Orchestrating a brighter world NEC

FlowBlaze

Stateful Packet Processing in Hardware

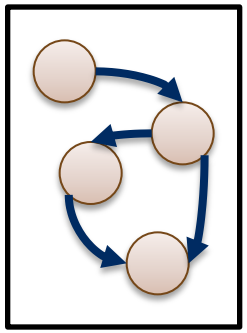
Salvatore Pontarelli, **Roberto Bifulco**, Marco Bonola, Carmelo Cascone, Marco Spaziani, Valerio Bruschi, Davide Sanvito, Giuseppe Siracusano, Antonio Capone, Michio Honda, Felipe Huici, Giuseppe Bianchi

UNIVERSITA' DEGLI STUDI
DI ROMA TOR VERGATA

consorzio nazionale
interuniversitario
per le telecomunicazioni

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 761493 ("5GTANGO") and No.762057 ("5G-PICTURE")

*Now at ONF



Roberto

State Machines

State Machines

State machines to the rescue of complex forms

Łukasz Makuch Follow

May 6, 2017 · 6 min read

Tackling UI State Mach

Carlos Galarza Foll

The Rise Of The State Machines

QUICK SUMMARY ↬ *The UI development became difficult in the*

20

UI,

Sha

Kinetic, NSDI'15

Figure 1: *Intrusion detection FSM.*

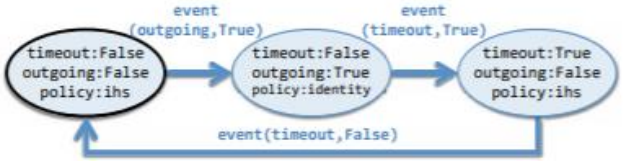


Figure 2: *Stateful firewall FSM.*

tion. Finally, to show Kinetic's generality, we present a MAC learning switch implementation.

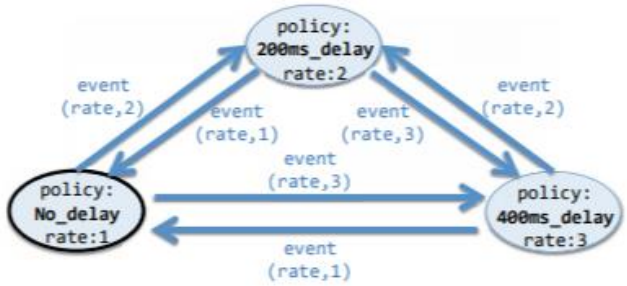
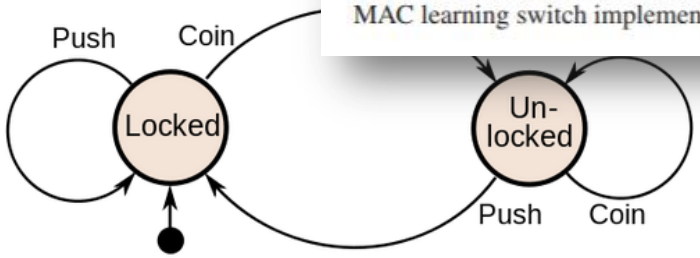


Figure 3: *Data usage-based rate limiter FSM.*

packets (*e.g.*, all packets from the same host, in the case of the previous example). Each group of packets has a separate FSM instance; packets in the same group will always be in the same state. We call such a group of packets a *located packet equivalence class (LPEC).*




```
        }
    }
    Unlock()
    for {
        ev := <-ch
        if ev.Push() {
            break
        }
    }
    Lock()
}
}
```

Coin

Coin

Push

Un-
locked

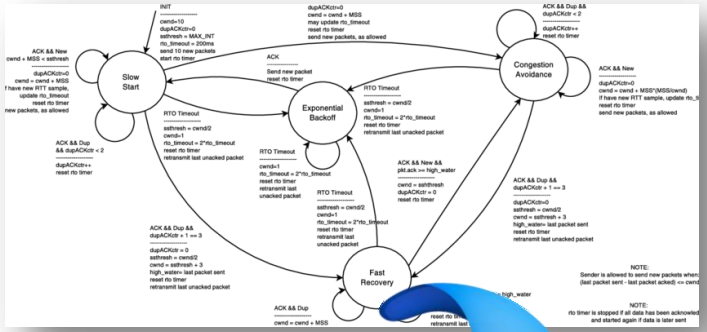
Programmable NICs

NICs with programmable ASICs, SoC, FPGAs…

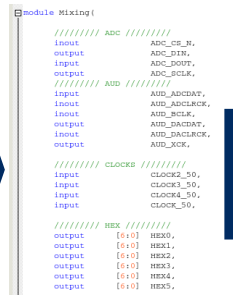
E.g., Microsoft [AccelNet NSDI '17, NSDI '18]

Programming them is Hard!

