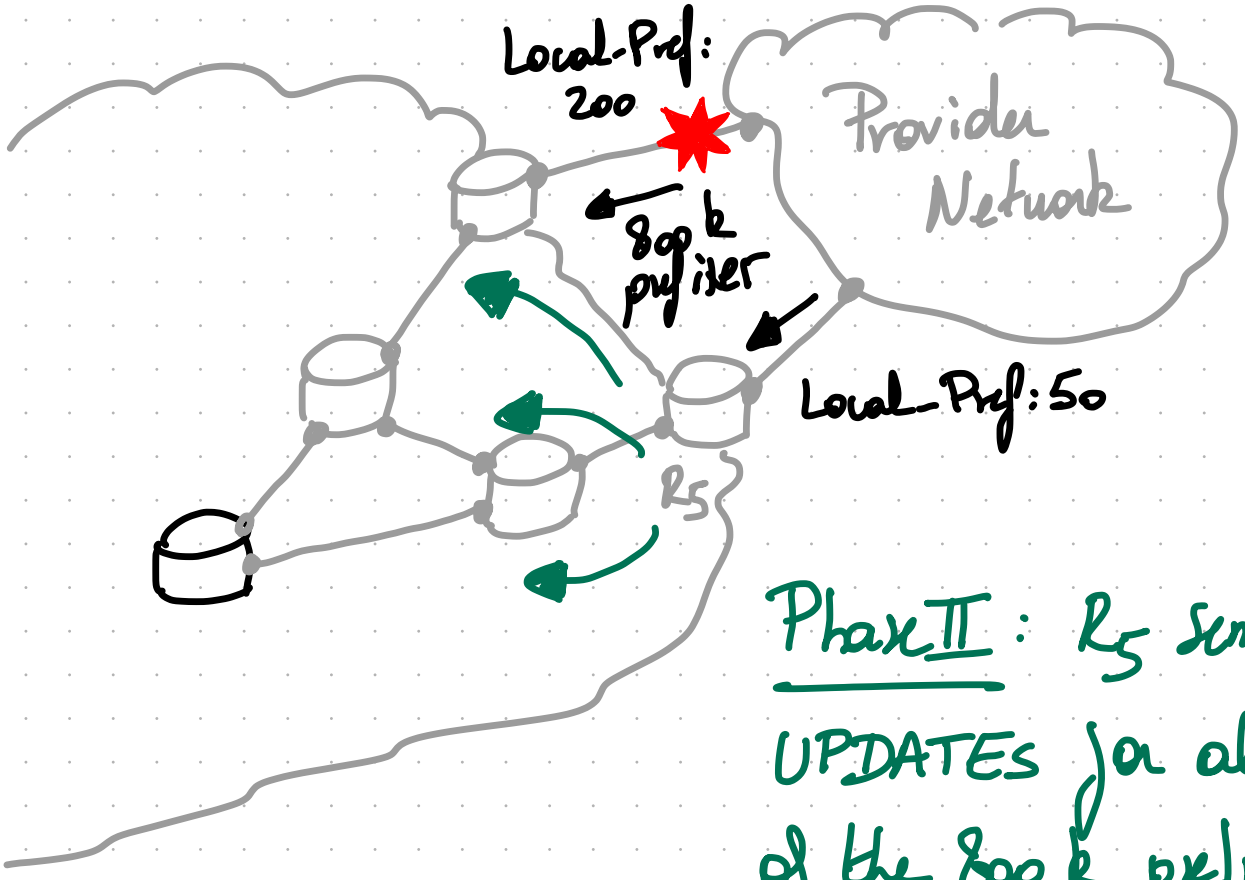


Prior to the failure, all of the internal routers in AS 1 only know one route to reach each of the 800k prefixes: via R_4.

This is because R_5 does not advertise its eBGP routes as they have a smaller local-preference.



Phase I: R_4 sends WITHDRAWs for all of the 800k prefixes...



Phase II: R_5 sends UPDATEs for all of the 800k prefixes...

One way to avoid this problem is to force R_5 to advertise its external routes, even if it does not choose them as best.

Multiple BGP extensions allow that. Of course, it comes at the cost of having internal routers carrying possibly WAY more routes in their routing table.

Note that the tradeoff between scalability and speed is rather typical in networking.

2.4 How do we quickly update the forwarding state?

Updating the Forwarding Information Base (FIB) is typically the key bottleneck to overcome in the convergence process.

