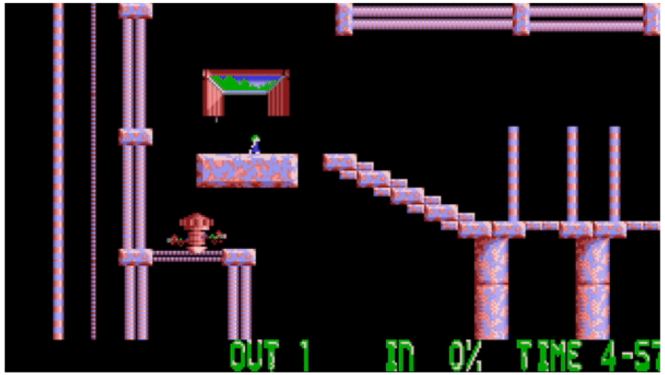
Fast Reroute in P4: Keep Calm and Enjoy Programmability



Marco Chiesa
KTH Royal Institute of Technology

Joint project with:

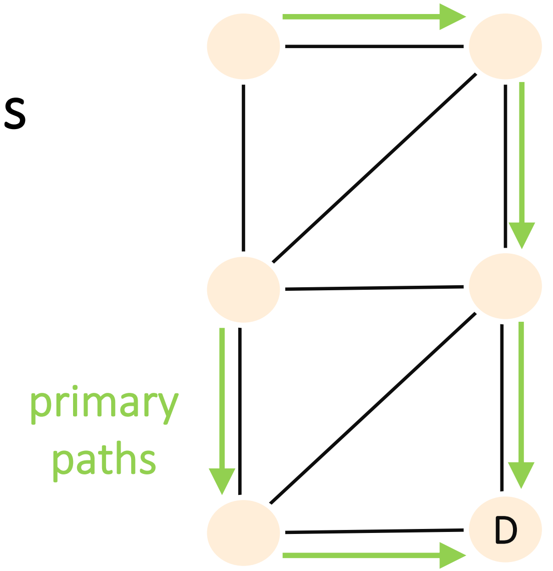
Roshan Sedar
Michael Borokhovich
Gianni Antichi
Stefan Schmid

Image credits: The Noun Project: Switch By Opher Aloni, Detour 2 By Pierre-Luc Auclair, CA, Right arrow By heri sugianto, ID, Fast clock By Piero Borgo, IT, Small table By IconMark, PH, Brain By Symbolon, IT Right arrow 2 By Roy, NL, Redo By Bluetip Design, US

Fast Reroute (FRR)

What is FRR?

- forwarding rules **conditional** on port status
- e.g., IP Loop-Free Alternates



Fast Reroute (FRR)

What is FRR?

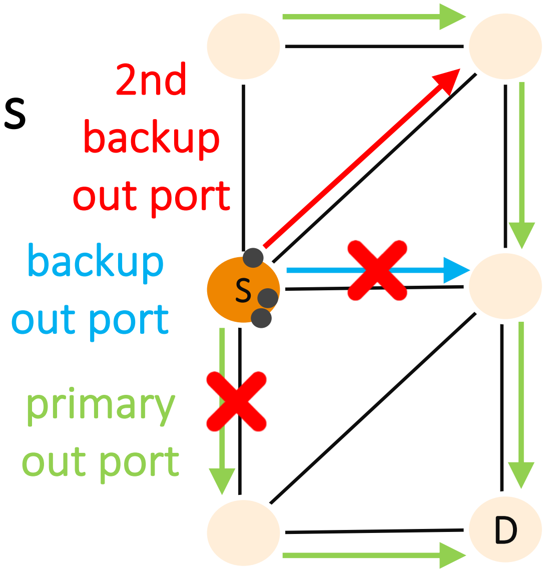
- forwarding rules **conditional** on port status
- e.g., IP Loop-Free Alternates

Pros:

- **reduce** network downtime (≤50ms)
    - no need to invoke control-plane

Cons:

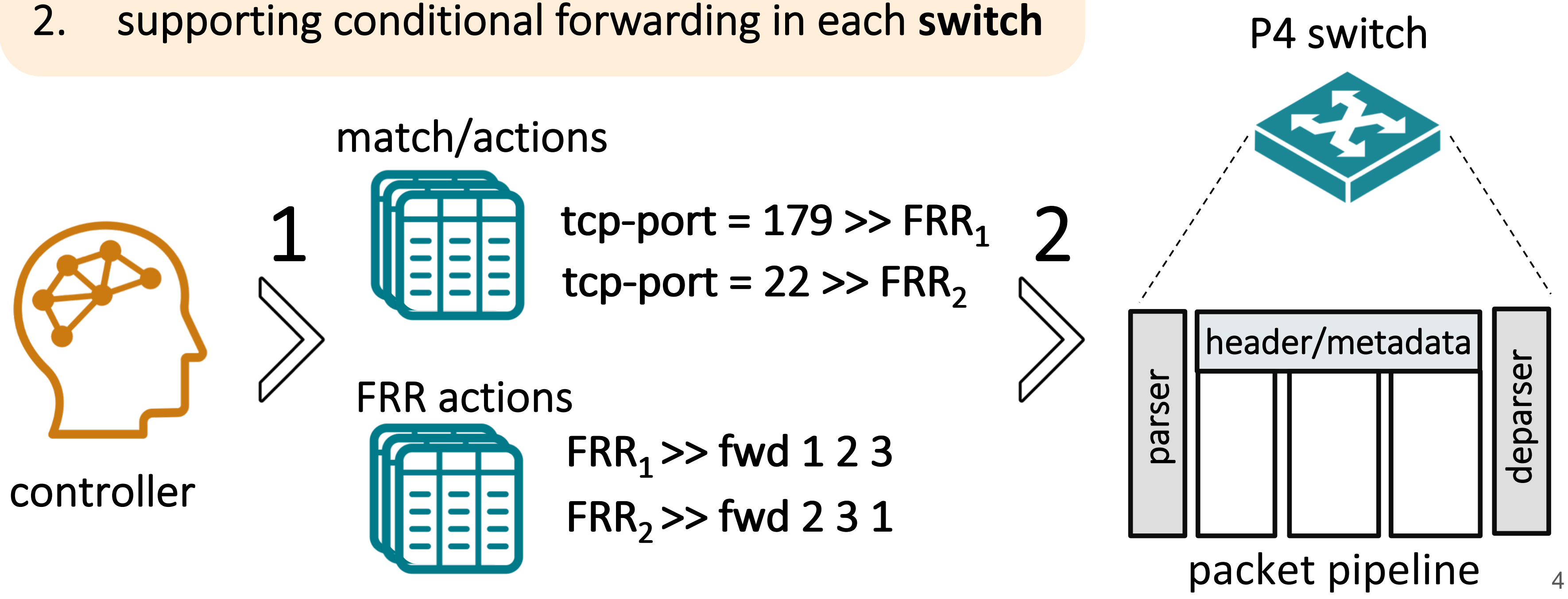
- **increase** forwarding space occupancy
    - proactively stores backup forwarding rules



How do we implement FRR in P4?

FRR entails solving two orthogonal problems:

1. computing **network-wide** conditional rules
2. supporting conditional forwarding in each **switch**