Network Function Logic



NetFPGA SUME

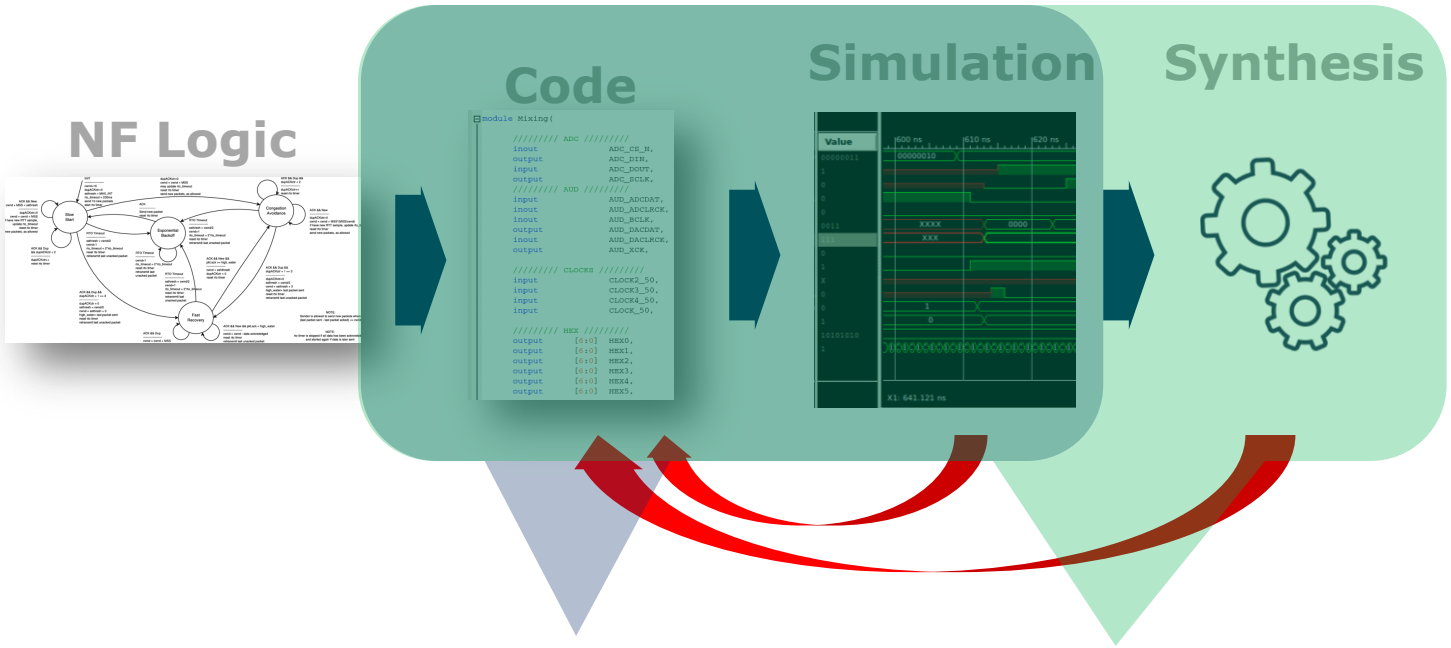


Code

Simulate

Synthesis

Making programming easier



High-level Synthesis

Faster programming
Expressive

Hardware expertise

ClickNP [Sigcomm '16],
Emu [ATC '17]

Match-Action Abstraction

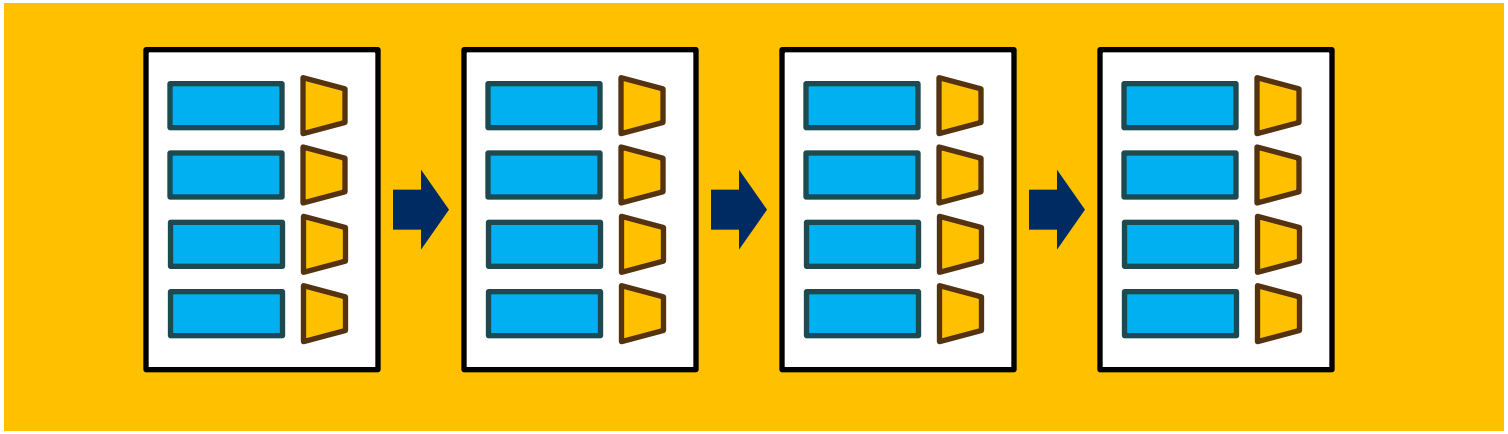
Faster programming
NF Logic focused

Limited support for state

P4 [CCR '14],
Domino [Sigcomm '16]

Match-Action Abstraction Limitations

Match-Action pipeline



State in tables

large

read only
(wr from cplane)

