S tateless
H ardware-
E nabled
L oad-aware
L oad-balancing

SHELL: Stateless Load-Aware Load Balancing in P4

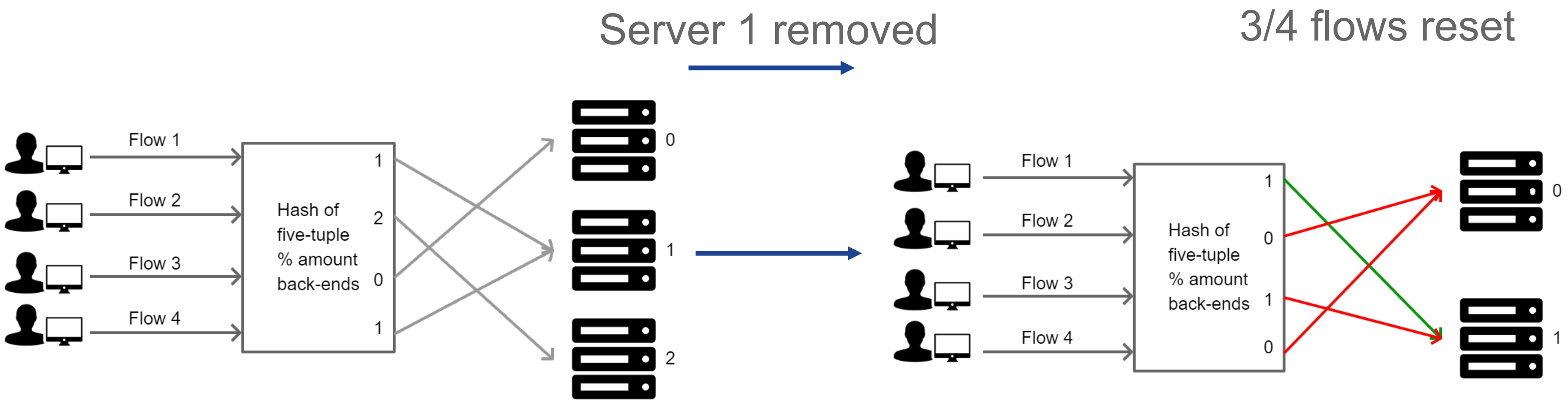
Benoît Pit-Claudel*†, Yoann Desmouceaux*†, Pierre Pfister†, Mark Townsley†*, Thomas Clausen*

*École Polytechnique † Cisco Systems

1st P4EU workshop @IEEE ICNP, Cambridge UK, 24th September 2018

Background and context

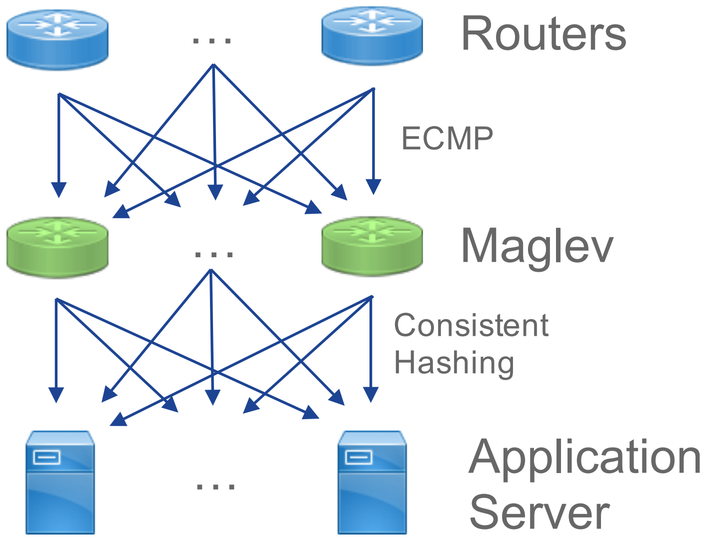
ECMP load-balancing



- ECMP [1] works statelessly on 5-tuples
- Weighted ECMP with active probing [2]
- Common problem: Not resilient to back-end configuration changes

Resilient L3 load-balancing: Maglev

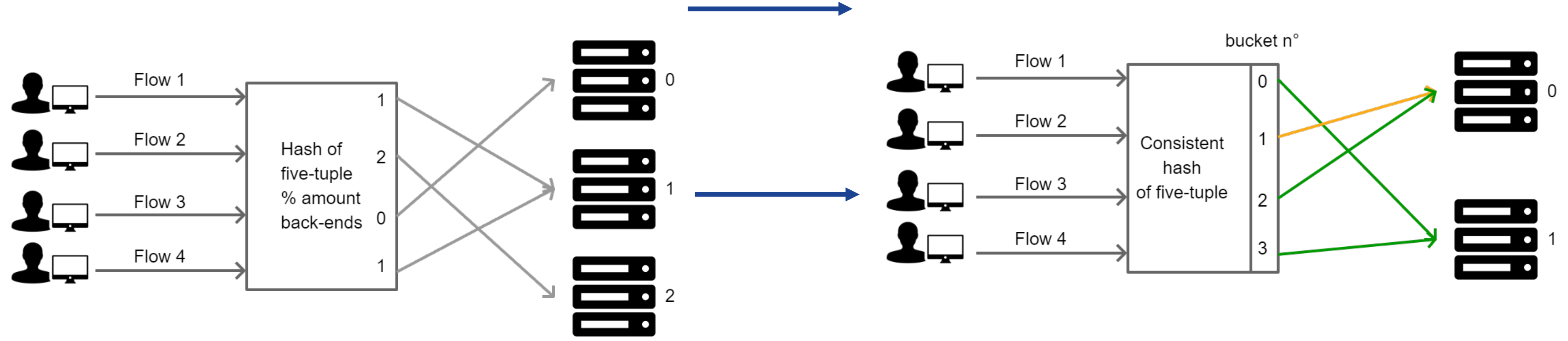
- Maglev [3]:
  - Routers dispatch flows (with ECMP) between Maglev instances
  - Softwarized instances, scalable at will
  - Consistent hashing (buckets): with high probability, flow-to-server assignment is consistent when adding/removing servers
  - Virtual IP address (VIP)
  - Direct Server Return (DSR)
  - Per-flow state (if memory permits)

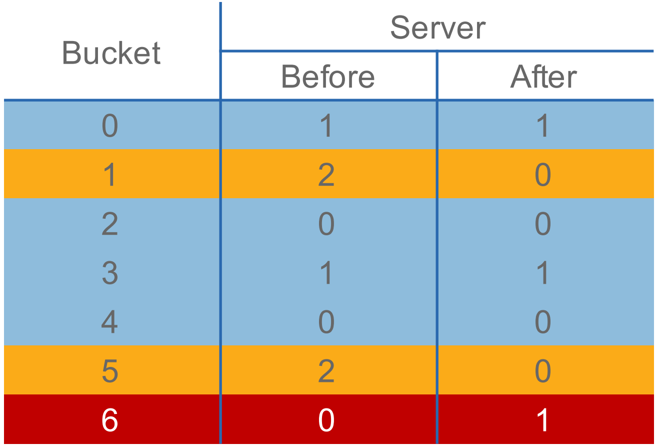


Maglev resiliency example

Server 2 removed

only 1/4 flows reset





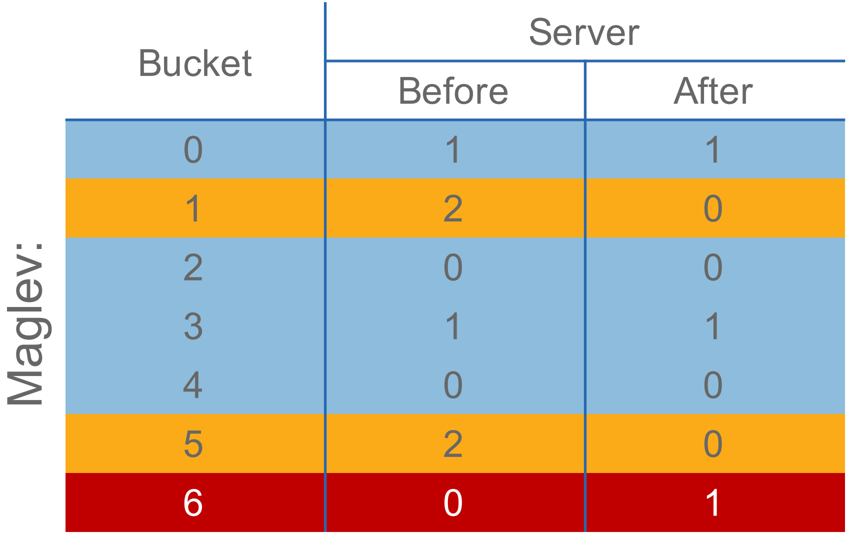
Bucket	Server	
	Before	After
0	1	1
1	2	0
2	0	0
3	1	1
4	0	0
5	2	0
6	0	1

