POLITECNICO
MILANO 1863

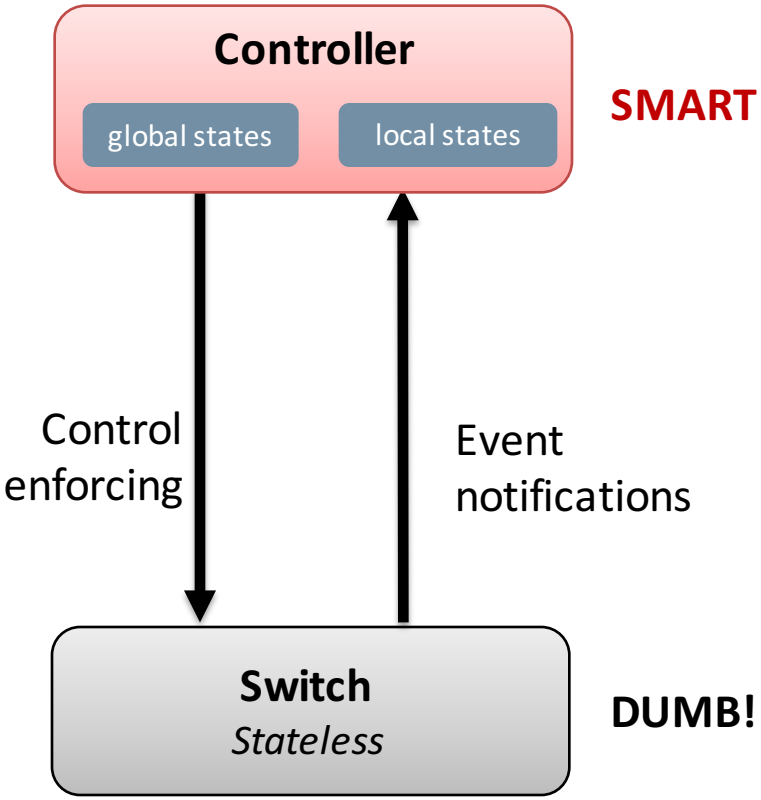
openstate.p4
Supporting Stateful Forwarding in P4

Antonio Capone, Carmelo Cascone

2nd P4 Workshop, Stanford, November 18, 2015

Stateless dataplane

Stateless model (e.g. OpenFlow)



SMART!

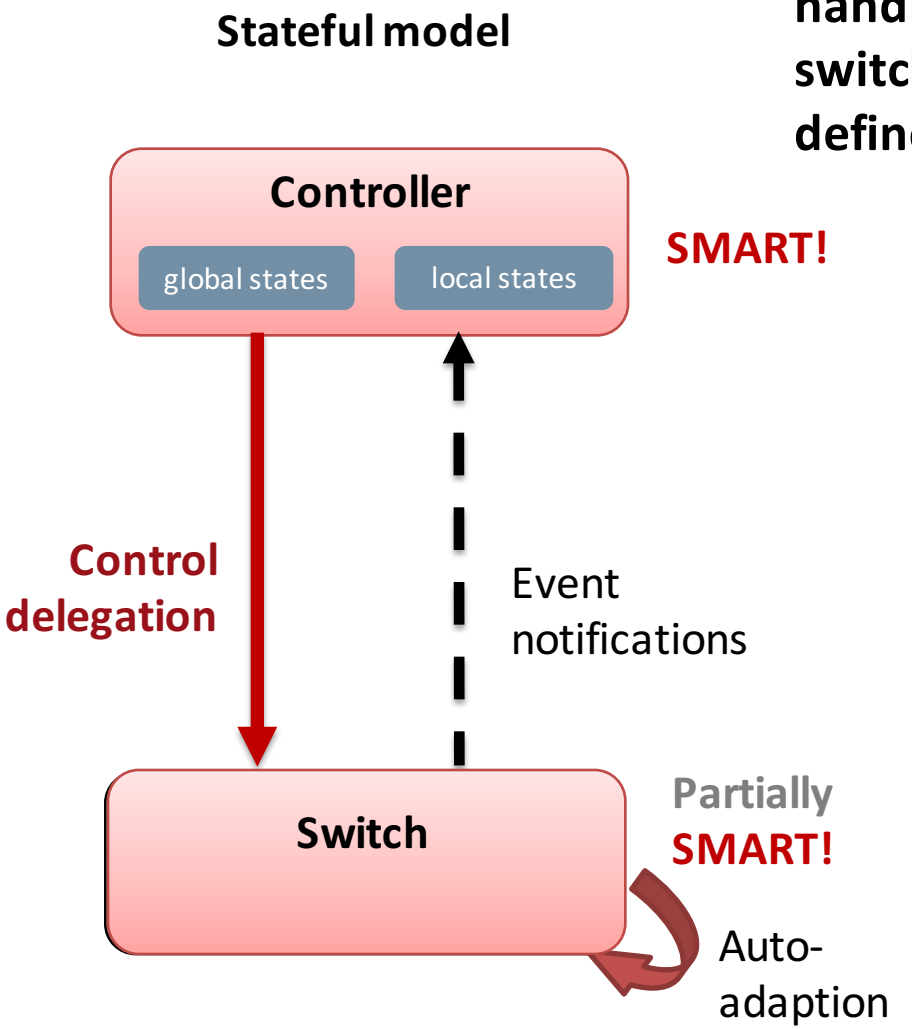
DUMB!

