SPIDER: Fault Resilient SDN Pipeline with Recovery Delay Guarantees

Carmelo Cascone* †, Luca Pollini^, Davide Sanvito^, Antonio Capone*, Brunilde Sansò†

* Politecnico di Milano, Italy
^CNIT – Consorzio Nazionale Universitario per le Telecomunicazioni, Italy
† Ecole Polytechnique de Montreal, Canada

Supported by EU Project

Beba
BEhavioural BAsed forwarding

Outline

- Motivations
- Goal
- SPIDER
- Numerical results

Horizon 2020
European Union Funding
for Research & Innovation

- European H2020 research project
- Started January 2015
- Goal: programmable stateful packet processing in the fast-path
  - Allow pre-configuration of different sets of forwarding rules to be applied according to the observed network state
  - in-switch fast state evolution according to packet-level events/time-based events/flow level measurements

cnit

Univesity of Rome Tor Vergata
Politecnico di Milano
University of Pisa

NEC

THALES

WIND

CESNET

KTH VETENSKAP OCH KONST

www.beba-project.eu

Fault Resiliency in SDN

- Weak support by current data plane abstractions
- OpenFlow
  - Stateless match+action
  - requires remote controller to reconfigure the forwarding
    - Additional overhead and latency → hard to obtain carrier-grade recovery times **(<50 ms)**
  - Fast-Failover group type:
    - limited to local failures protection
    - Liveness checking out of spec. → No guarantees on detection delays **(1ms – 500ms)**

Related Works

- Integration of a BFD deamon with OF Fast-Failover
- fine-tuning of parameters in Open vSwitch's BFD process
- OF extension to implement in-switch link monitoring functions
- OF extension with a flow entry auto-rejecting mechanism based on port status

Mainly based on patching OF Fast-Failover and BFD → slow-path

SPIDER Goal

- Provide a forwarding pipeline design to allow:
  - End-to-end proactive protection independent from controller reachability
  - Programmable sub-milliseconds detection delay
- Inspired by legacy technologies
  - BFD
  - MPLS Fast Reroute

Stateful Dataplane

- Switch mantains flow memory across different packets
- Forwarding is based on packet fields and current flow state
- The controller can delegate to switches local changes in the forwarding

OpenState

- Stateful dataplane
  - Statefulness in the fast-path → state updates at wirespeed!
- Stateful OpenFlow extension
- Pipeline of stateless/stateful stages
- Forwarding behaviour modeled as Finite State Machine (FSM)

[CCR '14] 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
[HPSR '15] S. Pontarelli, M. Bonola, G. Bianchi, A. Capone, C. Cascone, "Stateful Openflow: Hardware Proof of Concept", IEEE HPSR 2015, Budapest, July 1-4, 2015

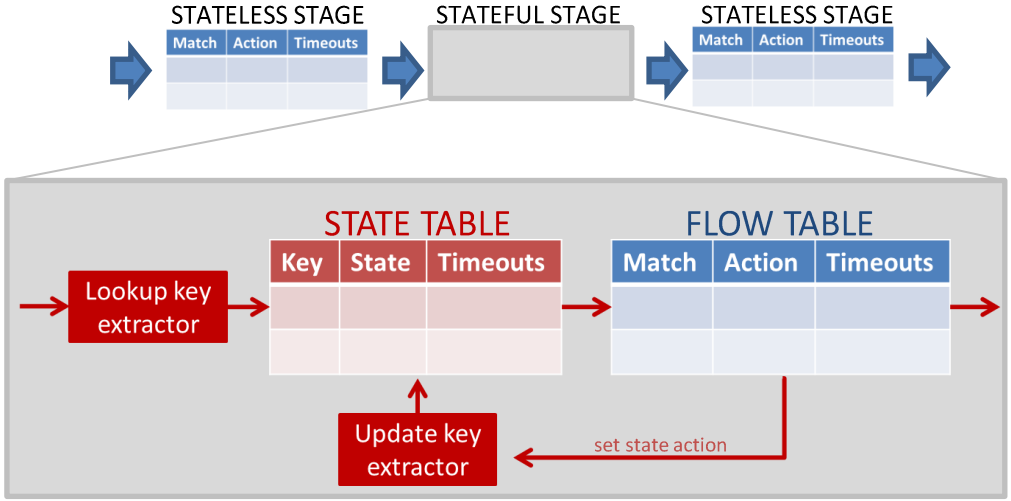
Other examples of stateful dataplane

- OVS: learn() action
  - OF extension → slow-path only
- P4: stateful memories
  - We can describe OpenState

OpenState Stateful Stage

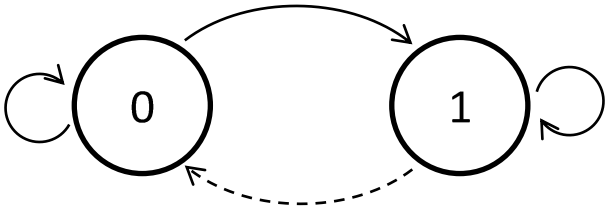
- State Table
  - Associates a flow key (exact match) with a state
  - Flow key extractor (lookup-scope and update-scope)