Consistent hashing table stays very similar upon rebuilding

Issues with Maglev

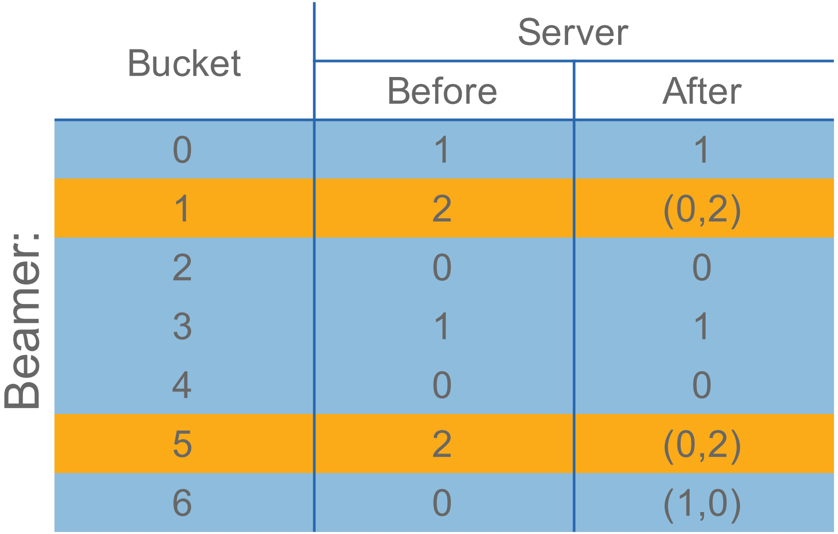
Resiliency	Statefulness	Fairness
>0.3% of bucket-to-server assignment change when server fails occur [3]	One entry per flow in the load-balancer => vulnerable to SYN floods	Does not take the current load of servers into account

Improving on Maglev resiliency: Beamer



Maglev:

Bucket	Server	
	Before	After
0	1	1
1	2	0
2	0	0
3	1	1
4	0	0
5	2	0
6	0	1



Beamer:

Bucket	Server	
	Before	After
0	1	1
1	2	(0,2)
2	0	0
3	1	1
4	0	0
5	2	(0,2)
6	0	(1,0)

- Beamer [4]: in the previous example, 1/7 wrong assignments
- Main idea: embed the **previous configuration** in the packet (w/ IP option)
- Allows for **stateless** implementation of the load-balancer => P4 prototype
- Agent in servers takes care of **daisy-chaining** upon reaching bad a server

Issues with Maglev

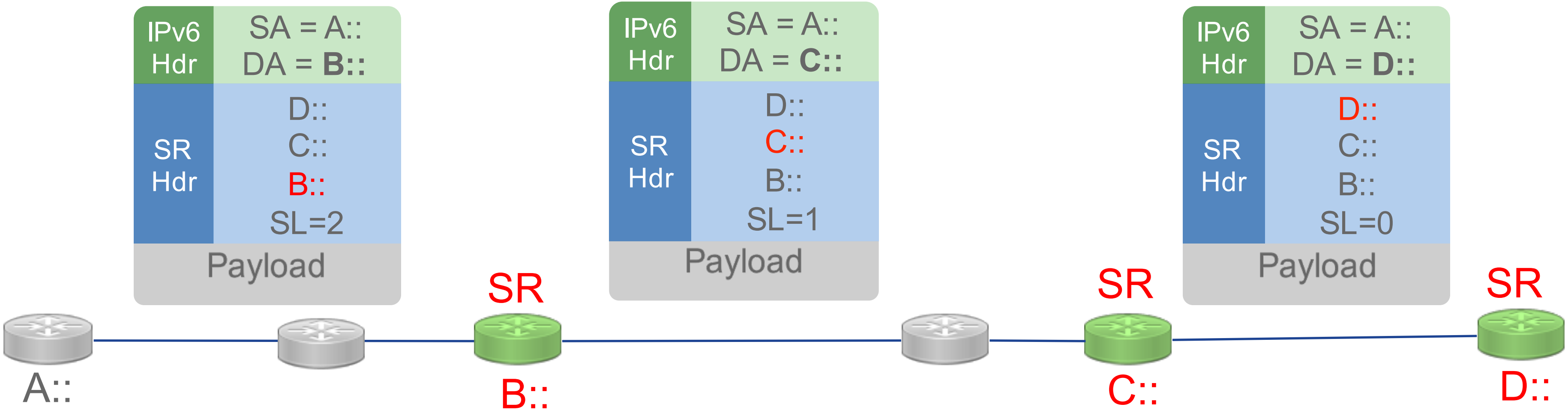
Resiliency	Statefulness	Fairness
>0.3% of bucket-to-server assignment change when server fails occur [3]	One entry per flow in the load-balancer => vulnerable to SYN floods	Does not take the current load of servers into account

Improving on Maglev fairness: 6LB

- 6LB [5]: Almost the same hashing algorithm as Maglev’s but…
- Uses the **power of two choices** [6] with Segment Routing (SR) to dispatch new flows among two pseudo-random candidates
- Goal: consider actual server capacities without control messages

Improving on Maglev fairness: 6LB

- 6LB [5]: Almost the same hashing algorithm as Maglev's but…
- Uses the **power of two choices** [6] with **Segment Routing** (SR) to dispatch new flows among two pseudo-random candidates
- Goal: consider actual server capacities without control messages



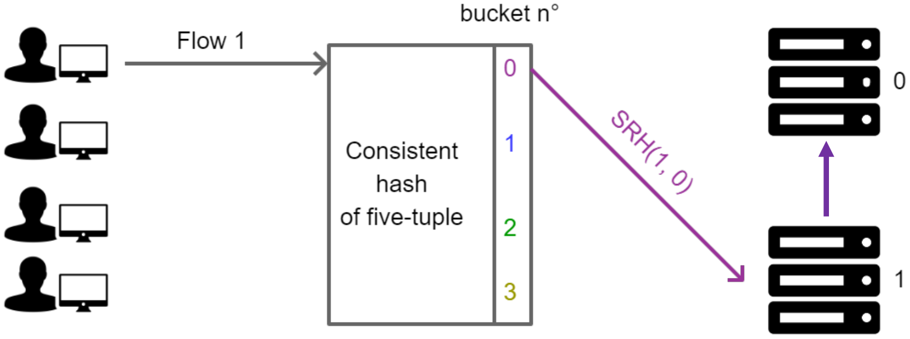
IPv6 Hdr	SA = A:: DA = **B::**
SR Hdr	D:: C:: B:: SL=2
Payload	