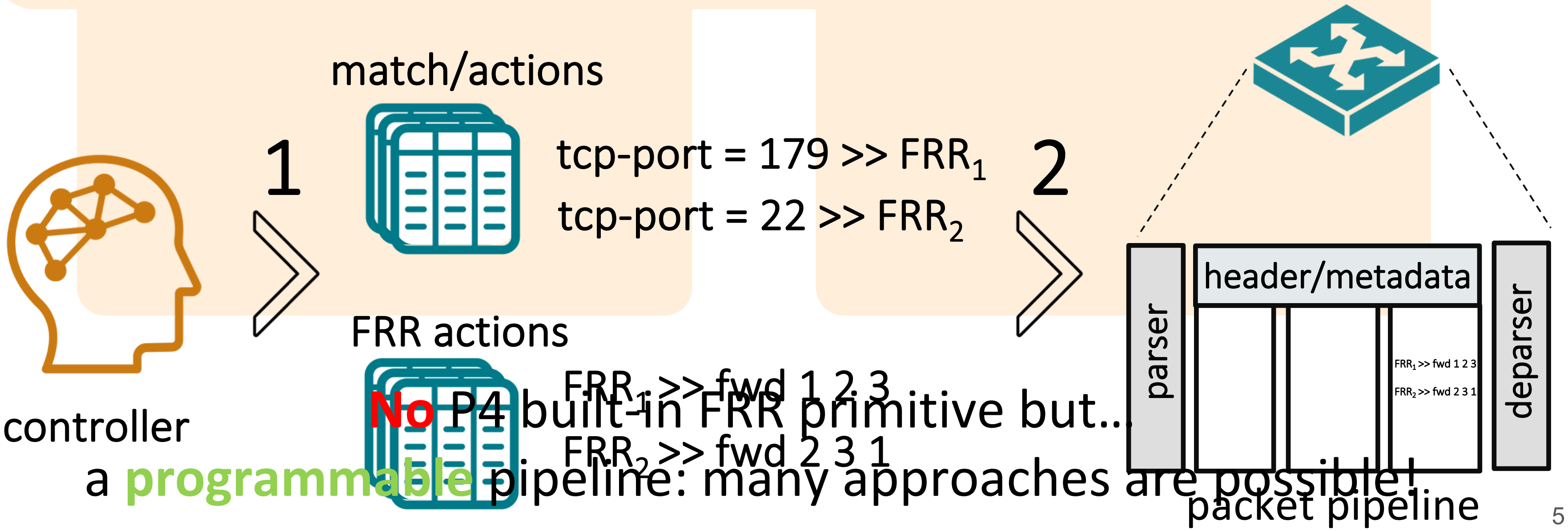
Supporting conditional forwarding in P4 switch

How do we implement FRR in P4?

FRR entails solving two orthogonal problems:

1. computing **network-wide** conditional rules
2. supporting conditional forwarding in each **switch**

utput



No P4 built-in FRR primitive but…

a **programmable** pipeline: many approaches are possible!

1st approach: recirculation

Input

FRR_1 = 1 2 3 4

FRR_2 = 2 3 4 1

FRR_3 = 3 4 1 2

FRR_4 = 4 1 2 3

a set of circular FRR actions

match	action	
out	fwd →	write & recirculate ↻
1	1	out := 2
2	~~2~~	out := 3
3	~~3~~	out := 4
4	4	out := 1

port 2 fails

port 3 fails

throughput reduction

1st requirement for FRR primitive

high
throughput

Second approach: sequential search

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

match	action
FRR = 1	fwd(1)

match	action
FRR = 1	fwd(2)

match	action
FRR = 1	fwd(3)

match	action
FRR = 1	fwd(4)

Second approach: sequential search

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

match	action
FRR = 1	fwd(1)
FRR = 2	**fwd(2)**

match	action
FRR = 1	fwd(2)
FRR = 2	**fwd(3)**

match	action
FRR = 1	fwd(3)
FRR = 2	**fwd(4)**

match	action
FRR = 1	fwd(4)
FRR = 2	**fwd(1)**

Second approach: sequential search

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1   **FRR_3 = 3 4 1 2** FRR_4 = 4 1 2 3

