IEEE CAMAD 2014 – Athens, 1-3 December

From dumb to smarter switches in software defined networks:

an overview of data plane evolution

Giuseppe Bianchi
University of Rome "Tor Vergata"

Antonio Capone
Politecnico di Milano - ANTLab

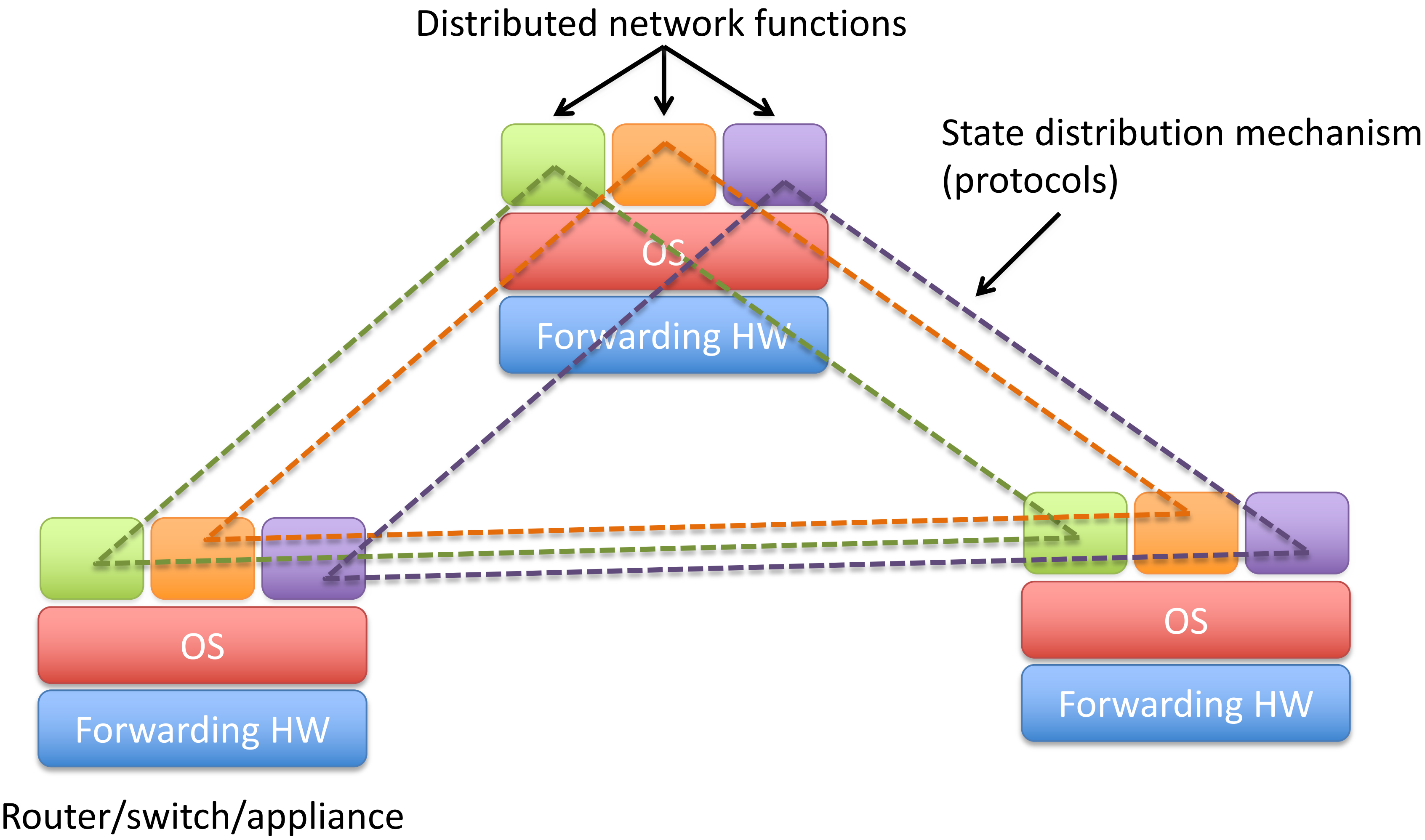
Agenda

1) ***Setting the scene:*** a brief intro to SDN and OpenFlow
2) ***Switches cannot remain dumb:*** Starting the process of data plane evolution
3) ***Not too much not too little:*** OpenState and statefull data planes
4) ***Applied smartness:*** statefull applications

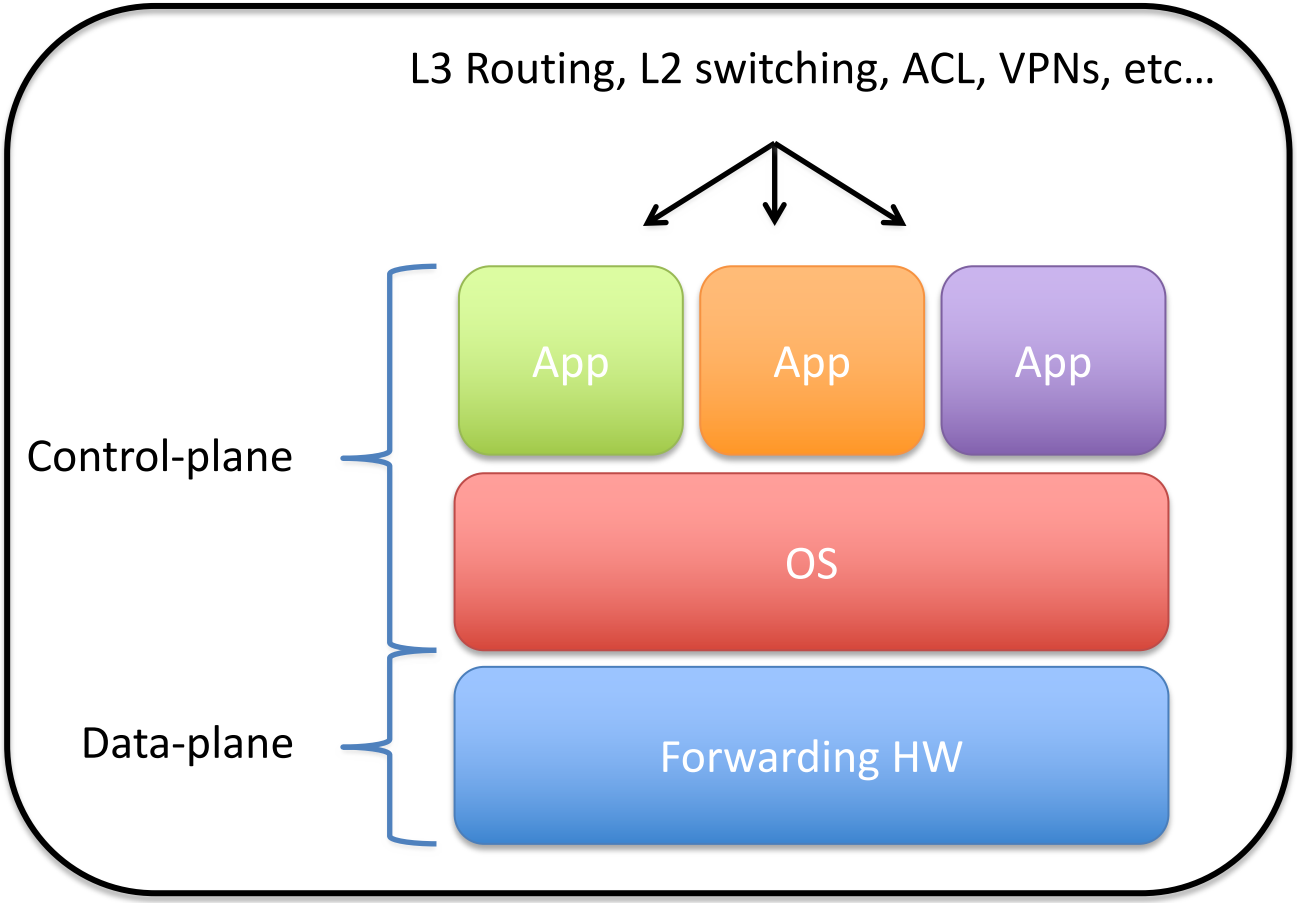
Setting the scene: a brief intro to SDN and OpenFlow

The future has already arrived. It's just not evenly **distributed** yet. [William Gibson]

Classic network paradigm


Distributed network functions
State distribution mechanism (protocols)
OS
Forwarding HW
OS
Forwarding HW
OS
Forwarding HW
Router/switch/appliance

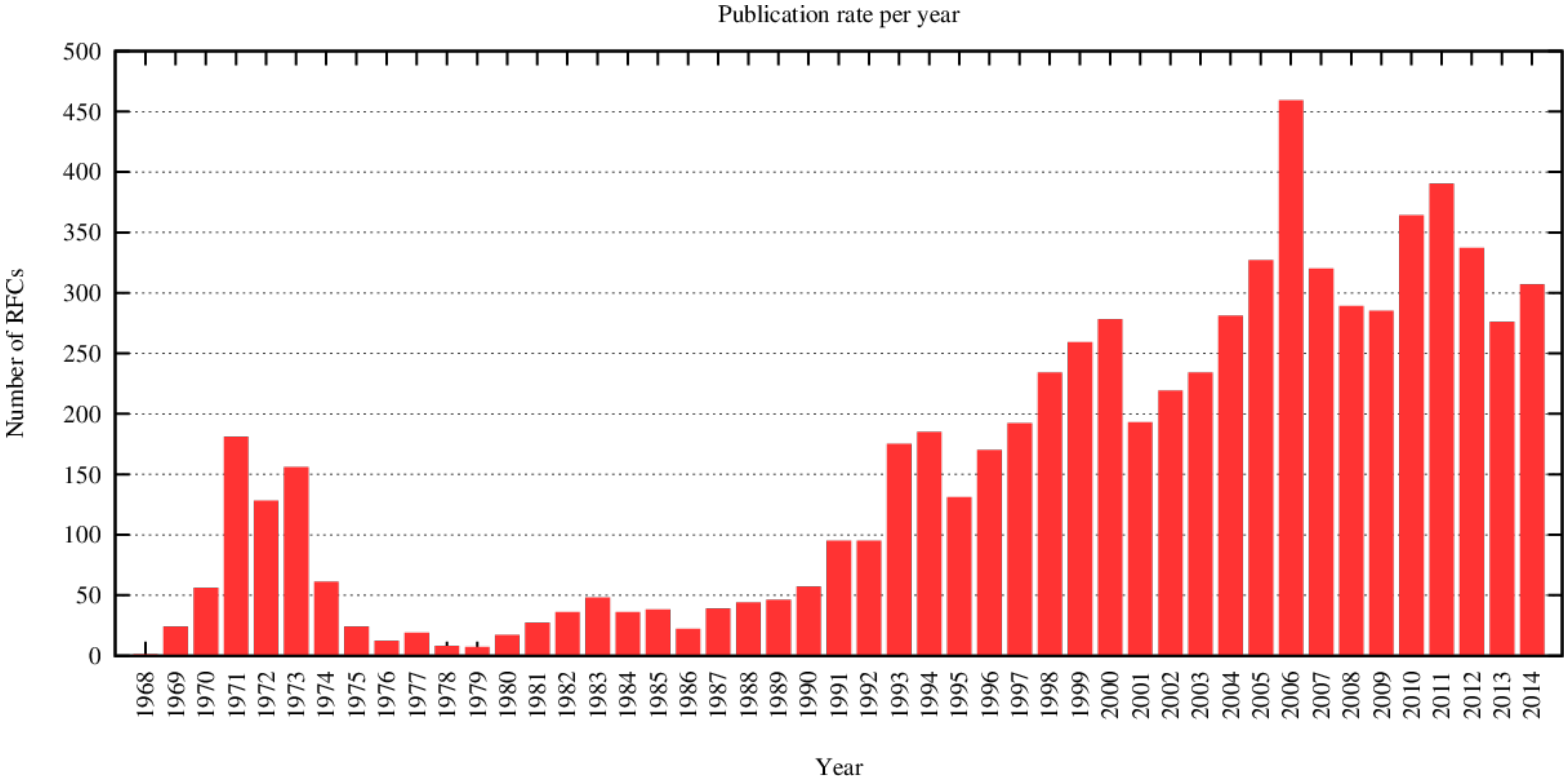
Vertically integrated



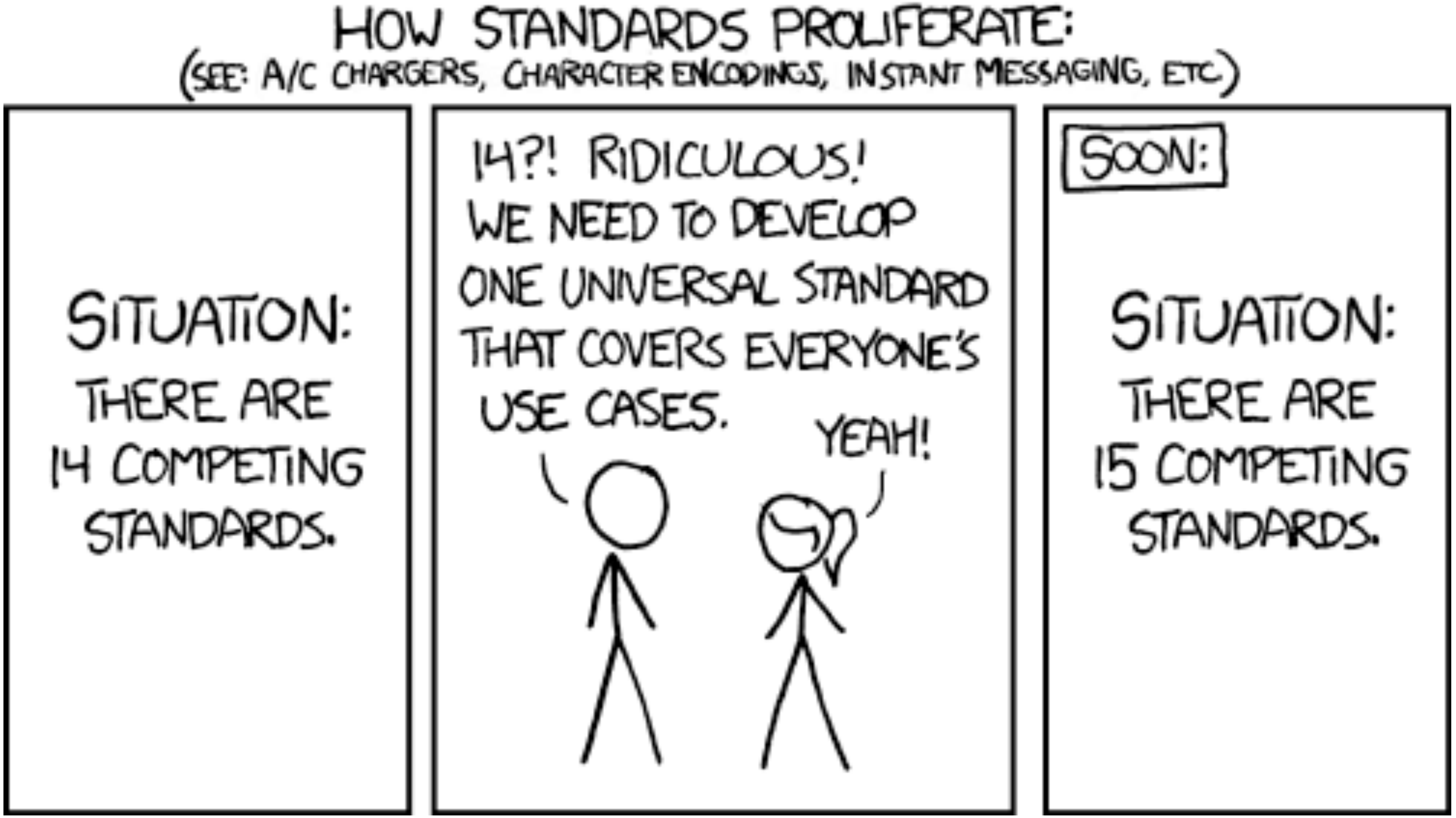
Closed platform!

Protocols guarantee interoperability…
But what's the drawback?

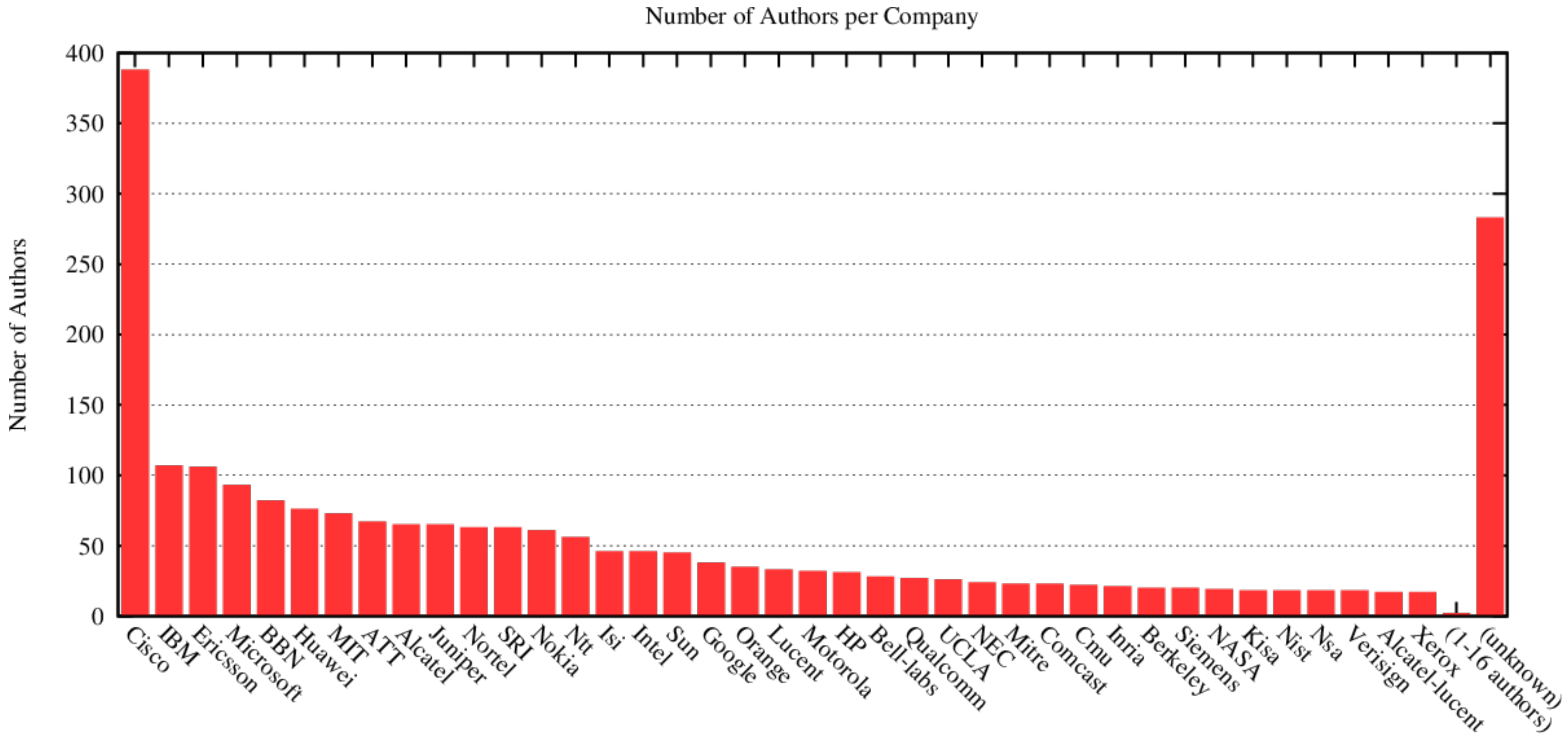
Way too many standards?



Source: IETF


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Vendors dominated?