Total Update Time	=	# prefixes	* average update time per prefix.

Our measurements on real network devices show average update times on the order of 100s of μsec.

=> For a full BGP table of around 800k prefixes:

800 k * 250 μs = 200 s

(see corresponding slide in the handout).

"Cheap trick": Prioritize FIB updates according to how much traffic each prefix sees.

Longer term solution:

Reorganize the FIB data structure so that it allows for fast incremental updates.

Step 1: Pre-compute the backup state.

Step 2: Pre-load it in the reorganized FIB.

Step 3: Activate the pre-loaded backup state upon detecting a failure.

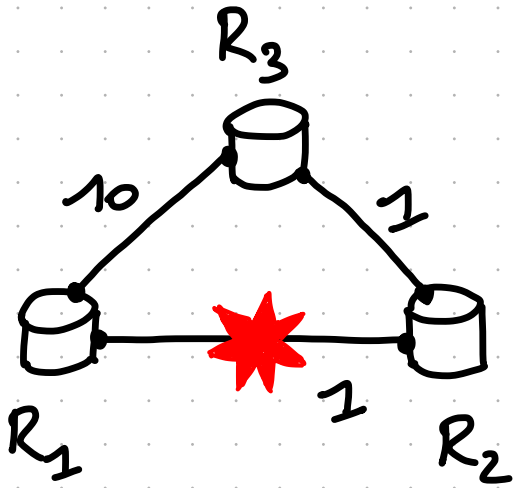
IGP → Loop-Free Alternates (LFA)

BGP → Prefix-Independent Convergence (PIC)

2.4.1 Loop-Free Alternates (LFA):

Intuition: A LFA is a neighbor for which you know that you can deviate the traffic impacted by a given failure and that traffic will not come back to you.

Example:



R_3 is a LFA of R_1 for the failure of $R_1 \rightarrow R_2$ since R_3 does not use R_1 to reach R_2.

Definition: A neighboring router N is a LFA for a router S to destination D if:

$$dist(N,D) < dist(S,D) + dist(S,N)$$

How to compute LFA on a router X:

- For all links (X, R):
  - For all direct neighbor (X, N):
    - Compute Shortest Path Tree of N.
    - If (X, R) ∉ SPT(N):
      - Add (X, R) as a candidate LFA. for all destinations.
    - Else:
      - Skip (X, R)

Note: The above algorithm computes per-link LFA. It can be extended to compute per-prefix LFA instead. For this, the evaluation of the next alternate is done for each destination prefix instead of per direct neighbor.

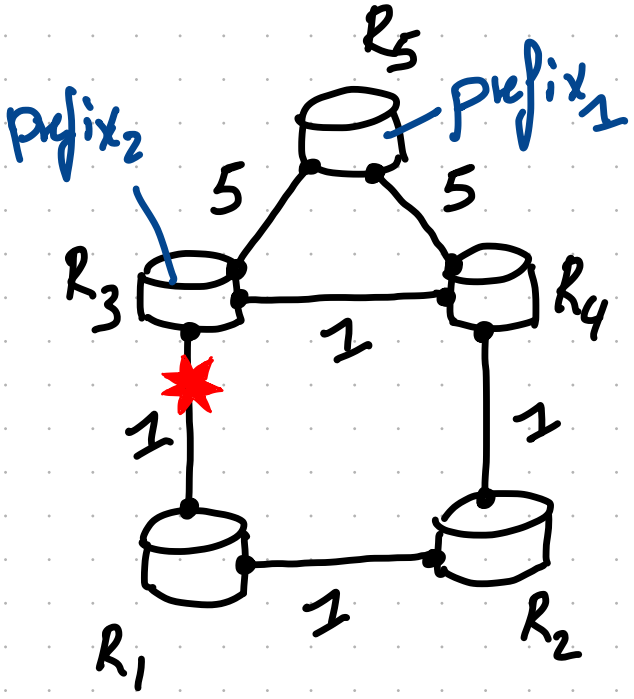
Depending on the network topology, a subset of the links and/or prefixes will be protectable using LFA / per-prefix LFA.

→ Some design patterns, such as triangles, lead to high coverage.

→ Also, coverage (per-link LFA) ≤ coverage (per-prefix LFA)

In this topology,

R_2 is not a per-link LFA for R_1 as R_2 uses R_1 to reach $prefix_2$.