match	action
FRR = 1	fwd(1)
FRR = 2	fwd(2)
FRR = 3	**fwd(3)**

match	action
FRR = 1	fwd(2)
FRR = 2	fwd(3)
FRR = 3	**fwd(4)**

match	action
FRR = 1	fwd(3)
FRR = 2	fwd(4)
FRR = 3	**fwd(1)**

match	action
FRR = 1	fwd(4)
FRR = 2	fwd(1)
FRR = 3	**fwd(2)**

Second approach: sequential search

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2   **FRR_4 = 4 1 2 3**

match	action
FRR = 1	fwd(1)
FRR = 2	fwd(2)
FRR = 3	fwd(3)
FRR = 4	**fwd(4)**

match	action
FRR = 1	fwd(2)
FRR = 2	fwd(3)
FRR = 3	fwd(4)
FRR = 4	**fwd(1)**

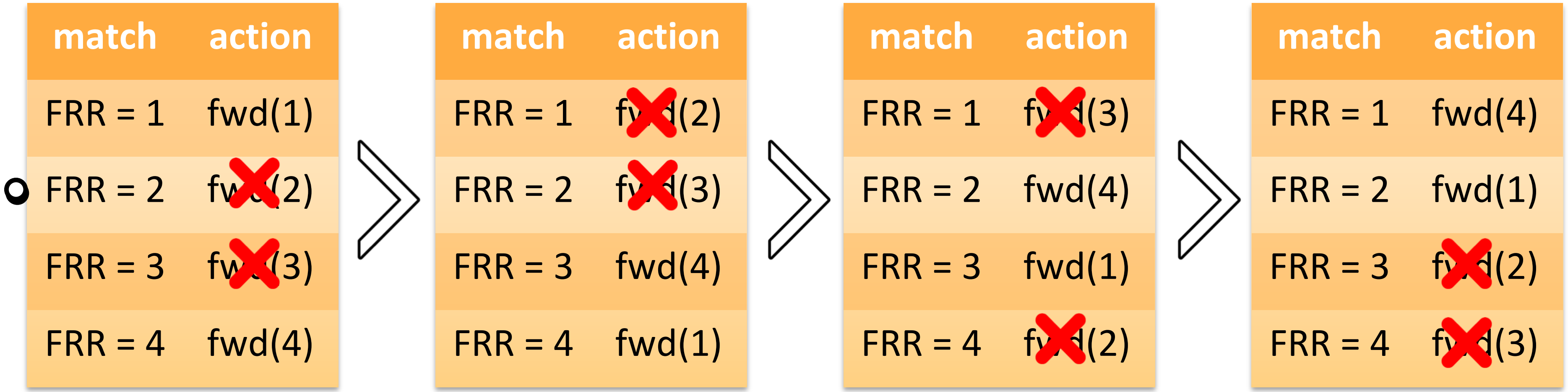
match	action
FRR = 1	fwd(3)
FRR = 2	fwd(4)
FRR = 3	fwd(1)
FRR = 4	**fwd(2)**

match	action
FRR = 1	fwd(4)
FRR = 2	fwd(1)
FRR = 3	fwd(2)
FRR = 4	**fwd(3)**

Second approach: sequential search

Input

$FRR_1 = 1\ 2\ 3\ 4$ $FRR_2 = 2\ 3\ 4\ 1$ $FRR_3 = 3\ 4\ 1\ 2$ $FRR_4 = 4\ 1\ 2\ 3$



match	action
FRR = 1	fwd(1)
FRR = 2	~~fwd(2)~~
FRR = 3	~~fwd(3)~~
FRR = 4	fwd(4)

match	action
FRR = 1	~~fwd(2)~~
FRR = 2	~~fwd(3)~~
FRR = 3	fwd(4)
FRR = 4	fwd(1)