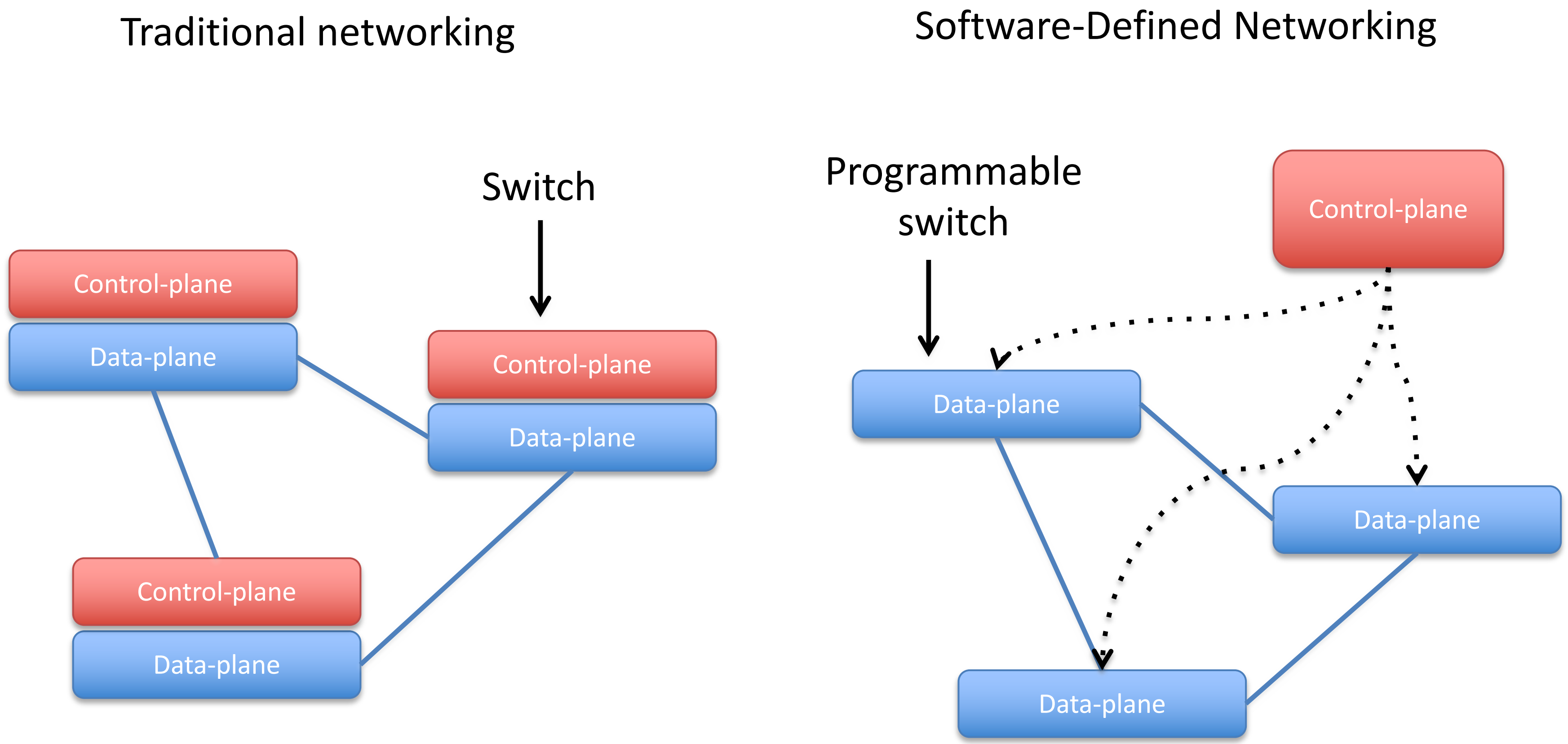


Source: IETF

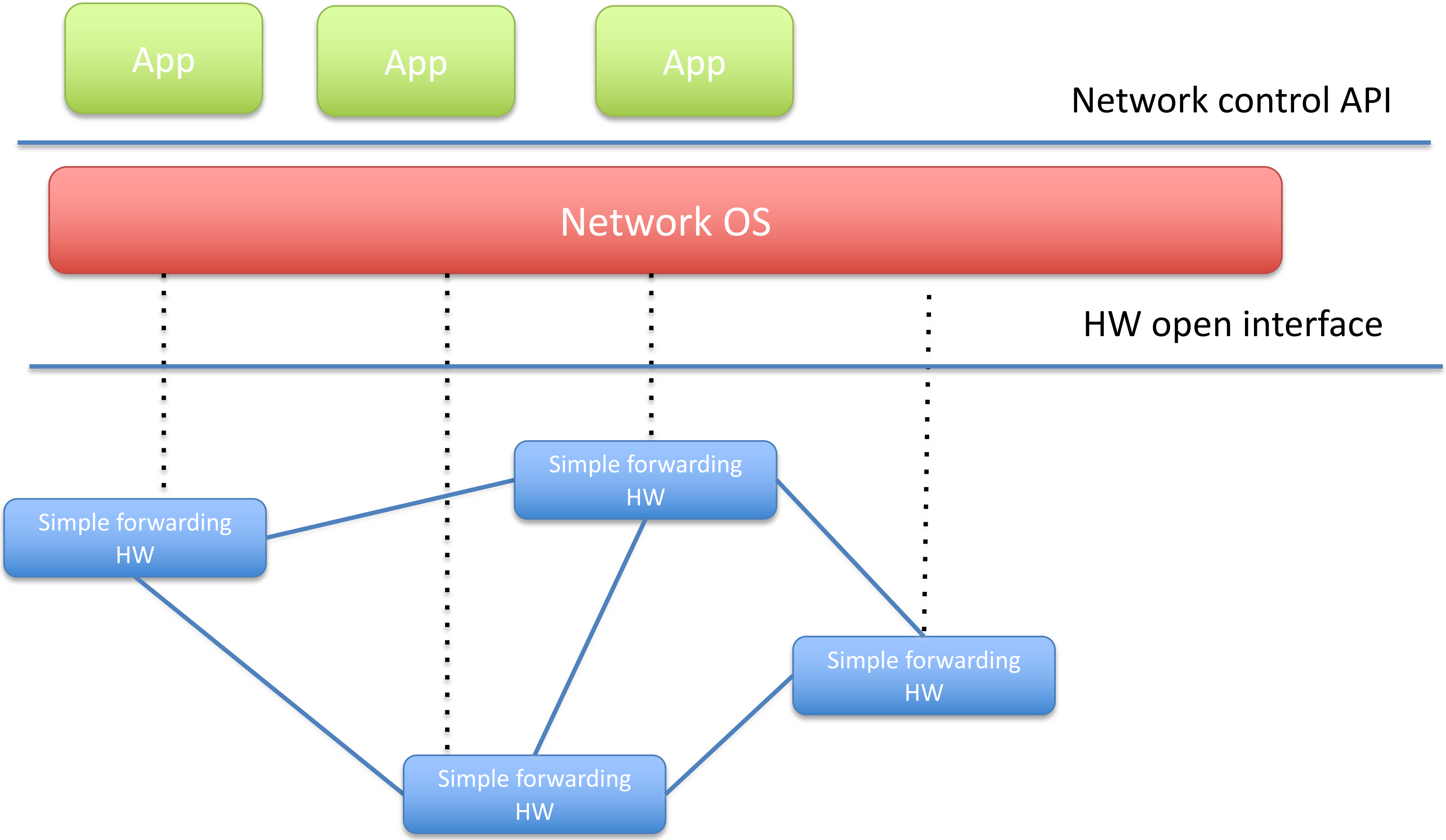
Non-standard management

- **Configuration interfaces** vary across:
  - Different vendors
  - Different devices of same vendor
  - Different firmware versions of same device!
- **SNMP fail**
  - Proliferation of non-standard MIBs
  - Partially implemented standard MIBS
  - IETF recently published a recommendation to stop producing writable MIB modules

The (new) paradigm



SDN architecture



From protocols to API

- HW forwarding abstraction
  - low-level primitives to describe packet forwarding
- Control plane API
  - Network topology abstraction
  - High-level access to switch programming
  - Common libraries
    - Host tracking
    - Shortest-path
    - Etc..

Success keys

- Low-level HW open interface
- Good, extensible and possibly open-source Network OS
- Open market for third-party network application developers
  - Network app store

- **Several attempts** (Active Networks, IETF ForCES), but one winner …

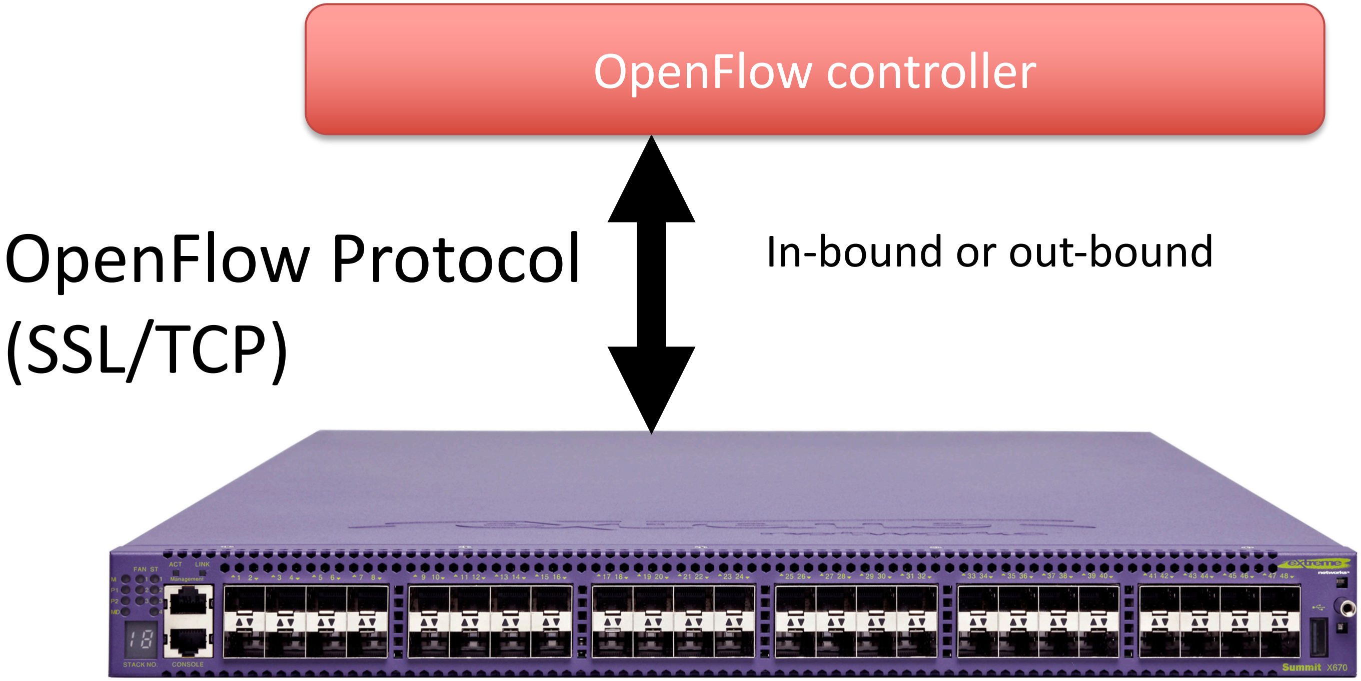
OpenFlow

- Stanford, 2008
- Clean Slate research program
- “With what we know today, if we were to start again with a clean slate, how would we design a global communications infrastructure?”

Is it really a clean slate approach?

OpenFlow

- OpenFlow is actually a **pragmatic approach** to SDN based on a simple HW abstraction that can be implemented with current HW commercial platforms



OpenFlow controller

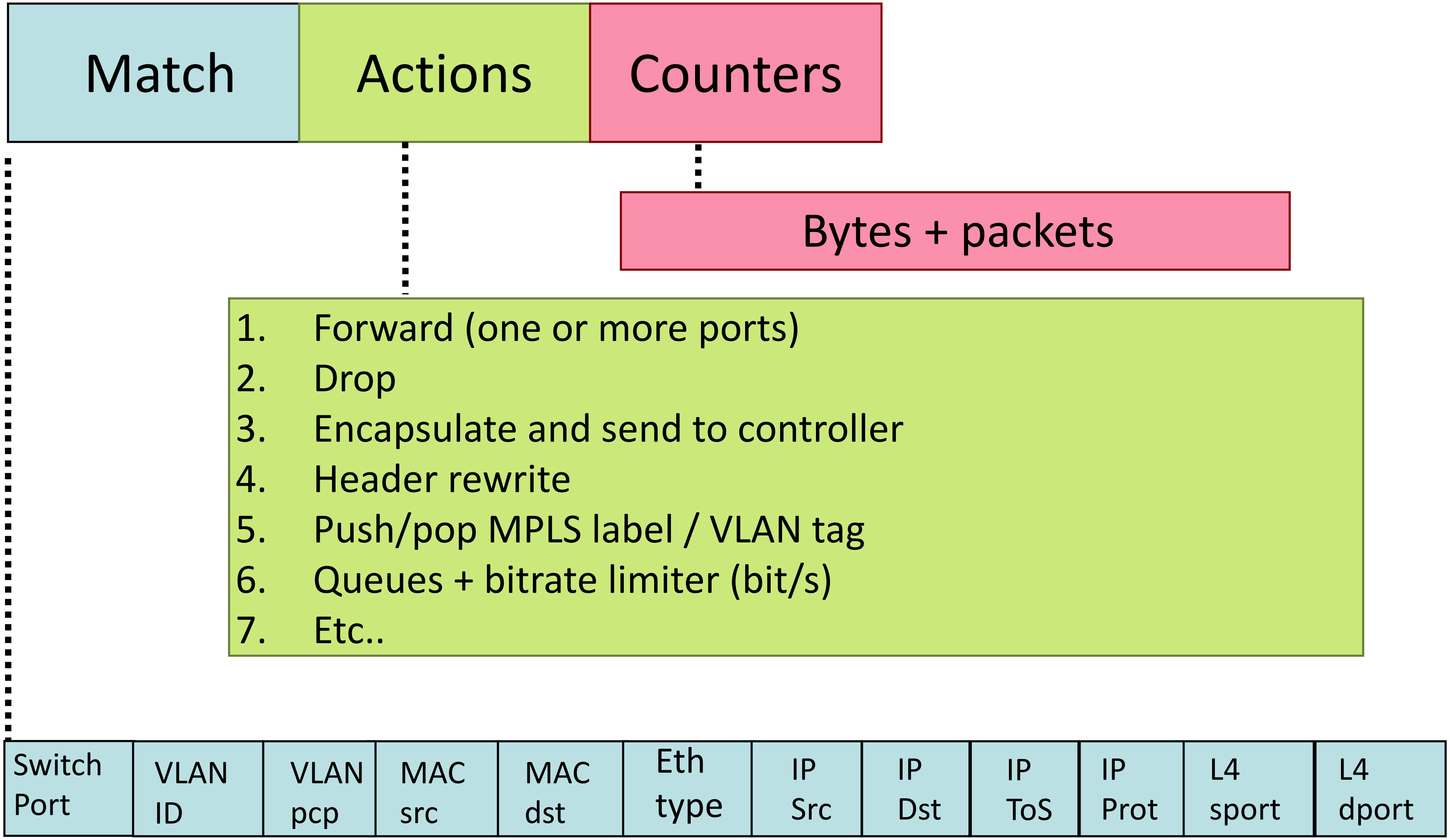
OpenFlow Protocol (SSL/TCP)

In-bound or out-bound

What is OpenFlow

- Switch abstraction
  - Match/action **flow table**
  - Flow counters
  - It doesn't describe how this should be implemented in switches (**vendor neutral !!!**)
- Application layer protocol
  - Binary wire protocol, messages to program the flow table
- Transport protocol
  - TCP, TLS

Flow table



Match	Actions	Counters

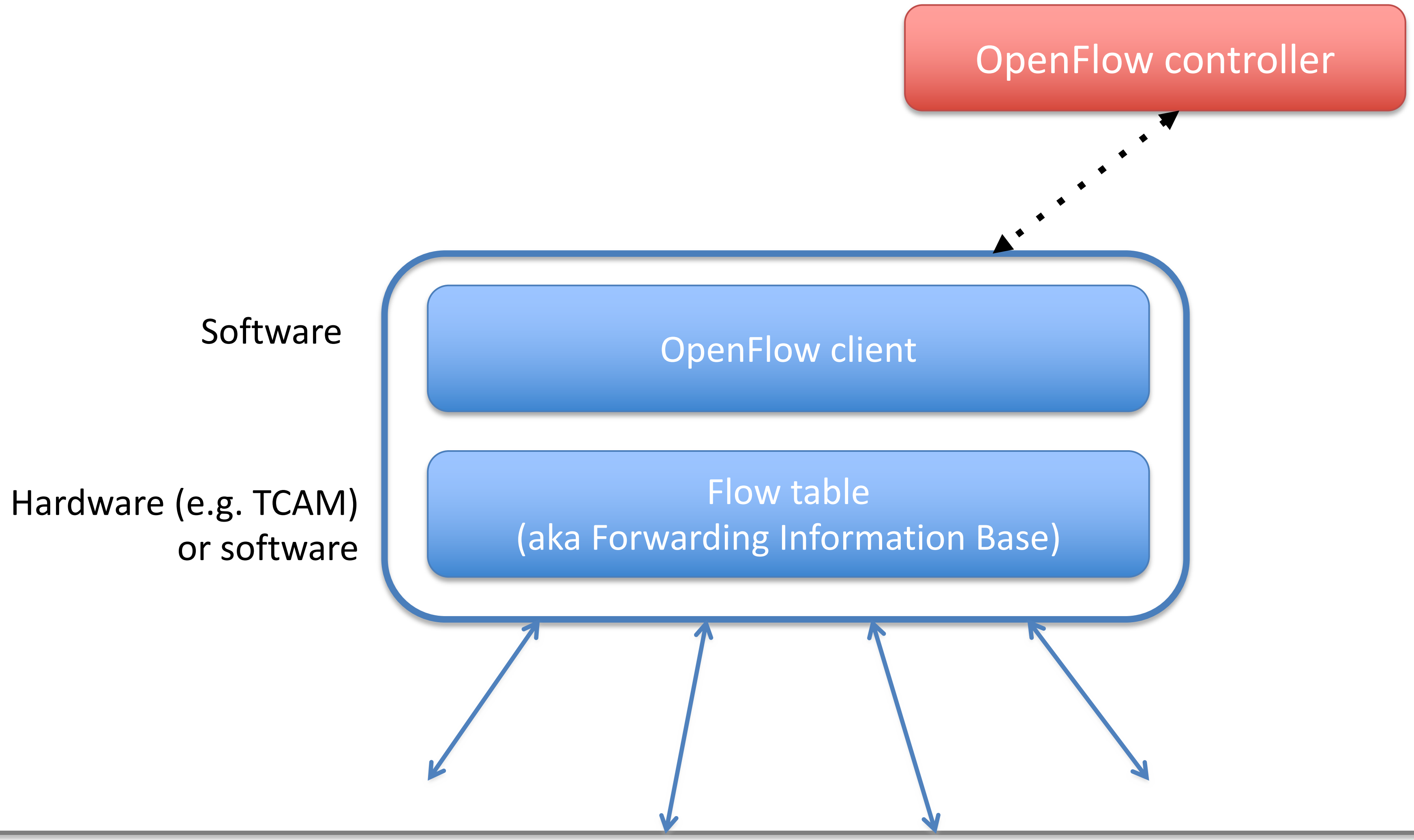
Bytes + packets

1. Forward (one or more ports)
2. Drop
3. Encapsulate and send to controller
4. Header rewrite
5. Push/pop MPLS label / VLAN tag
6. Queues + bitrate limiter (bit/s)
7. Etc..

