SDN +

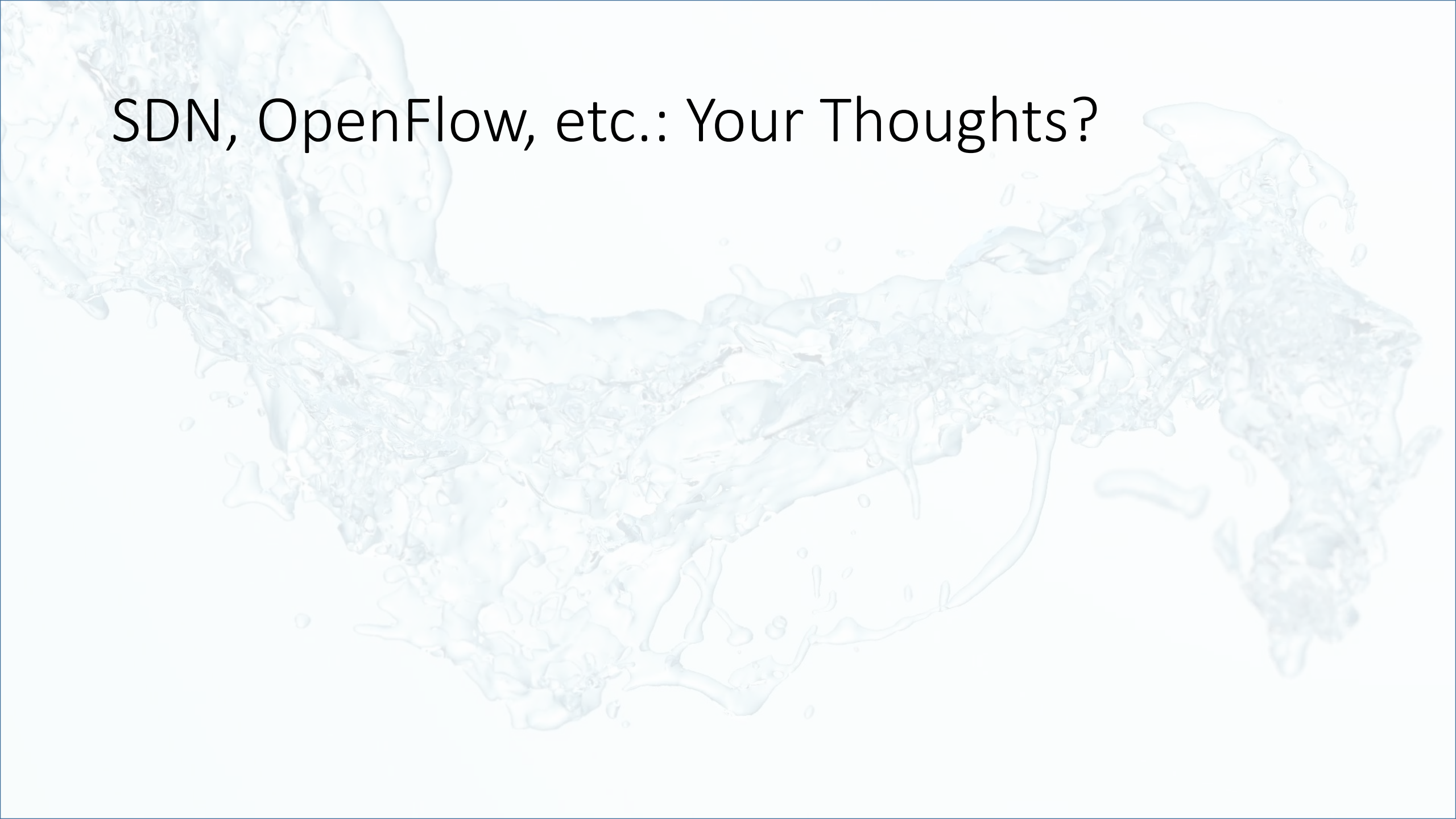
Gabriel Scalosub

Borrowed extensively from:

Jen Rexford, Nick McKeown, Scott Shenker, Raj Jain, Elisha Rosensweig, David Mahler, Vikrant Aggarwal, Brad Hedlund, Changhoon Kim

and various other papers/resources (see list at the end)

SDN, OpenFlow, etc.: Your Thoughts?



Outline

- Software Defined Networks (SDN)
- OpenFlow (OF)
- Open vSwitch (OVS)
- Mininet
- P4
- Virtual Extensible LAN (VXLAN)

Outline

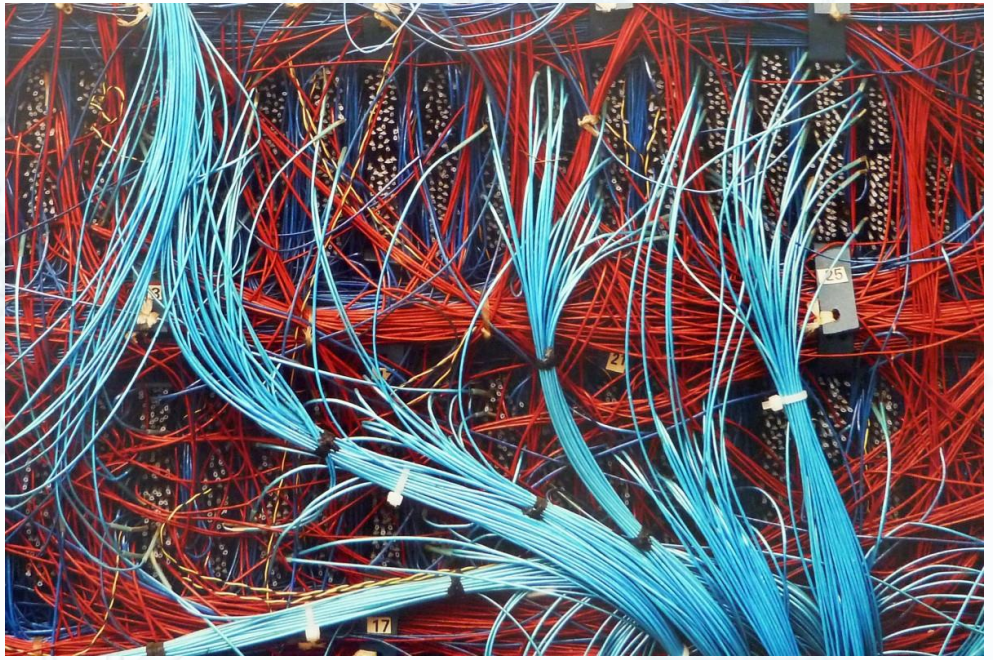
- Software Defined Networks (SDN)
- OpenFlow (OF)
- Open vSwitch (OVS)
- Mininet
- P4
- Virtual Extensible LAN (VXLAN)

SDN: Some Context

- 2000s
  - Internet is BIG
  - Many services / applications
    - Web, search, social networks, P2P, VOIP, …
- Meanwhile inside the network:
  - Proprietary HW-SW bundles, vendor-specific
  - Miniscule standardization
  - Very slow/expensive innovation
    - Performance, security, reliability, etc.
  - Buggy SW
    - Routers \w 20+ million lines of code
    - Failures, vulnerabilities
- Datacenters on the rise

SDN: Some Context

- People start to think:
  - These networks are quite a mess…
- A quest for better abstraction
  - Networks in general
  - Control/Management specifically
  - OSI was a good model (if not perfect)



OSI model

Layer	Name	Example protocols
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

SDN: What's It All About?