SDN applications dynamically adapt forwarding rules based on network states, either global and local

Event notifications:
- Packet arrivals
- Topology changes
- Traffic statistics

Control:
- Add rules
- Modify rules
- Delete rules
- Query statistics
- Send packets

Stateful dataplane



The idea of a stateful dataplane is to handle local states directly in the switches based on different sets of rules defined by the controller

Control delegation:

- Definition local states
- Definition of a "behavioral forwarding" (set of rules) per state
- Definition of "event" for state transitions

(state machine)

Event notifications:

- Notifications of event relevant for global states
- Local states synchronization (if needed)

Advantages and limitations

- In some network scenarios the control path to the controller is too slow for ensuring quick reaction to events
  - 1s delay in link failure reaction = 10M packets lost @ 100 Gbps
- And the signaling overhead generated quite large

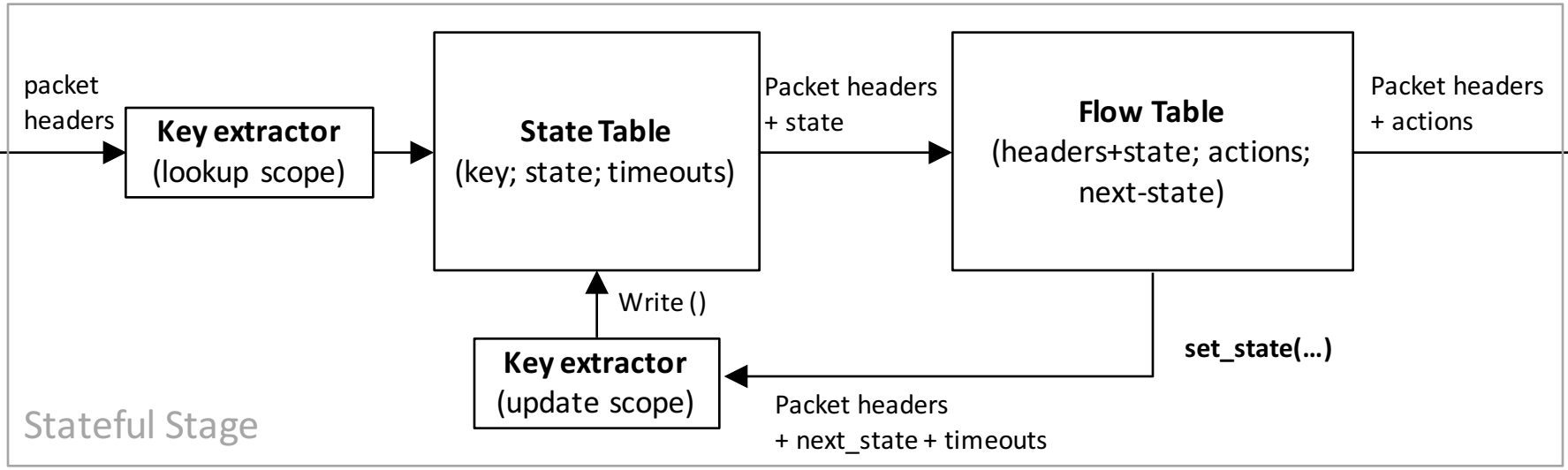
Quick local reaction

Low signaling overhead

- The set of supported local events must be compatible with the standard capabilities of a switch (header matches, timers, meters, …) – extensions for new gen hw
- State transitions must be implementable in a efficient way on hardware platforms (and high performance soft switches)

Not all state machines can be supported

How: OpenState architecture



Ingredients:

- new **set_state action**
- **lookup and update scope** as keys extracted from header (they may be different (cross flow state transitions)
- **state table** used for per-flow state information retrieval (easily implementable as a hash table) – [entries = # flows]
- **state machine** execution in a flow table – [entries = # states]

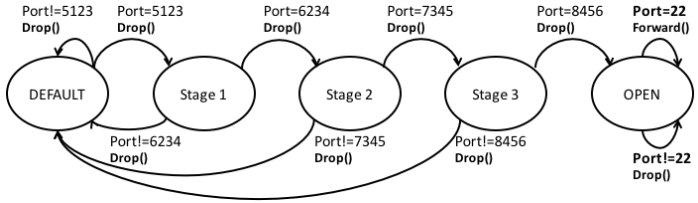
Abstraction:

- Mealy state machine $(\mathrm{T}: I \times S \rightarrow O \times S)$