Switch Port	VLAN ID	VLAN pcp	MAC src	MAC dst	Eth type	IP Src	IP Dst	IP ToS	IP Prot	L4 sport	L4 dport

Slide courtesy: Rob Sherwood

Switch abstraction



OpenFlow controller

Software

OpenFlow client

Hardware (e.g. TCAM)
or software

Flow table
(aka Forwarding Information Base)

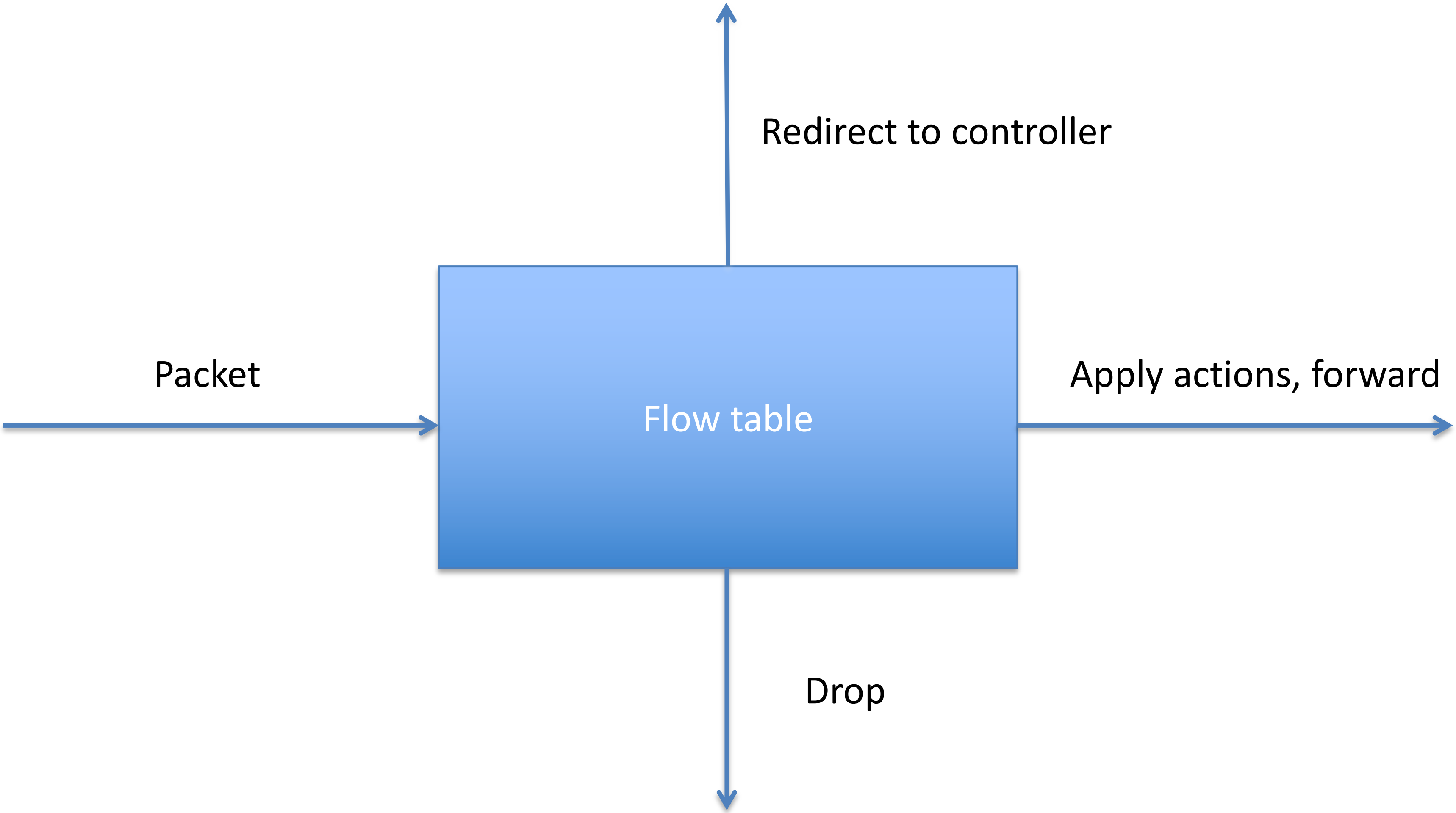
Example

Description	Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dest	TCP sport	TCP dport	Action
L2 switching	*	*	00:1f:..	*	*	*	*	*	*	Port6
L3 routing	*	*	*	*	*	*	5.6.*.*	*	*	Port6
Micro-flow handling	3	00:20..	00:1f..	0x800	Vlan1	1.2.3.4	5.6.7.8	4	17264	Port4
Firewall	*	*	*	*	*	*	*	*	22	Drop
VLAN switching	*	*	00:1f..	*	Vlan1	*	*	*	*	Port6, port7, port8

Reactive vs Proactive

- **Reactive**
  - Start with flow table empty
  - First packet of a flow sent to controller
  - Controller install flow entries
  - Good for stateful forwarding:
    - L2 switching, dynamic firewall, resource management
- **Proactive**
  - Flow entries installed at switch boot
  - Good for stateless forwarding:
    - L3 routing, static firewall, etc..

OpenFlow 1.0 recap

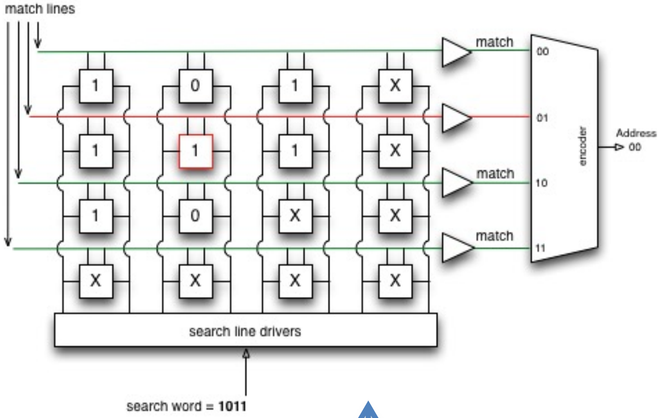


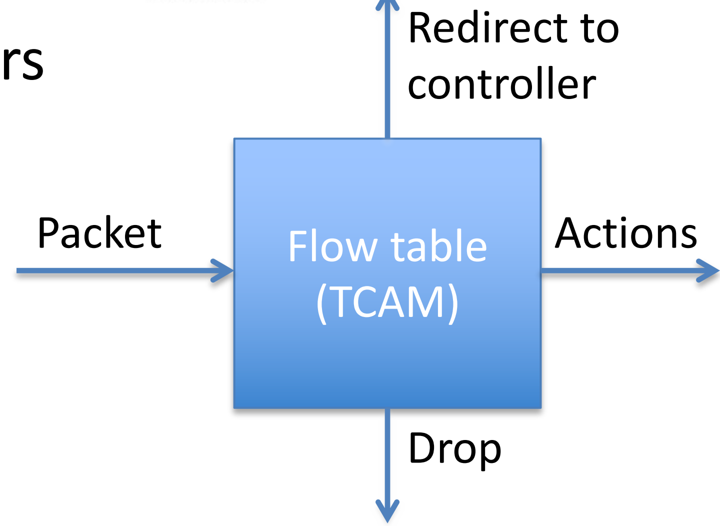
Models can be perfect and clean, reality is dirty!

- The match/action model can ideally be used to program any network behavior and to get rid of protocol limitations at any level
- But unfortunately, with OF:
  - Matches can be done only on a set of predefined header fields (Ethernet, IPv4, MPLS, VLAN tag, etc.)
  - Actions are limited to a rather small set
  - Header manipulation (like adding label/tags, rewriting of fields, etc.) is limited to standard schemes
- As a result, OF is not really protocol independent and standards (including OF standards) are still necessary

Where do OF limitations come from?

- OpenFlow has been designed having in mind current specialized HW architecture for switches
- Specialized HW is still fundamental in networking
  - General purpose HW (CPU) and soft-switches are still 2 order of magnitude slower
  - Architectures based network processors are also at least 1 order of magnitude slower
- The reference HW model for OF flow tables is TCAM (Ternary Content Addressable Memory)





Where do OF limitations come from?

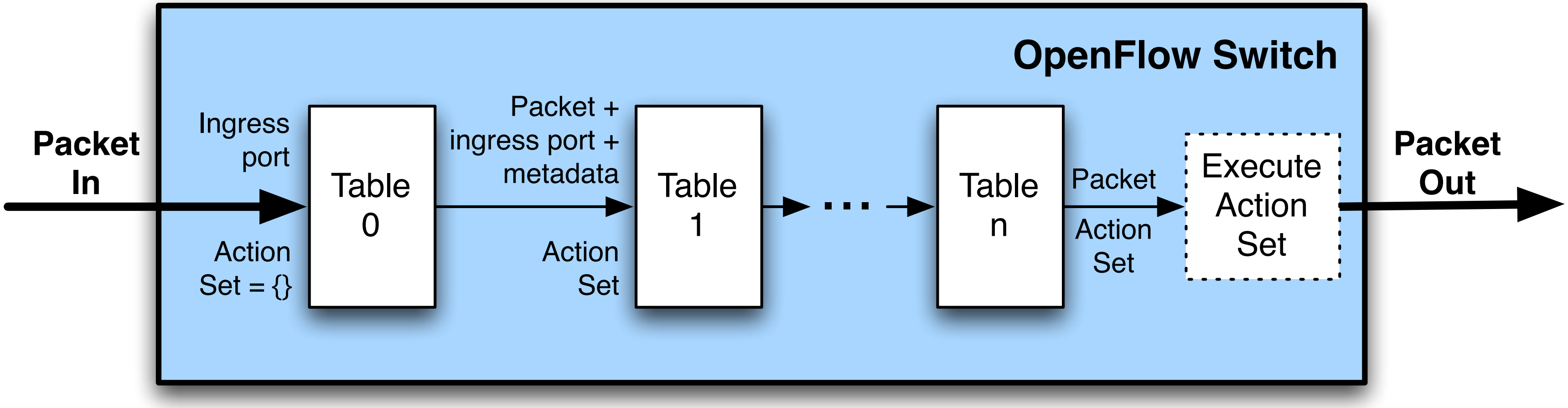
- TCAMs however are typically expensive components that are used by manufacturers only when strictly necessary
- Less expensive memory components based on predefined search keys are often used for most of the common functions of a switch
- OF success depends on its “vendor neutral” approach where implementations issues are completely opaque (including reuse of standard modules for e.g. MAC and IP forwarding)
- Specialized ASICs (Application-Specific Integrated Circuits) are typically complex with a number of hard limitations on table types, sizes, and match depth

Switches cannot remain dumb: Starting the process of data plane evolution

One man alone can be pretty **dumb** sometimes, but for real bona fide stupidity, there ain't nothin' can beat **teamwork**.
[Edward Abbey]

Evolution of the AL in OpenFlow: OF 1.1

- Single tables are costly: all possible combinations of header values in a single long table
- Solution: **Multiple Match Tables (MMT)**



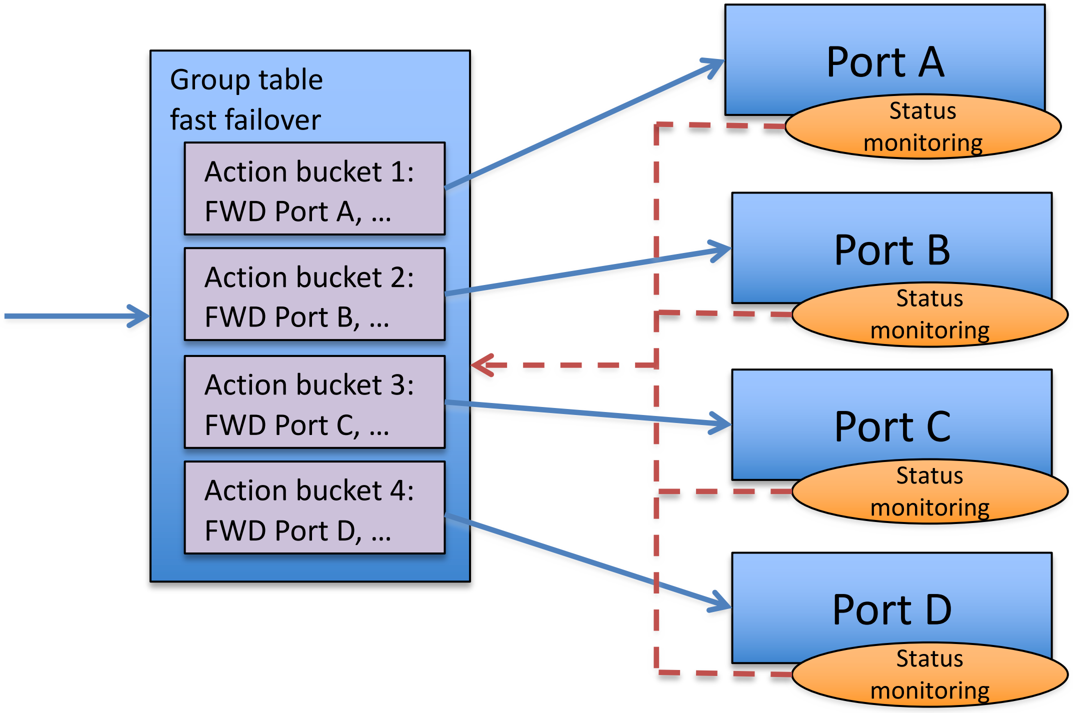
- New actions:
  - **Add metadata**: parameters added and passed to next table
  - **Goto table**: possibility to go to specific tables for further processing

Evolution of the AL in OpenFlow: OF 1.1

- Packets of the same flow are applied the same actions unless the table entry is modified by the controller
- Not good for some common and important cases (e.g. multicast, multipath load balancing, failure reaction, etc.)
- Solution: **Group tables**
  - Goto table “group table n”
  - List of buckets of actions
  - All or some of the buckets are executed depending on the type
- **Types** of Group tables
  - All (multicast)
  - Select (multipath)
  - Fast-failover (protection switching)

Evolution of the AL in OpenFlow: OF 1.1

- **Fast failover**
- Note that this is the first “stateful” behavior in the data plane introduced in OF !!!



Group table
fast failover

Action bucket 1:
FWD Port A, …

Action bucket 2:
FWD Port B, …

Action bucket 3:
FWD Port C, …

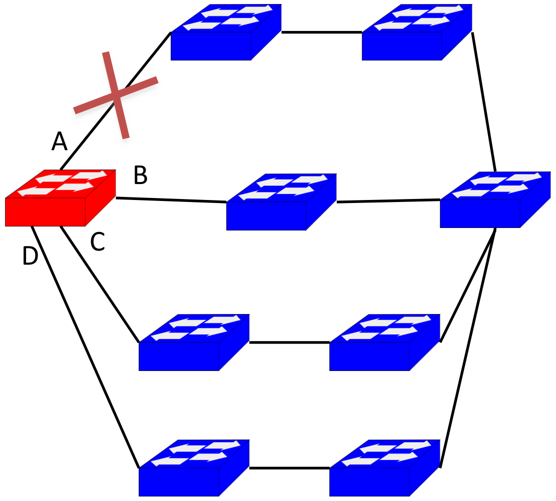
Action bucket 4:
FWD Port D, …

Port A
Status monitoring

Port B
Status monitoring

Port C
Status monitoring

Port D
Status monitoring

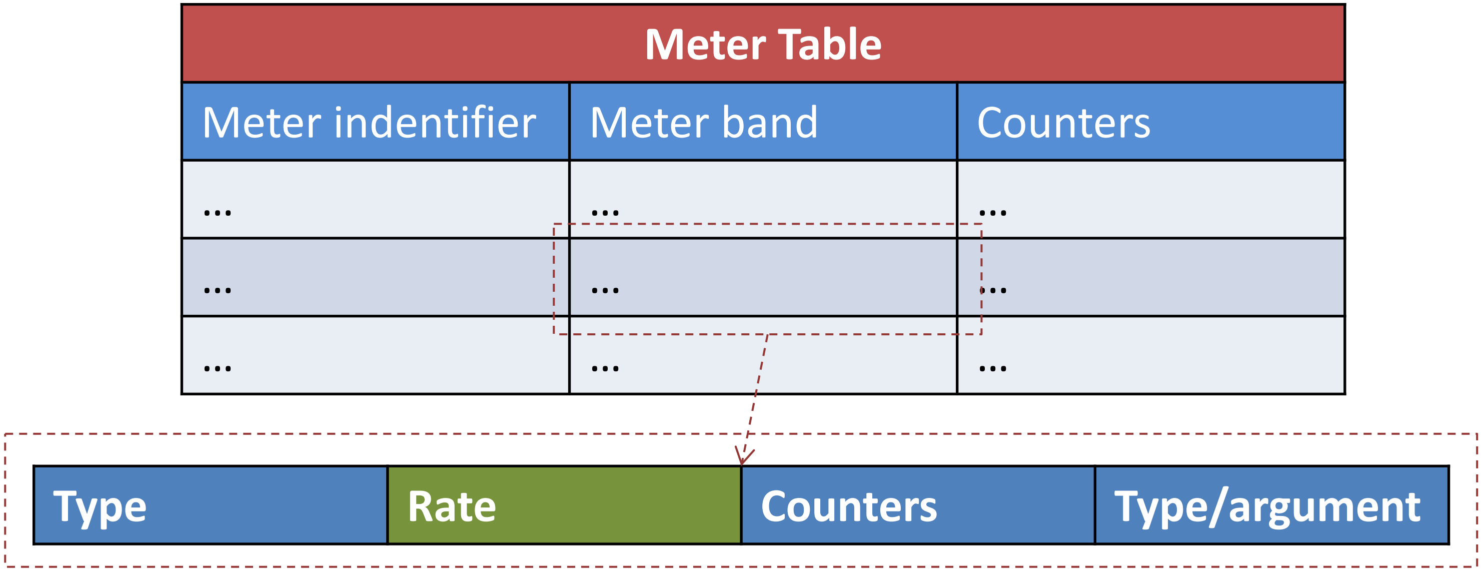


Evolution of the AL in OpenFlow: OF 1.2

- Support for IPv6, new match fields:
  - source address, destination address, protocol number, traffic class, ICMPv6 type, ICMPv6 code, IPv6 neighbor discovery header fields, and IPv6 flow labels
- Extensible match (Type Length Value)
- Experimenter extensions
- Full VLAN and MPLS support
- **Multiple controllers**

Evolution of the AL in OpenFlow: OF 1.3

- Initial traffic shaping and QoS support
  - **Meters:** tables (accessed as usual with “goto table”) for collecting statistics on traffic flows and applying rate-limiters



Meter Table		
Meter indentifier	Meter band	Counters
…	…	…
…	…	…
…	…	…

Type	Rate	Counters	Type/argument

Evolution of the AL in OpenFlow: OF 1.3

- More extensible wire protocol
- **Synchronized tables**
  - tables with synchronized flow entries
- **Bundles**
  - similar to transactional updates in DB
- Support for optical ports

Next OF: discussion started

Within ONF

- Tunnel support
- L4-L7 service support
- Error handling
- Fitness for carrier use
  - Support for OAM in its various forms
- **Flow state (…)**

Next OF: discussion started

- **Flow state**
  - The capability to store / access **flow metadata** that persists for lifetime of flow (not just current packet)
  - Potential to enable a variety of new capabilities:
    - Fragment handling without reassembly
    - Relation between bidirectional flows (e.g., RDI)
    - Autonomous flow learning + flow state tracking
    - MAC learning
    - TCP proxy
  - Hierarchies of flows
    - e.g. FTP control / data, all belonging to a user, etc.

*[MAC13] Ben Mack-Crane, "**OpenFlow Extensions**", US Ignite ONF GENI workshop, Oct 2013*

Also abstraction “involutions” (?): Typed tables

- “A step back to ensure wider applicability”
- A third way between reactive and proactive
- Pre-run-time description of switch-level “behavioral abstraction” (tell to the switch which types of flowmods will be instantiated at run time)
- Limit types supported according to HW type

Typed tables patterns: Forwarding Elements (F:E.)

OpenFlow 1.0

Layer 3 IPv4

Statefull Generic Tunnel

802.1D Forwarding

(…)

Constrained OpenFlow 1.1

Stateless Generic Tunnel

ONF Forwarding Abstractions WG

Further flexibility limitations due to specialized HW

- OF introduced the MMT model but does not mandate the width, depth, or even the number of tables
- OF allows the introduction of new match fields through a user-defined field facility
- But existing switch chips implement a small (4–8) number of tables whose widths, depths, and execution order are set when the chip is fabricated.
- This severely limits flexibility through specialized chips:
  - Chip for core routers: large 32-bit IP longest matching table and a small 128 bit ACL match table
  - Chip for enterprise router: small 32-bit IP table and large ACL table, with an additional MAC address match tables

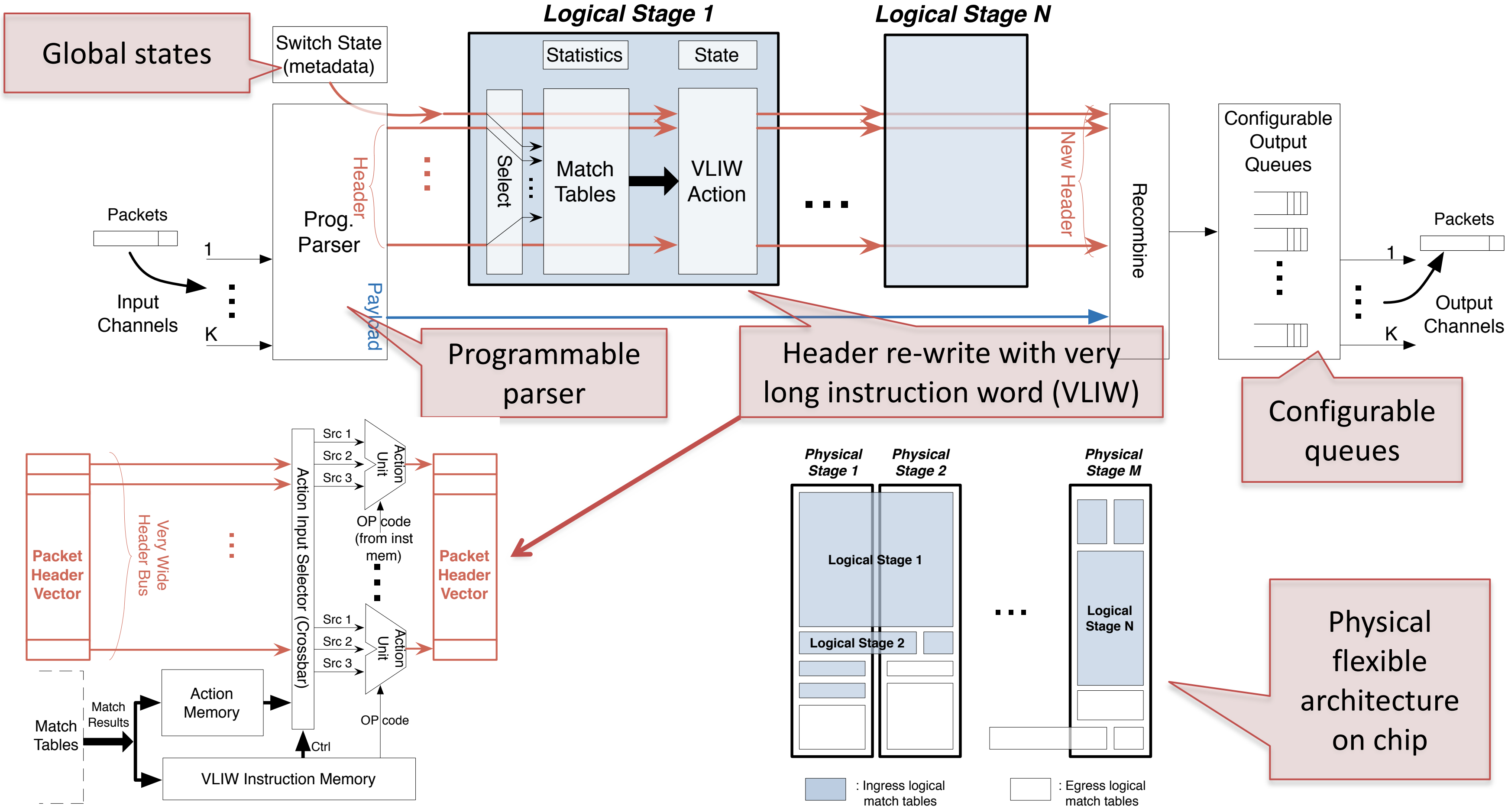
[BOS13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, ***"Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn",*** *in ACM SIGCOMM 2013.*

Reconfigurable Match Tables (RMT)

- Recent proposal by McKeown, Varghese et al. [BOS13]
- RMT
  - Field definitions can be altered and new fields added
  - Number, topology, widths, and depths of match tables can be specified, subject only to an overall resource limit on the number of matched bits
  - New actions may be defined
  - Arbitrarily modified packets can be placed in specified queues, for output at any subset of ports, with a queuing discipline specified for each queue.