- Flow Table
  - Classic OF match+action table
  - New *state* match field
  - New *set state* action



FSM

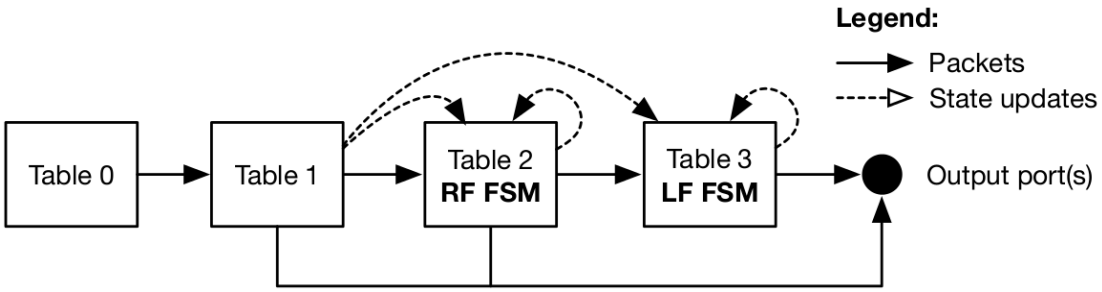
- OpenState stateful forwarding abstraction is based on FSM
- Forwarding depends on current state + packet header fields
- State transitions
  - Packet-driven
  - Time-based
- FSM structure is defined by the controller at boot-time
  - by inserting fow table entries
  - by configuring lookup-scope and update-scope
- The switch executes the FSM at run-time
  - by storing flow states in the state table
  - by updating the states



SPIDER

Stateful **P**rogrammable fa**I**lure **DE**tection and **R**ecovery

- Fault resilient SDN pipeline design
- Fully programmable failure detection and recovery in the fast-path
  - Sub-milliseconds detection&reroute (device timeout granularity)
- Based on stateful dataplane abstraction
  - Implementation in OpenState
- Instantaneous in-switch recovery from any pre-planned failure scenario
  - Controller intervention needed only in case of un-planned failures
- Programmable failure detection
  - BFD-like
- Fast reroute
  - Inspired by MPLS
  - For both local and non-local failures
  - Path probing
  - Flowlet-aware rerouting



Preplanning of Primary/Backup Paths

- Given:
  - network topology
  - set of demands

  We need to provide the controller with a set of primary path (PP) and backup paths (BP) for each possible failure affecting the PP of a given demand.

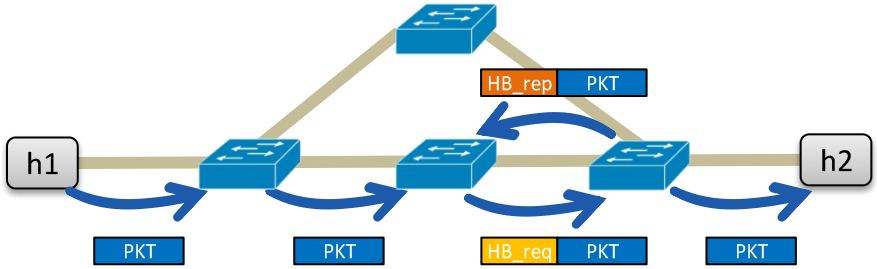
- The controller then creates the switch pipeline configuration
  - FSM instantiation
  - Flow table entries
  - Forwarding based on L2 src-dst addresses and MPLS label

[DRCN 2015] A. Capone, C. Cascone, A. Q.T. Nguyen, and B. Sansò, "Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState"

Failure Detection

Assumption:

As long as packets are received from a given port, that port can be also used to transmit packets