- How can we improve the architecture of our networks?
  - Let's draw intuition from computing!
- Separating the control plane from the data plane
- A (useful) abstraction of network functionality
- Not
  - Technology, SW, HW
- Yes
  - Paradigm, philosophy, approach

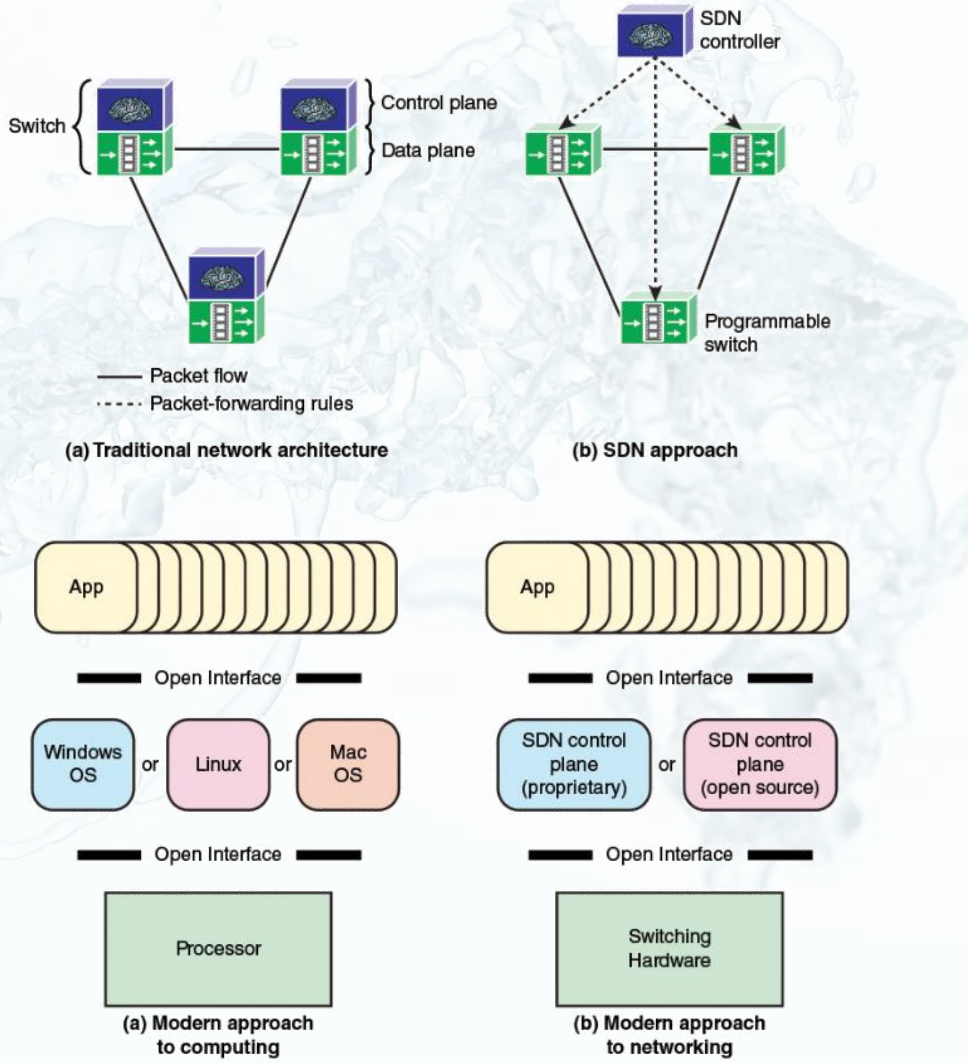


Image: Stallings (2016)

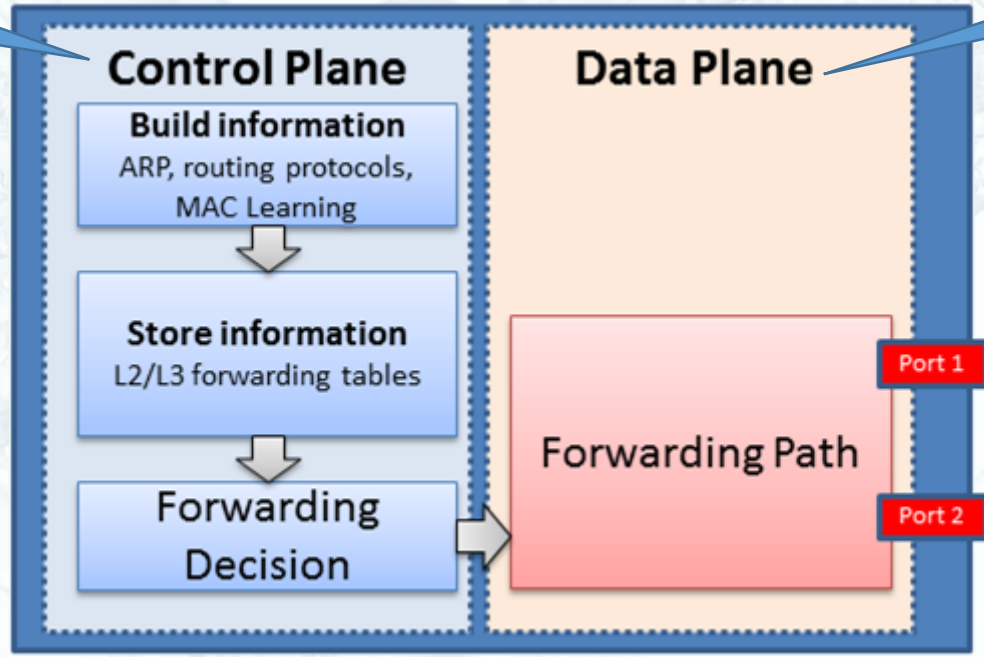
SDN: What Is It Good For?

- Network virtualization
  - Overlays, tunneling
- Managing ever-increasing network complexity
  - Routing, middleboxes, VPNs
- Vendor independence
- Catalyst of innovation on the ***network level***
  - Improved performance, agility, fault-tolerance, security, mobility support
  - Especially in datacenters
- An example: OpenPipes
  - https://www.youtube.com/watch?v=XWsV3ONfNyo

Traditional Switch Architecture

Management,
Distributed & Robust

- Resource Management
- Routing (protocols, state)
  - OSPF, RIP
  - BGP + policies
- ARP
- Vendor specific



Fast-Path
HW-accelerated

- Packet handling
  - Forwarding, en/de-capsulation, en/de-cryption,
- Quality of service (QoS)
  - Queuing, policing, shaping, dropping
- Access control
  - Maintain log, count

Image: http://bradhedlund.com

Separating Switch Control from Datapath

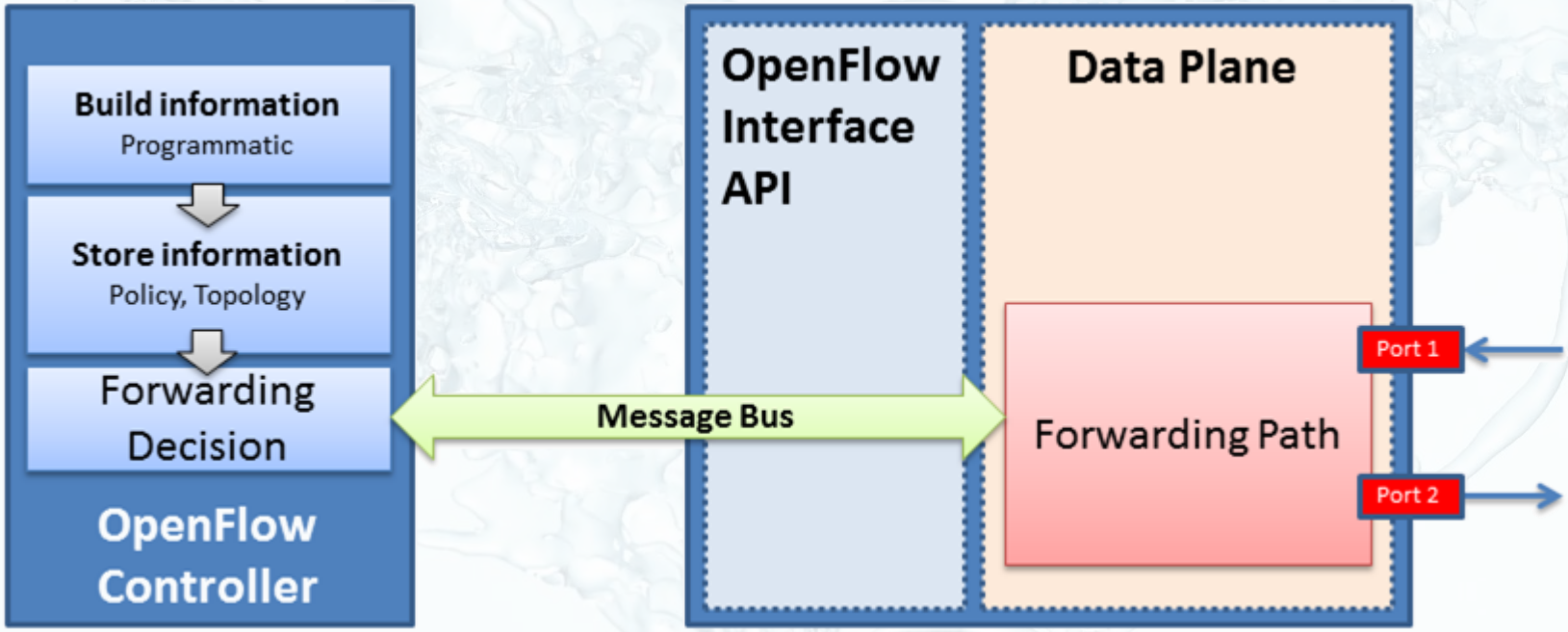


Image: http://bradhedlund.com

Metaview: Another Layer of Abstraction

- Abstraction for the data plane: OSI layers
- Abstraction for the control plane: ???
  - A bunch of protocols…
- SDN:
  - Providing a useful abstraction of the control plane
  - Network OS
- Mimicking the OS/HW separation in modern computing