*[BOS13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, **"Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn",** in ACM SIGCOMM 2013.*

Reconfigurable Match Tables (RMT)



Global states

Programmable parser

Header re-write with very long instruction word (VLIW)

Configurable queues

Physical flexible architecture on chip

*[BOS13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, **"Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn"**, in ACM SIGCOMM 2013.*

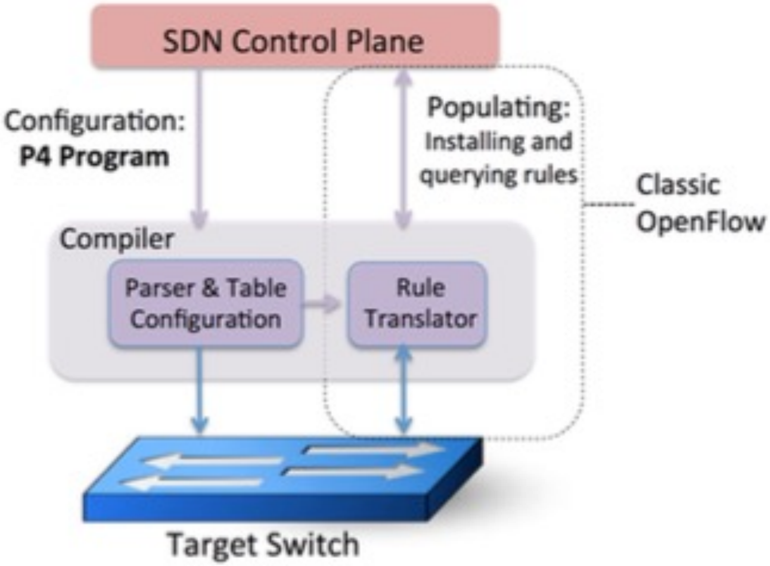
All the way down to programmability: P4

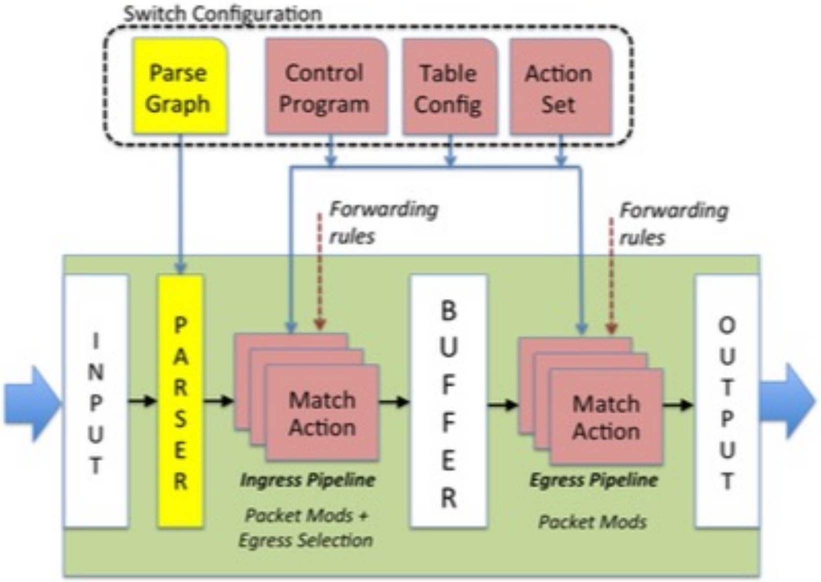
- Recent “strawman proposal” by McKeown, Rexford et al. [BOS13b]
- Towards real protocol independence:
  - **No predefined fields**, but reconfigurable fields
  - **Protocol independence**, switches not be tied to any specific network protocols
  - **Target independence**: flexible packet-processing functionality independently of the specifics of the underlying hardware.
- Advanced “configurability”

*[BOS13b] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, D. Talayco, A. Vahdat, G. Varghese, D. Walker, “**P4: Programming protocol-independent packet processors**,” arXiv:1312.1719.*

All the way down to programmability: P4

- Configurability achieved with a **programming language** describing parsing and control





- Programs are “compiled” according to the specific specialized HW

*[BOS13b] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, D. Talayco, A. Vahdat, G. Varghese, D. Walker, “**P4: Programming protocol-independent packet processors**,” arXiv:1312.1719.*

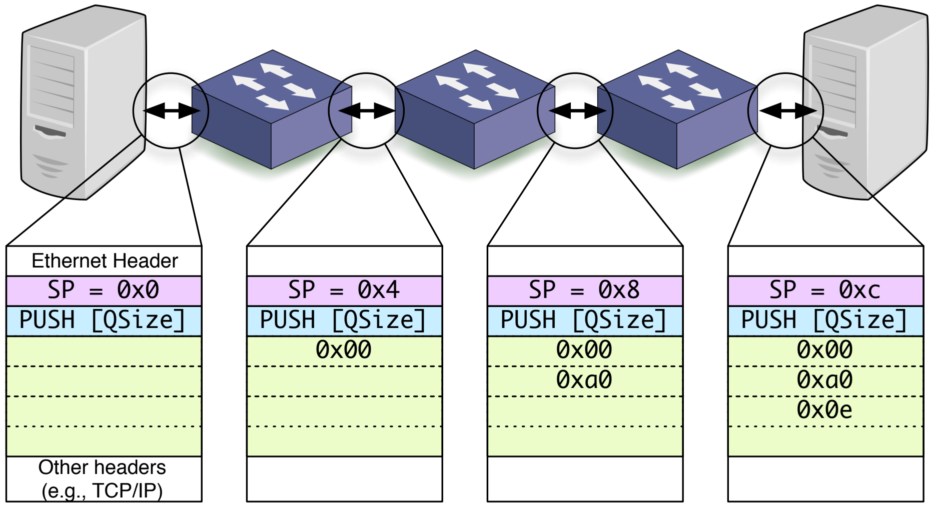
Protocol Oblivious Forwarding (POF)

- Removing/neglecting constraints on general HW can lead to extreme flexibility of a clean slate approach (not in the OF evolution track)
- POF proposal by Huawei [SON13]
- POF makes the forwarding plane totally protocol-oblivious
- The POF FE has no need to understand the packet format.
- POF FE execute instruction of its controller to:
  - extract and assemble the search keys from the packet header,
  - conduct the table lookups,
  - execute the associated instructions (in the form of executable code written in FIS or compiled from FIS).

*[SON13] H. Song, “**Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane**, HotSDN ’13. ACM, 2013, pp. 127–132.*

Tiny programs in the packets

- Taking programmability to the extreme …
- Remember "active networks" …

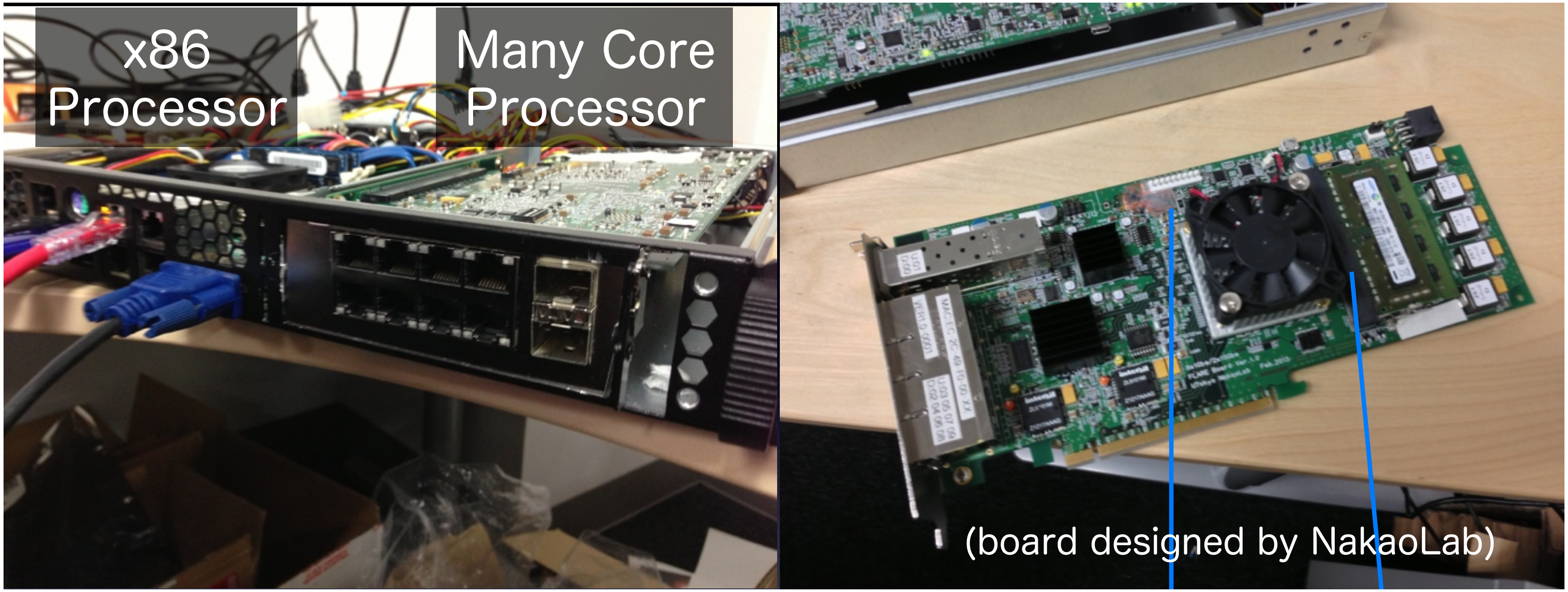


Packet memory is preallocated. The TPP never grows/shrinks inside the network.

*[JEY13] V. Jeyakumar, M. Alizadeh, C. Kim, and D. Mazieres, "**Tiny packet programs for low-latency network control and monitoring**,". HotSDN '13*

Deeply Programmable Networks (FLARE)

- Fully programmable control plane
- Fully programmable data plane
- Flexible and extensible API for both planes
- Experimental implementation



Aki Nakao, FLARE Project, NakaoLab, The university of Tokyo.

Not too much not too little: OpenState and stateful data planes

Too **clever** is **dumb**.
[Ogden Nash]

Looking for the “right” abstraction

- Programmability and real world viability
  - High levels of (deep) programmability in the data and control planes since ages
    - Active Networks
    - IETF ForCES
- Keywords for success:
  - Pragmatism
  - Compromise
  - “right” mix of programmability: right **level of abstraction**
    - Many wonderful programmable platforms buried in the “lab world”

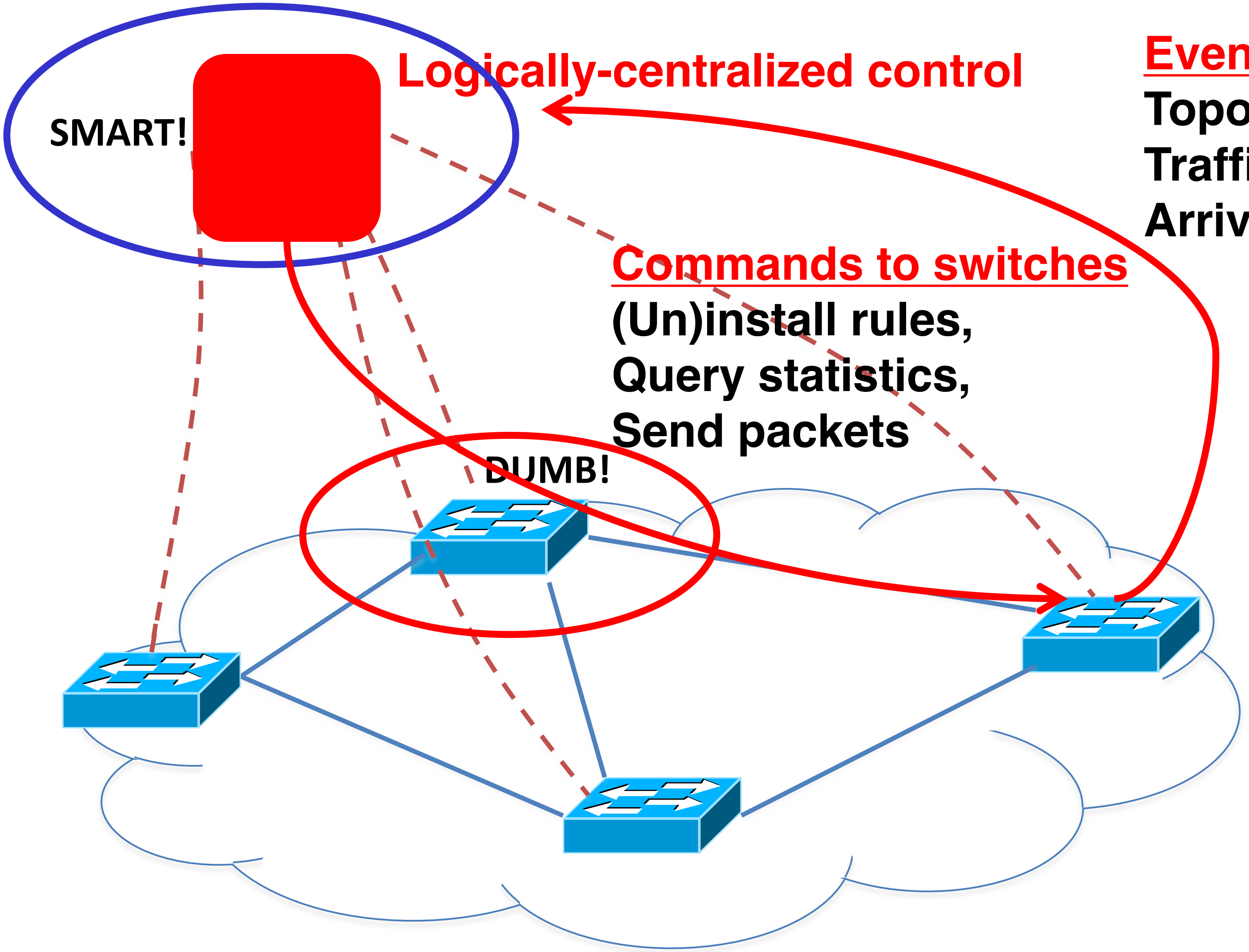
Remember: OF **meant to be** a compromise

[original quotes: from OF 2008 paper]

- **Best approach**: "persuade commercial name-brand equipment vendors to provide an open, programmable, virtualized platform on their switches and routers"
  - Plainly speaking: *open the box!! No way…*

- **Viable approach**: "compromise on generality and seek a degree of switch flexibility that is
  - High performance and low cost
  - Capable of supporting a broad range of research
  - **Consistent with vendors' need for closed platforms.**

OF **forces** separation of data and control



Logically-centralized control

Events from switches

Topology changes,
Traffic statistics,
Arriving packets

Commands to switches

(Un)install rules,
Query statistics,
Send packets

Centralized Control: pros and cons

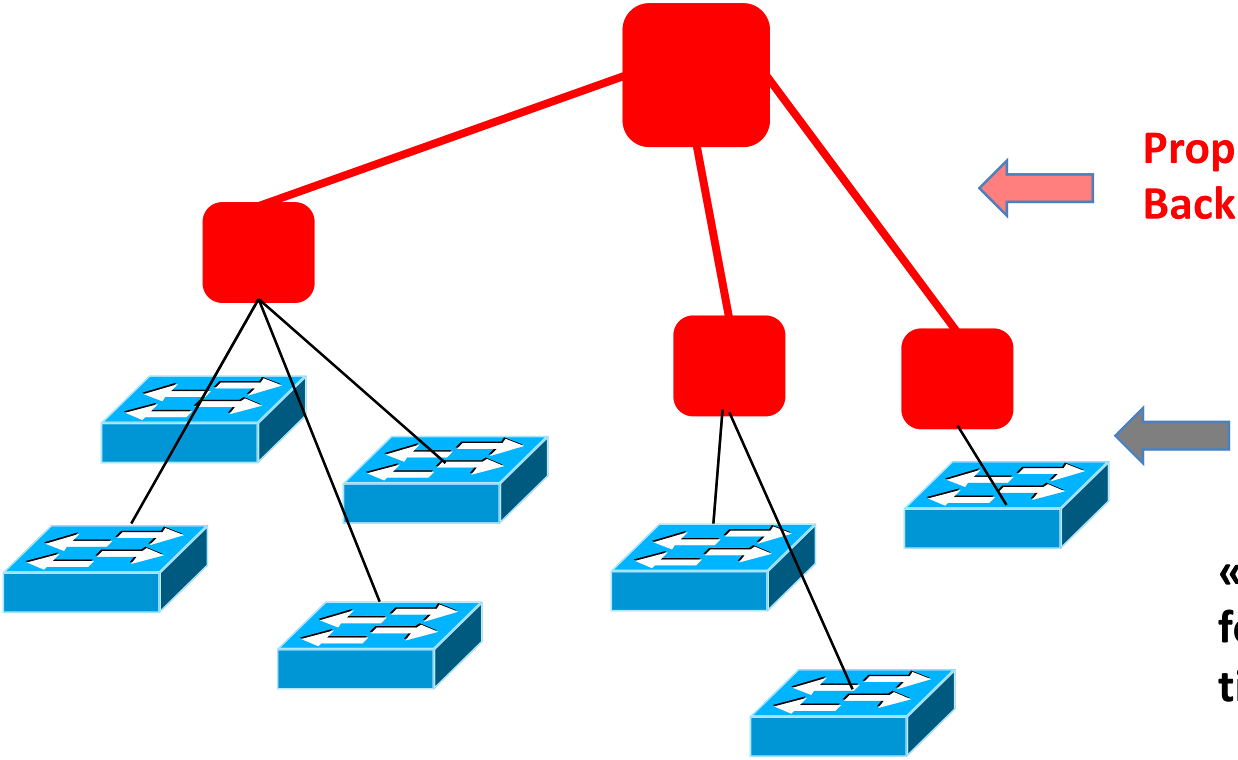
- PROS:
  - Central view of the network as a whole
    - Network states, etc
  - One-click network config/deploy
    - **Platform agnosting** switch API is key - e.g. OpenFlow forwarding abstraction
- CONS:
  - Control latency!!!
    - O(second)
      1s = 300M packets lost @ 100 gbps
  - Signalling overhead

Great idea for network-wide states and «big picture» decisions

Poor idea for local states/decision, (way!) better handled locally (less delay, less load)

Distributed controllers

the «common» way to address such cons



Proprietary controller extensions?
Back to Babel?

A non-solution!
still slow path latency!!