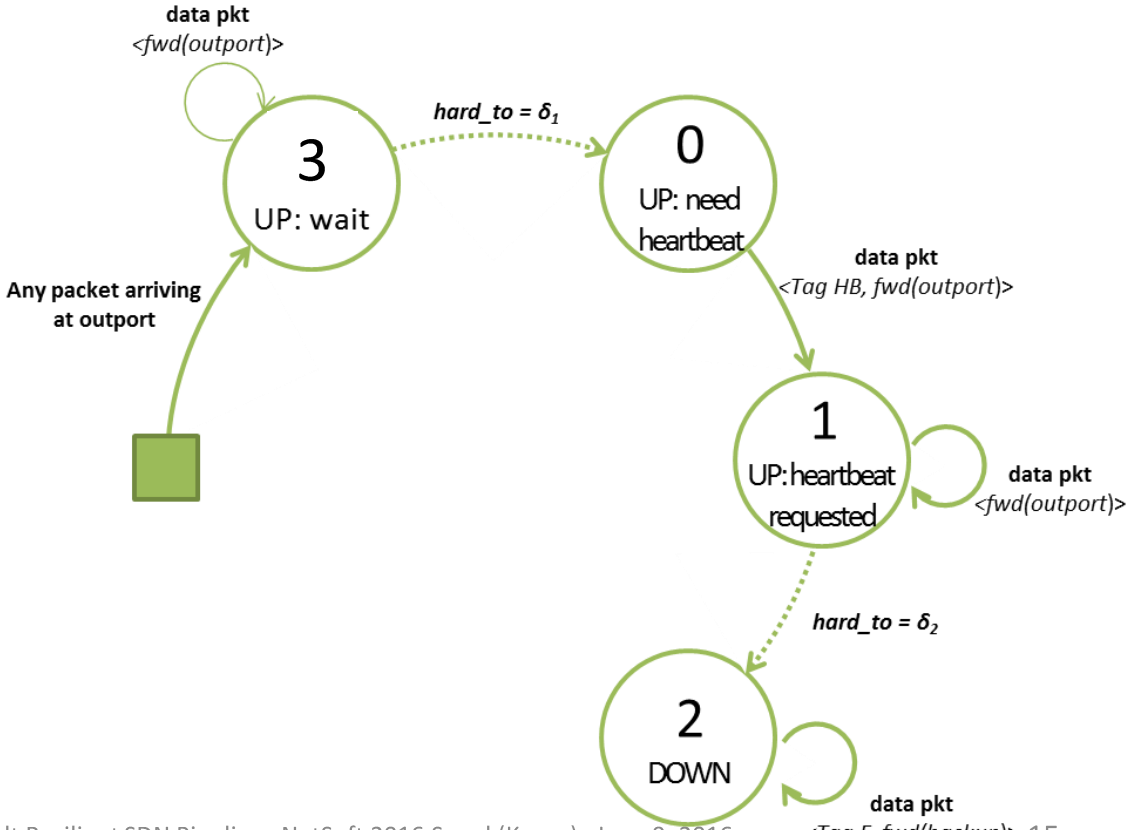


- If no packet is received from port x within a δ_1 interval:
  - Next data packet towards port x is tagged with a special value (Heartbeat request)
  - Port x is declared down if adjacent node does not send back a copy (Heartbeat reply) within a δ2 interval
- Configurable trade off: overhead vs. failover responsiveness
  - Heartbeat requests generation timeout (δ1)
  - Heartbeat reply timeout (δ2) before the port is declared down
- Guaranteed max detection delay: δ1 + δ2

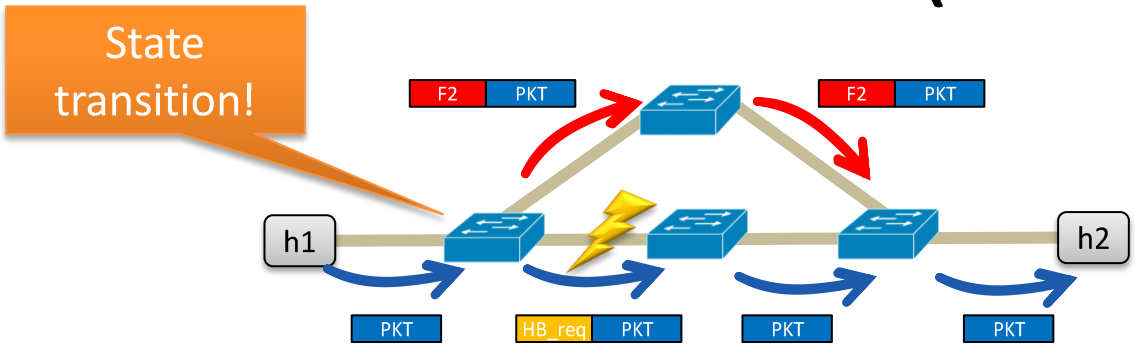
Failure Detection FSM

δ_1 = HB requests generation timeout
δ_2 = HB reply timeout

Lookup-scope = [metadata]
Update-scope = [metadata]



Fast Reroute (local)