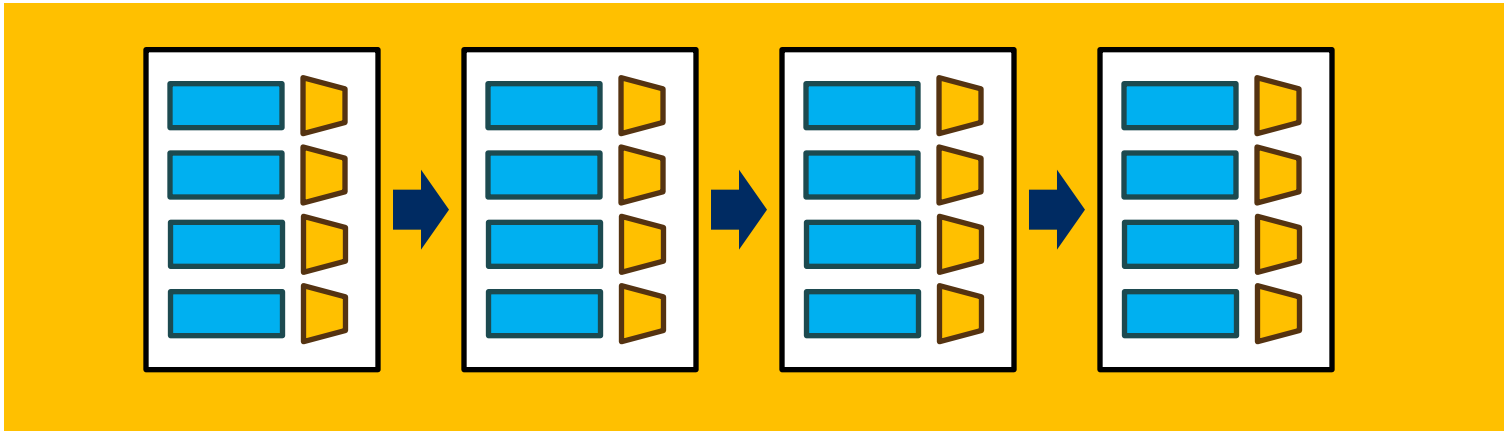
State in registers

small

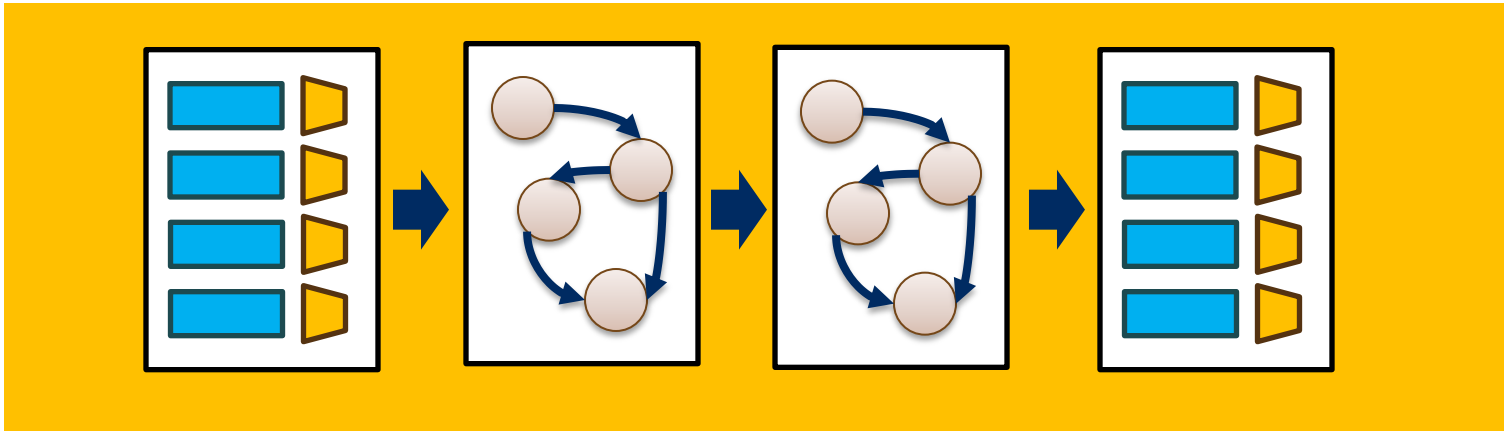
read/write

Extending Match-Action abstractions

Match-Action pipeline

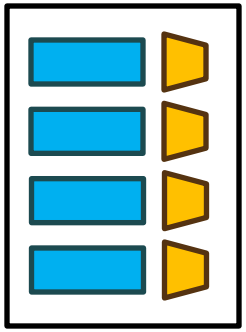


FlowBlaze

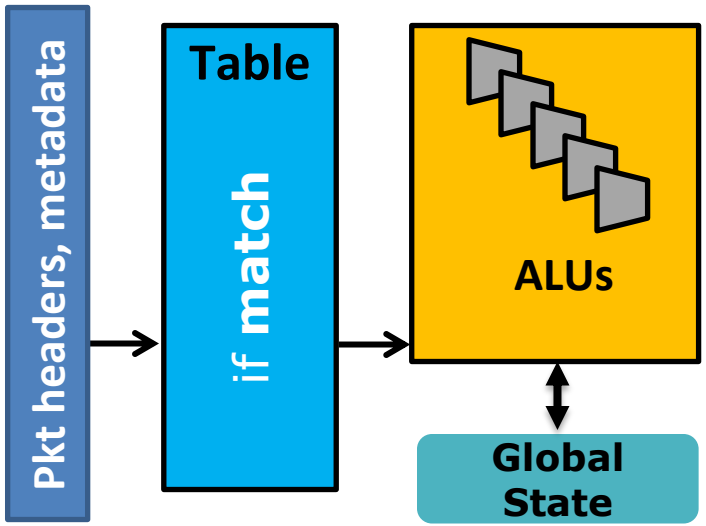


Match-Action vs Finite State Machine (FSM)