BUT

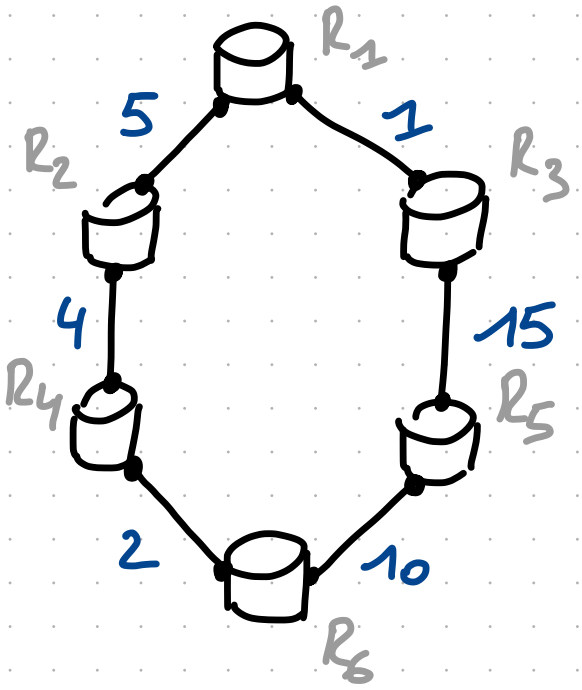
R_2 is a per-prefix LFA for R1 for $prefix_1$ because

$$dist(R_2, prefix_1) < dist(R_2, R_1) + dist(R_1, pfx_1)$$

$$6 \qquad 1 \qquad 6$$

Increasing LFA's coverage with Remote LFAs:

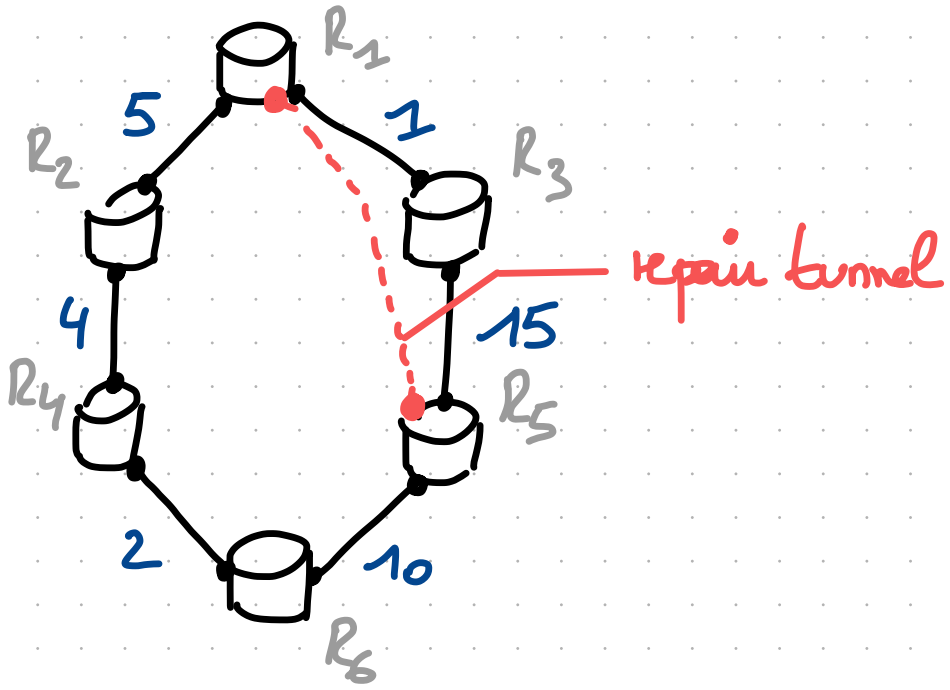
Let's look at another example:



In this topology, R_1 does not have a per-link, nor a per-prefix LFA to R_6. Why? Because R_3 uses R_1 to reach R_6 in the pre-convergence state.

Remote LFAs enable to increase the LFA coverage by allowing a router to use remote, non-neighboring routers as repair nodes by tunneling to them.