IPv6 Hdr	SA = A:: DA = **C::**
SR Hdr	D:: C:: B:: SL=1
Payload	

IPv6 Hdr	SA = A:: DA = **D::**
SR Hdr	D:: C:: B:: SL=0
Payload	

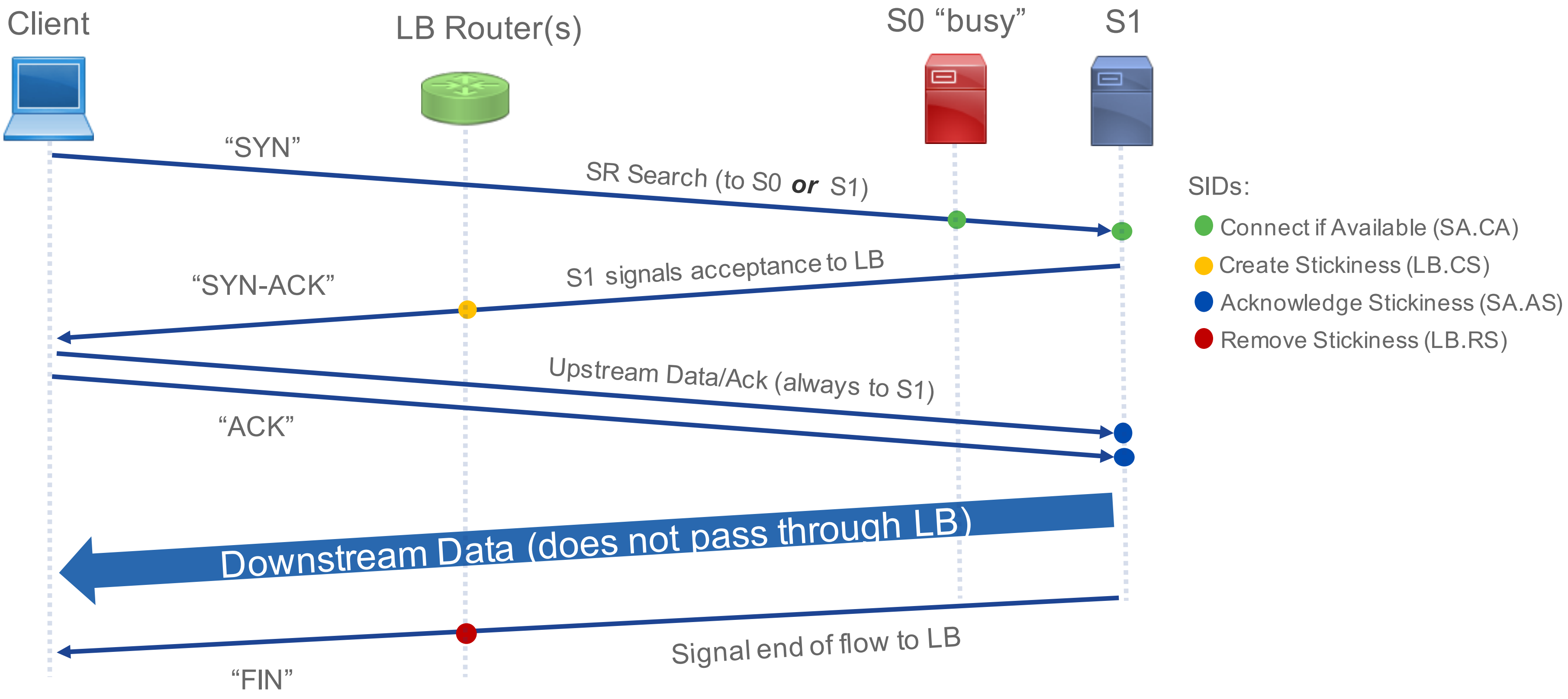
Improving on Maglev fairness: 6LB

Bucket	Candidates (6LB)	Candidate (Maglev)
0	(1, 0)	1
1	(0, 1)	0
2	(0, 1)	0
3	(0, 1)	0
4	(0, 1)	0
5	(1, 0)	1
6	(1, 0)	1



- Server 1 is already loaded, so it forwards the connection request to its next candidate, server 0.
- State is then installed in the LB to map the 5-tuple to server 0.

6LB requires state



Client

LB Router(s)

S0 “busy”

S1

“SYN”

SR Search (to S0 ***or*** S1)

S1 signals acceptance to LB

“SYN-ACK”

Upstream Data/Ack (always to S1)

“ACK”

Downstream Data (does not pass through LB)

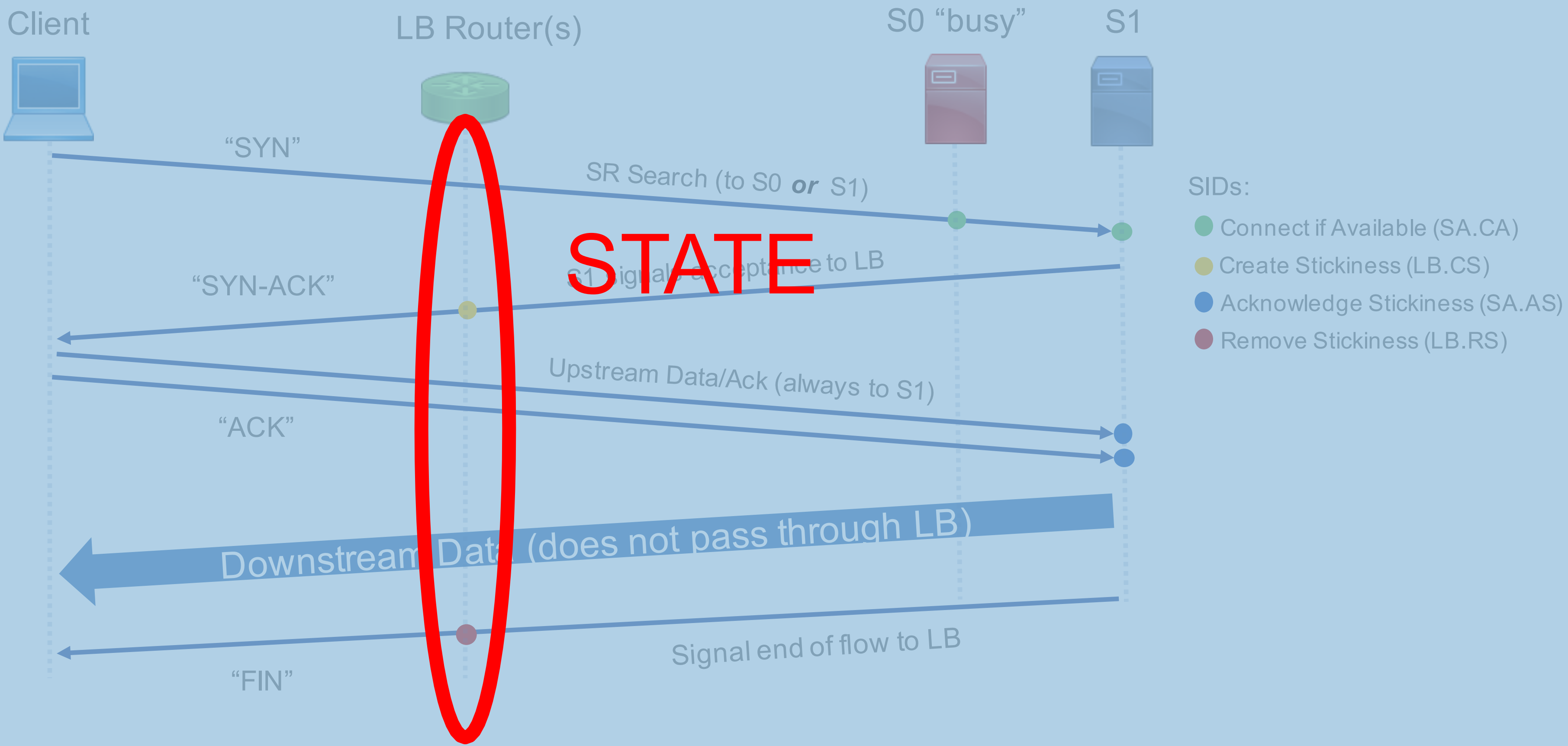
Signal end of flow to LB

“FIN”

SIDs:

- 🟢 Connect if Available (SA.CA)
- 🟡 Create Stickiness (LB.CS)
- 🔵 Acknowledge Stickiness (SA.AS)
- 🔴 Remove Stickiness (LB.RS)

6LB requires state



Client

LB Router(s)

S0 “busy”

S1

“SYN”

SR Search (to S0 ***or*** S1)

STATE

S1 Signal acceptance to LB

“SYN-ACK”

Upstream Data/Ack (always to S1)

“ACK”

Downstream Data (does not pass through LB)