[CCR14] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, *"OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch"* ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 44–51, 2014.

Toy example: Port knocking firewall

Drop all packets from an IP source until a sequence of packets with a given ("secret") sequence of port numbers is received; then open port 22.



1) State lookup

IPsrc=1.2.3.4	Port=8456

2) State transition

STAGE-3	IPsrc=1.2.3.4	Port=8456

State Table

Flow key	state
IPsrc= … …	… … …
Ipsrc= … …	… … …
IPsrc=1.2.3.4	Write: OPEN
IPsrc=5.6.7.8	OPEN
IPsrc= … …	… … …
IPsrc= no match	DEFAULT

Flow Table

Match fields		Actions	
state	headers	action	Next-state
DEFAULT	Port=5123	drop	STAGE-1
STAGE-1	Port=6234	drop	STAGE-2
STAGE-2	Port=7345	drop	STAGE-3
STAGE-3	Port=8456	drop	OPEN
OPEN	Port=22	forward	OPEN
OPEN	Port=*	drop	OPEN
*	Port=*	drop	DEFAULT

write

3) State update

OPEN

IPsrc=1.2.3.4	Port=8456

Work so far

Based on OpenState basic abstraction we have put together a European H2020 research project to prove its feasibility and extend it to more general stateful dataplane abstractions

www.beba-project.eu

Proof-of-concept [OS][HPSR15]:

- SW implementation:
  - based on CPqD softswitch
- HW implementation:
  - based on FPGA

Applications [EWSDN15][DRCN15]:

- MAC learning
- Label/address advertisement learning
- Flow-consistent Load Balancing
- Denial-of-Service mitigation
- **Failure detection & recovery**
- **…**

Come and see our demo!

[OS] http://openstate-sdn.org - public repository with openstate implementation and example applications (on mininet)

[HPSR15] S. Pontarelli, M. Bonola, G. Bianchi, A. Capone, C. Cascone, "Stateful Openflow: Hardware Proof of Concept", IEEE HPSR 2015, Budapest, July 1-4, 2015

[EWSDN15] C. Cascone, L. Pollini, D. Sanvito, A. Capone, "Traffic Management Applications for Stateful SDN Data Plane", EWSDN 2015 (4th European Workshop on Software Defined Networks) , Bilbao, Sept. 30-Oct 2, 2015,