Let's look at the previous example again:



- While R_3 uses R_1 to reach R_6, it does not use R_1 to reach R_5.
- R_5 reaches R_6 directly (i.e. not via R_3)

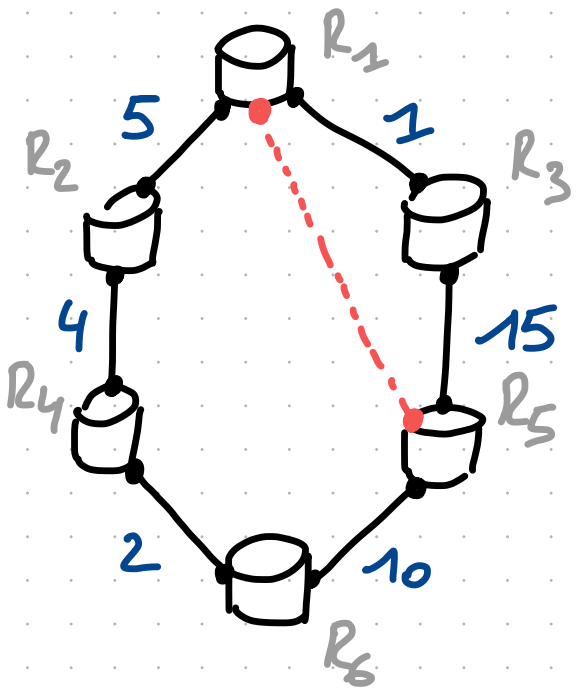
=> By encapsulating its R_6-directed traffic to R_5 and sending that encapsulated traffic to R_3, R_1 can retrieve connectivity!

Observe that this tunnel would typically be LDP-based.

How do we compute remote LFAs?

Given $D_{opt}(a,b)$, a function that returns the shortest-path distance between a and b.

- On router X:
  - $\forall$ destination Y:
    - Let nh be the pre-converge next-hop used by X to reach Y (according to Dopt)
    - Let P be the set of nodes that X can reach without traversing (X, nh).
    - Let Q be the set of nodes that can reach Y without traversing (X, nh)
    - Let candidates_RLFA = $P \cap Q$
    - return candidates_RLFA.



- Considering R_1 and R_6 again:
  - $P = \{R_3, R_5\}$ the set of nodes R_1 can reach NOT going via (R_1, R_2).
  - $Q = \{R_2, R_4, R_5\}$ the set of nodes which reach R_6 NOT via (R_1, R_2)
  - $P \cap Q = \{R_5\}$ → the only RLFA available to protect R_6 is R_5.

Note RLFA do NOT guarantee full coverage.

Consider what happens to P and Q with (R_3, R_5) set to 23 instead of 15...

(R)LFA gives us a simple condition a router can check locally to know to which neighbor it can safely redirect traffic to.

The next question now becomes:

"How do we organize the FIB to allow for a fast activation of the backup state?"

One possible solution:

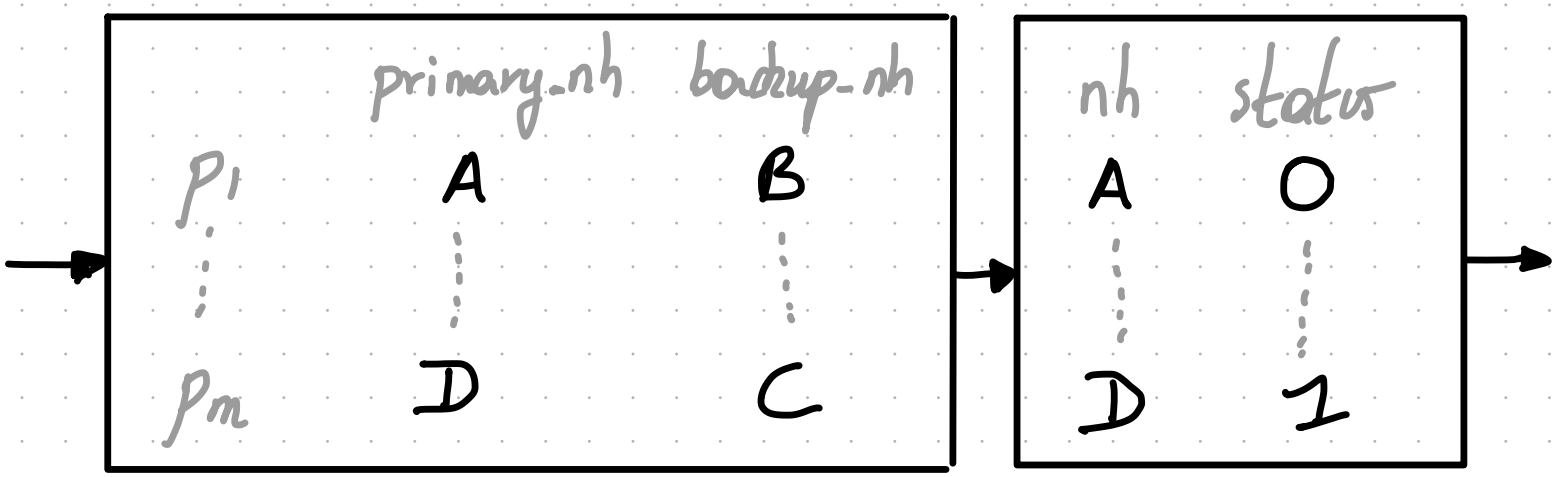


TABLE I

↓

put primary and backup nh in metadata

Implementable in P4!