match	action
FRR = 1	~~fwd(3)~~
FRR = 2	fwd(4)
FRR = 3	fwd(1)
FRR = 4	~~fwd(2)~~

match	action
FRR = 1	fwd(4)
FRR = 2	fwd(1)
FRR = 3	~~fwd(2)~~
FRR = 4	~~fwd(3)~~

port 2 fails

port 3 fails

increased latency
waste of resources at each stage

2nd requirements for FRR primitive

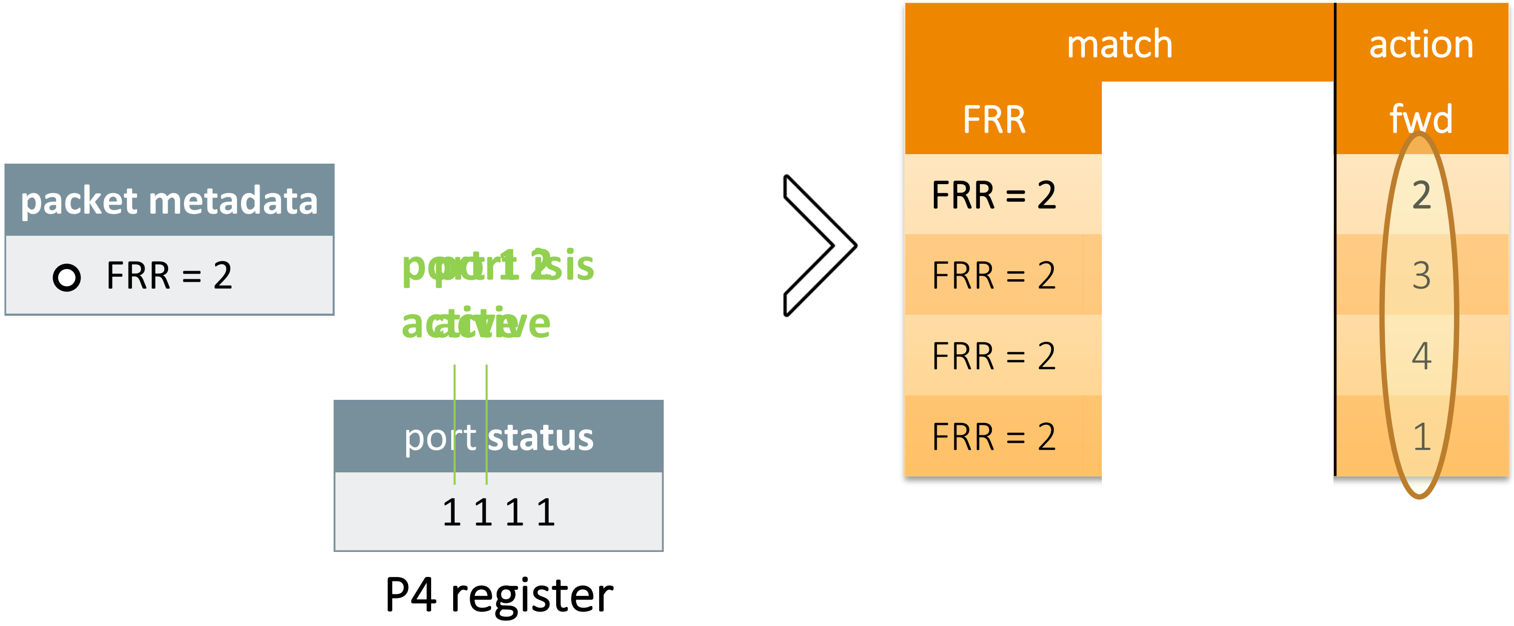
high
throughput

low forwarding
latency

Parallelism needed! Naïve TCAM approach

Input

FRR_1 = 1 2 3 4 **FRR_2** = 2 3 4 1 FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