«true» fast path solution: update forwarding rules in 1 packet time – 3 ns @ 40B x 100 Gbps

3 ns = 60cm signal propagation…

Our vision

Events from switches & central rule updates

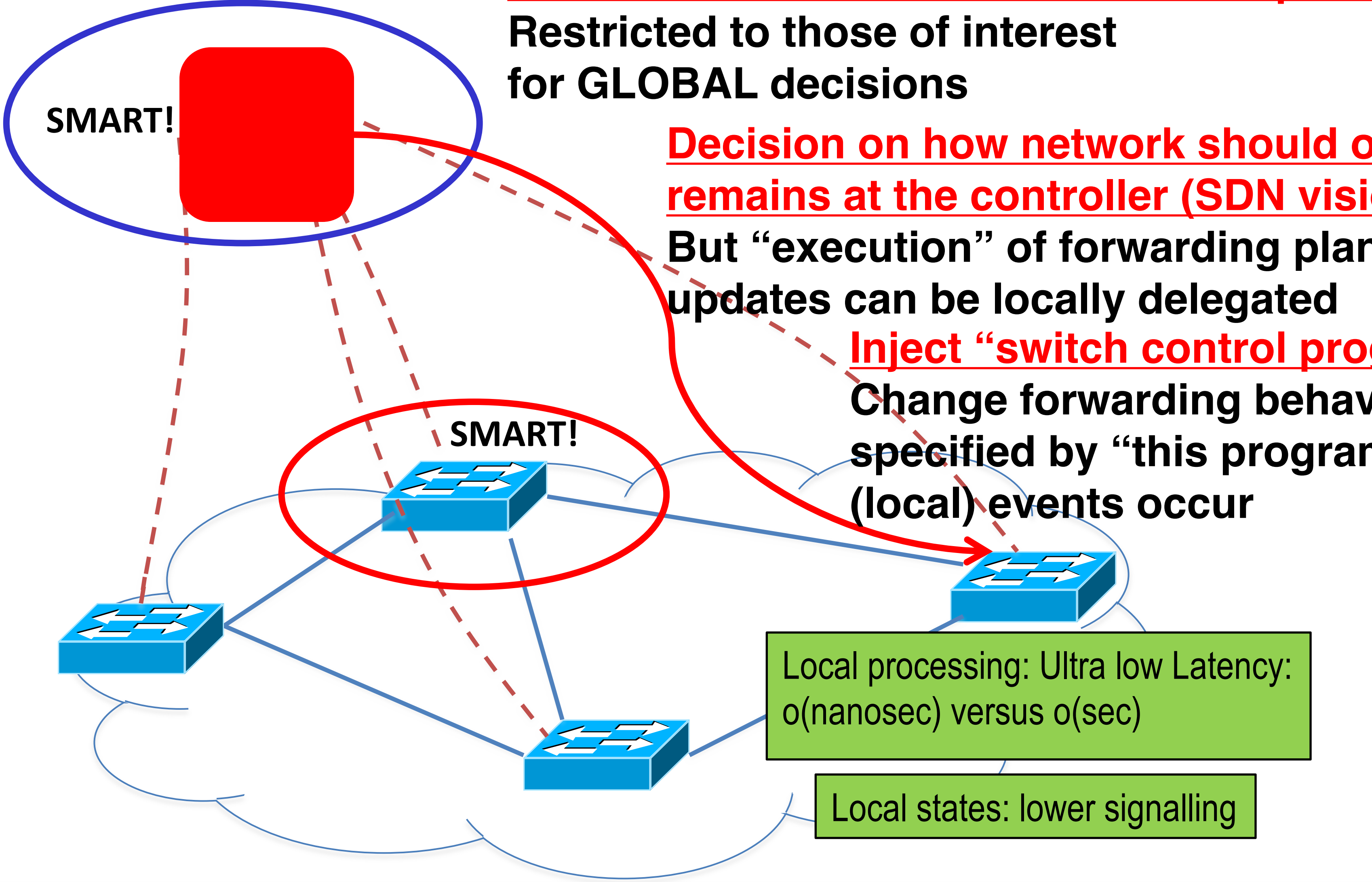
Restricted to those of interest for GLOBAL decisions

Decision on how network should operate remains at the controller (SDN vision)

But “execution” of forwarding plane updates can be locally delegated

Inject “switch control programs”

Change forwarding behavior as specified by “this program” IF (local) events occur



Local processing: Ultra low Latency: o(nanosec) versus o(sec)

Local states: lower signalling

What is missing in the picture

Behavioral Description

src=1.2.*.*, dest=3.4.5.* → drop

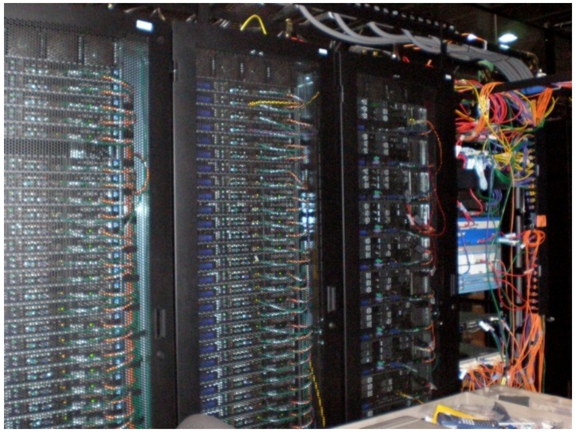
src = *.*.*.*, dest=3.4.*.* → forward(2)

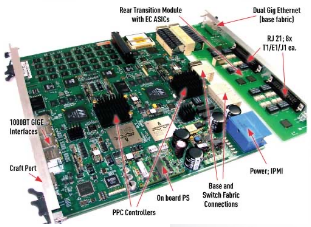
src=10.1.2.3, dest=*.*.*.* → send to controller

OF forwarding abstraction insufficient!!

Platform-agnostic stateful processing: how to?

«repurposed » device




Rear Transition Module with EC ASICs
Dual Gig Ethernet (base fabric)
RJ 21; 8x T1/E1/J1 ea.
1000BT GIGE Interfaces
Craft Port
Power; IPMI
Base and Switch Fabric Connections
On board PS
PPC Controllers

Any vendor, any size, any HW/SW platform…

Easier said than done

- We need a switch architecture and API which is…
  - **High performance**: control tasks executed at wire-speed (packet-based events)
  - **Platform-independent**: consistent with vendors' needs for closed platforms
  - **Low cost and immediately viable**: based on commodity HW

Apparently, far beyond OpenFlow switches…

**Our (perhaps surprising?) finding:
much closer to OF than expected!!**

Our findings at a glance

- Any control program that can be described by a Mealy (Finite State) Machine is already (!) compliant with OF1.3

- MM + Bidirectional flow state handling requires minimal hardware extensions to OF1.1+

- Proof of concept HW and SW implementation

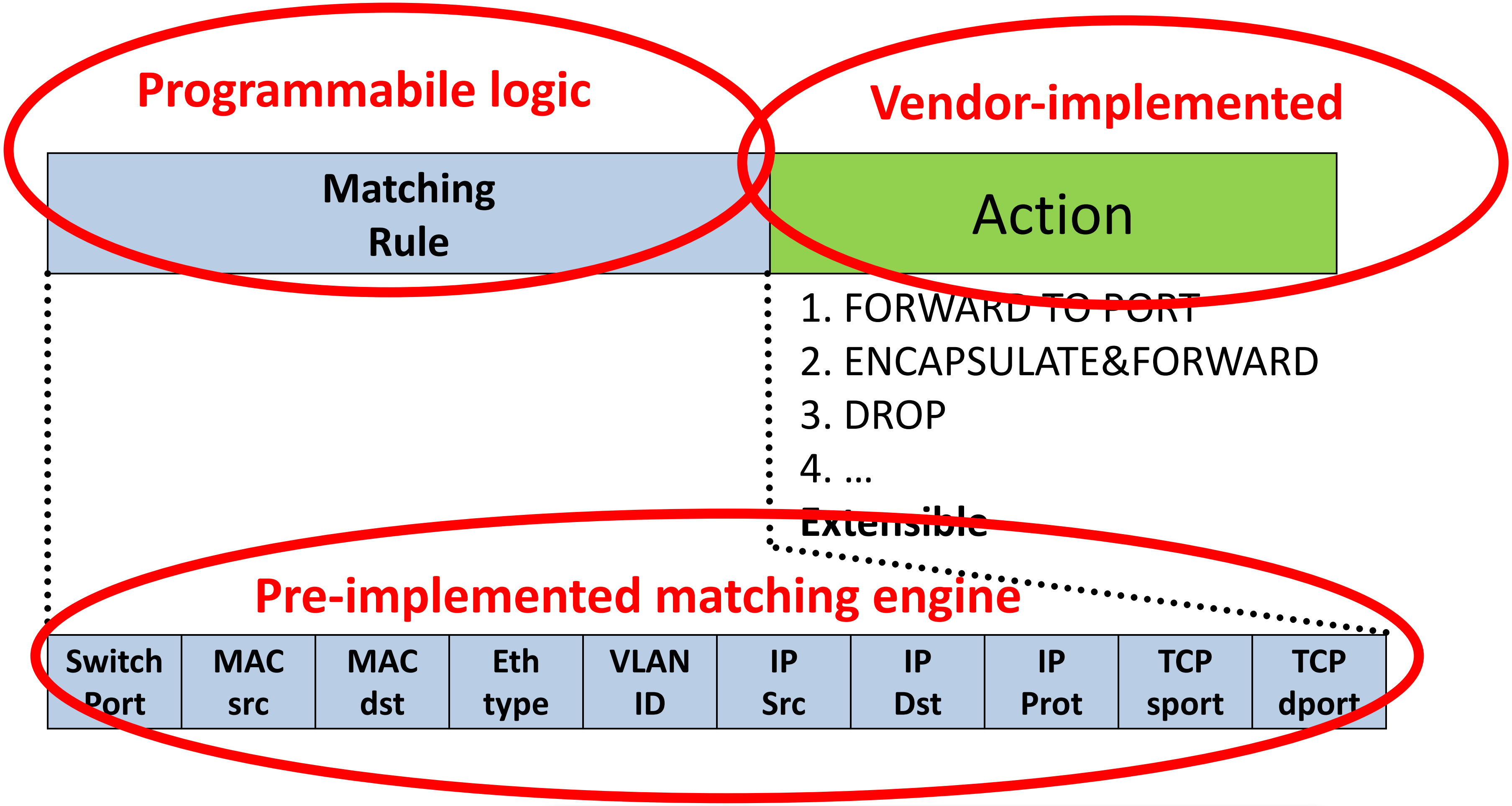
Our approach: **OpenState**

easier understood via a running example: port knocking

[CCR14] G. Bianchi, M. Bonola, A. Capone, C. Cascone, "**OpenState: programming platform-independent stateful OpenFlow applications inside the switch**", ACM Computer Communication Review, vol. 44, no. 2, April 2014.

[ARX14] G. Bianchi, M. Bonola, A. Capone, C. Cascone, S. Pontarelli, "**Towards Wire-speed Platform-agnostic Control of OpenFlow Switches**", available on ArXiv, 2014.

Remember OF match/action API



Programmabile logic

Vendor-implemented

Matching Rule

Action

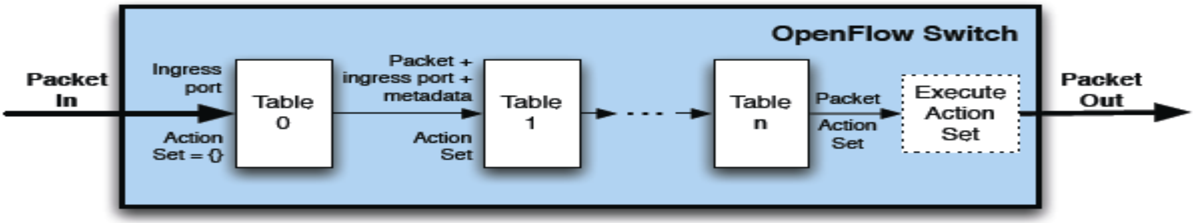
1. FORWARD TO PORT
2. ENCAPSULATE&FORWARD
3. DROP
4. …

Extensible

Pre-implemented matching engine

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport

Multiple flow tables since OF version 1.1

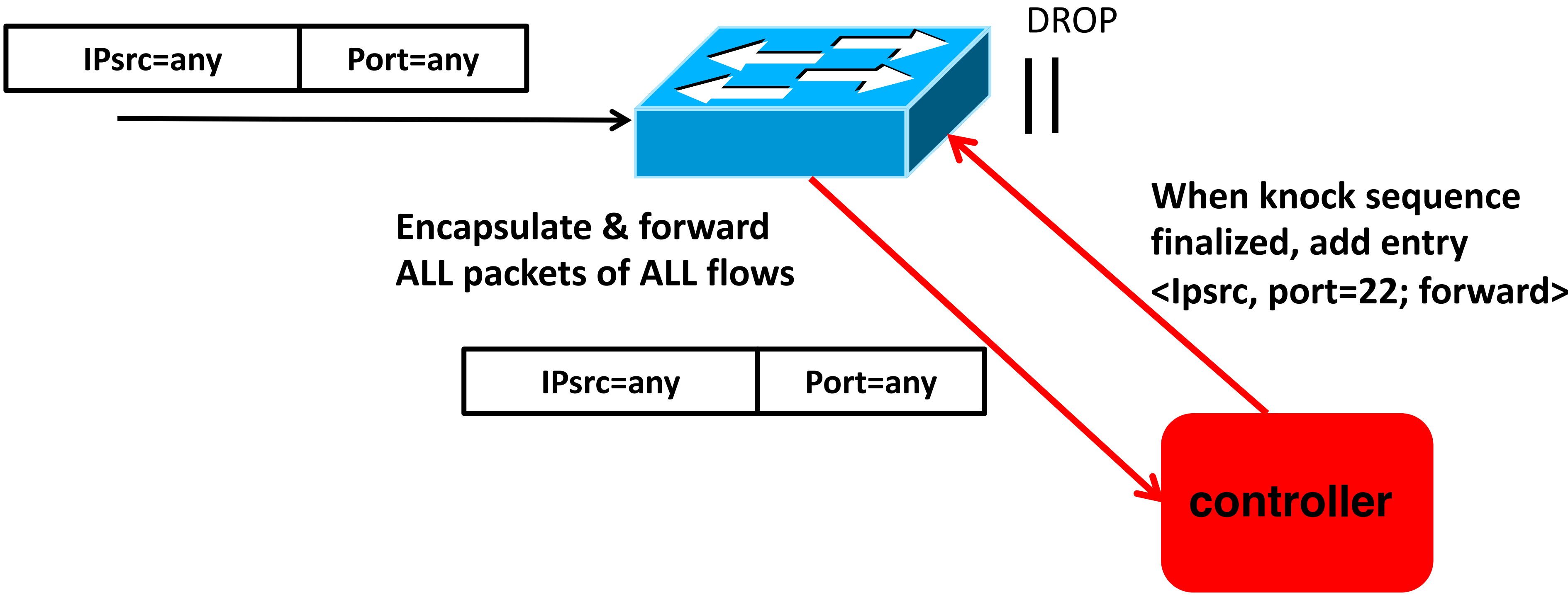


Background: Port Knocking firewall

knock «code»: 5123, 6234, 7345, 8456

IPsrc=1.2.3.4	Port=5123	Drop(); 1.2.3.4 → 1° knock OK
IPsrc=1.2.3.4	Port=6234	Drop(); 1.2.3.4 → 2° knock OK
IPsrc=1.2.3.4	Port=7345	Drop(); 1.2.3.4 → 3° knock OK
IPsrc=1.2.3.4	Port=8456	Drop(); 1.2.3.4 → OPEN port SSH
IPsrc=1.2.3.4	Port=22	Forward()

Port Knocking @ controller



IPsrc=any	Port=any

DROP

Encapsulate & forward
ALL packets of ALL flows

IPsrc=any	Port=any

When knock sequence
finalized, add entry
<Ipsrc, port=22; forward>

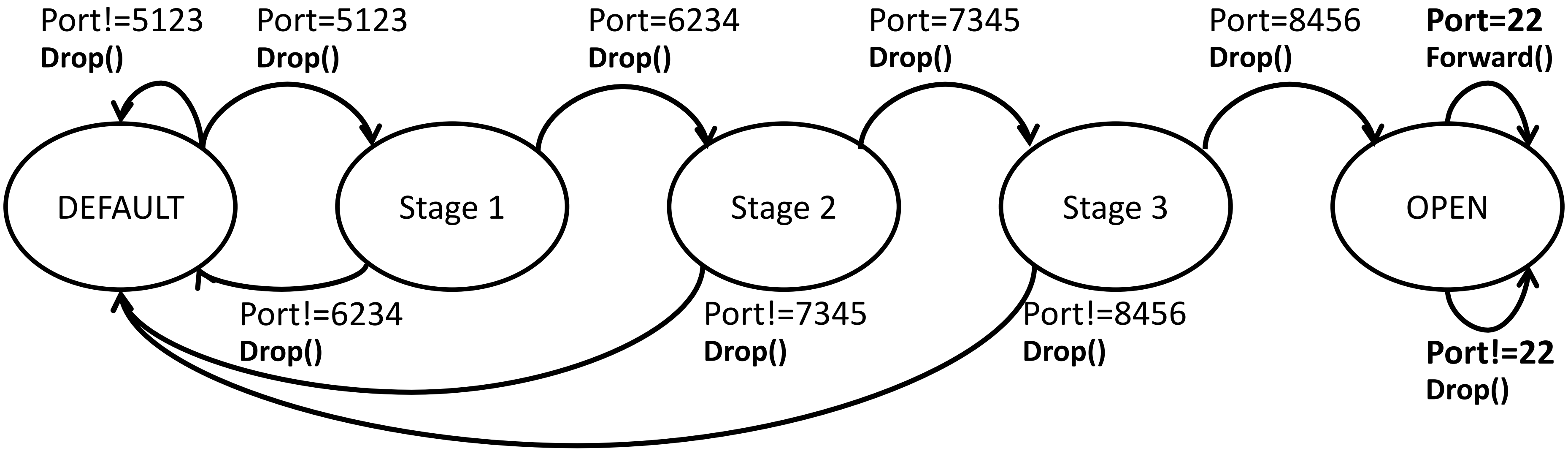
controller

Lots of overhead!!

Maintain Knock state per flow

Needed as no «knock» state handled in switch

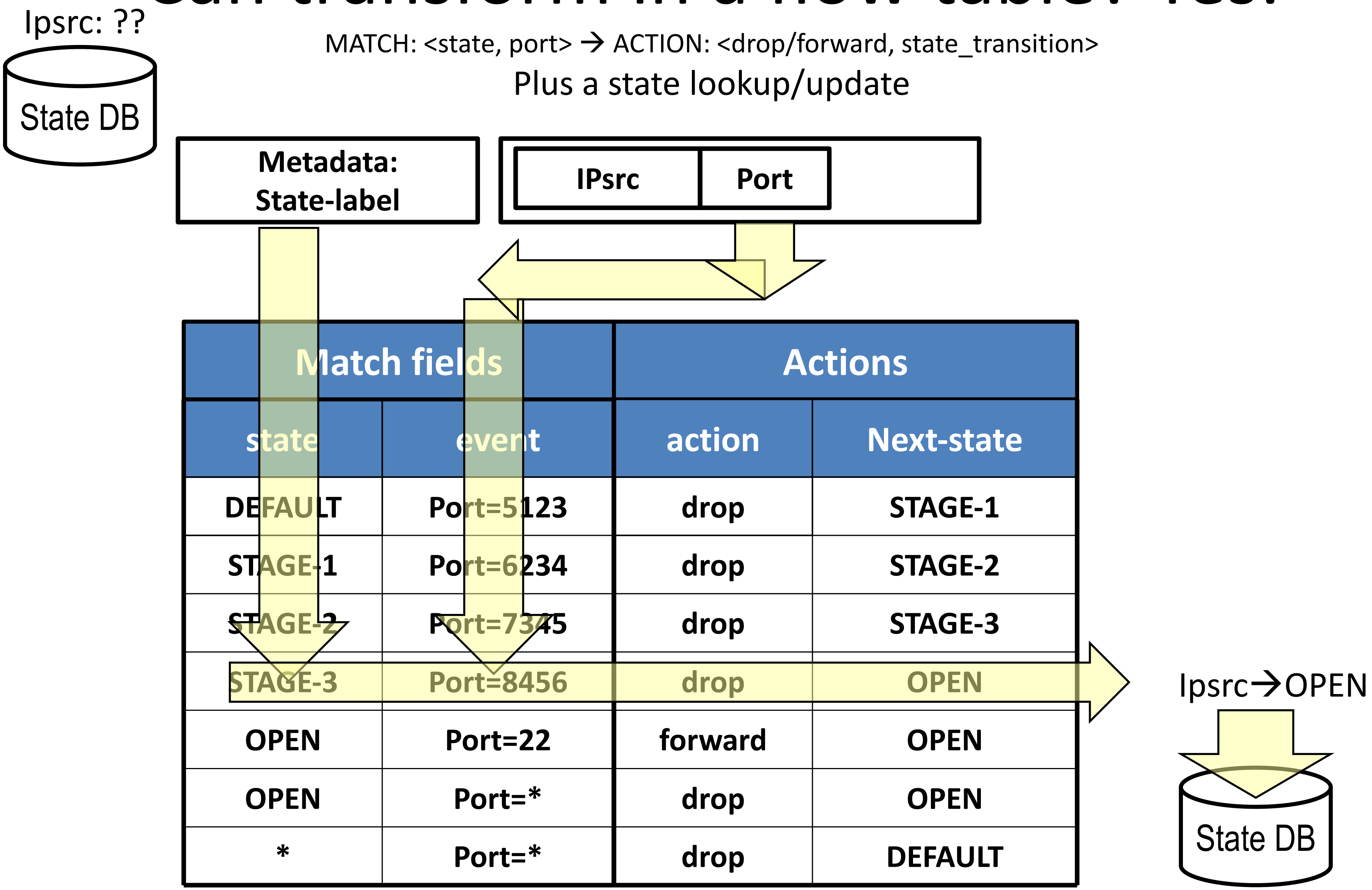
«Abstract» description for port knocking: Mealy Machine