OSI model

Layer	Name	Example protocols
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

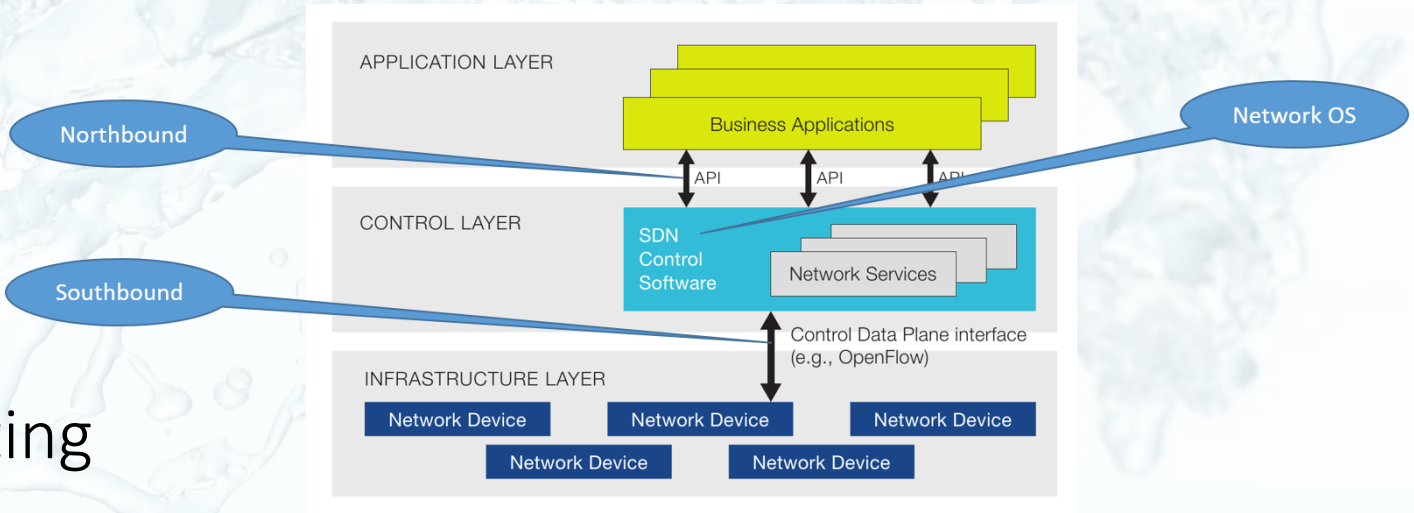
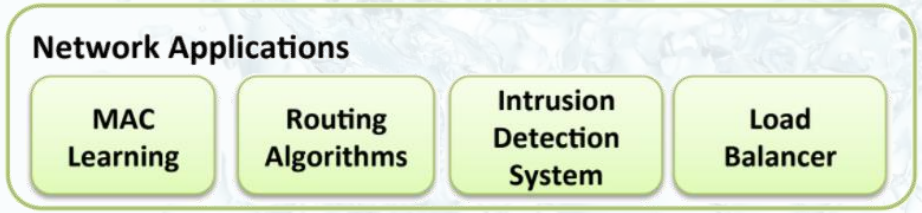


Image: https://www.opennetworking.org/

SDN: Supporting Innovative Applications

- Each network application gets its own view of the network
- Examples:



  - Topology abstraction
- Programmable API
  - Northbound..

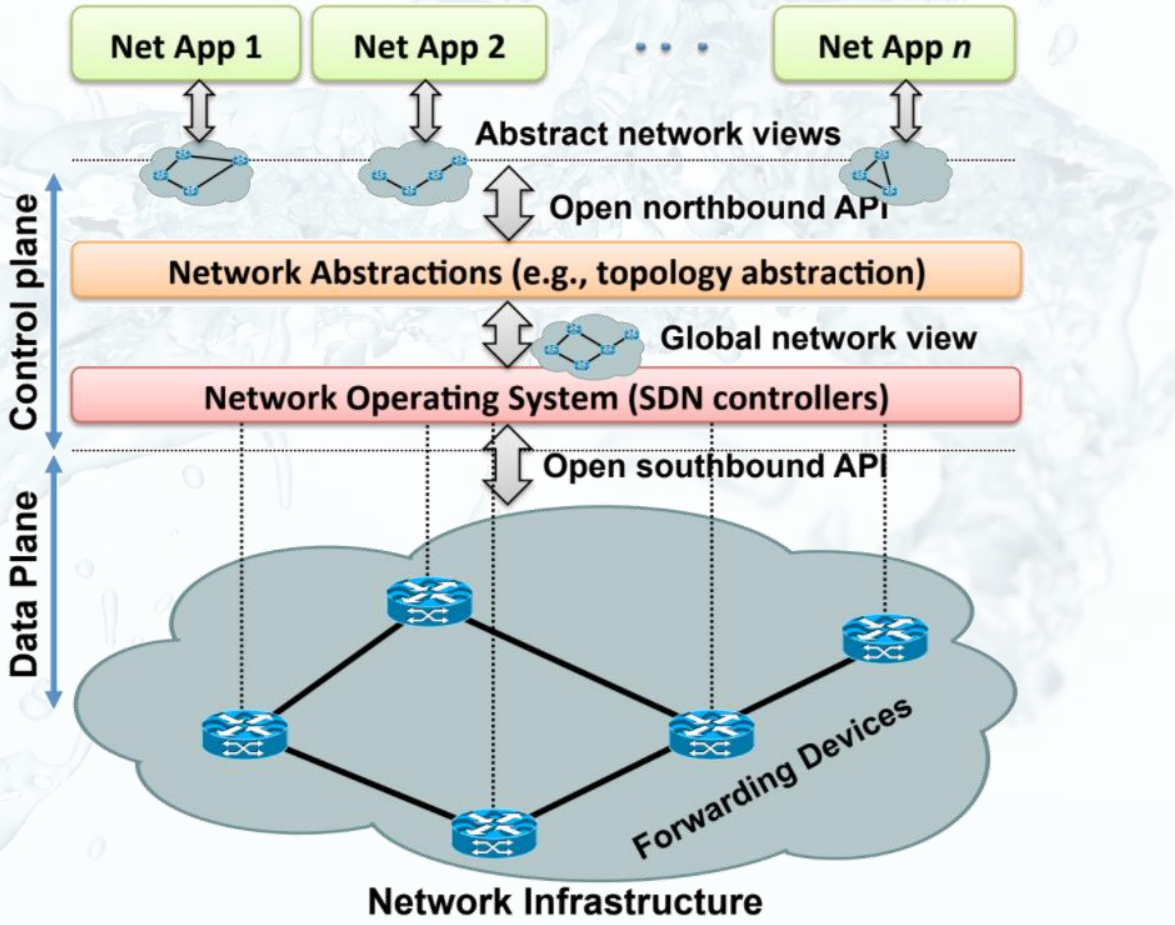


Image: Kreutz et al. (2015)