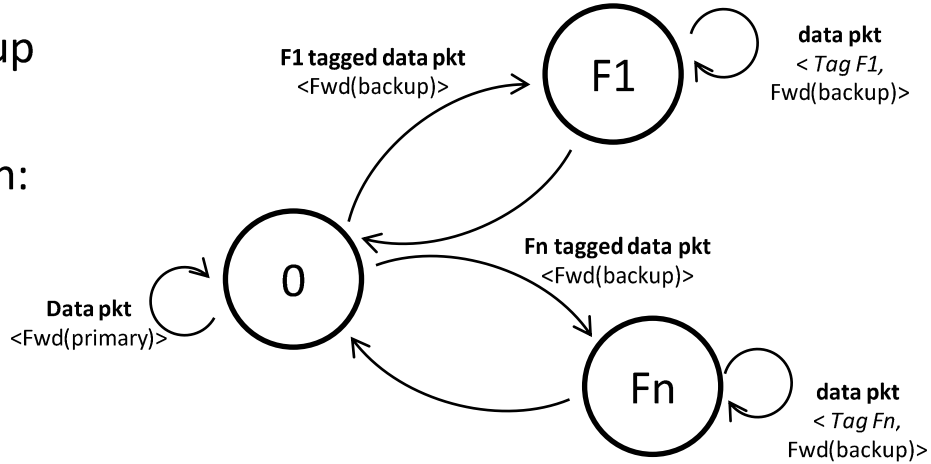


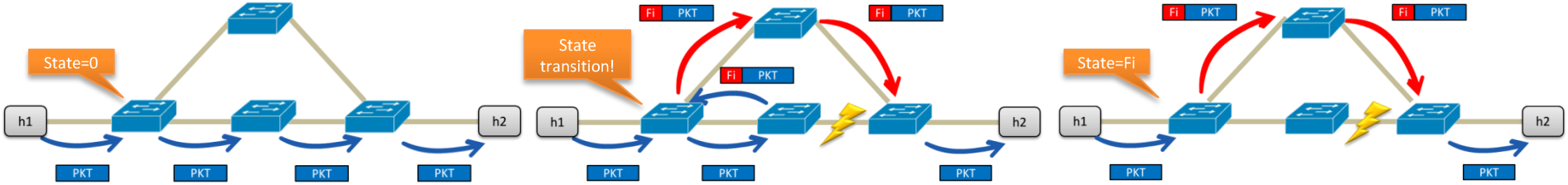
- MPLS label used to distinguish between different forwarding :
  - No tag → forward packet on the primary path
  - tag=Fi → forward packet on the detour for the i-th failure
- Zero losses after failure detection
- No controller intervention
- What if no local alternative path is available?

Fast Reroute (remote)

Lookup-scope = [ETH_SRC,ETH_DST]
Update-scope = [ETH_SRC,ETH_DST]

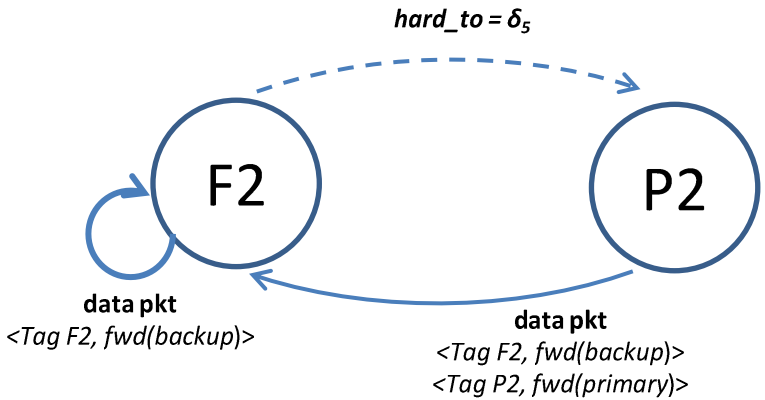
- Packets are tagged and bounced back up to a proper redirect point
- Tagged packets trigger a state transition:
  - updating the routing of the involved connections
- Still zero losses after failure detection!
- Tagged data packets as signalling
- No controller intervention!