Can transform in a flow table? Yes:

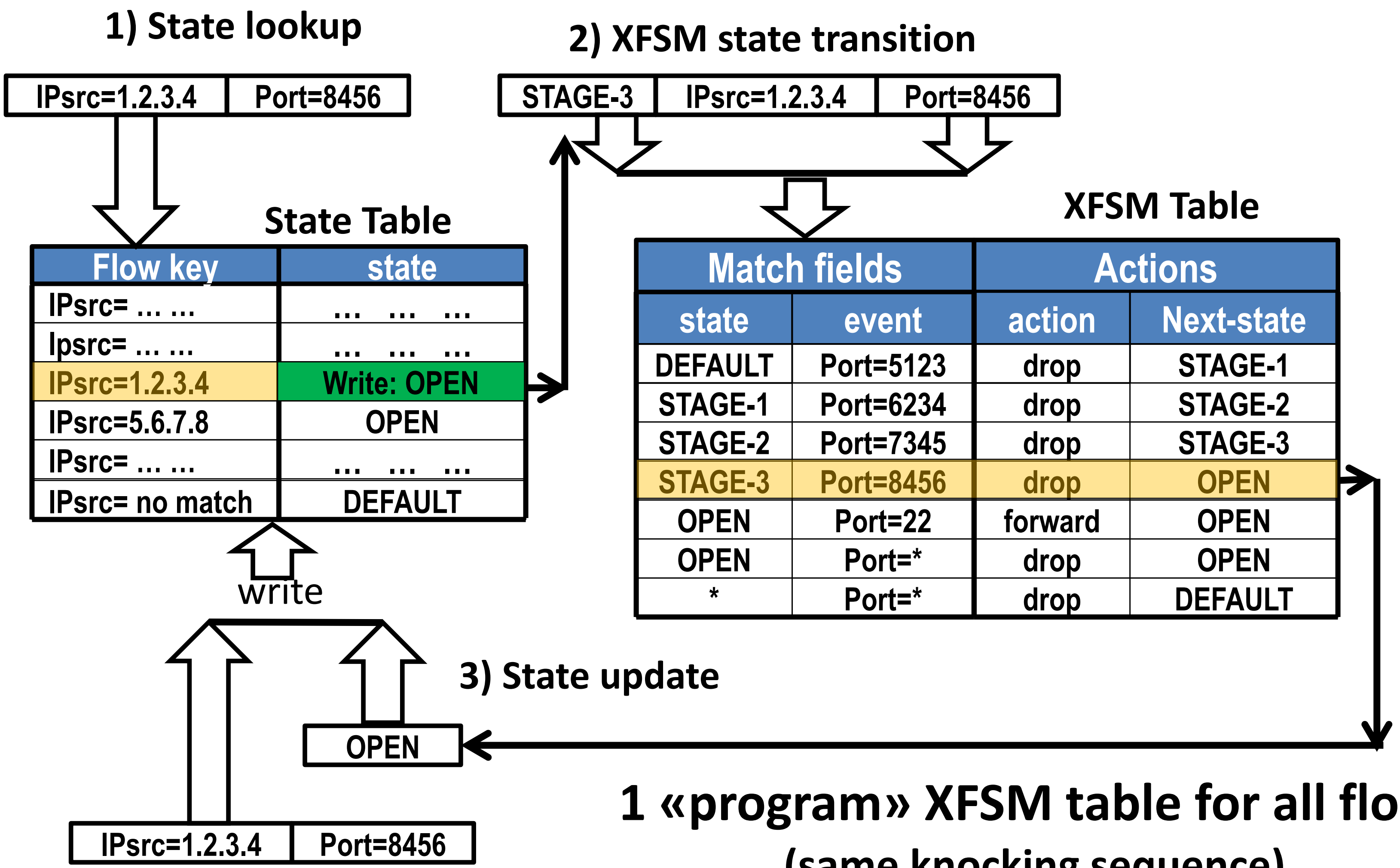
MATCH: <state, port> → ACTION: <drop/forward, state_transition>

Plus a state lookup/update



Match fields		Actions	
state	event	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

Putting all together



1) State lookup

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

2) XFSM state transition

STAGE-3	IPsrc=1.2.3.4	Port=8456

XFSM Table

Match fields		Actions	
state	event	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

3) State update

OPEN

IPsrc=1.2.3.4	Port=8456

1 «program» XFSM table for all flows

(same knocking sequence)

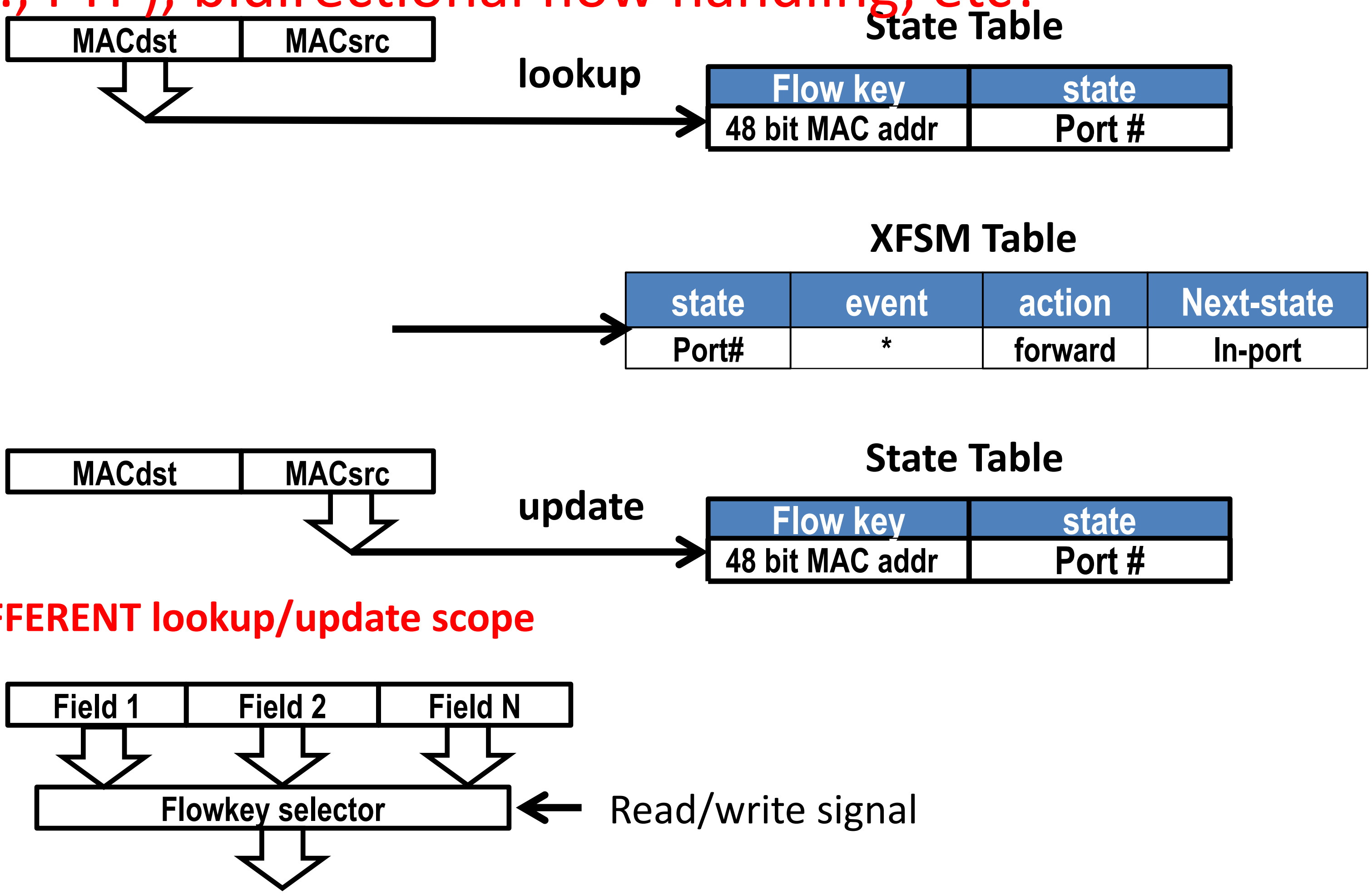
N states, one per (active) flow

Proof of concept

- SW implementation:
  - Trivial modifications to SoftSwitch
  - Public domain
- HW implementation:
  - 5 clock (2 SRAM read + 2 TCAM + 1 SRAM write)
  - 10 Gbps just requires 156 MHz clock TCAM, trivial
  - Optimization in progress (pipelining) for 100 Gbps.

Cross-flow state handling

- Yes but what about MAC learning, multi-port protocols (e.g., FTP), bidirectional flow handling, etc?



State Table

Flow key	state
48 bit MAC addr	Port #

XFSM Table

state	event	action	Next-state
Port#	*	forward	In-port

State Table

Flow key	state
48 bit MAC addr	Port #

DIFFERENT lookup/update scope

Read/write signal

Current challenge

Prove programmability of complex functions

DONE:

- Port Knocking
- MAC learning
- Label/address advertisement learning
- Reverse Path Forwarding
- Flow-consistent Load Balancing
- DDoS multi-stage flow marking
- …

All (!) otherwise not possible without explicit controller's involvement or custom distrib. control…

CHALLENGE:

→ IDS/DPI

→ TCP flow processing

→ Monitoring

→ …

Need new «actions»

Need extra logic (full XFSM)

Our challenge: towards an open «flow processor»?

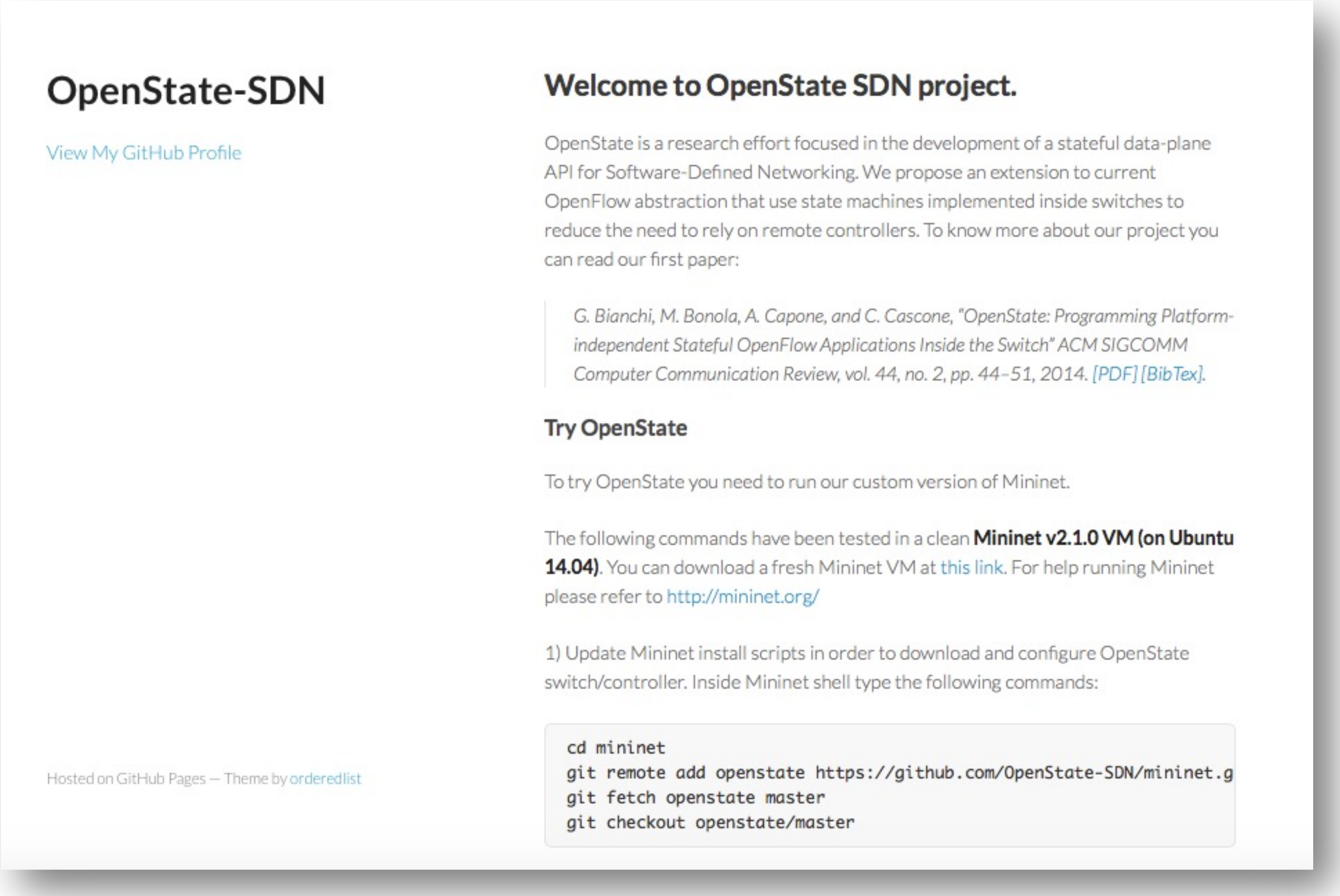
Aftermath

- Control intelligence in devices seems possible
  - Via Platform-independent abstraction
  - Retaining high speed & scalability
  - As «small» OpenFlow extension (?!)
- TCAM as «State Machine processor»
  - Now Mealy Machines
  - Currently working on full XFSM extension
- Rethinking control-data plane SDN separation?
  - Control = Decide! Not decide+enforce!

Applied smartness:
stateful applications

There are science and the **applications** of science,
bound together as the fruit of the tree which bears it.
[Louis Pasteur]

openstate-sdn.org



OpenState-SDN

View My GitHub Profile

Hosted on GitHub Pages — Theme by orderedlist

Welcome to OpenState SDN project.

OpenState is a research effort focused in the development of a stateful data-plane API for Software-Defined Networking. We propose an extension to current OpenFlow abstraction that use state machines implemented inside switches to reduce the need to rely on remote controllers. To know more about our project you can read our first paper:

> *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.* [PDF] [BibTex].

Try OpenState

To try OpenState you need to run our custom version of Mininet.

The following commands have been tested in a clean **Mininet v2.1.0 VM (on Ubuntu 14.04)**. You can download a fresh Mininet VM at this link. For help running Mininet please refer to http://mininet.org/

1) Update Mininet install scripts in order to download and configure OpenState switch/controller. Inside Mininet shell type the following commands:

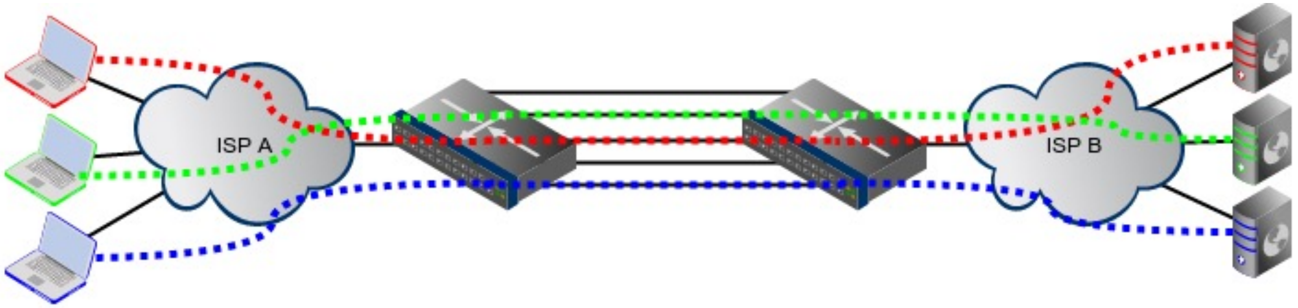
```
cd mininet
git remote add openstate https://github.com/OpenState-SDN/mininet.g
git fetch openstate master
git checkout openstate/master
```

Switch: **ofsoftswitch13**; Controller: **Ryu**

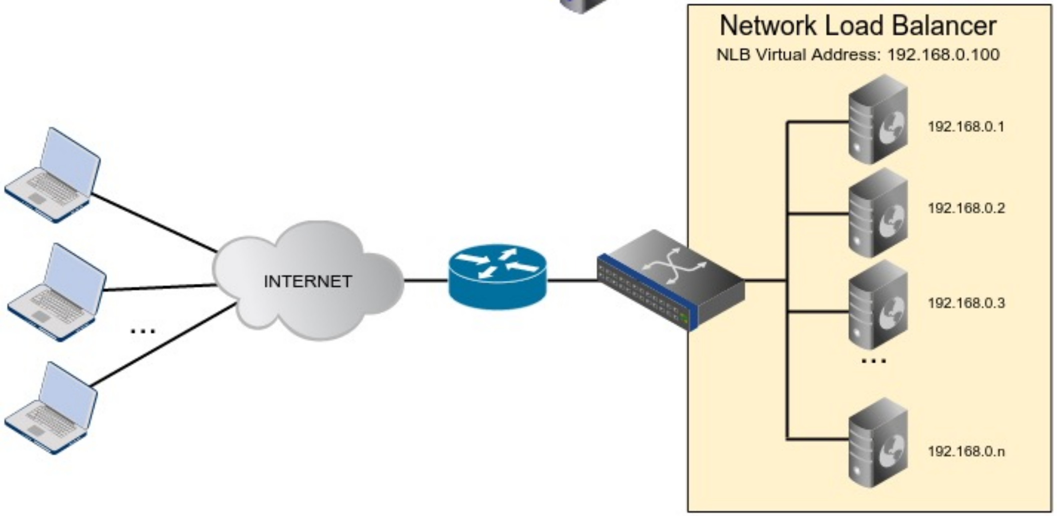
Forwarding Consistency

- Ensure consistency in forwarding decisions for packets of a same transport layer flow

Example: LAG @Internet Exchange Point



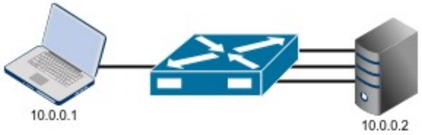
Example: Server Load Balancer



Forwarding Consistency

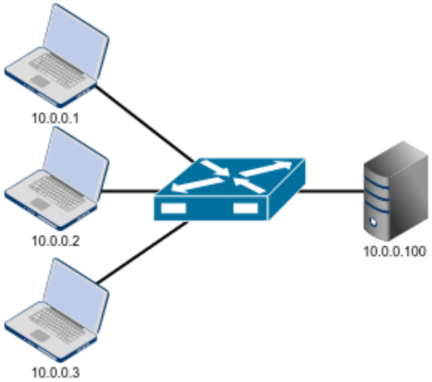
One to Many:

Intra-flow state handling



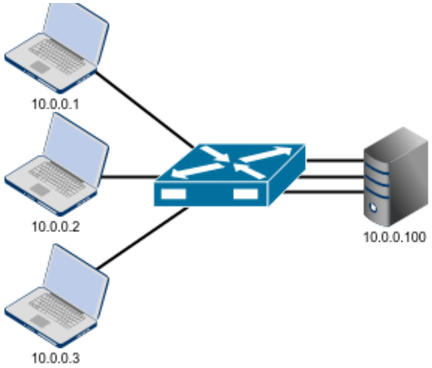
Many to One:

Cross-flow state handling

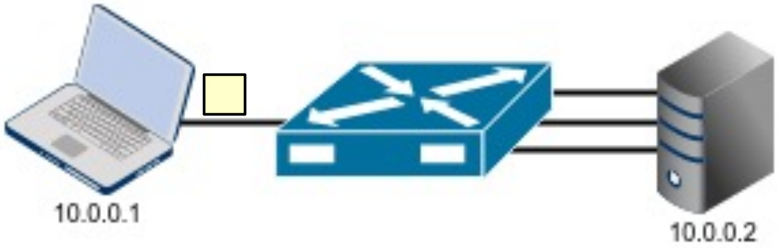


Many to Many:

Inter-stage cross-flow state handling



Forwarding Consistency: **One to Many**

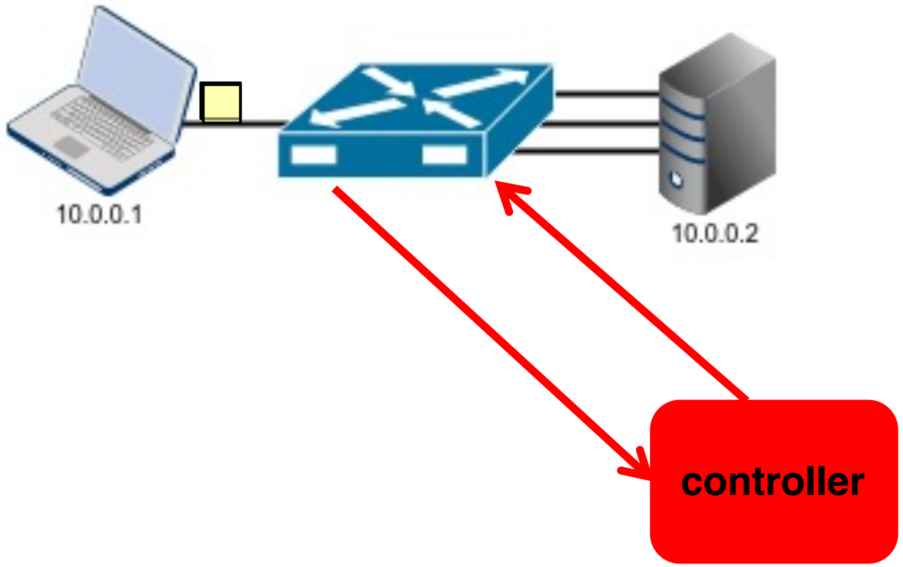


The first packet of a TCP connection coming from the input port is sent to one of many possible output ports.