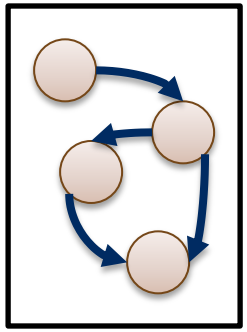
if **match**
then **action**



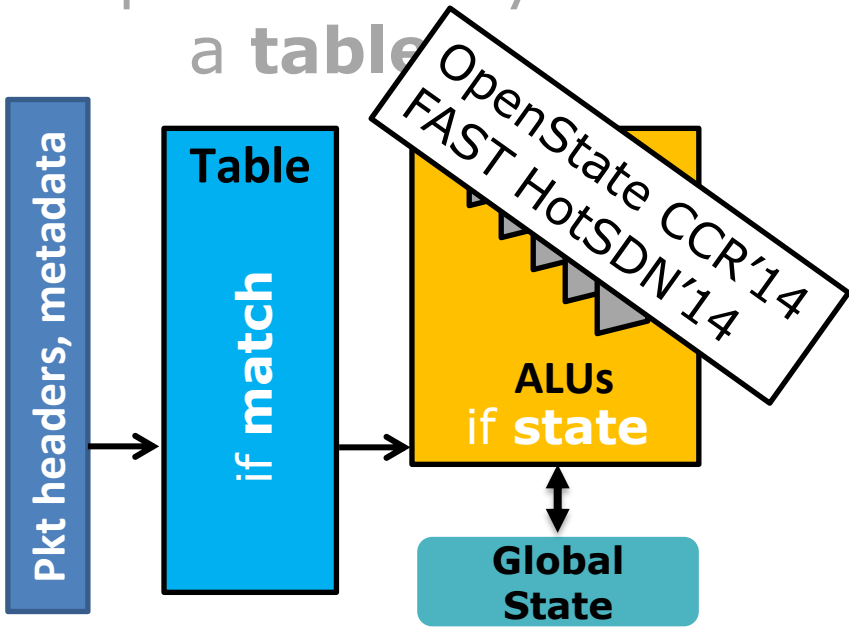
a **table**



if (**match**, **state**)
then **action**

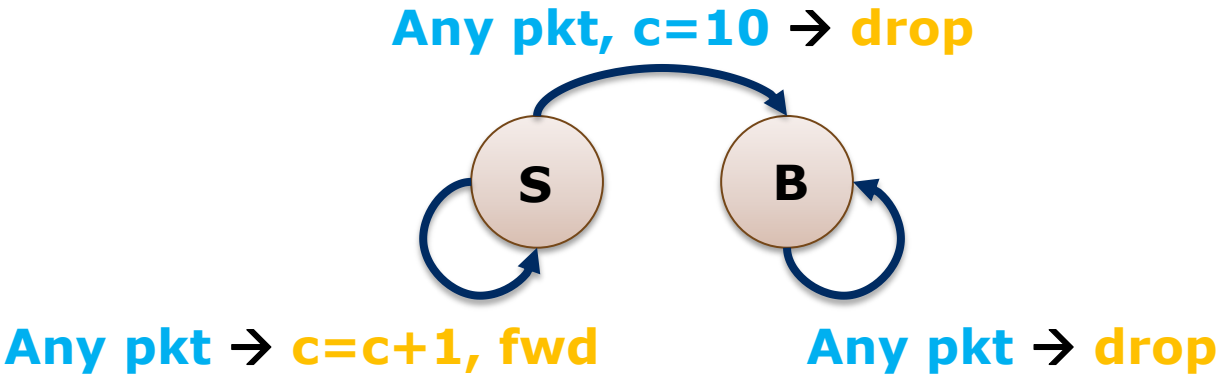


represented by
a **table**



Multiple state machines?

Example: Drop a flow after its 10th packet



Flow ID	State
IPdst = 192.168.0.1	S
IPdst = 192.168.0.2	S
IPdst = 192.168.0.3	B

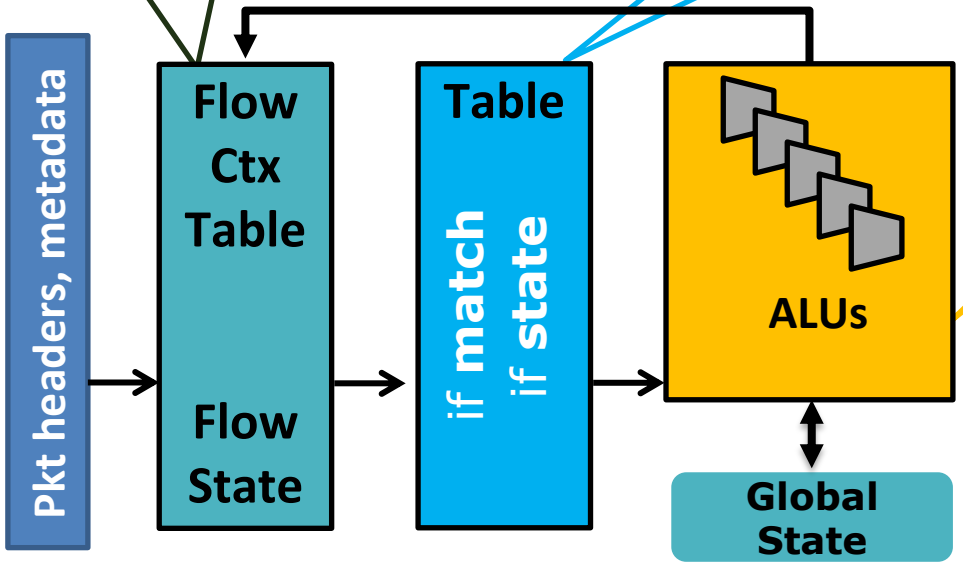
Each flow's FSM evolves on its own

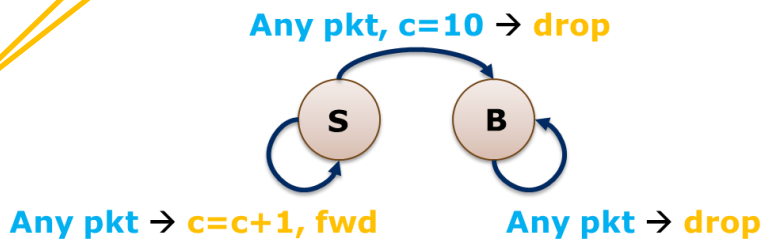
Per-flow state is common in network functions

Introducing per-flow state

Flow ID (IP dst)	State	Register
192.168.0.1	S	c=7
192.168.0.2	S	c=4
192.168.0.3	B	c=10

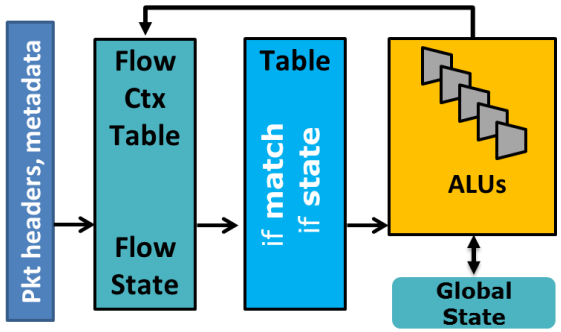
Pkt Header	State	Cond.	Action
*	B	*	drop
*	S	c=10	State=B, drop
*	S	c<10	fwd