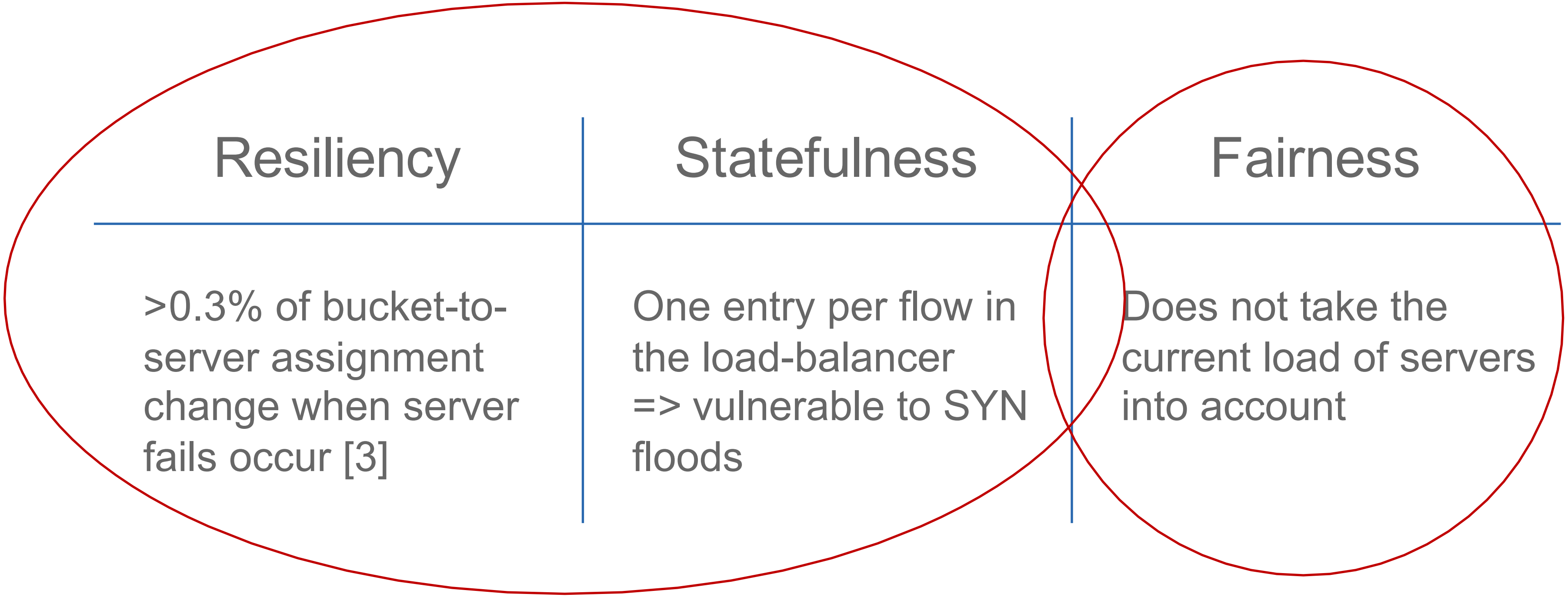
Signal end of flow to LB

“FIN”

SIDs:

- ● Connect if Available (SA.CA)
- ● Create Stickiness (LB.CS)
- ● Acknowledge Stickiness (SA.AS)
- ● Remove Stickiness (LB.RS)

Issues with Maglev



Resiliency	Statefulness	Fairness
>0.3% of bucket-to-server assignment change when server fails occur [3]	One entry per flow in the load-balancer => vulnerable to SYN floods	Does not take the current load of servers into account

Can we get both?

SHELL Overview

What information do we need?

Consistent hashing table

Size ~ 20 x number of servers

bucket	Potential servers	
	Choice 1	Choice 2
1	281	12
2	1	8
…	2	42
…	42	90
…	12	20
…	21	16

What information do we need?

Consistent hashing table

Size ~ 20 x number of servers

SYN

bucket	Potential servers	
	Choice 1	Choice 2
1	**281**	**12**
2	1	8
…	2	42
…	42	90
…	12	20
…	21	16

Lookup = bucket (from 5 tuple)
Build SRH with some (e.g. 2) candidates for bucket

What information do we need?

Consistent hashing table

Size ~ 20 x number of servers

SYN

bucket	Potential servers	
	Choice 1	Choice 2
1	**281**	**12**
2	1	8
…	2	42
…	42	90
…	12	20
…	21	16

Lookup = bucket (from 5 tuple)
Build SRH with some (e.g. 2) candidates for bucket

Flow table

Five tuple	Assigned server
(fd00::0, fd00::1, TCP, 9999, 80)	**281**
…	7
…	23[illegible]
…	182
…	12
…	16
…	…
…	…
…	…
…	…
…	…
…	…
…	…

Size ~ number of flows

Where to store?

Where to store flow information?

- We don’t want state (flow table) in the LB
- The server accepting the connection (1 or 2) must find a way (a field in the packet) to communicate that to the client, which will be reflected and used by the LB

- i.e. we need a **covert channel**
- An agent runs in the servers
  - records the index of the accepting server of SYN packets
  - transmits it back to the client on subsequent packets, through the covert channel

What information do we need?

Consistent hashing table

Size ~ 20 x number of servers

bucket	Potential servers	
	Choice 1	Choice 2
1	281	12
2	1	8
…	2	42
…	42	90
…	12	20
…	21	16

Flow table

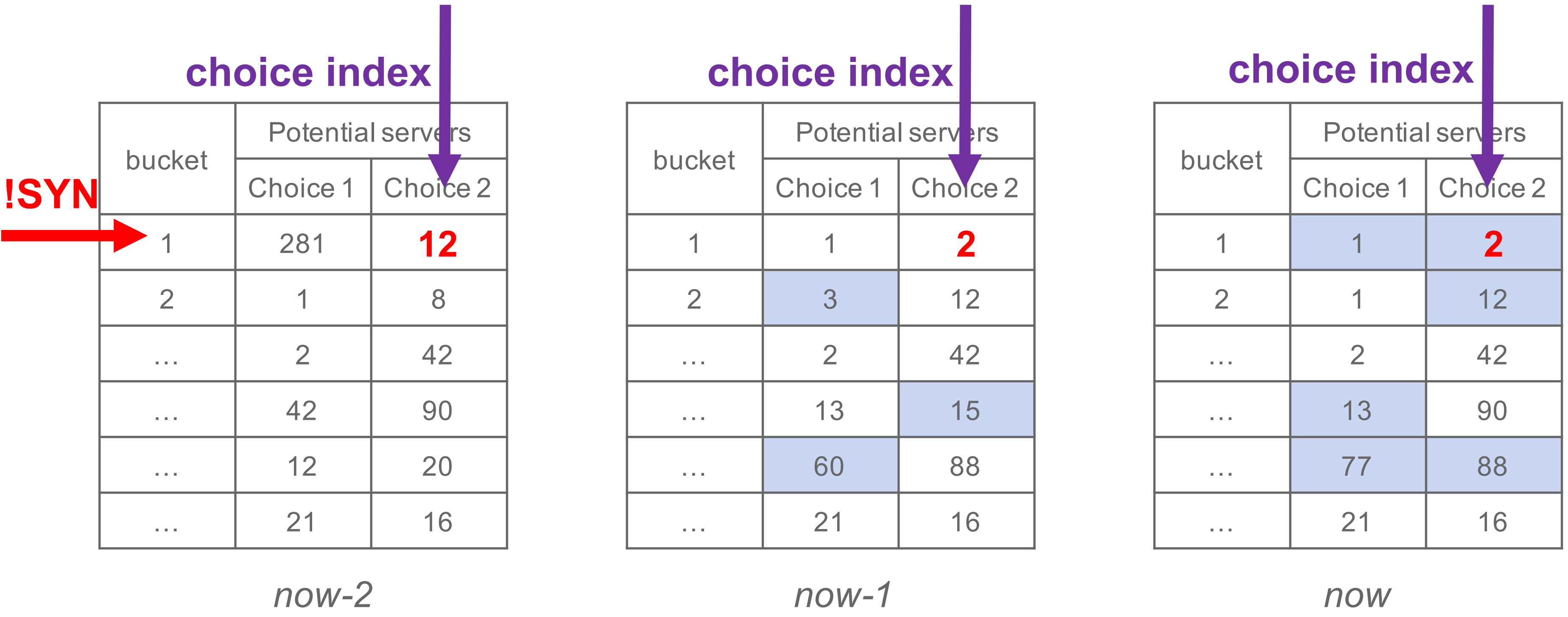
Five tuple	Assigned server
(fd00::0, fd00::1, TCP, 9999, 80)	1
…	7
…	[illegible]
…	[illegible]
…	12
…	16
…	…
…	…
…	…
…	…
…	…
…	…
…	…