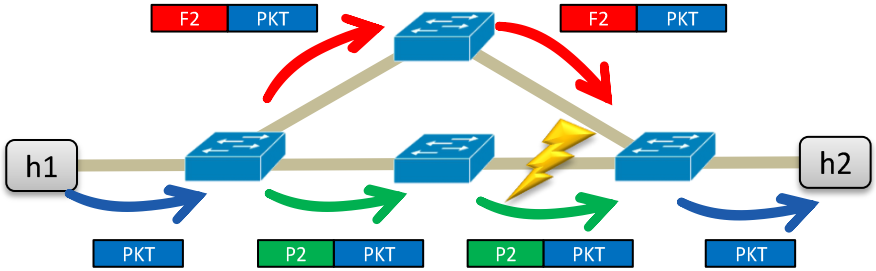




Path Probing

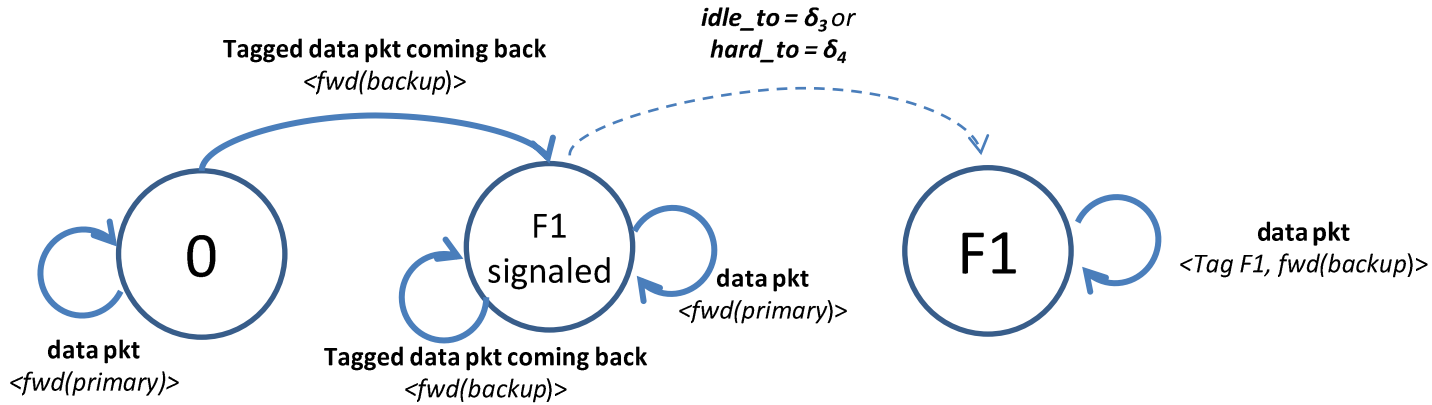
- How to restore the forwarding on the primary path?
- Programmable periodic probing for primary path availability



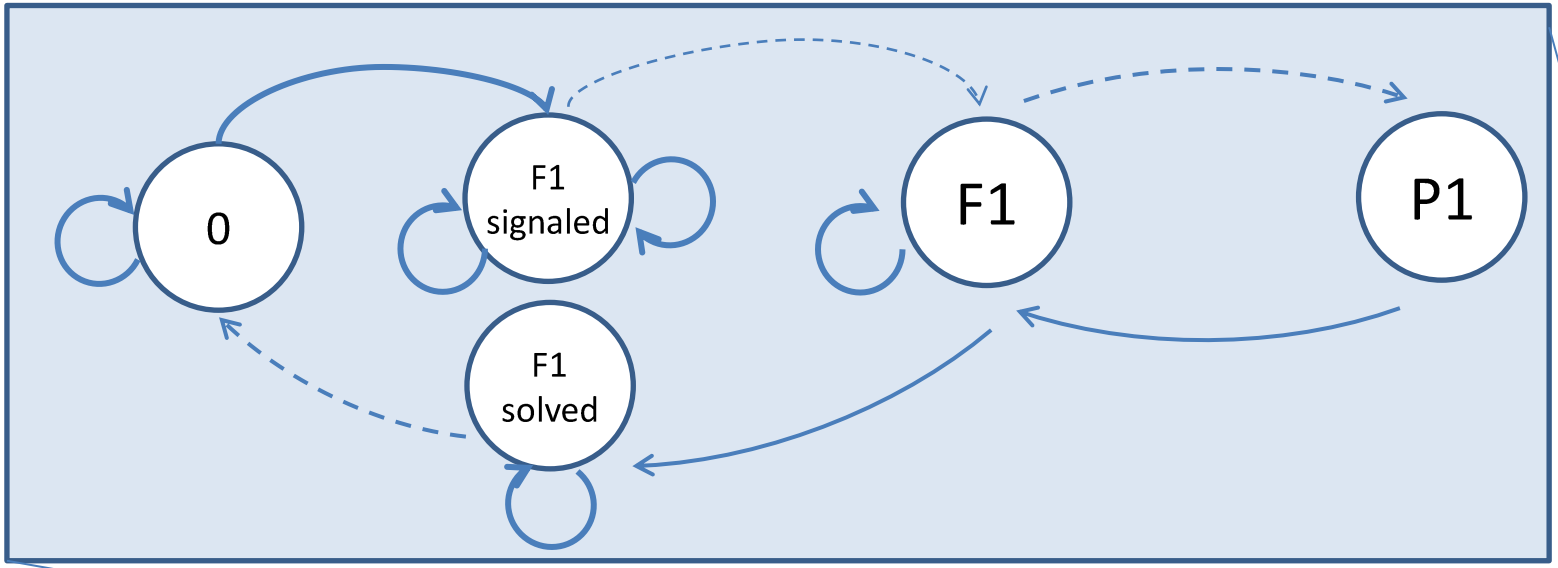


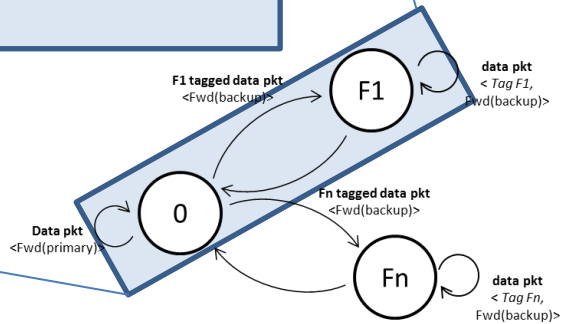
Flowlet-aware Rerouting

- Failover activation/deactivation can be post-poned
  - In order to minimize out-of-sequence, packets are kept on the primary path up to expiration of a burst of packets
  - Programmable idle timeout/hard timeout