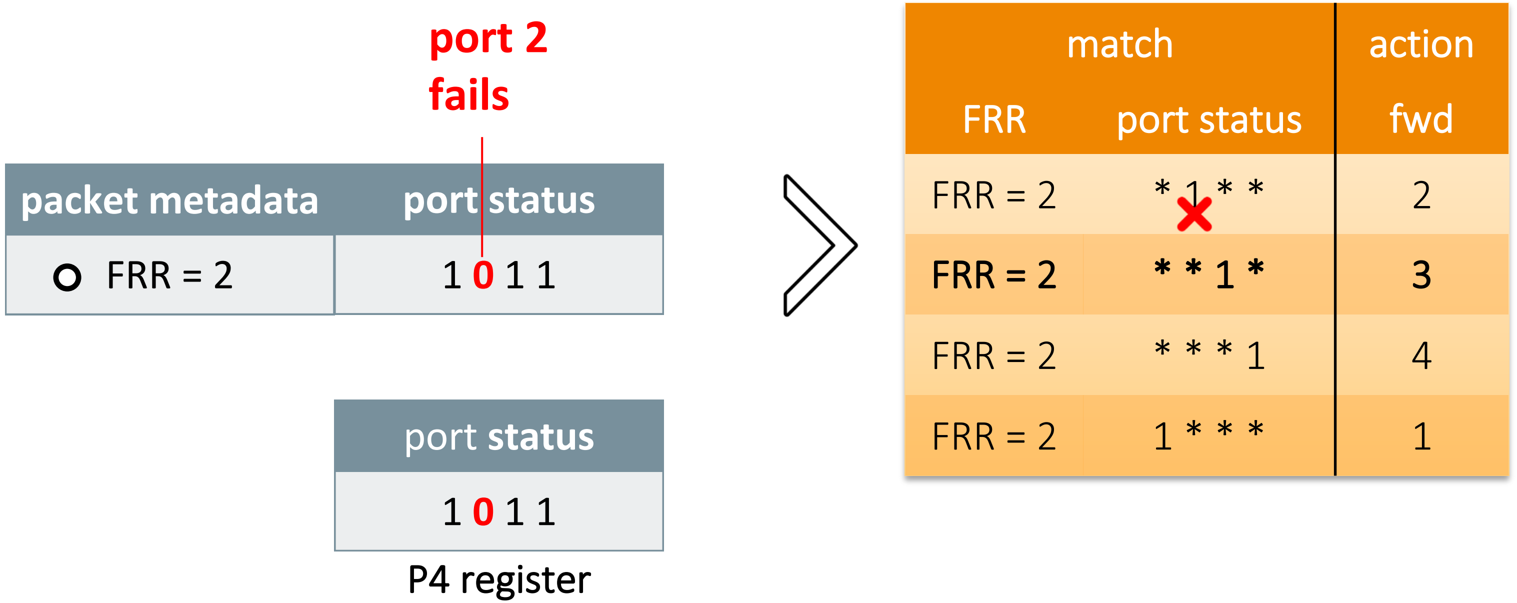


match		action
FRR		fwd
FRR = 2		2
FRR = 2		3
FRR = 2		4
FRR = 2		1

Parallelism needed! Naïve TCAM approach

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3



packet metadata	port status
○ FRR = 2	1 **0** 1 1

port status
1 **0** 1 1

match		action
FRR	port status	fwd
FRR = 2	~~* 1 * *~~	2
FRR = 2	*** * 1 ***	**3**
FRR = 2	* * * 1	4
FRR = 2	1 * * *	1

Parallelism needed! Naïve TCAM approach

Input

FRR$_1$ = 1 2 3 4 **FRR$_2$ = 2 3 4 1** FRR$_3$ = 3 4 1 2 FRR$_4$ = 4 1 2 3

ports 2 and 3 fail

packet metadata	port status
○ FRR = 2	1 **0 0** 1

port status
1 **0 0** 1

P4 register

match		action
FRR	port status	fwd
FRR = 2	* ~~1~~ * *	2
FRR = 2	* * ~~1~~ *	3
FRR = 2	*** * * * 1**	**4**
FRR = 2	1 * * *	1

Parallelism needed! Naïve TCAM approach

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
○ FRR = 4	1 **0 0** 1

port status
1 **0 0** 1

P4 register

match		action
FRR	port status	fwd
FRR = 1	1 * * *	1
FRR = 1	* 1 * *	2
FRR = 1	* * 1 *	3
FRR = 1	* * * 1	4
...	...	...
FRR = 4	*** * * 1**	**4**
FRR = 4	1 * * *	1
FRR = 4	* 1 * *	2
FRR = 4	* * 1 *	3

Final requirements for FRR primitive

high throughput

low forwarding latency

efficient reroute

”port status” P4 register

How much state for “naïve TCAM”?