TABLE II

↓

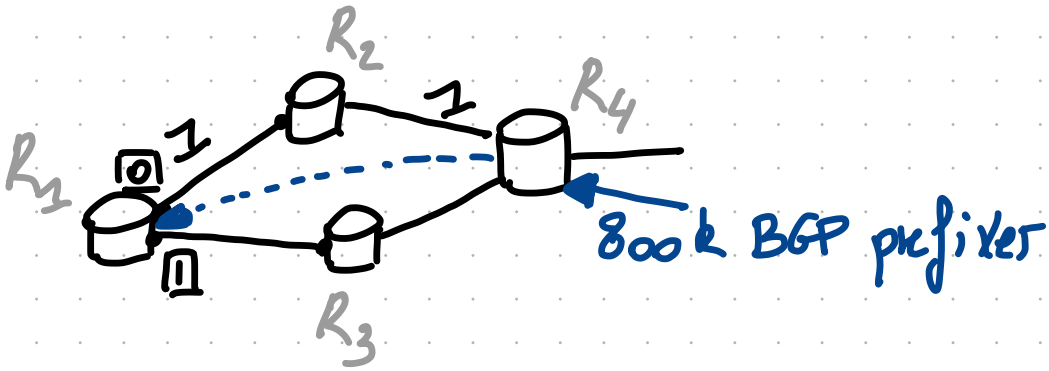
```
if status==0:
  egress_port=backup;
else
  egress_port=primary;
```

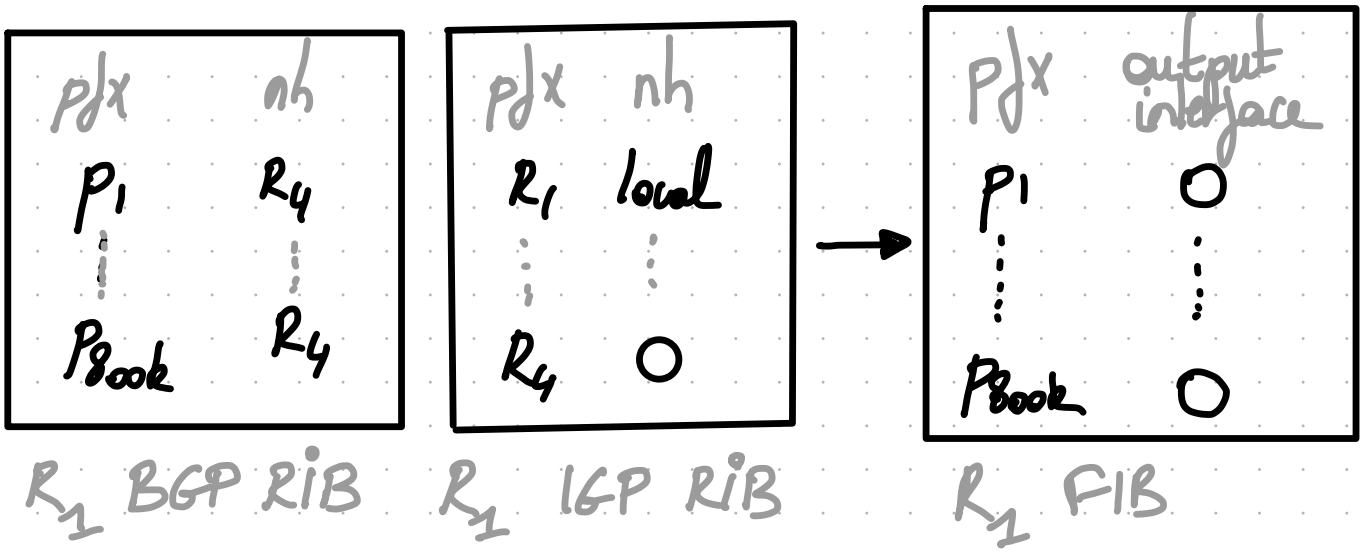
2.4.2. BGP Prefix Independent Convergence (PIC):

Goal: Enable router to quickly switchover to pre-installed alternate paths upon failures that affect BGP routes.