SDN: A Unified Framework

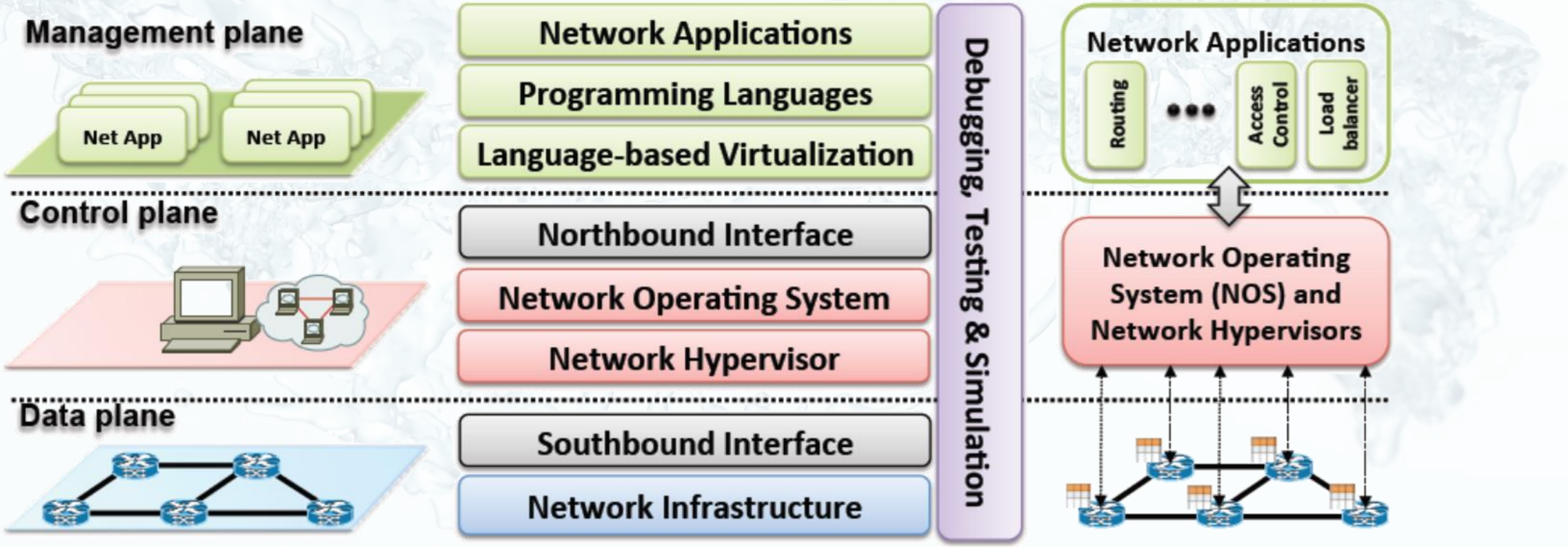
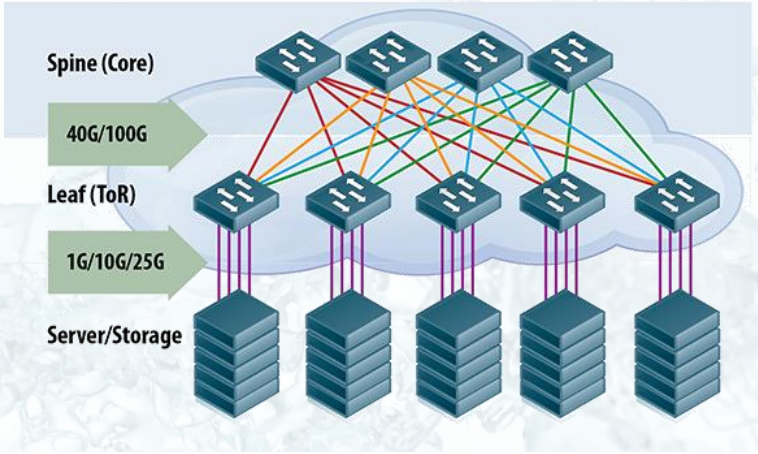


Image: Kreutz et al. (2015)

Who Cares (Most)? Datacenters!

- Schematics:
  - Spine (core), Leaf (ToR) switches
  - Server racks
- An example of some concrete numbers:
  - 1024 spine switches
  - 2048 leaf switches
  - 40 servers/rack
  - Now add VMs
  - Serious amount of dynamic links and flows
    - Congestion, failures, …
- A real challenge to manage (without SDN)



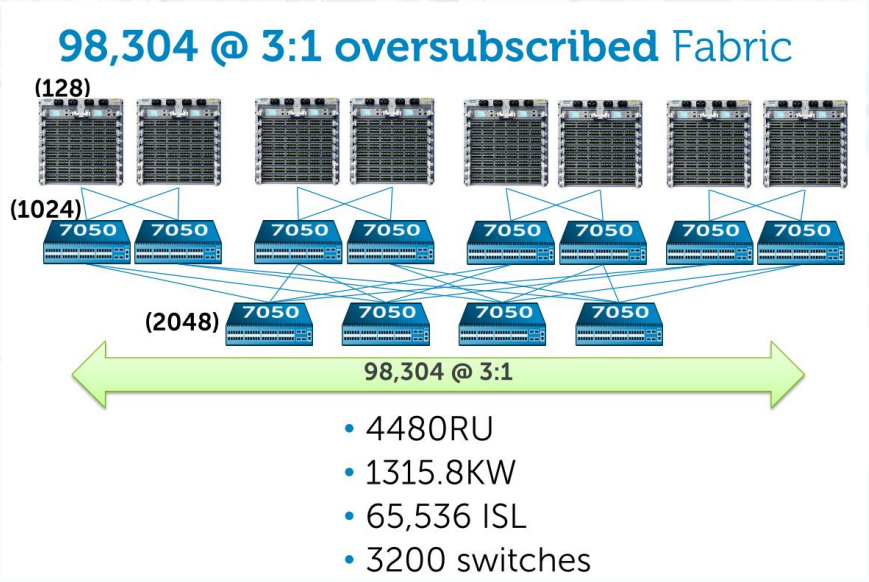


Image: Hedlund (2012)

A Simple Example: Ping

- Initially: S1 flow table empty
- H1 pings H2
  - Ping message arrives at S1
  - S1 doesn't have an entry to reach H2
  - S1 passes ping message to C0
  - C0 adds entry to S1 flow table (returns ping message)
  - S1 forwards ping message to H2
- H1 pings H2 again
  - S1 has flow entry
  - Forwards ping message directly to H2
    - Doesn't go through C0

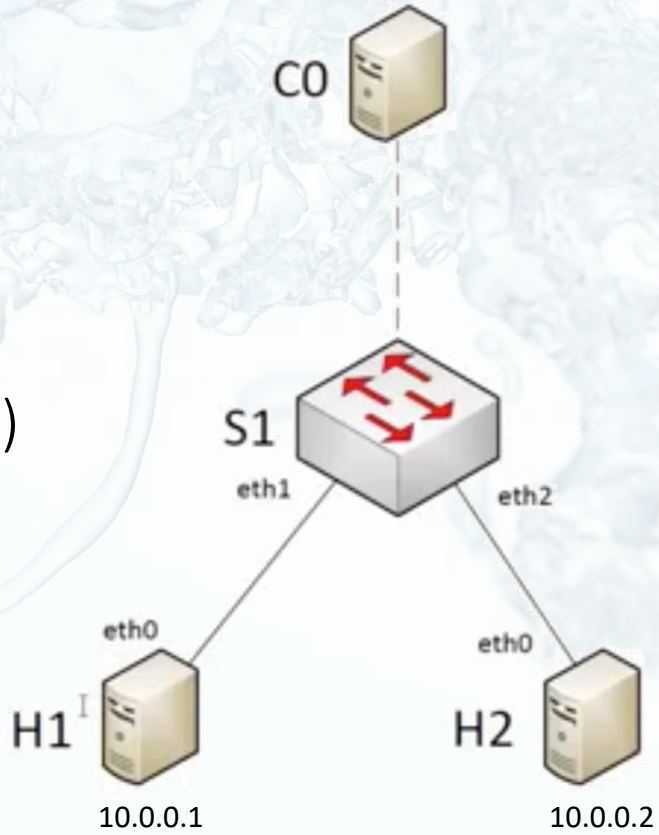


Image: Mahler (2013)

SDN: Not All Rosy Pink

- Robustness & Scalability
  - Response time
  - Controller failure and recovery
    - No distributed control protocol
  - Distributed controllers
    - Maintain consistency
- Interoperability
  - APIs, vendor support

Outline

- Software Defined Networks (SDN)
- OpenFlow (OF)
- Open vSwitch (OVS)
- Mininet
- P4
- Virtual Extensible LAN (VXLAN)

OpenFlow: A Key Component

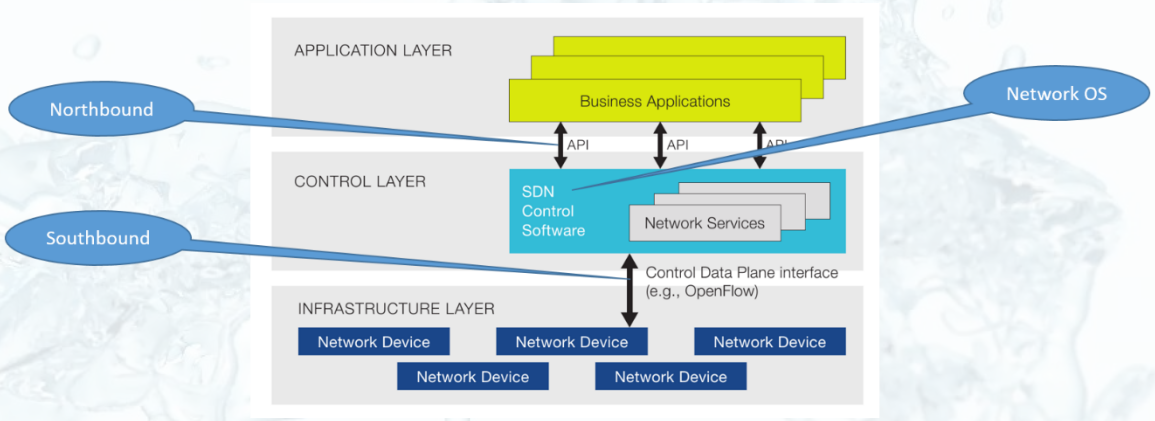


Image: https://www.opennetworking.org/

- OpenFlow ≠ SDN!!
- OF gives a way to actually implement the concept of SDN
- Protocol developed for "connecting" control & data planes
  - Southbound…
- De-facto standard today
  - Although there were/are other alternatives
- Defines requirements and capabilities of an OF-enabled network/switch
- Takes into account that:
  - Data plane: fast, dumb
    - Uses specialized HW & Logic (e.g., TCAM – Ternary Content Addressable Memory)
  - Control plane: slow, smart