Input:

- switch with *k* ports
- 10 circular set of FRR actions

Naïve TCAM FRR:

- k^2 number of TCAM entries

For *k* = 24

- 5.760 TCAM entries!

For *k* = 48

- 23.040 TCAM entries!

- 10 pods in a datacenter with F10 FRR [nsdi-13]
- 10 destinations with the ″k arc-disjoint″ FRR mechanism [ton-16]

[nsdi-13] *V. Liu et al. "F10: A Fault-Tolerant Engineered Network" in NSDI 2013*
[ton-16] *M. Chiesa et al. "On the Resiliency of Randomized Routing Against Multiple Edge Failures" in Transactions on Networking 2016*

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4   **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 1 1 1

frr_ports
$b_1b_2b_3b_4b_5b_6b_7$

Encoding FRR input:

- add a packet metadata field *frr_ports*
- map bits to the switch ports

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 1 1 1

frr_ports
$b_1b_2b_3b_4b_5b_6b_7$

$$\underbrace{\hphantom{b_1b_2b_3b_4b_5b_6b_7}}$$

bit-to-port mapping is
1 **2 3 4 1** 2 3

Encoding FRR input:

- add a packet metadata field *frr_ports*
- map bits to the switch ports
- set bit to 1 to include a port
- set bit to 0 to skip a port

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 1 1 1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	0 1 1 1 1 0 0
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

Encoding FRR input:

- add a packet metadata field *frr_ports*
- map bits to the switch ports
- set bit to 1 to include a port
- set bit to 0 to skip a port

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
FRR = 2	1 1 1 1

match		action
frr_ports		fwd
1 * * * * * *		1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	0 1 1 1 1 0 0
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
○ FRR = 2	1 1 1 1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	**0 1 1 1 1 0 0**
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

match		action
frr_ports	port status	fwd
✗ 1 * * * * * *	1 * * *	1
*** 1 * * * * ***	*** 1 * ***	**2**
* * 1 * * * *	* * 1 *	3
* * * 1 * * *	* * * 1	4
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
○ FRR = 2	1 **0** 1 1

port 2 fails

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	**0 1 1 1 1 0 0**
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

match		action
frr_ports	port status	fwd
✗ 1 * * * * * *	1 * * *	1
* 1 * * * * *	✗ * 1 * *	2
*** * 1 * * * ***	*** * 1 ***	**3**
* * * 1 * * *	* * * 1	4
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 **FRR_2 = 2 3 4 1** FRR_3 = 3 4 1 2 FRR_4 = 4 1 2 3

packet metadata	port status
O FRR = 2	1 **0 0** 1

ports 2 and 3 fail

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	**0 1 1 1 1 0 0**
FRR = 3	0 0 1 1 1 1 0
FRR = 4	0 0 0 1 1 1 1

bit-to-port mapping is: 1 2 3 4 1 2 3

match		action
frr_ports	port status	fwd
~~1~~ * * * * * *	1 * * *	1
* 1 * * * * *	* ~~1~~ * *	2
* * 1 * * * *	* * ~~1~~ *	3
* * * **1** * * *	* * * **1**	**4**
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

Encoding FRR in the packet metadata

Input

FRR_1 = 1 2 3 4 FRR_2 = 2 3 4 1 FRR_3 = 3 4 1 2   **FRR_4 = 4 1 2 3**

packet metadata	port status
○ FRR = 4	1 1 1 1

match	action write frr_ports
FRR = 1	1 1 1 1 0 0 0
FRR = 2	0 1 1 1 1 0 0
FRR = 3	0 0 1 1 1 1 0
FRR = 4	**0 0 0 1 1 1 1**

bit-to-port mapping is: 1 2 3 4 1 2 3

match		action
frr_ports	port status	fwd
~~1~~ * * * * * *	1 * * *	1
* ~~1~~ * * * * *	* 1 * *	2
* * ~~1~~ * * * *	* * 1 *	3
* * * **1** * * *	* * * **1**	**4**
* * * * 1 * *	1 * * *	1
* * * * * 1 *	* 1 * *	2
* * * * * * 1	* * 1 *	3

How much space does this encoding save?