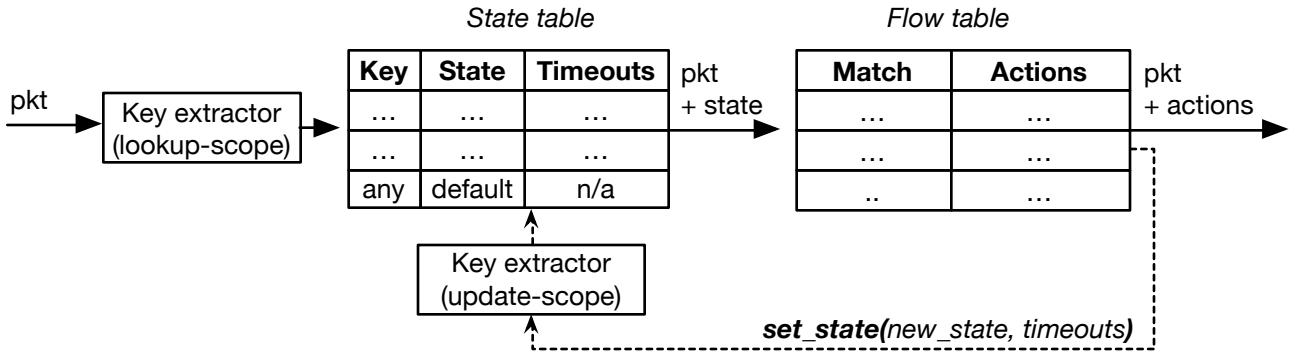
[DRCN15] A. Capone, C. Cascone, A.Q.T. Nguyen, B. Sansò, "Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState", IEEE DRCN 2015, Kansas City, USA, March 24-27, 2015.

OpenState & P4

- With the experience acquired in analyzing opportunities and limitations of stateful dataplanes it was quite natural to attempt the implementation of OpenState in P4
- It turned out that it is actually possible to use P4 to describe an OpenState stateful stage
- However:
  - The workarounds we have used point out possible improvements in P4 for the support of stateful dataplanes
  - Some open questions remain on the level of flexibility that is reasonable assuming for the target

Goal: #include “openstate.p4”

OpenState abstraction



What we need:

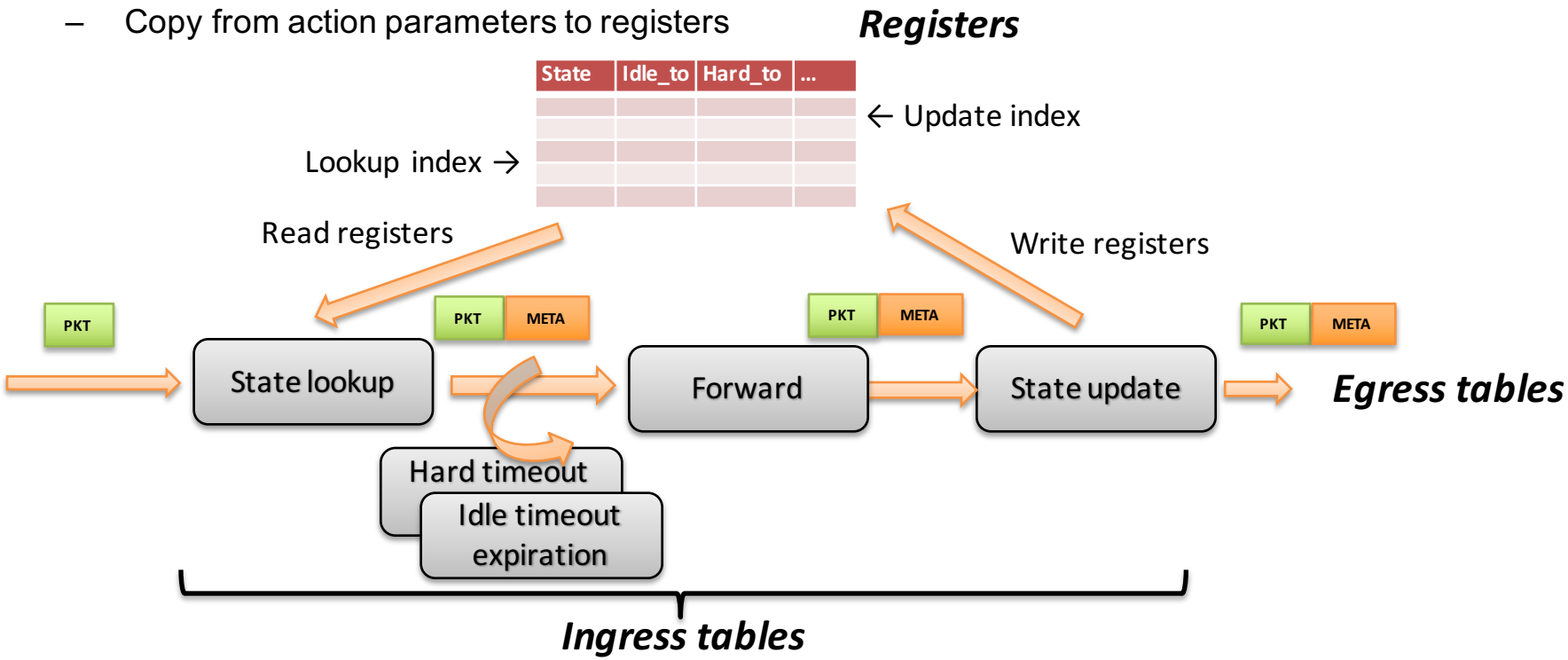
1. State table
2. Key extractors (lookup/update)
3. State idle/hard timeout handling
4. Set-state action