OpenFlow: Requirements and Switch Types

- OpenFlow-supporting switch should:
  - Have a secure channel to the controller
  - Be able to manage its ports (state)
  - Maintain a flow table
- Protocol:
  - TLS channel: TCP/6633 or TCP/6653
  - Limited set of messages
- Switch Types:
  - Pure OpenFlow: runs OpenFlow only! All decisions rely on controller
  - Hybrid OpenFlow: supports OpenFlow, but also other legacy protocols
    - Most commercially available switches are hybrid…

OpenFlow: Flow Tables Basics (v1.0)

- Header-match -> Action

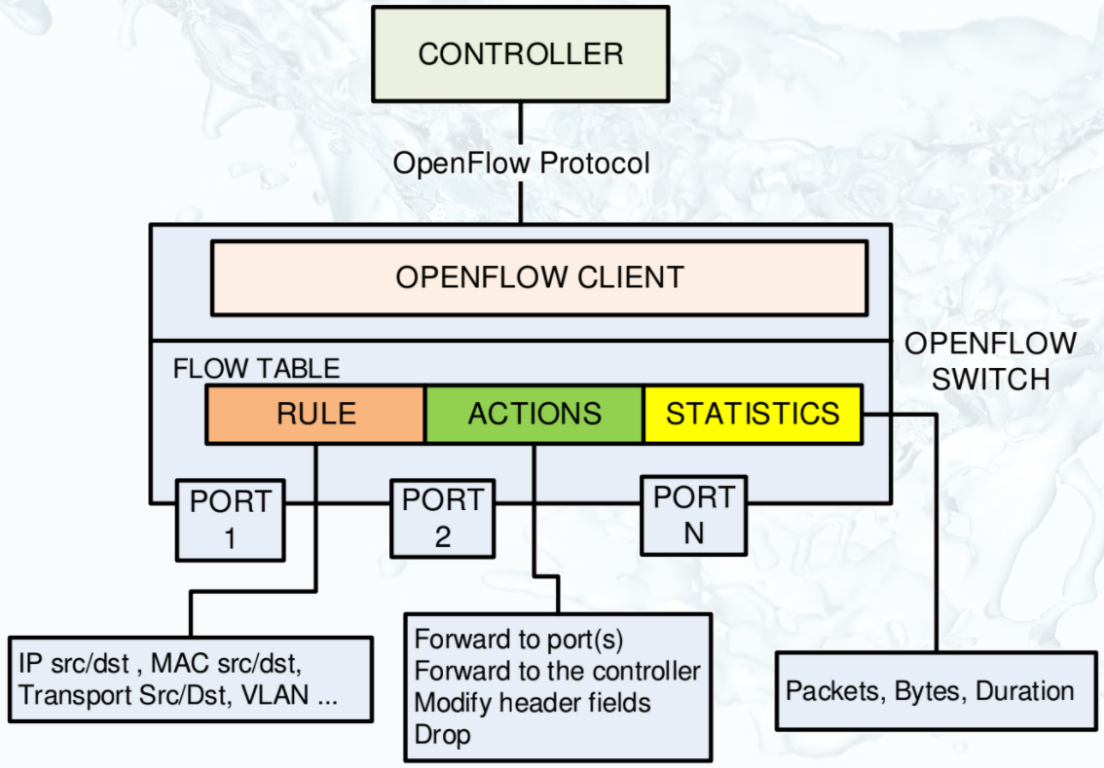


Image: Nunes et al. (2014)

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall

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

Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

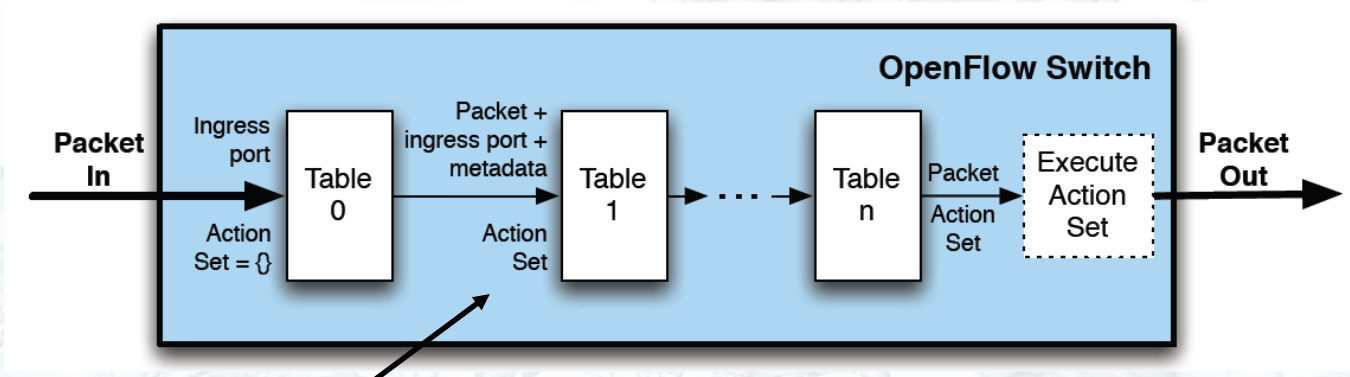
VLAN Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f..	*	vlan1	*	*	*	*	*	port6, port7, port9

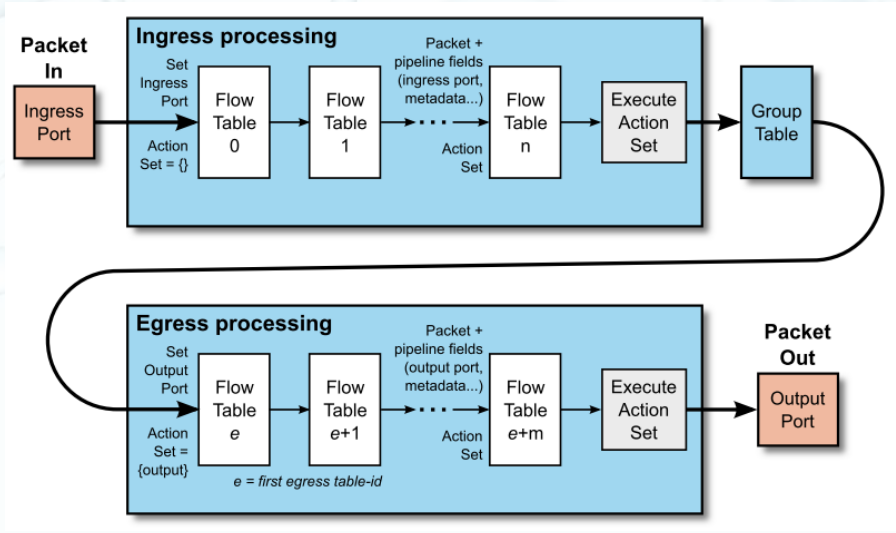
Image: Seetharaman (2012)

OpenFlow: Pipelines and Additional Properties

- Every packet arriving:
  - Checked against Table0
  - No match: Drop
  - Match:
    - Update counters for matched rule
    - Perform action
- Some actions:
  - Forward to controller
  - Forward via specified port
  - Forward to next flow table
  - Modify fields: VLAN ID, IP (think NAT!)
  - Drop packet