Size ~ number of flows

Flow info distributed in packets

Lookup = bucket (from 5 tuple)
+ choice index (from covert channel in packet)

Resiliency – SHELL History Matrix



!SYN

choice index

bucket	Potential servers	
	Choice 1	Choice 2
1	281	**12**
2	1	8
…	2	42
…	42	90
…	12	20
…	21	16

now-2

choice index

bucket	Potential servers	
	Choice 1	Choice 2
1	1	**2**
2	3	12
…	2	42
…	13	15
…	60	88
…	21	16

now-1

choice index

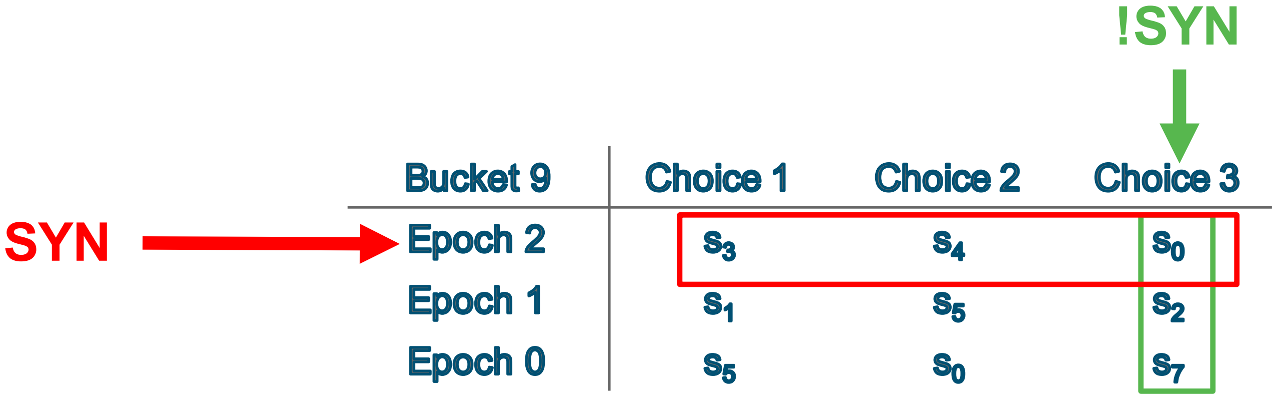
bucket	Potential servers	
	Choice 1	Choice 2
1	1	**2**
2	1	12
…	2	42
…	13	90
…	77	88
…	21	16

now

Build SRH with some (e.g. 2) old values of bucket+choice index

History Matrix: Summary

- SYN: build SRH with candidates for bucket (row in matrix)
- Non-SYN: build SRH with history for both bucket *and* choice index as found in covert channel (column in matrix)



Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 2	s_3	s_4	s_0
Epoch 1	s_1	s_5	s_2
Epoch 0	s_5	s_0	s_7

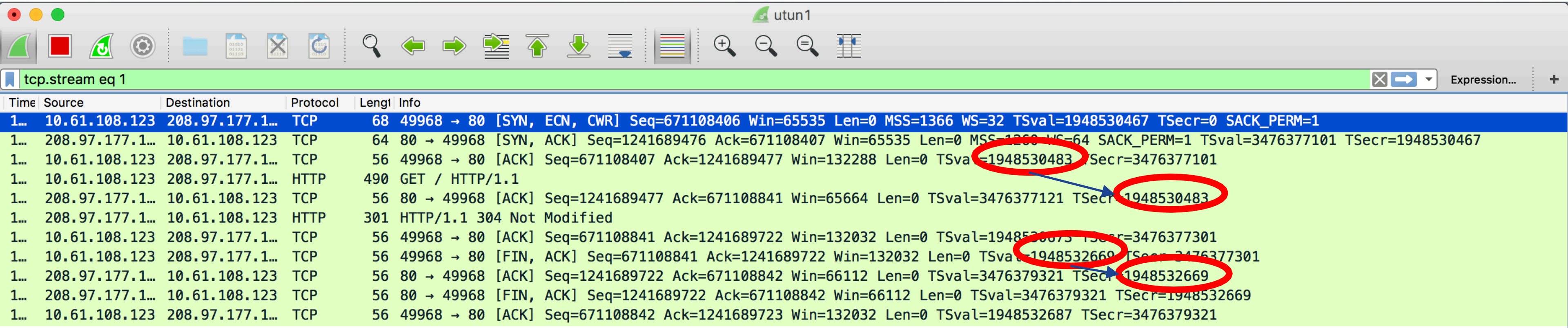
Covert channel

- Covert channel: field echoed by the client without him knowing SHELL encodes data in it
- Easy in QUIC (64 bits in connection ID), but QUIC isn't universally adopted

→ Challenge: Use TCP: what fields are echoed in a TCP session?

→ SHELL implementation uses TCP Timestamp, with only a few bits

→ Other possibilites…



```
tcp.stream eq 1
```

```
Time Source         Destination    Protocol Lengt Info
1…   10.61.108.123  208.97.177.1…  TCP        68  49968 → 80 [SYN, ECN, CWR] Seq=671108406 Win=65535 Len=0 MSS=1366 WS=32 TSval=1948530467 TSecr=0 SACK_PERM=1
1…   208.97.177.1…  10.61.108.123  TCP        64  80 → 49968 [SYN, ACK] Seq=1241689476 Ack=671108407 Win=65535 Len=0 MSS=1260 WS=64 SACK_PERM=1 TSval=3476377101 TSecr=1948530467
1…   10.61.108.123  208.97.177.1…  TCP        56  49968 → 80 [ACK] Seq=671108407 Ack=1241689477 Win=132288 Len=0 TSval=1948530483 TSecr=3476377101
1…   10.61.108.123  208.97.177.1…  HTTP      490  GET / HTTP/1.1
1…   208.97.177.1…  10.61.108.123  TCP        56  80 → 49968 [ACK] Seq=1241689477 Ack=671108841 Win=65664 Len=0 TSval=3476377121 TSecr=1948530483
1…   208.97.177.1…  10.61.108.123  HTTP      301  HTTP/1.1 304 Not Modified
1…   10.61.108.123  208.97.177.1…  TCP        56  49968 → 80 [ACK] Seq=671108841 Ack=1241689722 Win=132032 Len=0 TSval=1948530673 TSecr=3476377301
1…   10.61.108.123  208.97.177.1…  TCP        56  49968 → 80 [FIN, ACK] Seq=671108841 Ack=1241689722 Win=132032 Len=0 TSval=1948532669 TSecr=3476377301
1…   208.97.177.1…  10.61.108.123  TCP        56  80 → 49968 [ACK] Seq=1241689722 Ack=671108842 Win=66112 Len=0 TSval=3476379321 TSecr=1948532669
1…   208.97.177.1…  10.61.108.123  TCP        56  80 → 49968 [FIN, ACK] Seq=1241689722 Ack=671108842 Win=66112 Len=0 TSval=3476379321 TSecr=1948532669
1…   10.61.108.123  208.97.177.1…  TCP        56  49968 → 80 [ACK] Seq=671108842 Ack=1241689723 Win=132032 Len=0 TSval=1948532687 TSecr=3476379321
```