Implementation issues

Insertion in the flow context table



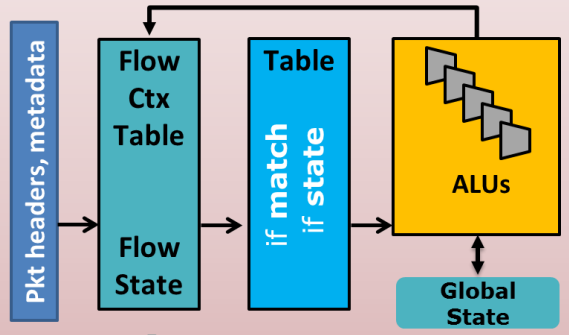
Cuckoo hash table

Insertion = new flow

Variable insertion time

Implementation issues

Insertion in the flow context table



Cuckoo hash table

Insertion = new flow

Variable insertion time

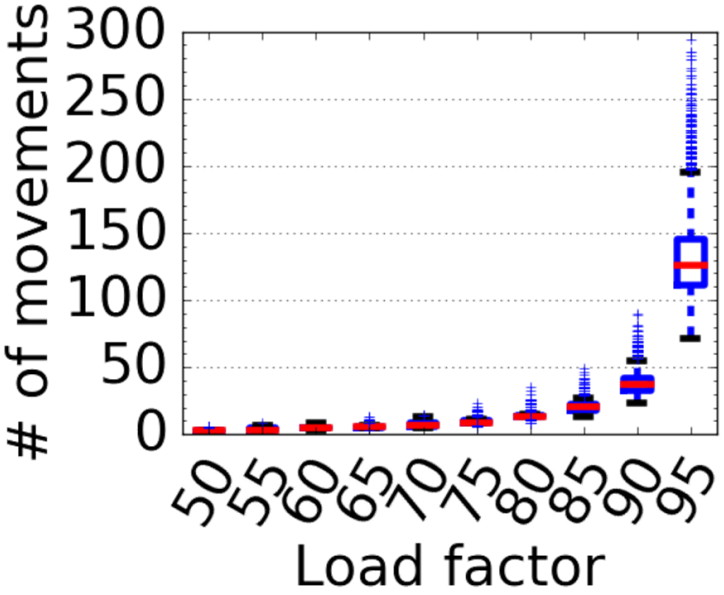
Handling variable insertion time

Flow table: Cuckoo hash

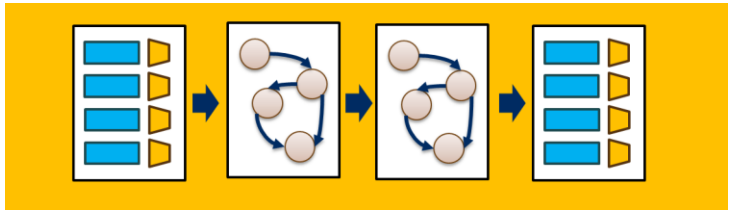
Efficient

Constant lookup-time

Variable insertion-time



Waiting for Insertion!!



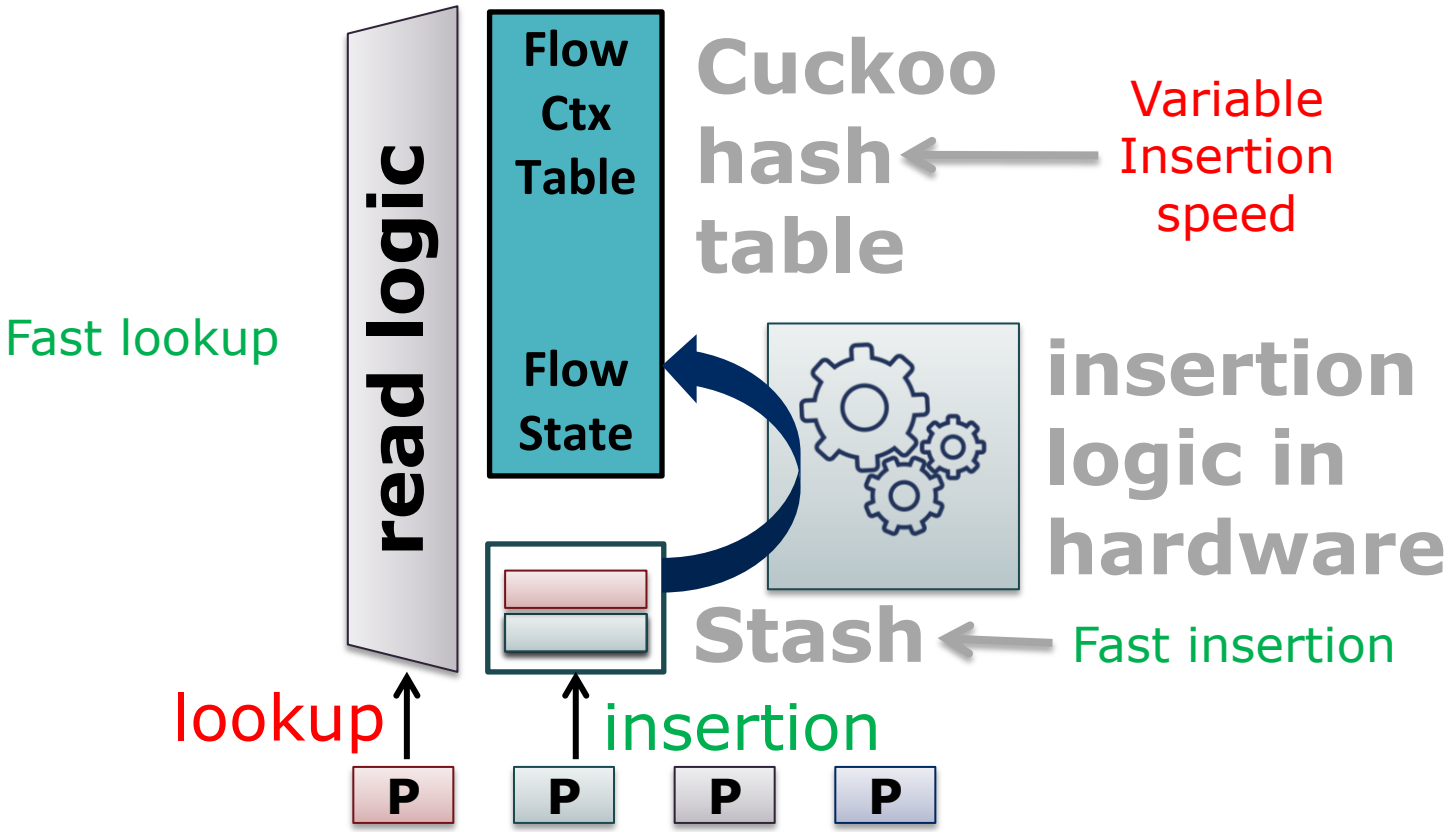
Throughput reduction

Latency increase

Flow context insertion handling

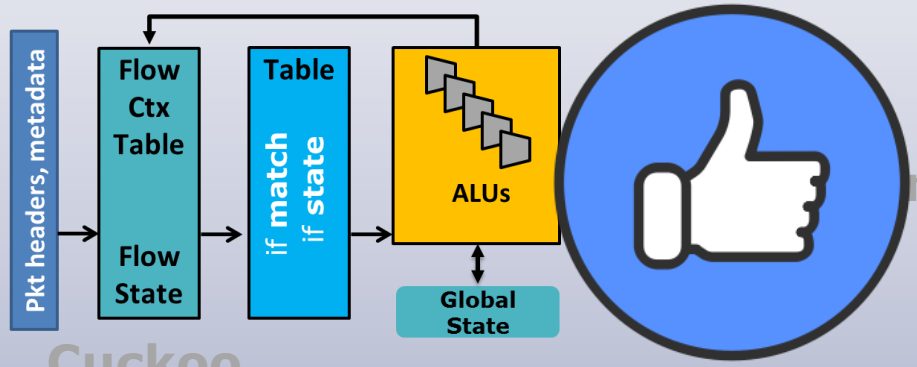
lookup time scales with pkt arrival rate