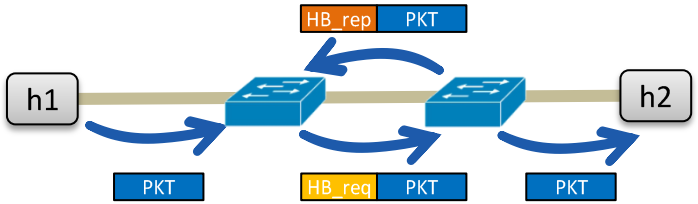


Putting all together: Fast reroute FSM





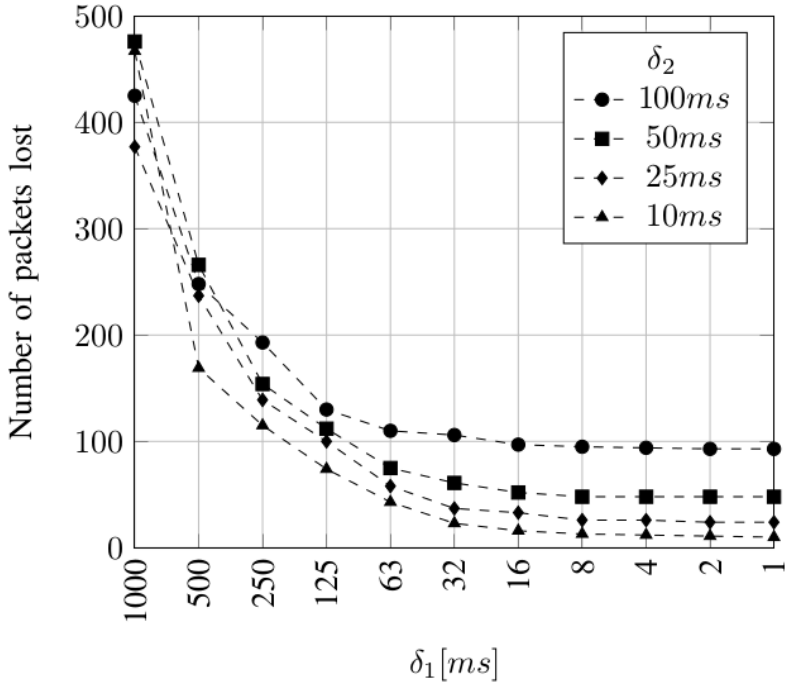
Results: Detection Mechanism



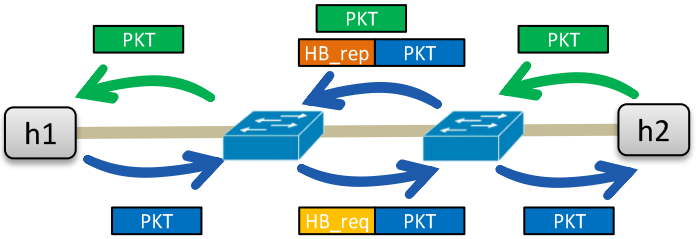
Unidirectional demand h1->h2 @1000 pkt/s

The plot shows the number of packets lost by tuning:

- Heartbeat requests generation timeout (δ_1)
- Heartbeat reply timeout (δ_2)

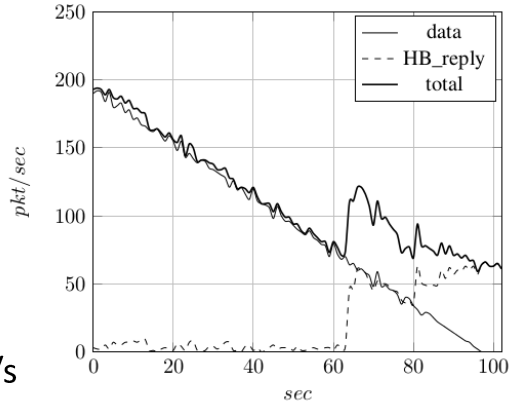


Results: Heartbeat Overhead

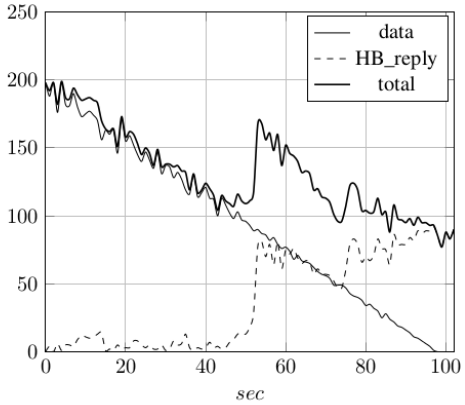


Unidirectional demand h1->h2 @100 pkt/s
Unidirectional demand h2->h1 from 200 to 0 pkt/s