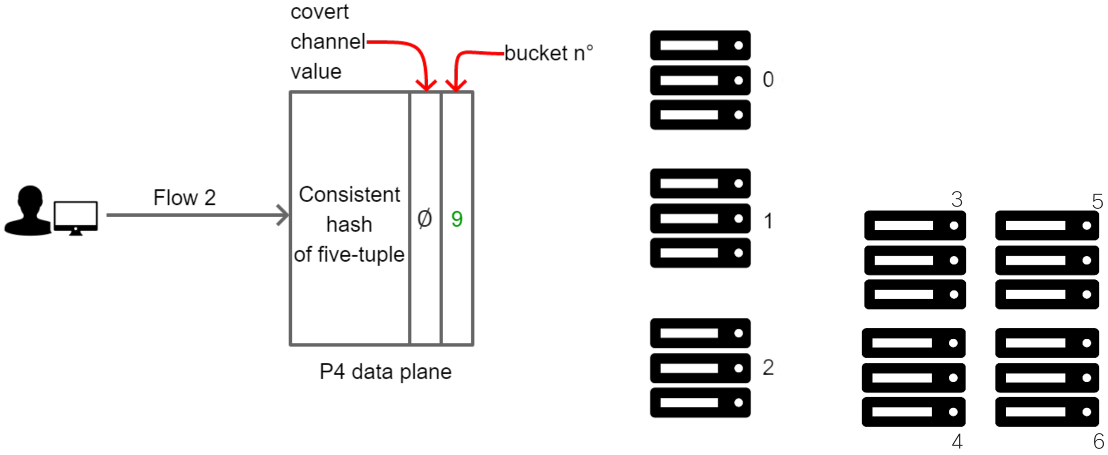
Life of a flow

Life of a flow (1/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 0	s_1	s_0	s_2

Data plane

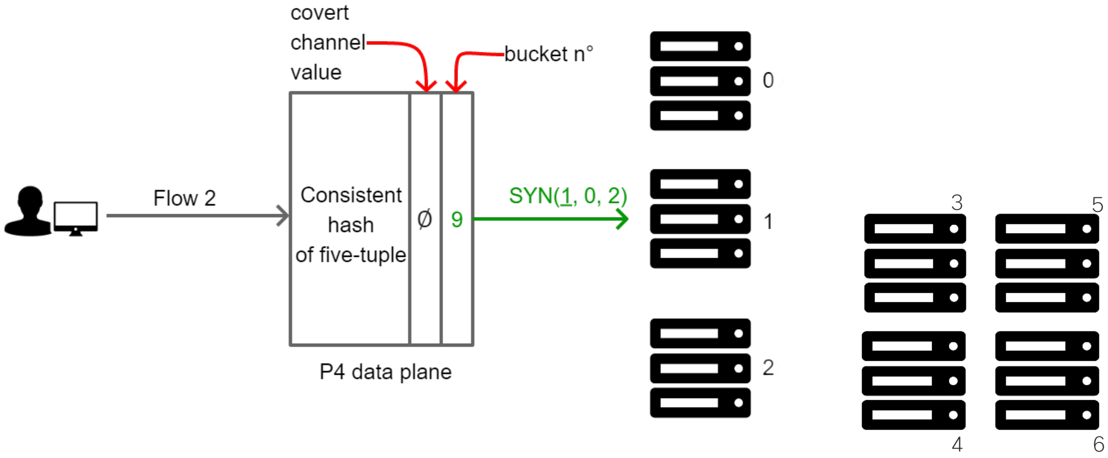


Life of a flow (2/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 0	s_1	s_0	s_2

Data plane

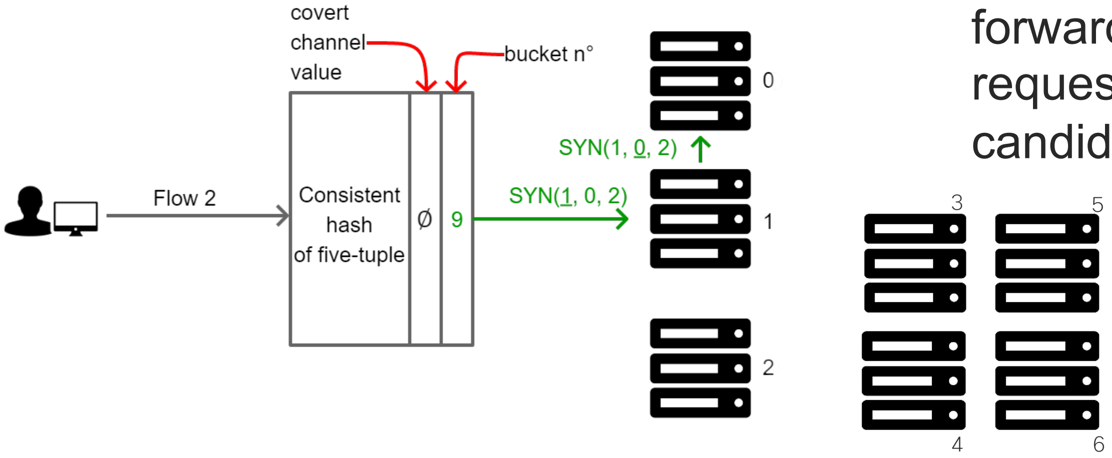


Life of a flow (3/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 0	s_1	s_0	s_2

Data plane



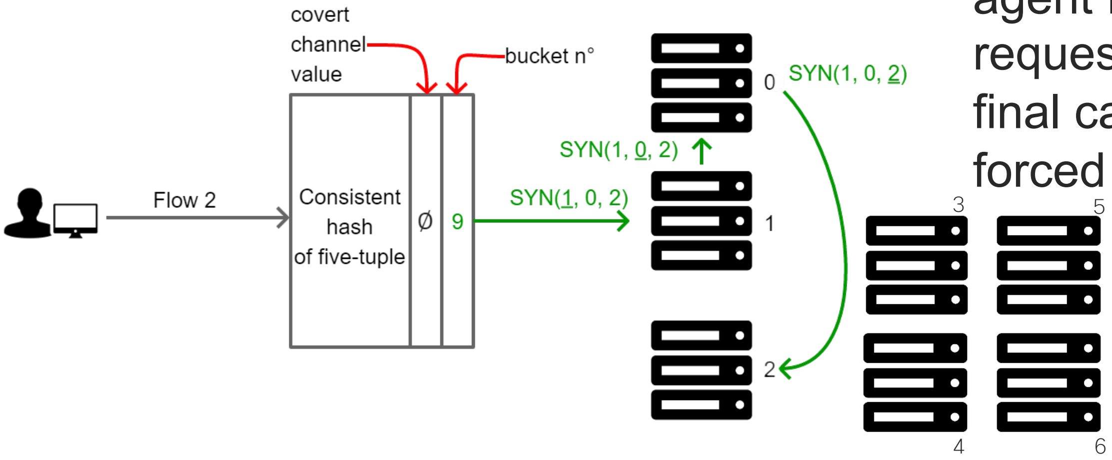
Server 1 is overloaded, the server agent forwards the SYN request to the second candidate

Life of a flow (4/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 0	s_1	s_0	s_2

Data plane



Server 0 is also overloaded, the server agent forwards the SYN request to the third and final candidate, who is forced to accept it

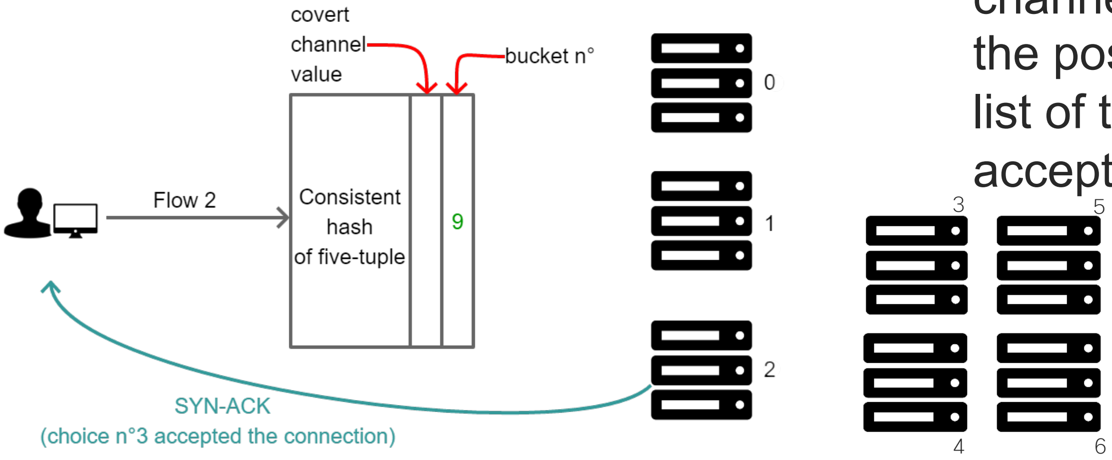
Life of a flow (5/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 0	s_1	s_0	s_2