insertion time scales with flow arrival rate



Implementation issues

Insertion in the flow context table



rtion = new flow

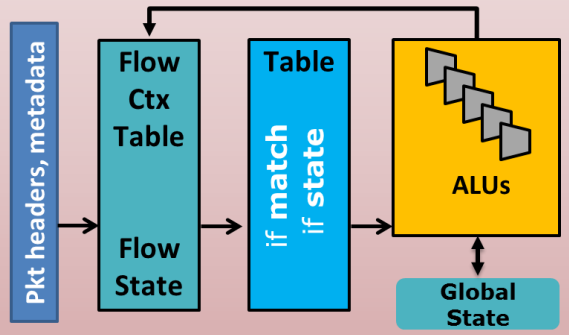
Cuckoo hash table

Variable insertion time

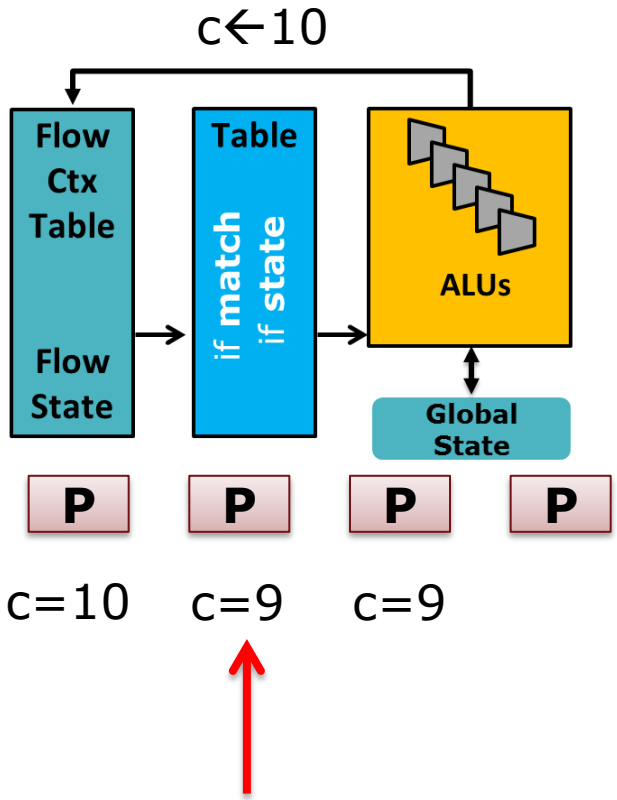
State update latency

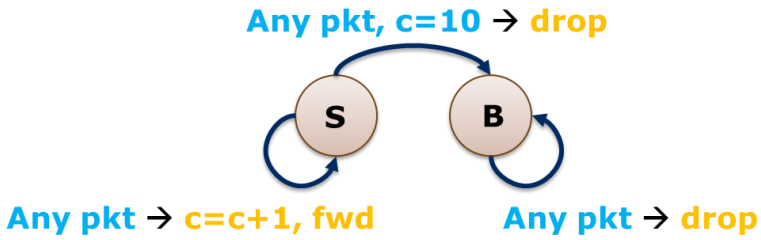
Feedback loop

Race conditions

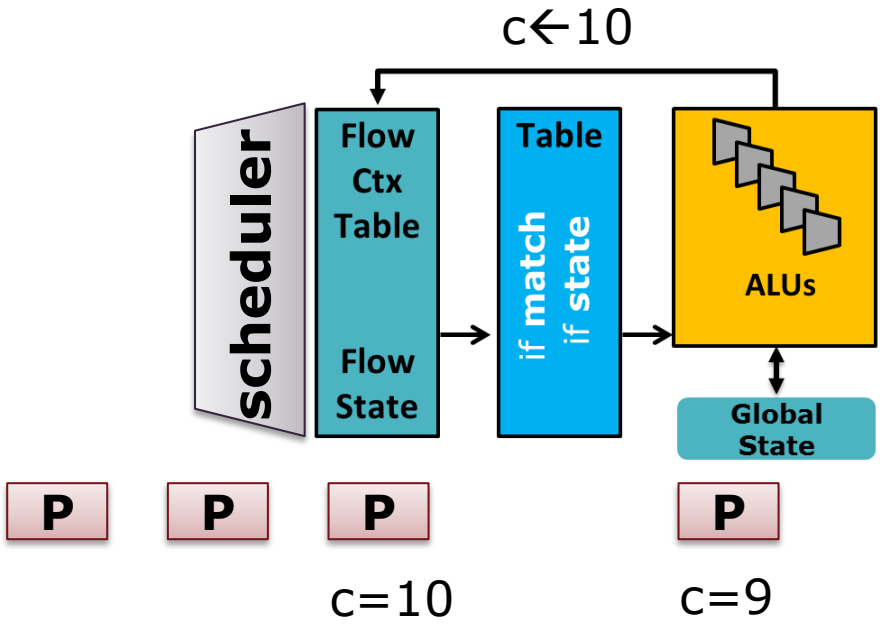


Avoiding race conditions



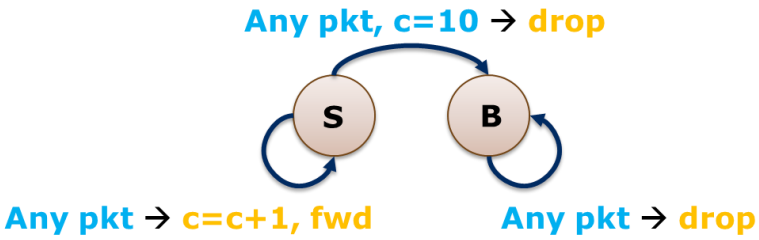


Avoiding race conditions



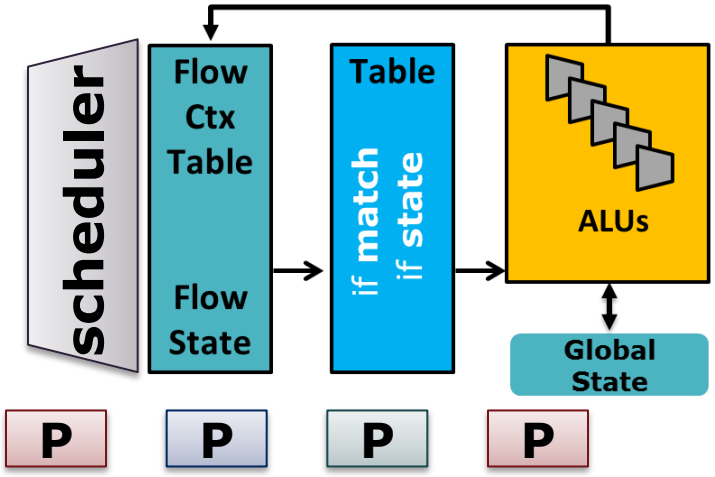