What we have (P4 abstractions):

1. Registers (stateful memories)
2. Hash generators
3. Packet ingress timestamp
4. Primitive actions (read/write registers)

Control flow at a glance

1. **State lookup**
   - hash generator produce a index to access registers
   - copy from registers to packet metadata
2. **Timeouts handling** at control flow
3. **User-defined forwarding logic**
4. **State update** (state transitions defined by user)
   - Update index might be different from lookup
   - Copy from action parameters to registers



openstate.p4

State table

Requirement:

- store flow states, possibly a very large number

Our workaround:

- registers (one for each column of the state table)

Issues:

- addressing constrained by register size

Ideal abstraction:

- state table as a dedicated table type
- exact match on 1 field, a “flow key” of arbitrary length
- only 1 action that writes metadata (state, timeouts, etc.)
- **need data plane driven insert/update**
  (i.e. OVS “learn” action)

openstate.p4

Key extractors (lookup / update)

Requirement:

- uniquely access state entries, optionally different in lookup/update operations
  Cross-flow state handling (e.g. MAC learning)

Our workaround:

- hash generators on different field lists

Issues:

- collisions → state inconsistency

Ideal abstraction:

1. programmable hash generators
2. simple fields concatenation

openstate.p4

```
field_list_calculation lookup_hash {
    input {
        lookup_scope;
    }
    algorithm : crc32;
    output_width : 32;
}

field_list_calculation update_hash {
    input {
        update_scope;
    }
    algorithm : crc32;
    output_width : 32;
}
```

maclearning.p4

```
field_list lookup_scope {
    ethernet.dstAddr;
}

field_list update_scope {
    ethernet.srcAddr;
}
```

openstate.p4

State timeouts

Requirement:

- enable time-based state transitions
  i.e. OpenFlow-like idle/hard state timeouts

Our workaround:

- timestamp comparison at control flow
  E.g.
```
if (ingress_timestamp > idle_exp_timestamp)
        apply(idle_to_expiration) …
```

Issues:

- expired registers not flushed

Ideal abstraction:

- support for transparent timeout handling in the state table
- expose target timestamp resolution (ms, µs, ns, etc.)
  Short timeouts critical for applications like failure detection, Denial-of-Service mitigation, etc.

Conclusions

- Shown feasibility in P4 of a stateful data plane abstraction like OpenState
- openstate.p4 is based on a number of workarounds
- Room for improvements in P4 specification
- E.g. we need support for a proper state table !

Download & try:

http://github.com/OpenState-SDN/openstate.p4

http://www.openstate-sdn.org