The server agent writes encodes in the covert channel of the SYN-ACK the position in the choice list of the server that accepted the connection

Data plane

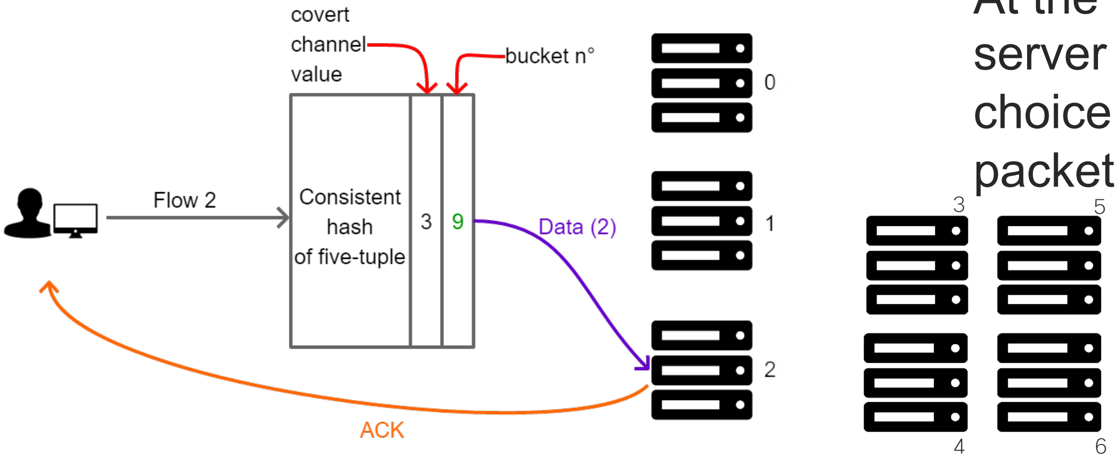


Life of a flow (6/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 0	s_1	s_0	$\boxed{s_2}$

Data plane



The load-balancer reads the covert channel value. At the moment, the only server that can be third choice is s2, so the packet is sent to s2

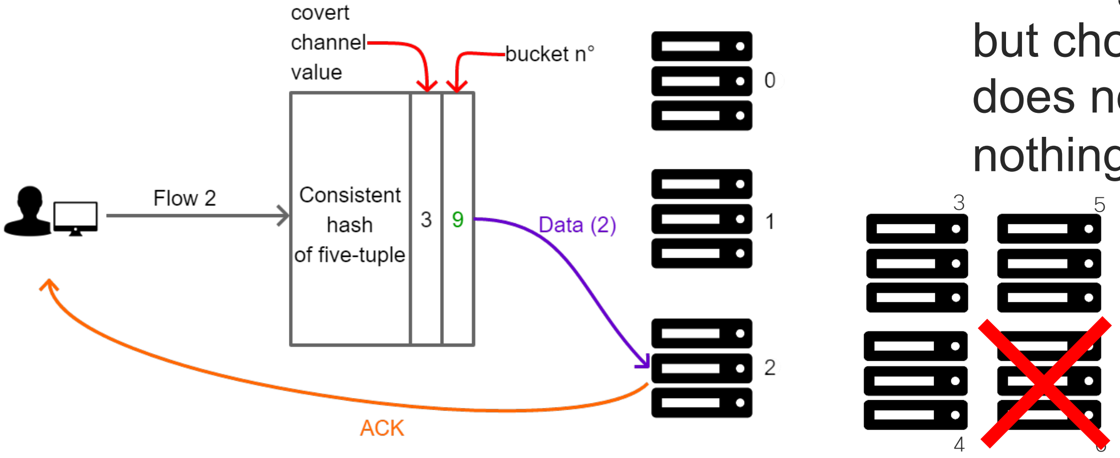
Life of a flow (7/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 1	s_1	s_5	s_2
Epoch 0	"	s_0	"

One backend reconfiguration happens, but choice 3 for bucket 3 does not change, so nothing changes

Data plane

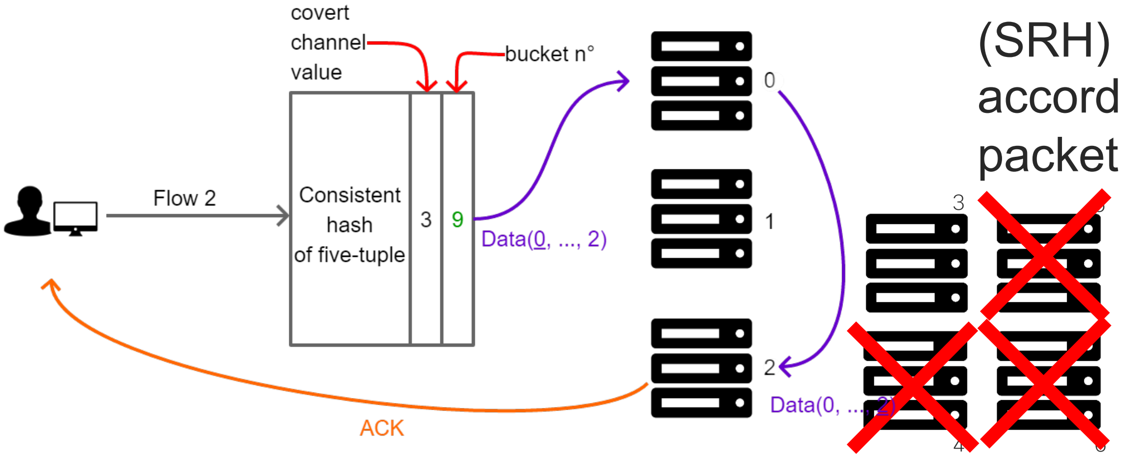


Life of a flow (8/8)

History Matrix

Bucket 9	Choice 1	Choice 2	Choice 3
Epoch 2	s_3	s_4	s_0
Epoch 1	s_1	s_5	s_2
Epoch 0	"	s_0	"

Data plane

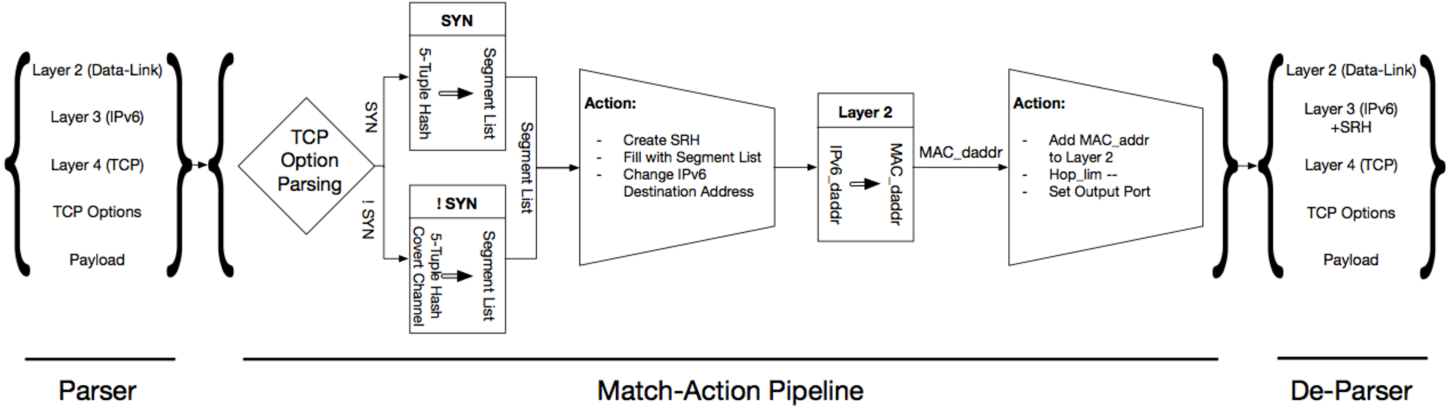


Other changes happen, the inserted SR header (SRH) is modified accordingly, and the packet reaches s2