All the next packets of the same TCP connection must be forwarded on the same selected port.

Forwarding Consistency: **One to Many**

OpenFlow solution: the controller is in charge of states management.



First packet of each new TCP connection is sent to the controller in order to:

-select an output port (e.g. randomly) and forward the packet

-install a flow entry for subsequent packets

Forwarding Consistency: **One to Many**

OpenState solution:

the switch itself handles connection’s state.

A flow is identified by [IP_SRC,IP_DST,TCP_SRC,TCP_DST]
FLOW STATE: output port

First packet of each new TCP connection is sent to the **Group Table** in order to:

- select an output port randomly
- store the selected port in the State Table for subsequent packets

Controller is not involved!

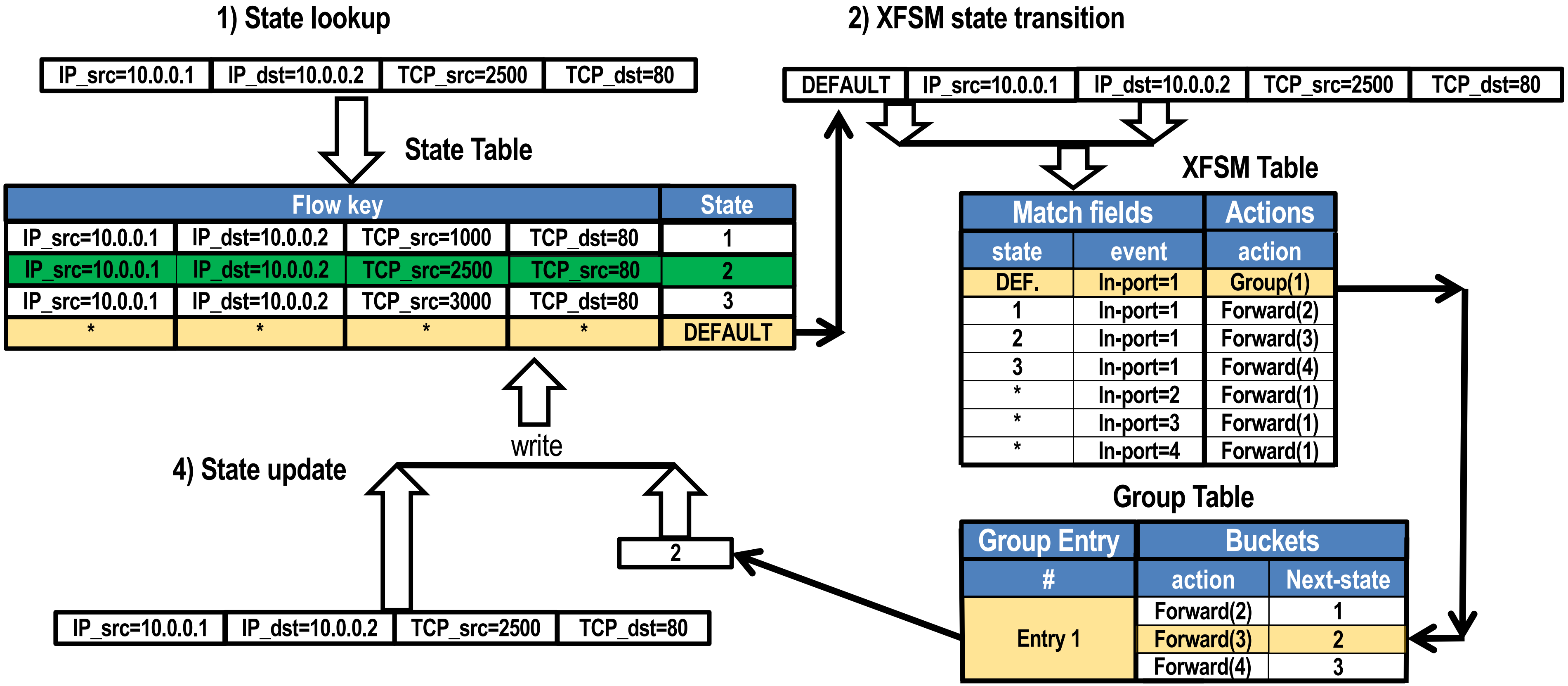
Forwarding Consistency: **One to Many**

- In this case the state is set in the group table
- Random Group Entry

10.0.0.1
10.0.0.2

Lookup Scope: IP_src, IP_dst, TCP_src,TCP_dst

Update Scope: IP_src, IP_dst, TCP_src,TCP_dst



1) State lookup

IP_src=10.0.0.1	IP_dst=10.0.0.2	TCP_src=2500	TCP_dst=80

State Table

Flow key				State
IP_src=10.0.0.1	IP_dst=10.0.0.2	TCP_src=1000	TCP_dst=80	1
IP_src=10.0.0.1	IP_dst=10.0.0.2	TCP_src=2500	TCP_src=80	2
IP_src=10.0.0.1	IP_dst=10.0.0.2	TCP_src=3000	TCP_dst=80	3
*	*	*	*	DEFAULT

2) XFSM state transition

DEFAULT	IP_src=10.0.0.1	IP_dst=10.0.0.2	TCP_src=2500	TCP_dst=80

XFSM Table

Match fields		Actions
state	event	action
DEF.	In-port=1	Group(1)
1	In-port=1	Forward(2)
2	In-port=1	Forward(3)
3	In-port=1	Forward(4)
*	In-port=2	Forward(1)
*	In-port=3	Forward(1)
*	In-port=4	Forward(1)

Group Table

Group Entry	Buckets	
#	action	Next-state
Entry 1	Forward(2)	1
	Forward(3)	2
	Forward(4)	3

4) State update

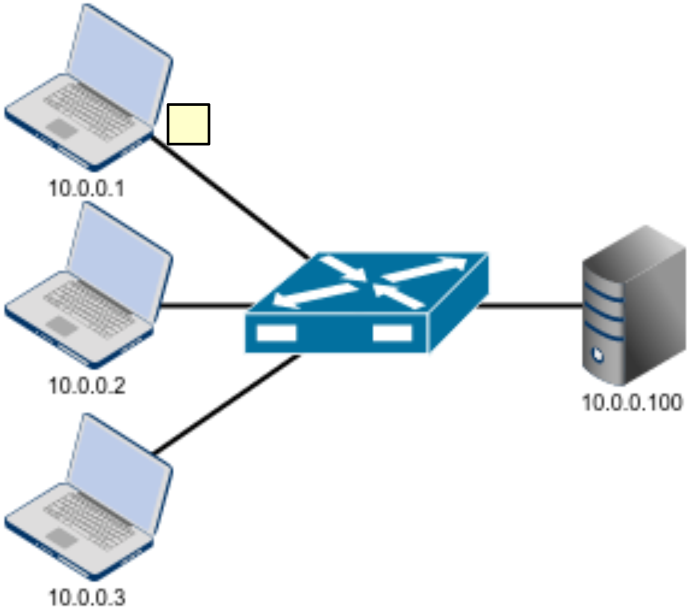
write

2

IP_src=10.0.0.1	IP_dst=10.0.0.2	TCP_src=2500	TCP_dst=80

Forwarding Consistency: **Many to One**

Forwarding consistency must be ensured according to packets received in the reverse direction.



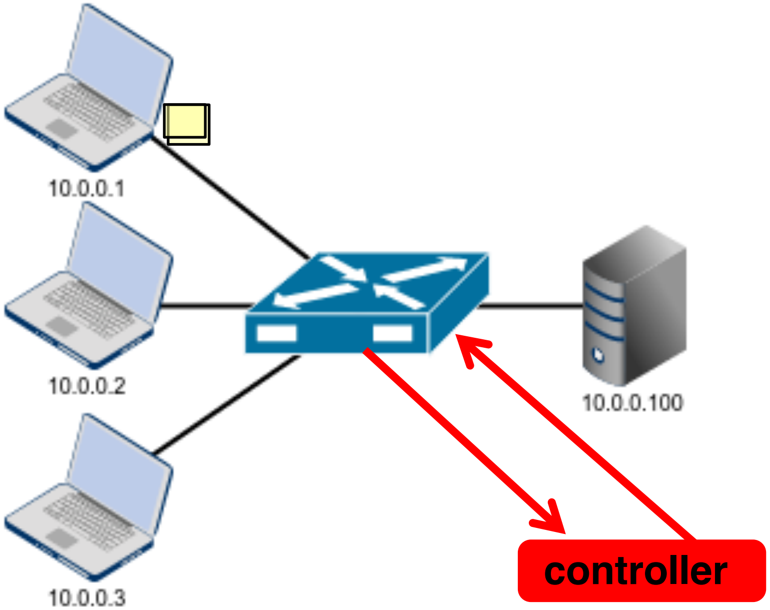
The first packet of a TCP connection coming from one of the many input ports is forwarded on the only output port.

All packets of the reverse flow of the same TCP connection must be forwarded on the same ingress port.

Forwarding Consistency: **Many to One**

OpenFlow solution:

the controller is in charge of states management.



First packet of each new TCP connection is sent to the controller in order to:

-forward the packet

-install a flow entry for reverse flow’s packets

Forwarding Consistency: **Many to One**

OpenState solution:

the switch itself handles connection's state.

A flow is identified by [IP_SRC,IP_DST,TCP_SRC,TCP_DST]
FLOW STATE: input port

First packet of each new TCP connection is forwarded and the input port is stored to forward response packets

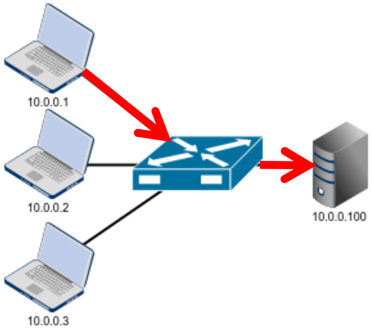
Cross-flow state
Controller is not involved!

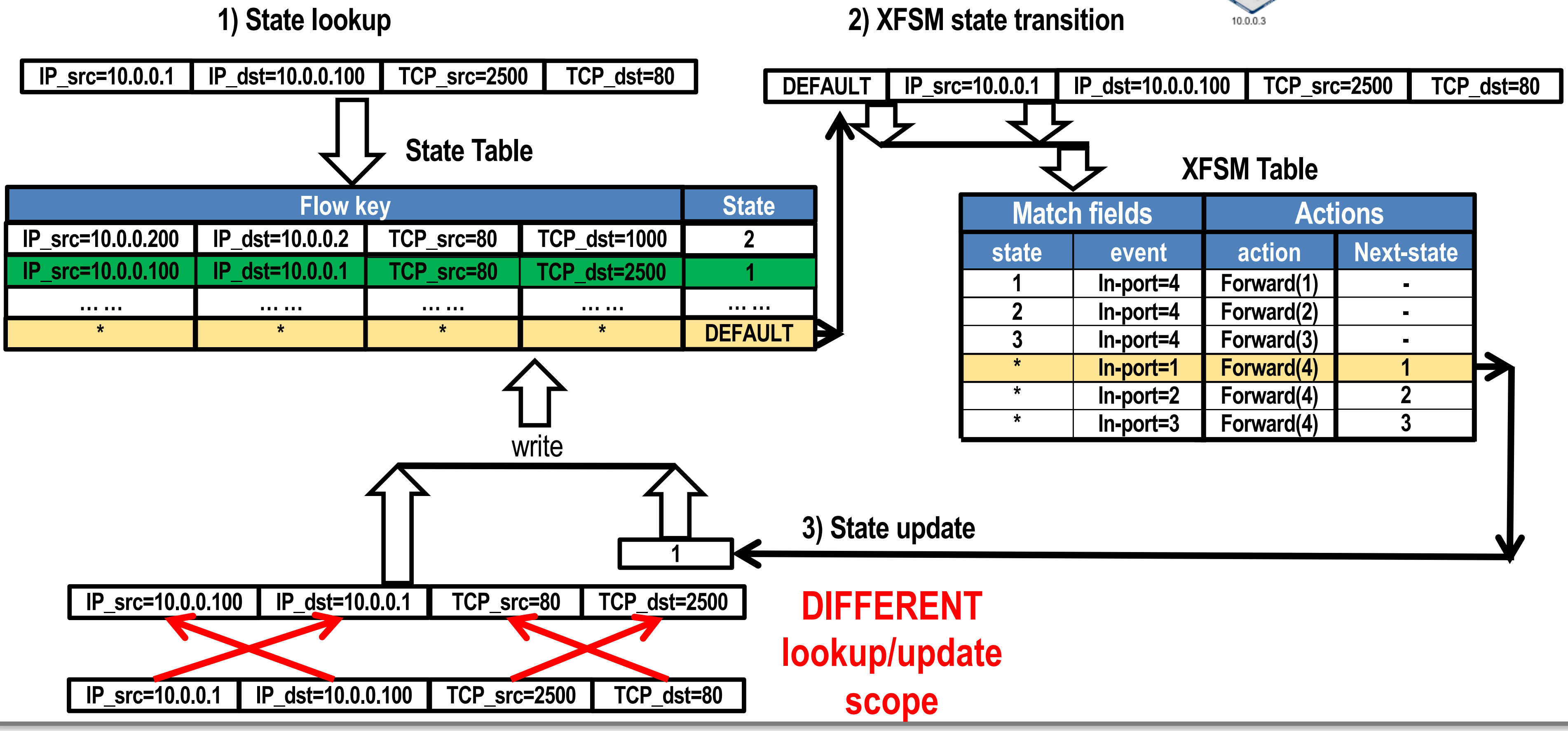
Forwarding Consistency: **Many to One**

Communication Host -> Server

Lookup Scope: IP_src, IP_dst,TCP_src,TCP_dst

Update Scope: IP_dst,IP_src,TCP_dst,TCP_src





1) State lookup

IP_src=10.0.0.1	IP_dst=10.0.0.100	TCP_src=2500	TCP_dst=80

State Table

Flow key				State
IP_src=10.0.0.200	IP_dst=10.0.0.2	TCP_src=80	TCP_dst=1000	2
IP_src=10.0.0.100	IP_dst=10.0.0.1	TCP_src=80	TCP_dst=2500	1
… …	… …	… …	… …	… …
*	*	*	*	DEFAULT

2) XFSM state transition

DEFAULT	IP_src=10.0.0.1	IP_dst=10.0.0.100	TCP_src=2500	TCP_dst=80

XFSM Table

Match fields		Actions	
state	event	action	Next-state
1	In-port=4	Forward(1)	-
2	In-port=4	Forward(2)	-
3	In-port=4	Forward(3)	-
*	In-port=1	Forward(4)	1
*	In-port=2	Forward(4)	2
*	In-port=3	Forward(4)	3

write

3) State update

1

IP_src=10.0.0.100	IP_dst=10.0.0.1	TCP_src=80	TCP_dst=2500

IP_src=10.0.0.1	IP_dst=10.0.0.100	TCP_src=2500	TCP_dst=80

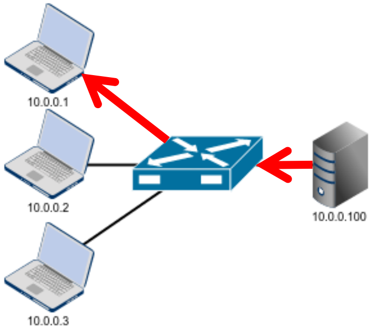
DIFFERENT lookup/update scope

Forwarding Consistency: **Many to One**

Communication Server -> Host

Lookup Scope: IP_src, IP_dst,TCP_src,TCP_dst

Update Scope: IP_dst,IP_src,TCP_dst,TCP_src



1) State lookup

IP_src=10.0.0.100	IP_dst=10.0.0.1	TCP_src=2500	TCP_dst=80

State Table

Flow key				State
IP_src=10.0.0.200	IP_dst=10.0.0.2	TCP_src=80	TCP_dst=1000	2
IP_src=10.0.0.100	IP_dst=10.0.0.1	TCP_scr=80	TCP_dst=2500	1
… …	… …	… …	… …	… …
*	*	*	*	DEFAULT

2) XFSM state transition

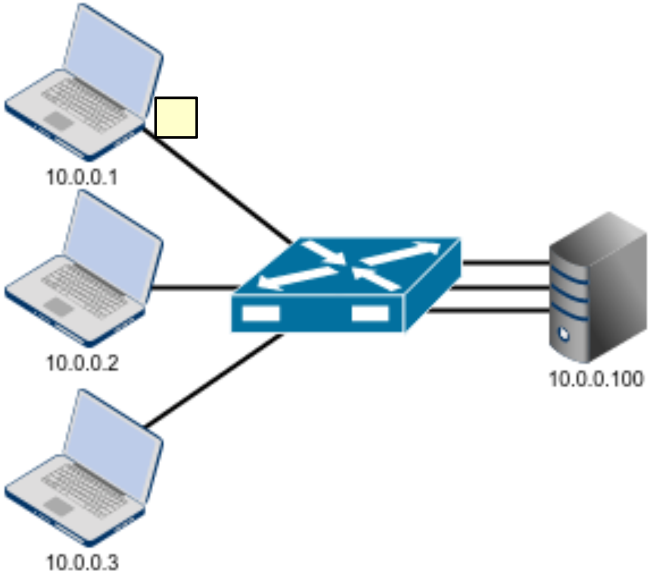
1	IP_src=10.0.0.1	IP_dst=10.0.0.100	TCP_src=2500	TCP_dst=80

XFSM Table

Match fields		Actions	
state	event	action	Next-state
1	In-port=4	Forward(1)	-
2	In-port=4	Forward(2)	-
3	In-port=4	Forward(3)	-
*	In-port=1	Forward(4)	1
*	In-port=2	Forward(4)	2
*	In-port=3	Forward(4)	3

Forwarding Consistency: **Many to Many**

Combining the first two, we want here to load balance on the output ports while doing reverse path forwarding on the input port



The first packet of a TCP connection coming from one of the many input ports is forwarded to one of many possible output ports.

All the next packets of the same TCP connection must be forwarded on the same selected output port, while all packets of the reverse flow of the same TCP connection must be forwarded on the same ingress port.