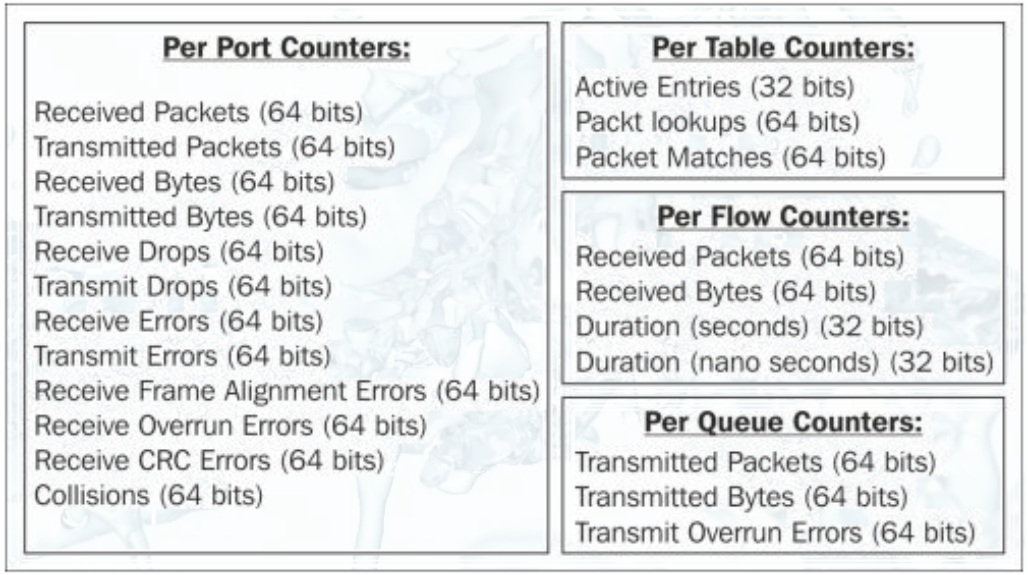
OpenFlow v1.3



OpenFlow v1.5

OpenFlow: Pipelines and Additional Properties

- Every packet arriving:
  - Checked against Table0
  - No match: Drop
  - Match:
    - Update counters for matched rule
    - Perform action
- Some actions:
  - Forward to controller
  - Forward via specified port
  - Forward to next flow table
  - Modify fields: VLAN ID, IP (think NAT!)
  - Drop packet



- Soft states:
  - Rules have (possible) timeouts
- Counters & Statistics
  - Queried by controller
    - Proactive/reactive

OpenFlow: Versions

- Mainly support additional matches / protocols
  - Allows for more complex rules

OF Version	Match (additional) Fields
V1.0	[previous slides…]
V1.1	Metadata, VLAN Tagging, MPLS label, MPLS traffic class, …
V1.2	OXM, IPv6 (src, dst, flow label, ICMPv6), …
V1.3	IPv6 extension headers, …
V1.4 - V1.5	Optical ports, table synchronization, output-port pipeline, …

OF Extensible Match: User defined

combine MAC learning of source/dest

OpenFlow: Controller-Switch Messages

- Three message types
  - Controller-to-switch
    - **Initiated by the controller** to manage or inspect the state of the switch
  - Symmetric
    - **Initiated by either the controller or the switch** to communicate with the other
  - Asynchronous
    - **Initiated by the switch** to update the controller on network events or changes to switch state

OpenFlow: Controller-to-Switch Messages

- *features* (on turning on the switch)
  - Controller requests list of supported features at switch
- *modify-state*
  - Modify rules/state at the switch
- *read-state*
  - Get state (e.g., counters info)
- *send-packet*
  - Instruction to send a specific packet out a specific switch port
- *barrier*
  - Controller sends message to switch.
  - Switch completes all previously received messages and sends a barrier reply.
  - Only after barrier reply will new messages be processed.
  - Allows controller to sync the switch state

OpenFlow: Symmetric Messages

- *hello*
  - Connection setup messages
- *echo*
  - Request and reply
  - Can be used to measure properties of switch/controller connection
  - E.g., latency, liveness
- *vendor*
  - Vendor-specific messages
  - Configured by the vendor per need

OpenFlow: Asynchronous Messages

- *packet-in*
  - Notification of the arrival of a packet that has send-to-controller flow-rule
    - Usually there is a default rule stating this for all packets with no specific flow-rule
  - Request for instructions from controller
- *flow-removal*
  - Updates on flow-rules being removed
  - Via timeout or direct commands from controller
- *port-status*
  - Updates on port status (e.g., port disabled)
- *error*
  - Notify controller about errors

Outline

- Software Defined Networks (SDN)
- OpenFlow (OF)
- Open vSwitch (OVS)
- Mininet
- P4
- Virtual Extensible LAN (VXLAN)

Open v(irtual)Switch

- Essentially, a SW network MUX
- Open source
  - Modular
  - Extendible
- Works on Linux-based hypervisors
  - Xen, KVM
- Many contributors

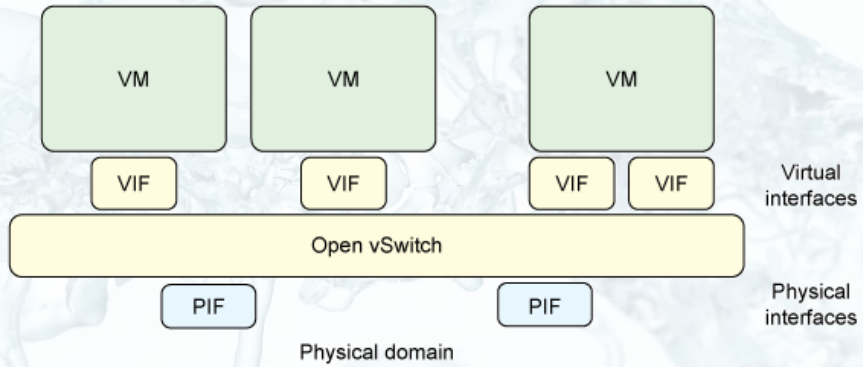


Image: Pettit and Gross (2011)

OVS: High Level View of Components

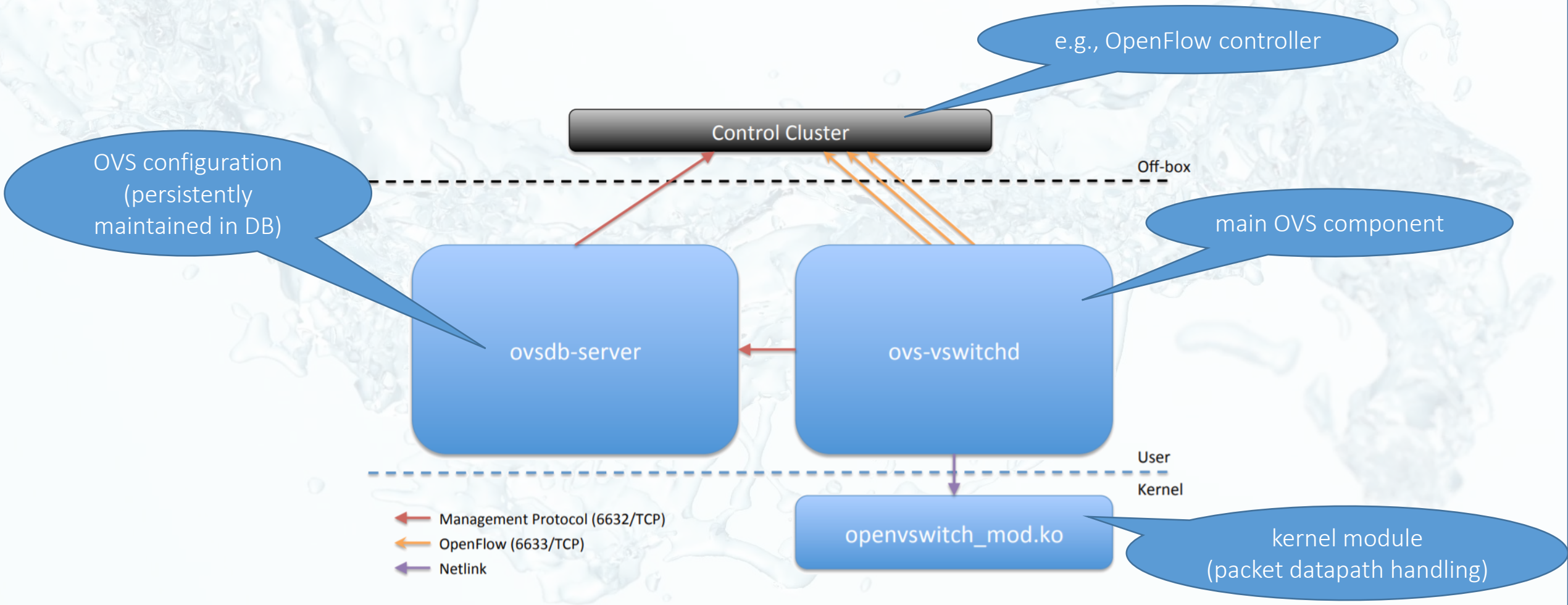


Image: Pettit and Gross (2011)

OVS: Architecture

- Works on a single host
- Follows the concept of SDN
  - Flow table in datapath (kernel)
    - Cached entries matched are handled fast!
  - No-match is reverted to vswitchd (user)
  - (OpenFlow optionally consulted)
  - Flow tables updated for future traffic
- Table match lookup:
  - L2-L4