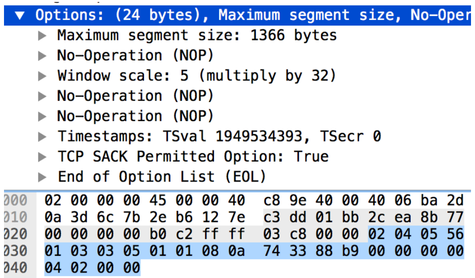
Evaluation

P4-NetFPGA Implementation

- P4 dataplane for NETFPGA-SUME: TCP timestamp parsing + SRH insertion



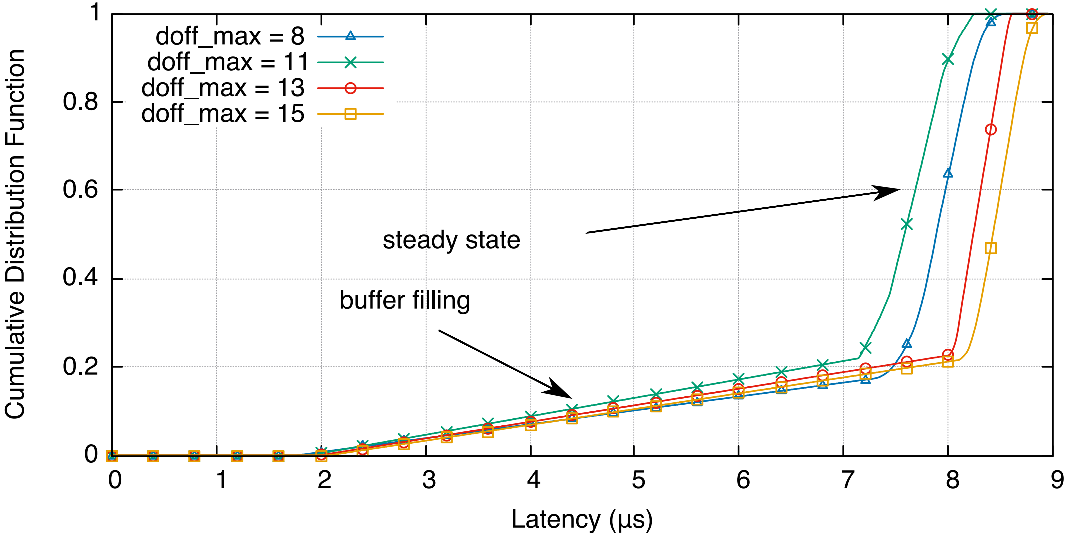
- A bit tricky due to “TLV” (type/length/value) fields
  - Only a subset of TCP options parsed
  - Namely SACK (different lengths) and timestamps
  - Different maximum parsing depths evaluated



```
▼ Options: (24 bytes), Maximum segment size, No-Oper
  ▶ Maximum segment size: 1366 bytes
  ▶ No-Operation (NOP)
  ▶ Window scale: 5 (multiply by 32)
  ▶ No-Operation (NOP)
  ▶ No-Operation (NOP)
  ▶ Timestamps: TSval 1949534393, TSecr 0
  ▶ TCP SACK Permitted Option: True
  ▶ End of Option List (EOL)
000  02 00 00 00 45 00 00 40  c8 9e 40 00 40 06 ba 2d
010  0a 3d 6c 7b 2e b6 12 7e  c3 dd 01 bb 2c ea 8b 77
020  00 00 00 00 b0 c2 ff ff  03 c8 00 00 02 04 05 56
030  01 03 03 05 01 01 08 0a  74 33 88 b9 00 00 00 00
040  04 02 00 00
```

P4-NetFPGA dataplane evaluation

- Latency = 9µs; Throughput = 60 million packets/s
- Different "TCP option parsing complexities" (maximum size of TCP option field) implemented/evaluated



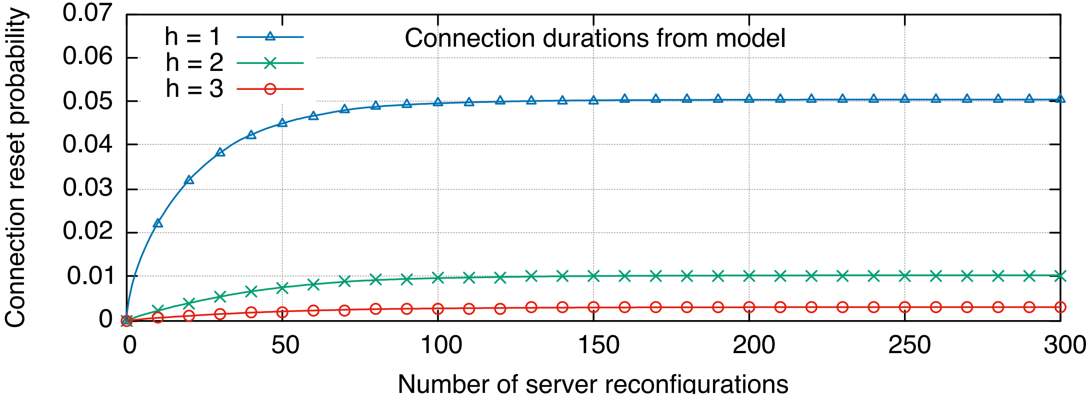
P4-NetFPGA dataplane evaluation

- Latency = 9µs; Throughput = 60 million packets/s
- Different “TCP option parsing complexities” (maximum size of TCP option field) implemented/evaluated

Max data offset	LUT	LUT as RAM	FF	BRAM
8	36.9%	19.4%	33.3%	59.3%
11	40.1%	22.0%	36.4%	63.2%
13	43.8%	24.9%	40.2%	67.7%
15	48.7%	28.6%	45.8%	74.1%

Consistent hashing resiliency evaluation

- Connection duration model built from:
  - A model of the number of back-end reconfigurations per second [7]
  - A model of connection durations [8]
- In real life situations, about 5 times less connections lost than with Maglev equivalent (where history depth = 1)



Conclusion/References

- No monitoring, but application-informed decisions
- Using SRv6 to direct one query to multiple candidates
- Using covert channel to steer to server having accepted
- Consistent hashing history matrix for resiliency
- Stateless P4-NetFPGA prototype => low latency/high throughput
- Future work: large-scale experiment on actual H/W
- Future work: hybrid stateful/stateless approach

[1] Thaler, D., & Hopps, C. (2000). *Multipath issues in unicast and multicast next-hop selection*. *IETF RFC 2991*.
[2] Aghdai, A., *et al.* (2018). Spotlight: Scalable Transport Layer Load Balancing for Data Center Networks. *arXiv preprint arXiv:1806.08455*.
[3] Eisenbud, D. E., *et al.* (2016). Maglev: A Fast and Reliable Software Network Load Balancer. In *USENIX NSDI* (pp. 523-535).
[4] Olteanu, V., *et* al. (2018). Stateless datacenter load-balancing with Beamer. In *USENIX NSDI* (pp. 125-139).
[5] Desmouceaux, Y., *et al.* (2018). 6LB: Scalable and Application-Aware Load Balancing with Segment Routing. *IEEE/ACM TON 26*(2), 819-834.
[6] Mitzenmacher, M. (2001). The power of two choices in randomized load balancing. *IEEE TPDS*, *12*(10), 1094-1104.
[7] Miao, R *et al.* (2017). Silkroad: Making stateful layer-4 load balancing fast and cheap using switching ASICs". In *ACM SIGCOMM* (pp. 15–28).
[8] Roy, A. *et al.* (2015) Inside the social network's (datacenter) network. In *ACM SIGCOMM* (pp. 123–137)