HB_req_rate = $1/\delta_1$



(a) HB_req rate = 70 pkt/sec

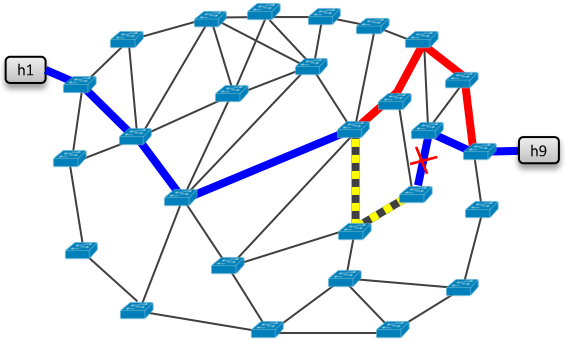


(b) HB_req rate = 100 pkt/sec

Hearbeat packets are requested **only if** incoming traffic rate is lower than $1/\delta_1$

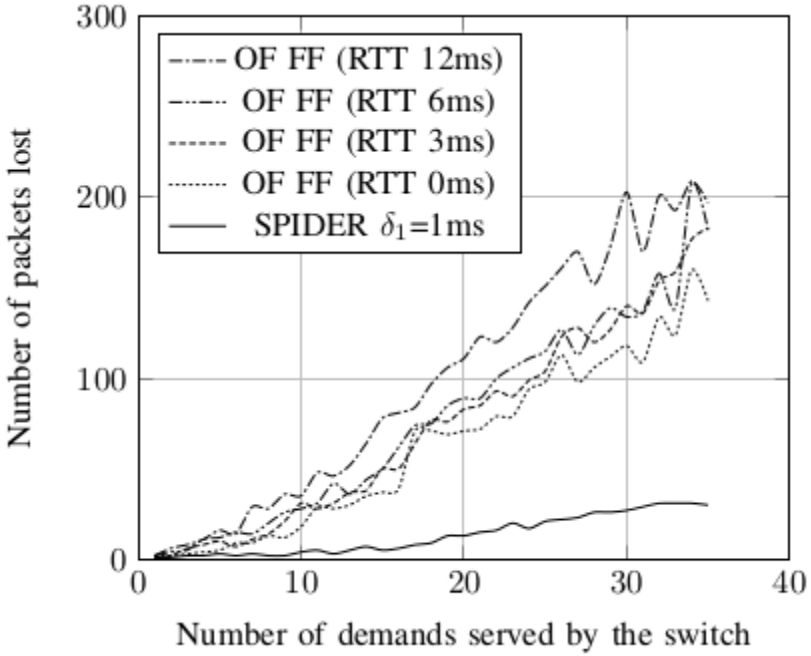
Overhead does not affect link available capacity!

Results: comparison with OpenFlow



We compared SPIDER to a reactive OF application:

- failure detection with Fast-Failover Group Table
- controller installs new forwarding rules



Losses in SPIDER: detection phase only

Losses in OF FF: detection phase + failover phase

Results: Complexity Analysis

n x *n* grid networks with a traffic demand for each pair of outer nodes of the grid

Number of flow entries per node is $O(E^2xN)$

Worst case scenario: E2E path protection

With a more efficient protection scheme (segment) we can even obtain a lower number of rules per node

NUMBER OF FLOW ENTRIES PER NODE.

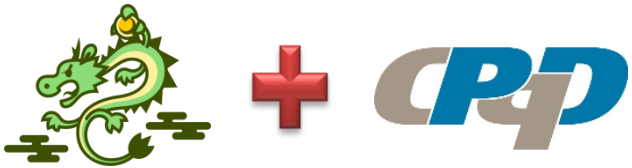
Net	D	E	C	min	avg	max	$E^2 \times N$
5x5	240	16	9	443	775	968	6400
6x6	380	20	16	532	1115	1603	14400
7x7	552	24	25	795	1670	2404	28224
8x8	756	28	36	1069	2232	3726	50176
9x9	992	32	49	1368	2884	4509	82944
10x10	1260	36	64	1188	3584	6153	129600
11x11	1560	40	81	1409	4249	7558	193600
12x12	1892	44	100	1185	5124	9697	278784
13x13	2256	48	121	2062	6218	11025	389376
14x14	2652	52	144	1467	7151	15436	529984
15x15	3080	56	169	3715	8461	16347	705600

SPIDER worst case scenario

Big-O analysis

Software Implementation

- SW implementation based on OpenState
  - Ryu* controller
  - CPqD OpenFlow 1.3 softswitch*
  - https://github.com/OpenState-SDN/spider



- SW implementation in P4 based on openstate.p4 library
  - https://github.com/OpenState-SDN/openstate.p4

*modified with OpenState support http://openstate-sdn.org

Conclusions

- Failure detection and recovery in SDN (OpenFlow) is a major problem
- Statefulness in the data plane allows to implement fast detection and rerouting (<1ms)
  - Indipendent on controller reachability
  - With guaranteed detection delays
- SPIDER is an example of a pipeline design providing such features

Thank you!