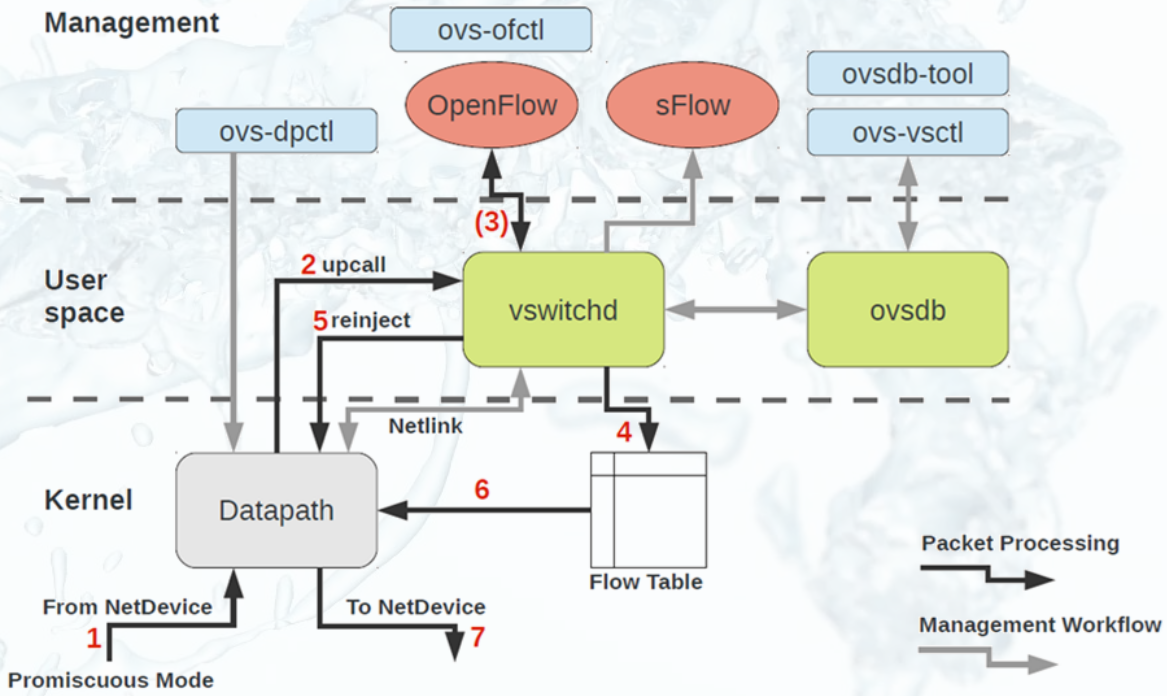


Image: Graf (2013)

OVS: Properties and Features

- Monitoring inter-VM communications
- VLAN tagging and 802.1q trunking
- Tunneling
  - VXLAN, GRE, IPsec
- Fine grained QoS
- IPv6
- OpenFlow & multi-table pipeline

OVS: CLI Commands

- *ovs-vsctl*
  - Management of the ovsdb which maintains the switch states and configuration
  - By this, configures the switch
  - Examples
    - *show*: displays status
    - *dump-ports-desc sX*: show port numbers of switch sX as they will appear in flowtables
- *ovs-ofctl*
  - Querying and controlling OpenFlow behavior
  - Examples
    - *dump-flows*: shows flowtable
    - *add-flow sX [match],actions=[action]*: adds a flow to switch sX with match-action specification
    - *del-flows sX --strict [match]*: delete flow entry for match
- *ovs-appctl*
  - Querying and controlling the ovs daemon
  - Useful for logging/monitoring

Outline

- Software Defined Networks (SDN)
- OpenFlow
- Open vSwitch (OVS)
- Mininet
- P4
- Virtual Extensible LAN (VXLAN)

Mininet

- Network emulator
  - Not simulator!!
    - Real packets going through, not just traces
    - E.g., can be monitored by Wireshark
- Run virtual networks on single host
  - Switches, hosts, controllers, applications
- Learning/testing/prototyping
  - Supports OpenFlow, various controllers
  - Python interpreter built-in
- Easiest setup:
  - mininet-vm over VirtualBox

Mininet: Additional Features

- Create custom topologies
- Run real programs
  - E.g., anything that's available on Linux (servers, client, wireshark, …)
- Customize packet forwarding
  - E.g., via OpenFlow / OVS
- Simple / Complex scenarios
  - CLI / Python scripts / P4…
- Active open source project
  - Support, forums, etc.

Mininet: Some Noteworthy Limitations

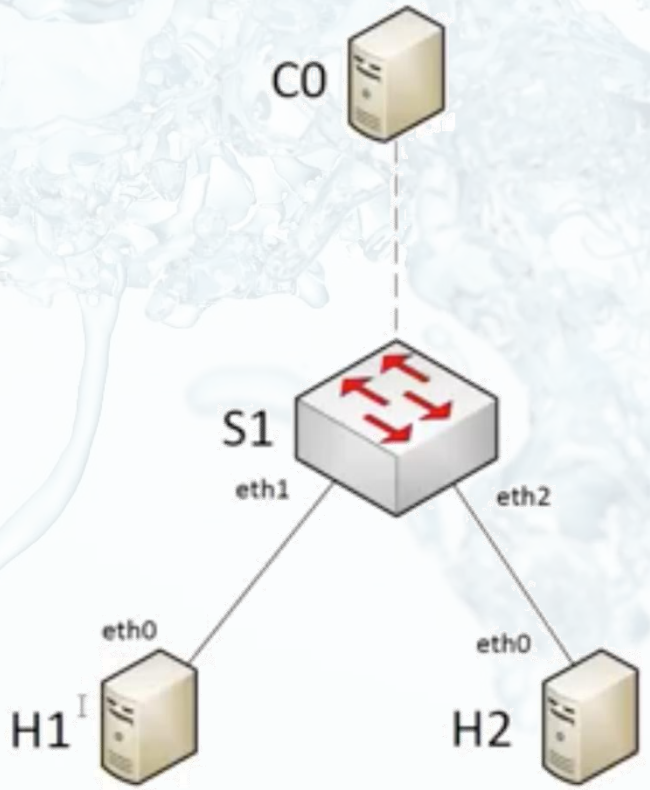
- Single host restrictions
  - Host resources (most notably CPU/RAM) shared among virtual hosts/switches
- Linux-based
  - SW for other OS cannot be run (at least not easily… VMs are possible)
- Isolated virtual network
  - Can be connected to Internet via NAT
- Virtual hosts/switches are processes on host
  - Be careful with PIDs, file system, etc.

Mininet: Simple CLI

```
Ubuntu 14.04.4 LTS mininet-vm tty1

mininet-vm login: mininet
Password:
Last login: Sat Apr  7 16:37:37 PDT 2018 on tty1
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic i686)

 * Documentation:  https://help.ubuntu.com/
mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```



Mininet: Programming

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)

def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```

Code

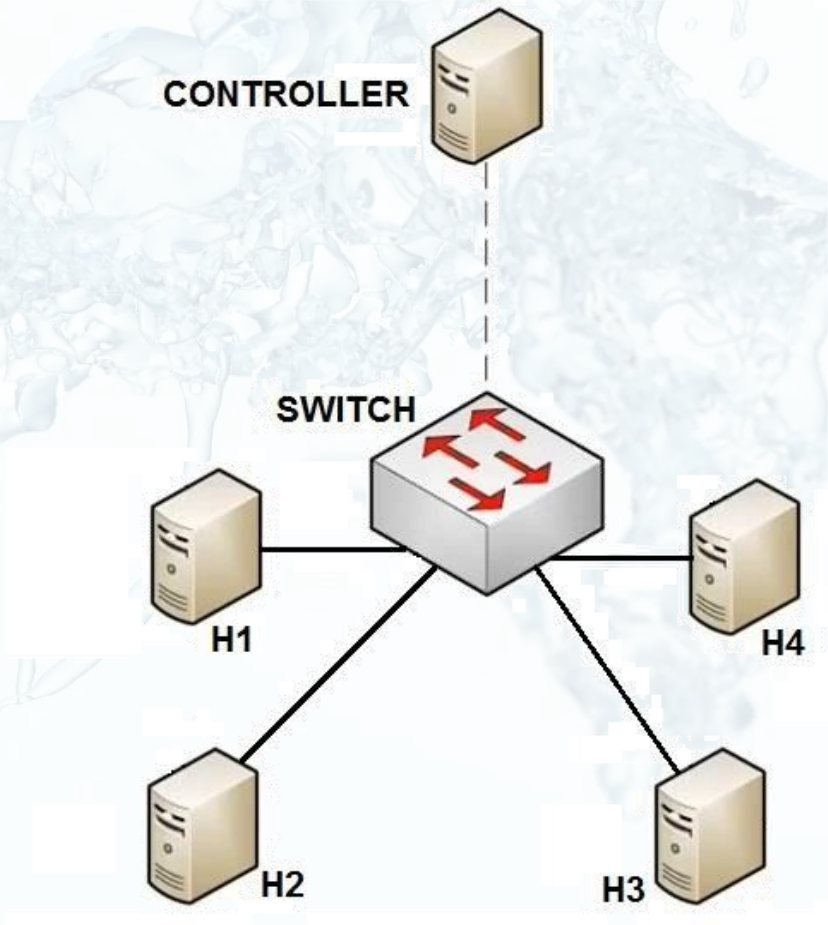
Execution output