Throughput reduction

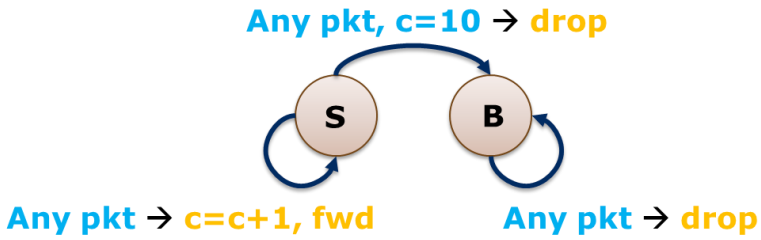
Latency increase



Avoiding race conditions

Lock pipeline for packets from the same flow

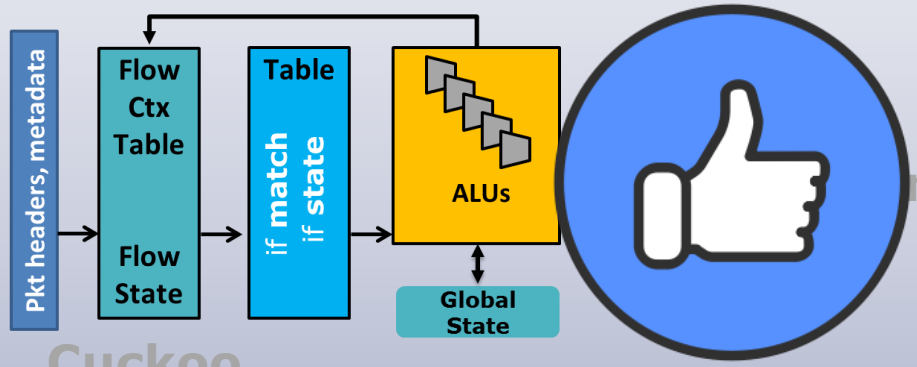




Performance degradation only in unlikely cases

Implementation issues

Insertion in the flow context table



rtion = new flow

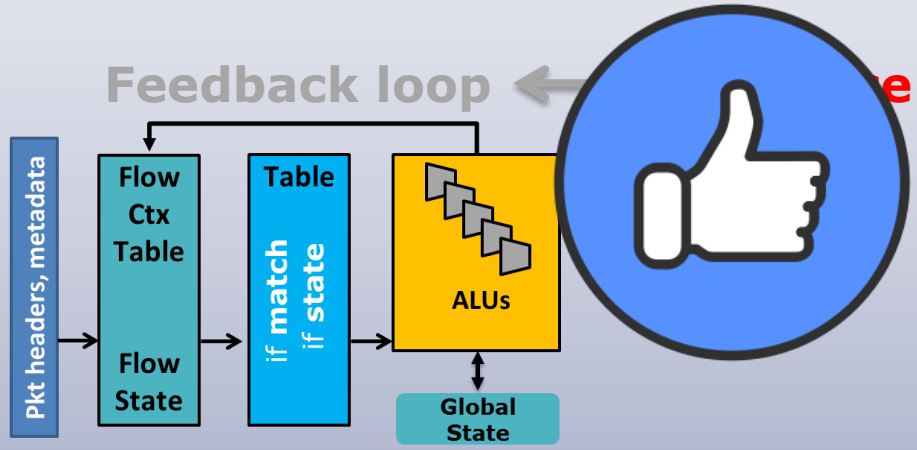
Cuckoo hash table

Variable insertion time

State update latency

Feedback loop

e conditions



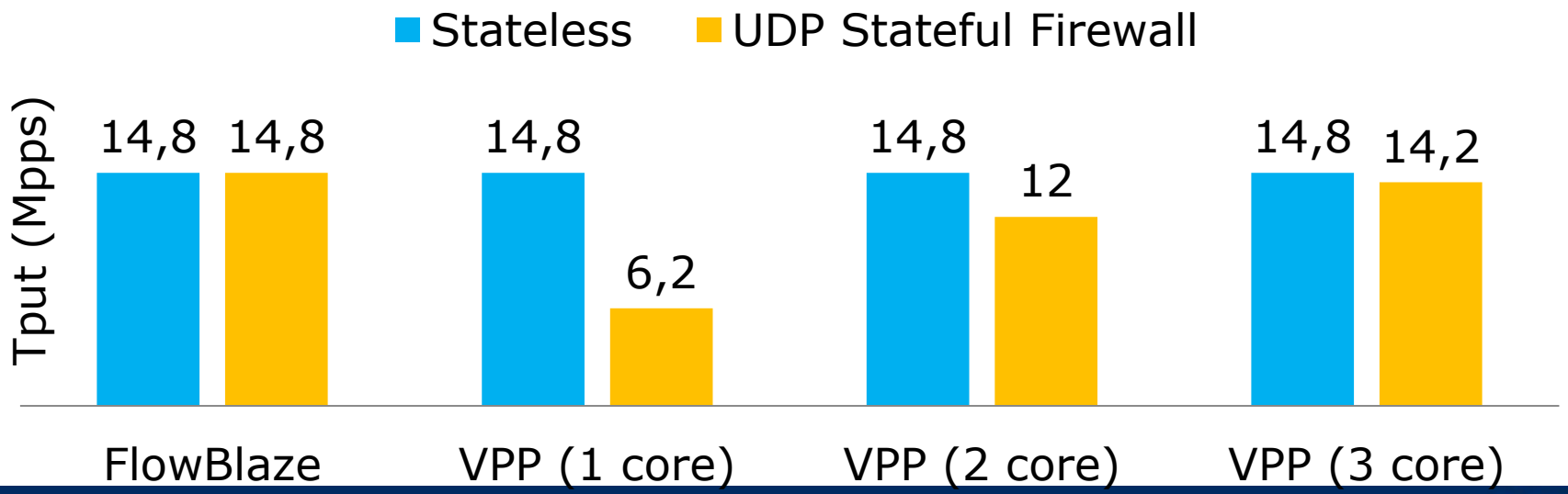
Does it work?