Forwarding Consistency: **Many to Many**

OpenState solution

A flow is identified by [IP_SRC,IP_DST,TCP_SRC,TCP_DST]

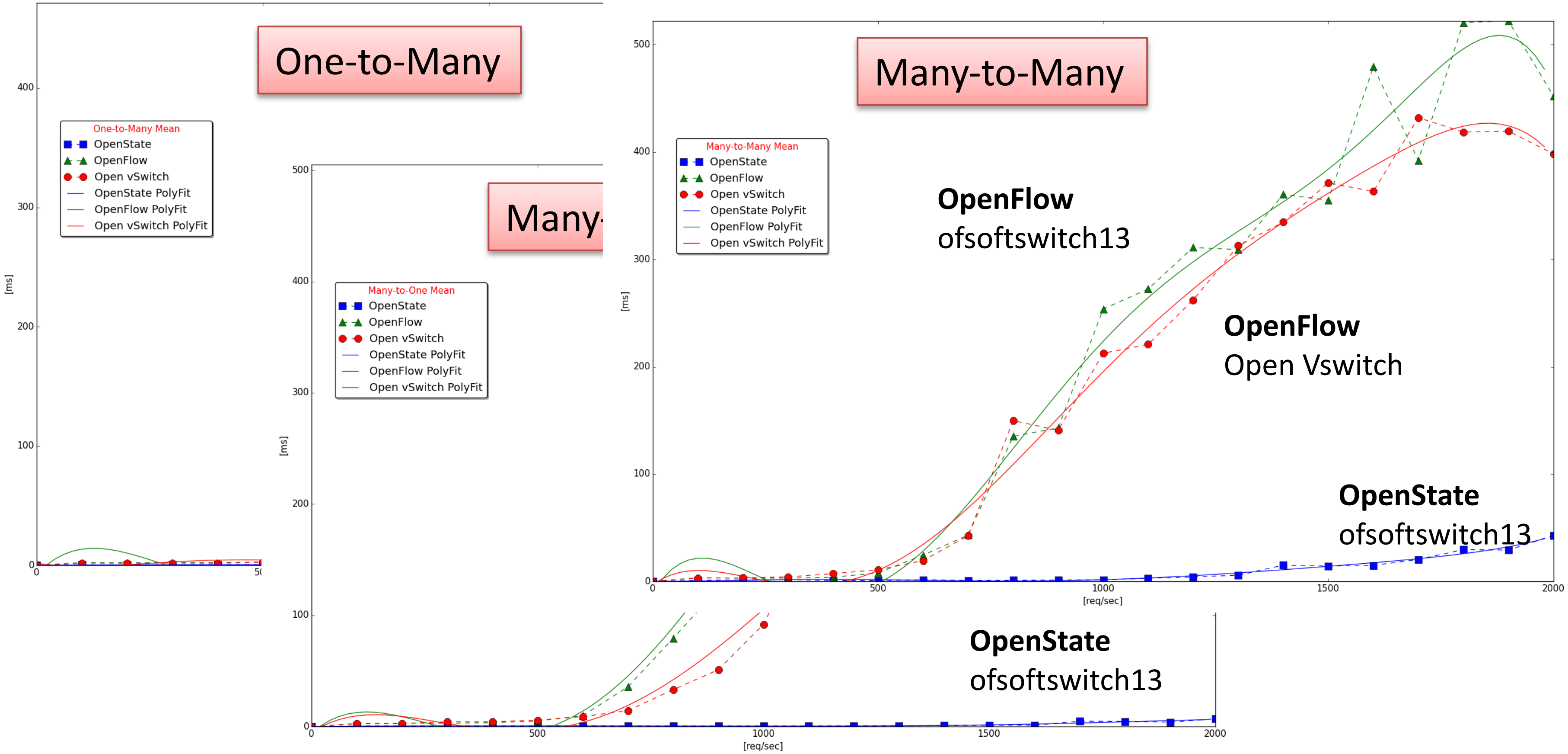
Two states are needed for each bidirectional flow:
FLOW STATE 1: output port
FLOW STATE 2: input port

For each first packet of each new TCP connection:

- packet is forwarded to a random output port
- the selected output port is stored in the State Table 0
- the input port is stored in the State Table 1

Forwarding Consistency: **Example Results**

Results will show the average value of the
time required by 1000 **TCP SYN packets** to cross the switches at increasing rate.



One-to-Many

Many-to-Many

OpenFlow
ofsoftswitch13

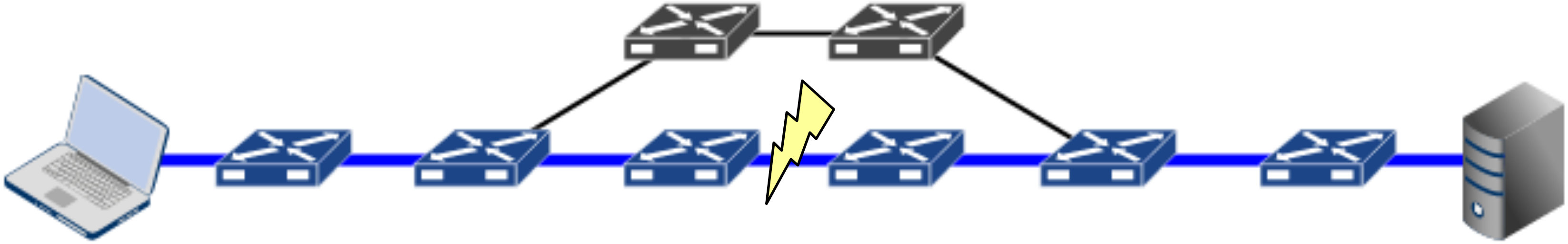
OpenFlow
Open Vswitch

OpenState
ofsoftswitch13

OpenState
ofsoftswitch13

Switch: **ofsoftswitch13**; Controller: **Ryu**

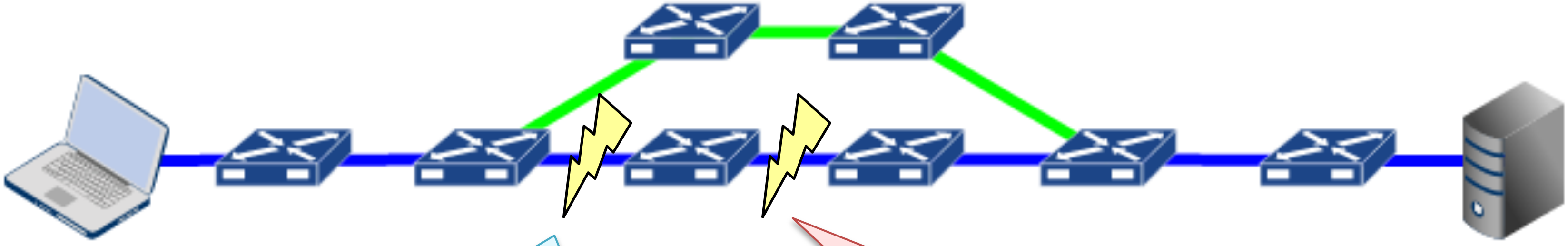
Fault Tolerance



- Ensure the network failure resiliency, quickly readapting the routing after a failure
- Fundamental function in any network (telco operators, data centers)
- Weak support in OpenFlow

Fault Tolerance

- OpenFlow

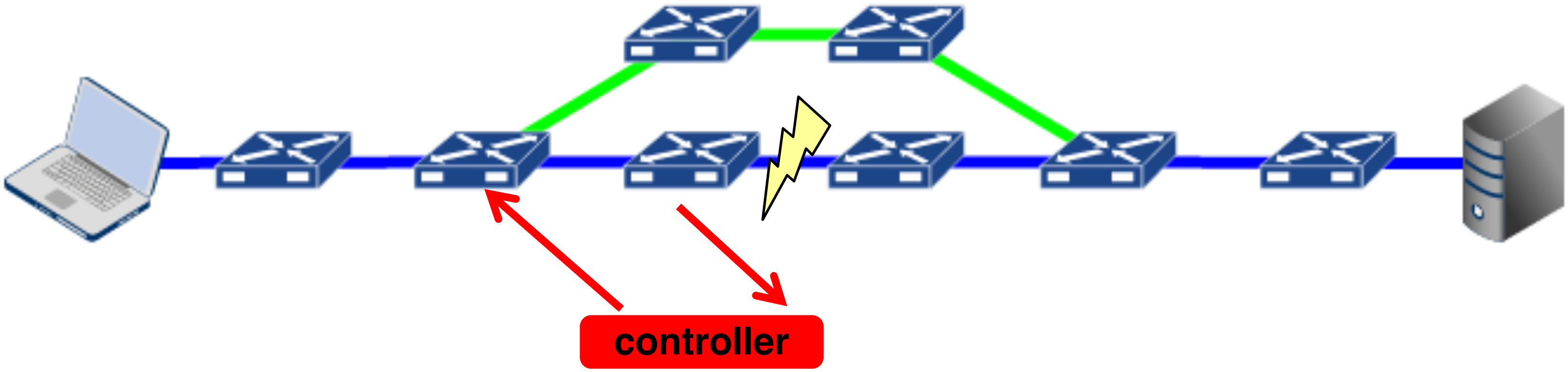


Fast failover: local reroute based on port states (Group table)

But what if a local reroute in not available ???

Fault Tolerance

- OpenFlow

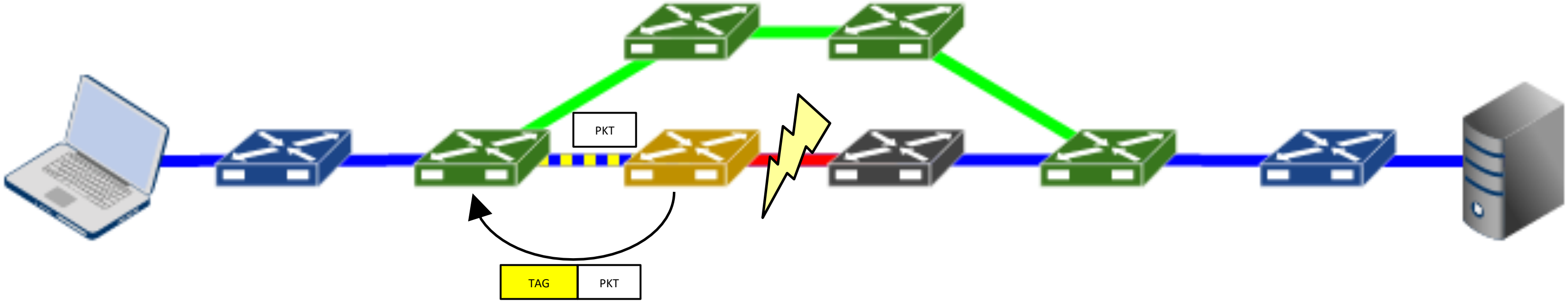


Obviously it is always possible to rely on the controller to:

- forward the packet on the backup path
- install flow entries for the backup path

Fault Tolerance in OpenState

With **OpenState** the switch itself can react to the fault



Proposed solution:

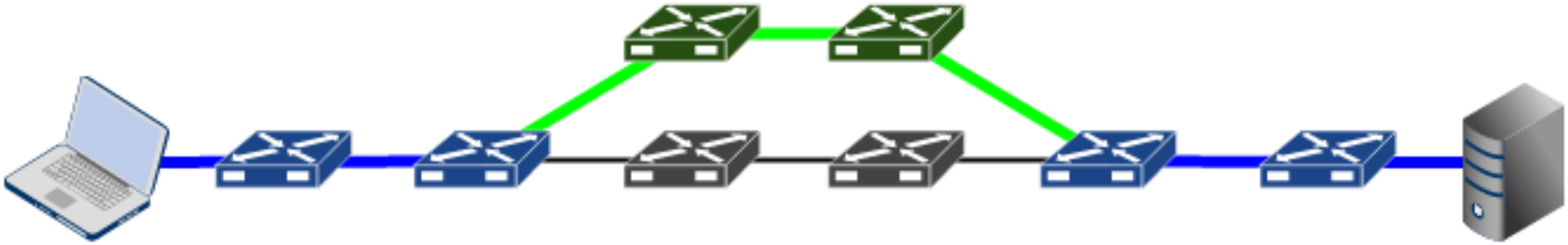
➔ Faults are signaled using the same data packets

➔ Packets are tagged and sent back

➔ Packets are sent back until matched against a node able to respond to that fault

Fault Tolerance

OpenState

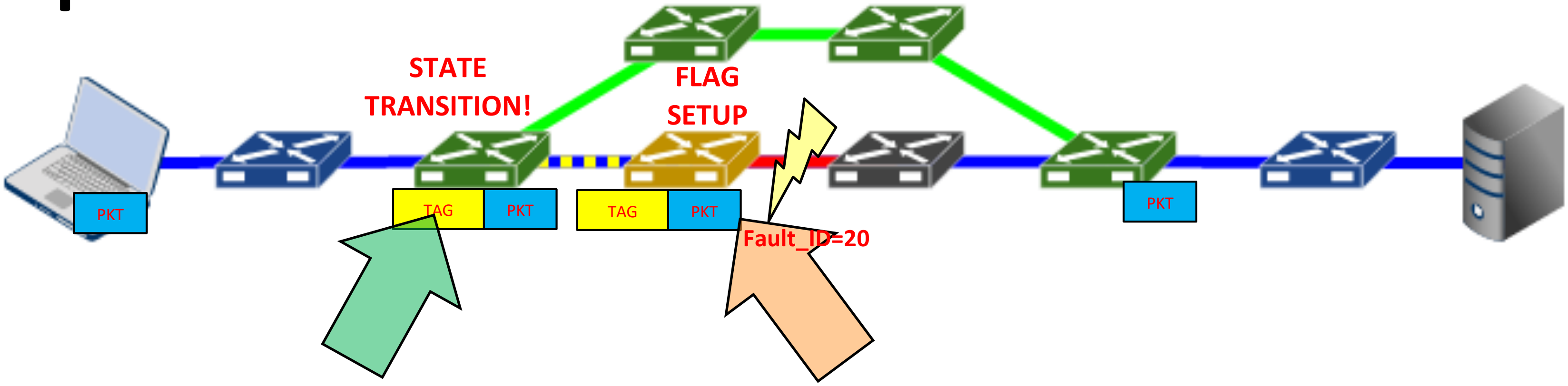


- A DETOUR is enabled based on the specific failure without constraints
- Backup paths can be pre-computed and installed by the controller (traffic engineering and quality/congestion control)
- The controller is entitled to restore the primary path once the fault has been resolved

[CAP14] A. Capone, C. Cascone, A. Nguyen, B. Sansò, “Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState”, available on ArXiv, Nov. 2014.

Fault Tolerance: Fault Reaction Example

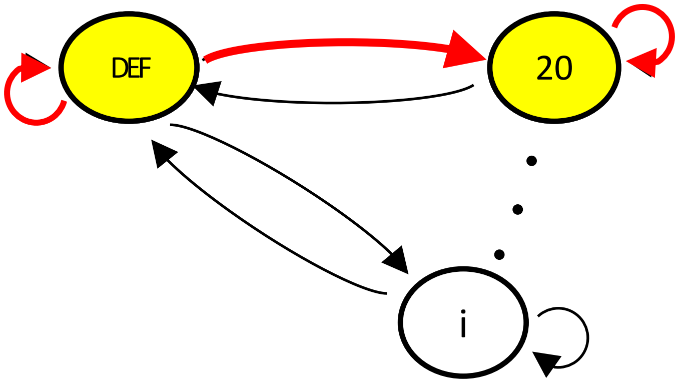
OpenState



Redirect Node:

FLOW STATE = DEF 20

TAG = STATE = 20

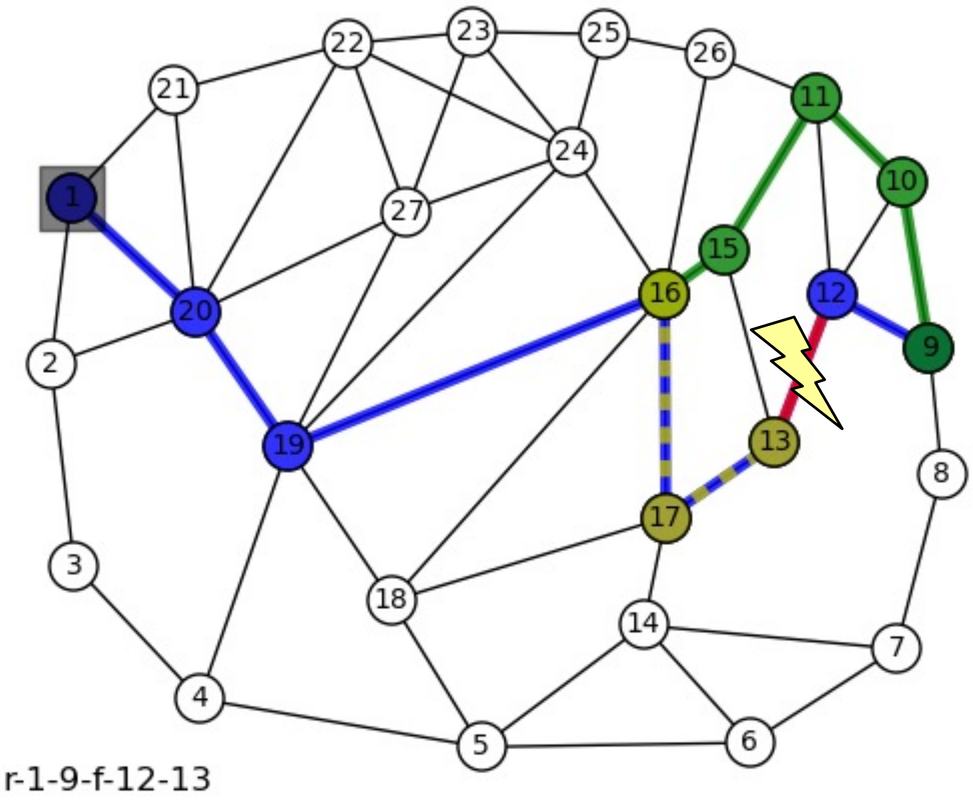


Detect Node:

GLOBAL REGISTERS = 00 01

TAG = FAULT_ID = 20

Fault Tolerance: Example on larger network



19 **Primary node**

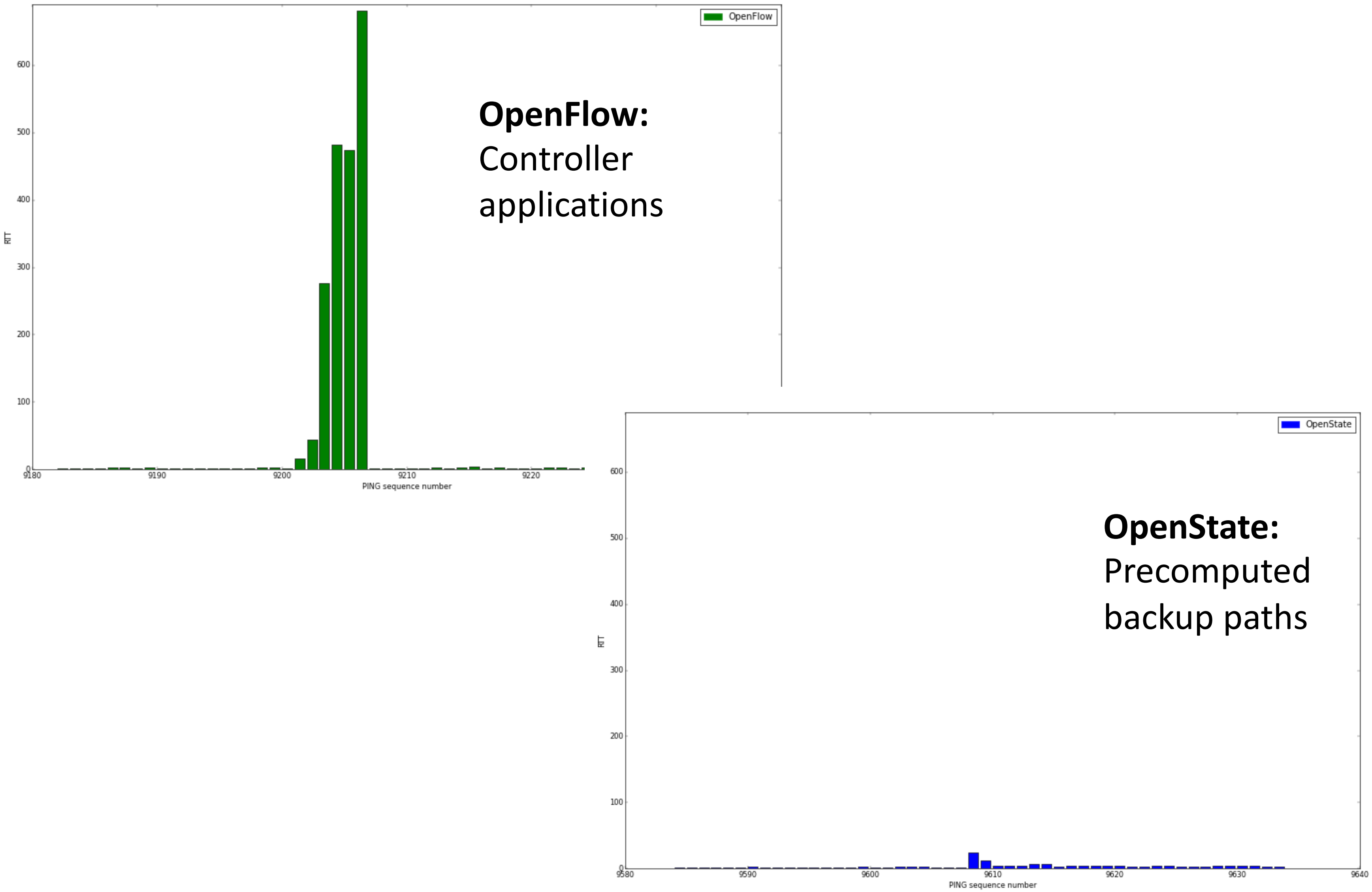
13 **Detect node**

17 **Forward-back node**

16 **Redirect node**

15 **Detour node**

Fault Tolerance: Exampe Results



OpenFlow:
Controller
applications

OpenState:
Precomputed
backup paths

Thanks

Giuseppe Bianchi

Email: giuseppe.bianchi@uniroma2.it

Antonio Capone

Email: antonio.capone@polimi.it

OpenState: openstate-sdn.org

EU project BEBA (web site available soon) – follow us!

This slide-set soon available on OpenState web site!