Input:

- switch with *k* ports
- 10 circular set of FRR actions

Without encoding:

- k^2 number of TCAM entries

With encoding:

- *2k-1* number of TCAM entries

For *k* = 24

- **92%** less TCAM entries
- 470 instead of 5.760

For *k* = 48

- **96%** less TCAM entries
- 950 instead of 23.040

Implementing existing FRR mechanisms

Why circular FRR sequences?

- Provide resiliency to multiple link failures with small overhead

F10 FRR [NSDI’13]:

- iterates through upward and downward datacenter links

Depth-First-Search FRR [HotSDN’13]:

- iterates through children nodes

K-arc disjoint FRR [Infocom’16]:

- iterates through *k* spanning trees

[nsdi-13] *V. Liu et al. "F10: A Fault-Tolerant Engineered Network" in NSDI 2013*
[HotSDN’13] *Borokhovich et al ””Graph exploration algorithms” in HotSDN 2013*
[ton-16] *M. Chiesa et al. "On the Resiliency of Randomized Routing Against Multiple Edge Failures" in Transactions on Networking 2016*

Implementing existing FRR mechanisms

Why circular FRR sequences?

- Provide resiliency to multiple link failures with small overhead

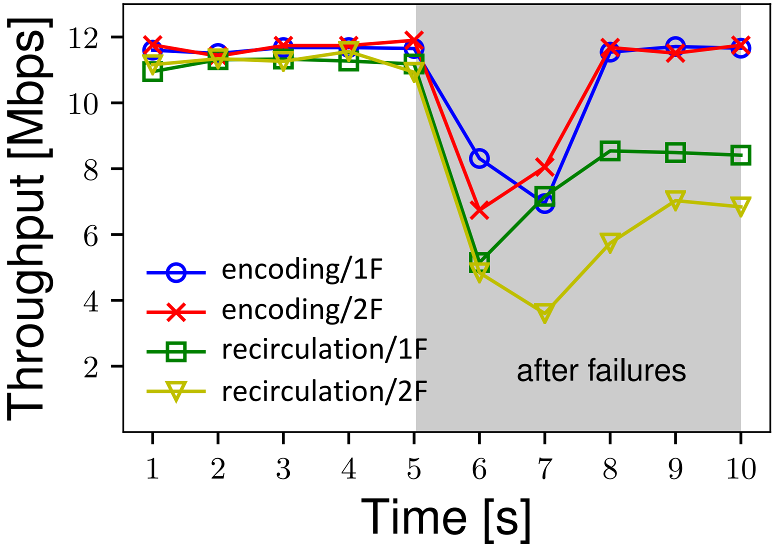
Depth-First-Search FRR [HotSDN'13]:

- iterates through children nodes
- caution: skip parent node

OpenFlow vs P4 (k = 48):

- 56.448 vs 190 TCAM entries
- 99.7% reduction

Preliminary Mininet evaluation:



Towards a P4 FRR primitive

high throughput

low forwarding latency

efficient reroute

small forwarding state

Transform FRR input into a P4 program:

- based on a mix of naïve and encoding TCAM-based approaches
- many challenging optimization problems

Summary

Fast Reroute is a critical functionality in today's network

- requires high throughput, low latency, fast reactivity, small state overhead

P4 does not define an FRR built-in primitive

- compilers must program the P4 pipeline

We propose a relatively simple TCAM-based FRR primitive

- based on bit-level mapping of ports
- no FRR-tailored hardware support
- future work:
  - optimize mapping computation
  - evaluation on real switches

R. Sedar et al.
"Supporting Emerging Applications With Low-Latency Failover in P4"
In ACM SIGCOMM workshop ***NEAT 2018***

Thank you!