Use case
Server Load Balancer
UDP Stateful Firewall
Port Knocking Firewall
Flowlet load balancer
Traffic Policer
Big Flow Detector
SYN flood Detection and Mitigation
TCP optimistic ACK detection
TCP super spreader detection
Dynamic NAT
vEPC subscriber's quota verification
Switch Paxos Coordinator
Switch Paxos Acceptor
In-network KVS cache

FlowBlaze provides the same performance for all use cases

Test:
10Gb/s@64B
flow definition: 5-tuple

FlowBlaze:
NetFPGA@156.25MHz
Compared to:
DPDK-VPP on Xeon X3470@2.93GHz, Intel 82599 10GbE NIC

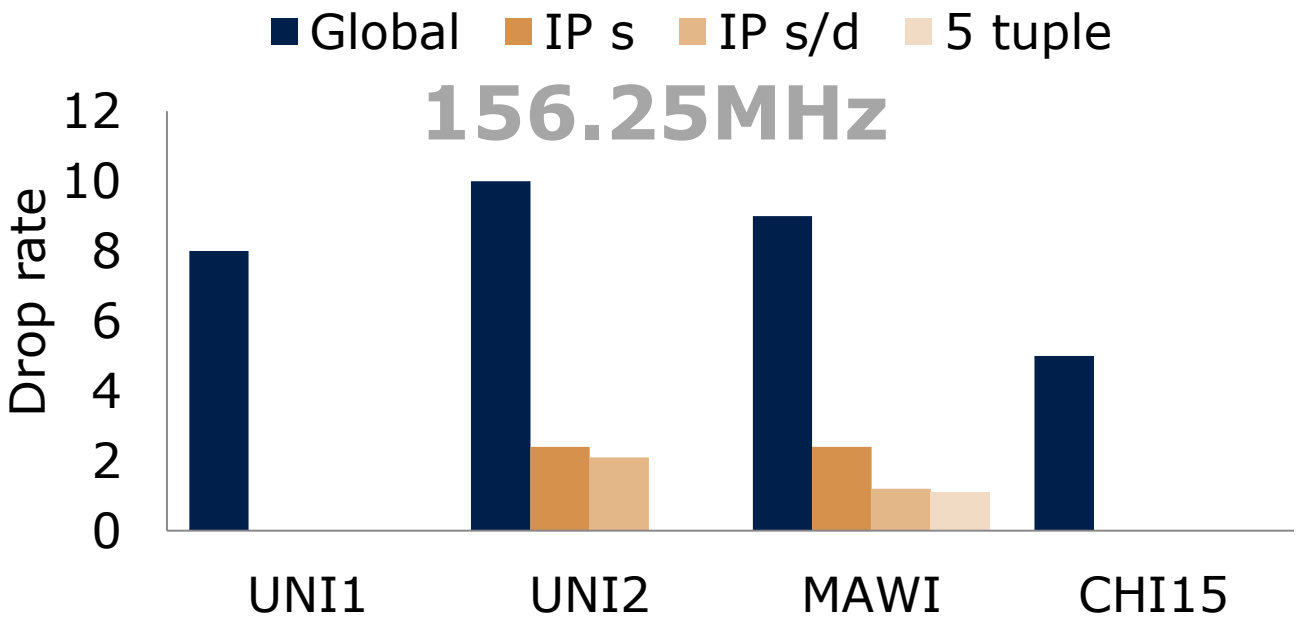


Stress test

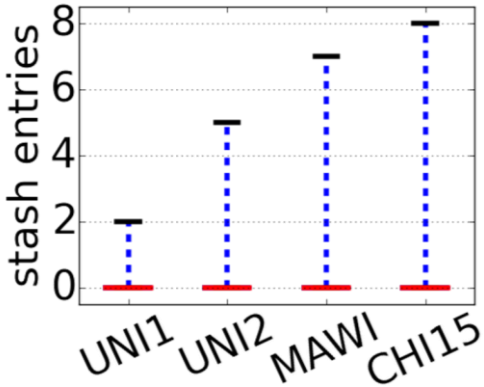
Test:
40Gb/s@64B
(NetFPGA line rate)

	Max # active flows			Max # new flows/ms		
Trace	IP s	IP s,d	5 tpl	IP s	IP s,d	5 tpl
UNI1	575	997	4k	13	19	39
UNI2	948	3k	7k	20	42	42
MW15	12k	130k	152k	38	112	114
CHI15	92k	147k	178k	135	144	144

Flow distributions



Stash



Conclusion

FlowBlaze

- FSM Abstraction for packet processing
- Efficient FPGA implementation

Benefits

- Can keep state for 100Ks flows in flow tables
- Save several CPU cores for stateful NFs
- Power efficient (check the paper!)
- Low latency (check the paper!)

Check the paper, there's a lot more!

FlowBlaze is open

Both software and hardware implementations maintained by

Axbryd

https://github.com/axbryd/FlowBlaze

Thank you!
visit us and check our demo at the poster session


Orchestrating a brighter world
NEC