(Make BGP as fast to converge as the IGP)

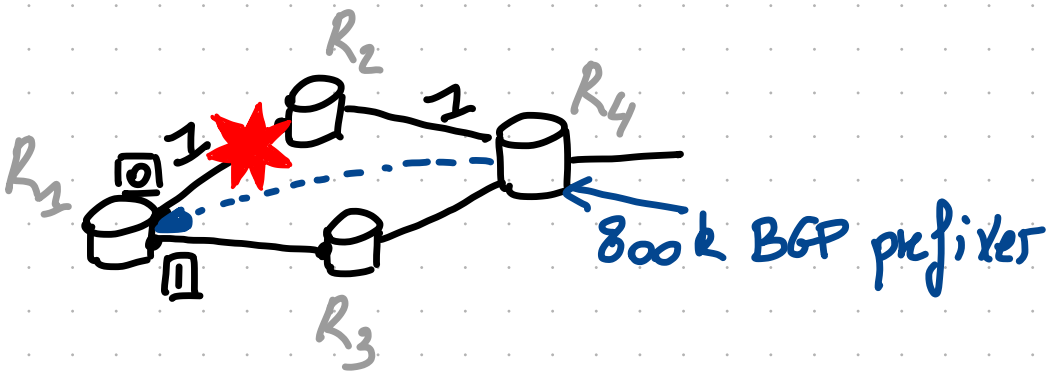
Problem: "Flat" FIBs





R_1 BGP RIB R_1 IGP RIB R_1 FIB

Problem: "Flat" FIBs



pfx	nh
P_1	R_4
⋮	⋮
P_{800k}	R_4

R_1 BGP RIB

pfx	nh
R_1	local
⋮	⋮
R_4	1

R_1 IGP RIB

→

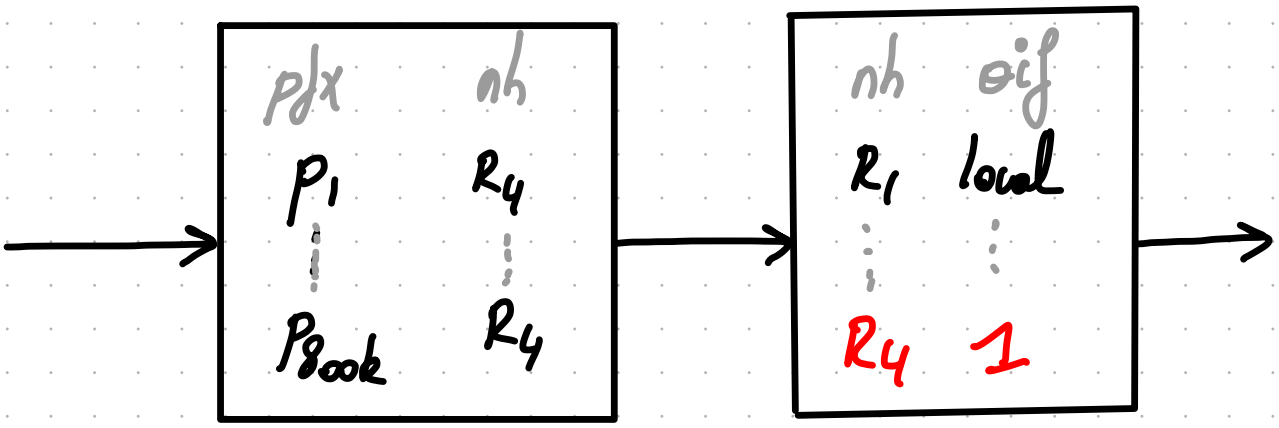
pfx	output interface
P_1	1
⋮	⋮
P_{800k}	1

R_1 FIB

Upon the failure of (R_1, R_2) link, R_1 has to perform 800k updates to its FIB...

The fundamental problem is that the dependency between BGP next hops and the IGP one is NOT maintained in the FIB. It is "flattened".

Solution: Maintain the hierarchy between BGP next-hops and IGP next-hops in the FIB as well.



R_1 BGP FIB (Table I)

R_1 IGP FIB (Table II)

set nh in metadata

match on nh metadata and set the output interface accordingly.