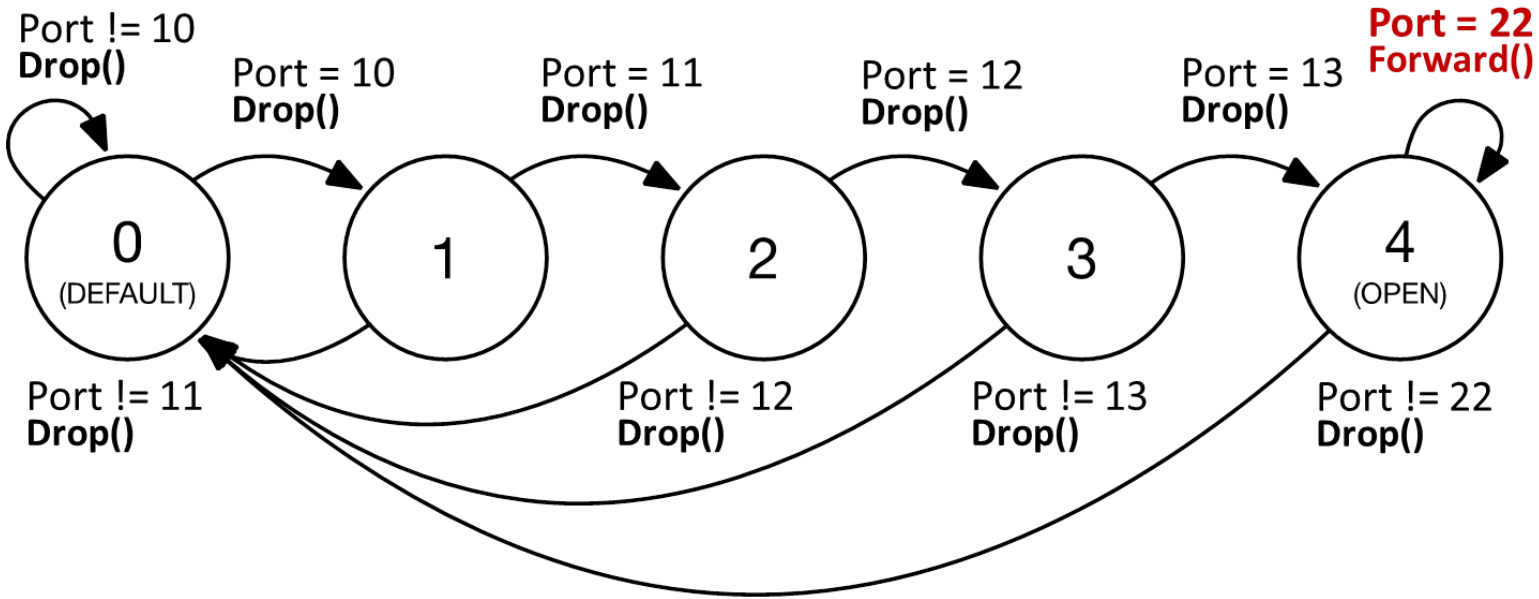
davide.sanvito@polimi.it

BACKUP SLIDE

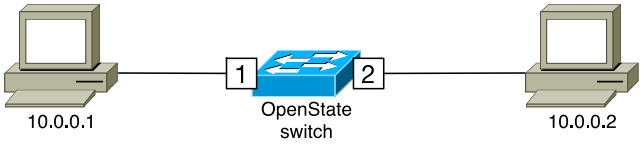
Example: port knocking

An IP address is allowed to access an UDP server after a secret knock sequence is received.

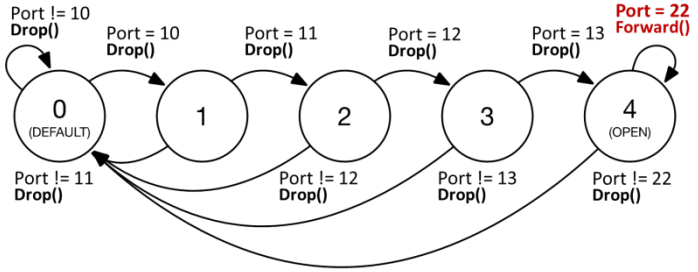
UDP secret sequence: 10,11,12,13
UDP server port: 22



Example: port knocking (2)



UDP secret sequence: 10,11,12,13
UDP server port: 22



Lookup key extractor

Key	State
*	0
10.0.0.1	4

Priority	Match	Actions
100	arp	flood()
10	state=0, ip, udp_dest=10	set_state(1), drop()
10	state=1, ip, udp_dest=11	set_state(2), drop()
10	state=2, ip, udp_dest=12	set_state(3), drop()
10	state=3, ip, udp_dest=13	set_state(4), drop()
10	state=4, ip, udp_dest=22	output(2)
0	ip, udp	set_state(0), drop()

Key extractors:
Lookup-scope = {ip_src}
Update-scope = {ip_src}

Update key extractor

State table memory requirements

- Failure detection state machine

P state entries (where P is the number of ports)

5 possible states

- Failover state machine

D_n state entries (D_n is the number of demands for which node n is a reroute node)

$1 + 4F_n$ possible states (where F_n is the number of remote failures managed by node n)