```
$ sudo python simpletest.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
Testing network connectivity
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Putting (Some of) It Together

- Mininet Demo
  - OpenFlow
  - OVS

Aggarwal (2015)



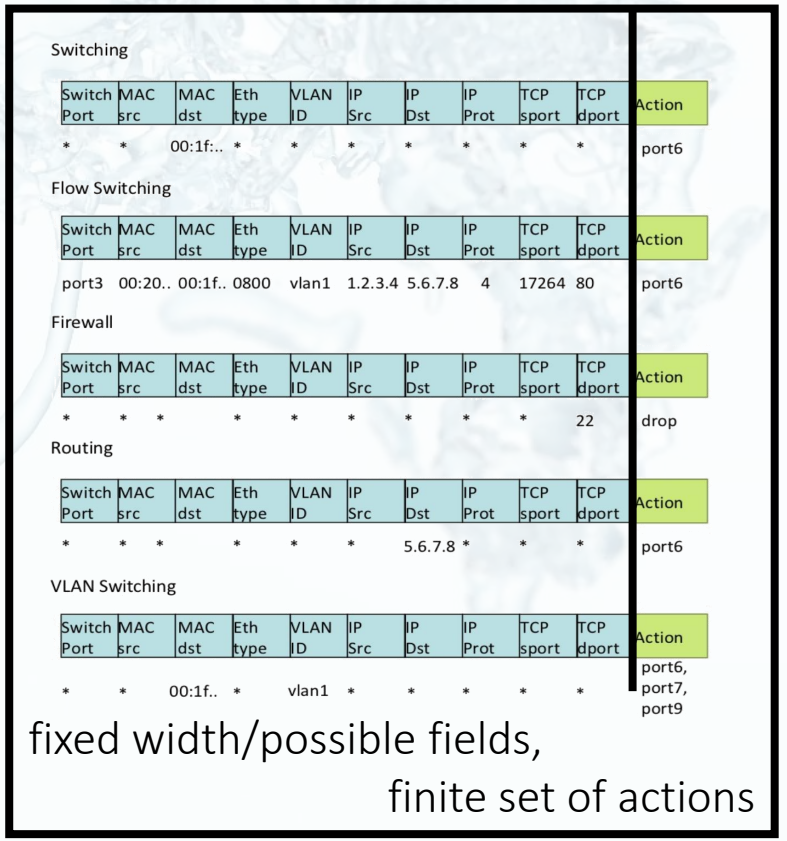
Outline

- Software Defined Networks (SDN)
- OpenFlow
- Open vSwitch (OVS)
- Mininet
- P4
- Virtual Extensible LAN (VXLAN)

OpenFlow Limitations

- OF essentially provides ***data plane configuration/manipulation***
  - Based on flow table rules creation/deletion/update
  - Controller-managed via OF API
  - Southbound-only
- But, tables are “fixed”
  - Determined by ASICs’ abilities to support protocols
    - Ethernet, IPv4, VLAN, …
  - OF newer versions: support more protocols
    - IPv6, VXLAN, …
- Why?
  - Traditionally, networking chips were not programmable
    - Speed(traditional ASICs) >> Speed(programmable chips)
  - What if this were no longer the case?

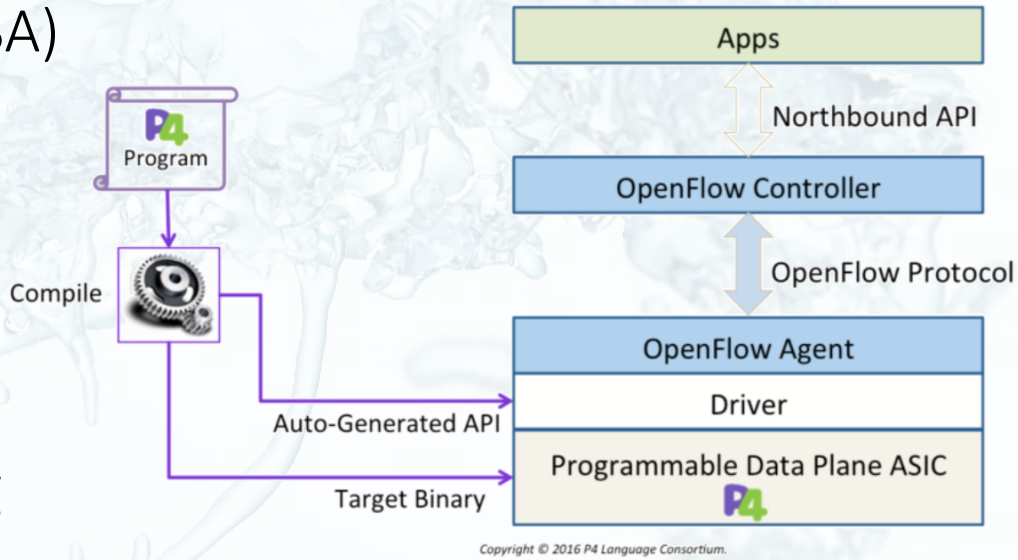
Application-specific integrated circuit



Beyond OpenFlow: P4

- P4: Programming Protocol-independent Packet Processors
  - Protocol Independent Switch Architecture (PISA)
- Network Design
  - From requirements, to ….

    … how switches handle packets
  - Layers and layers of protocols in between
- P4: A programming language for specifying how switches handle packets
  - E.g., OpenFlow is just a program to be compiled
  - Datapath programming independent of SDN
    - Actually, even independent of “protocols”… (program “what you want”)



Protocol Independent Switch Architecture (PISA)

- Programmable network devices
  - CPUs (e.g., OVS): ~10 Gbps (baseline)
  - FPGAs (e.g., Xilinx): 10x
  - PISA chips (e.g., Intel Flexpipe, Barefoot Tofino): 100x !!
    - Comparable to ASICs (size, cost, power, …)
    - Domain-specific: designed for networking, not general purpose…
- Following similar trends in other domains

Computers | Graphics | Signal Processing | Machine Learning | Networking

Image: Kim (2017)

Protocol Independent Switch Architecture (PISA)

- Allows applying software engineering approaches to network design
  - CI/CD, bug-fixing after deployment, differentiating “secret sauce”
- How is this done in other domains?
  - High-level programming language + compiler
- For PISA, it is P4
  - Current version is $P4_{16}$ (or P4_16)

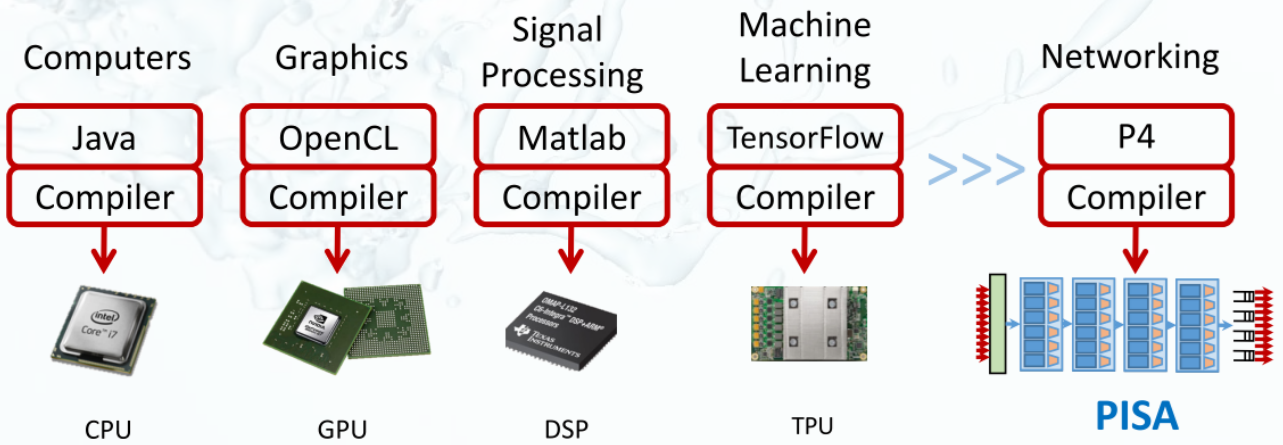
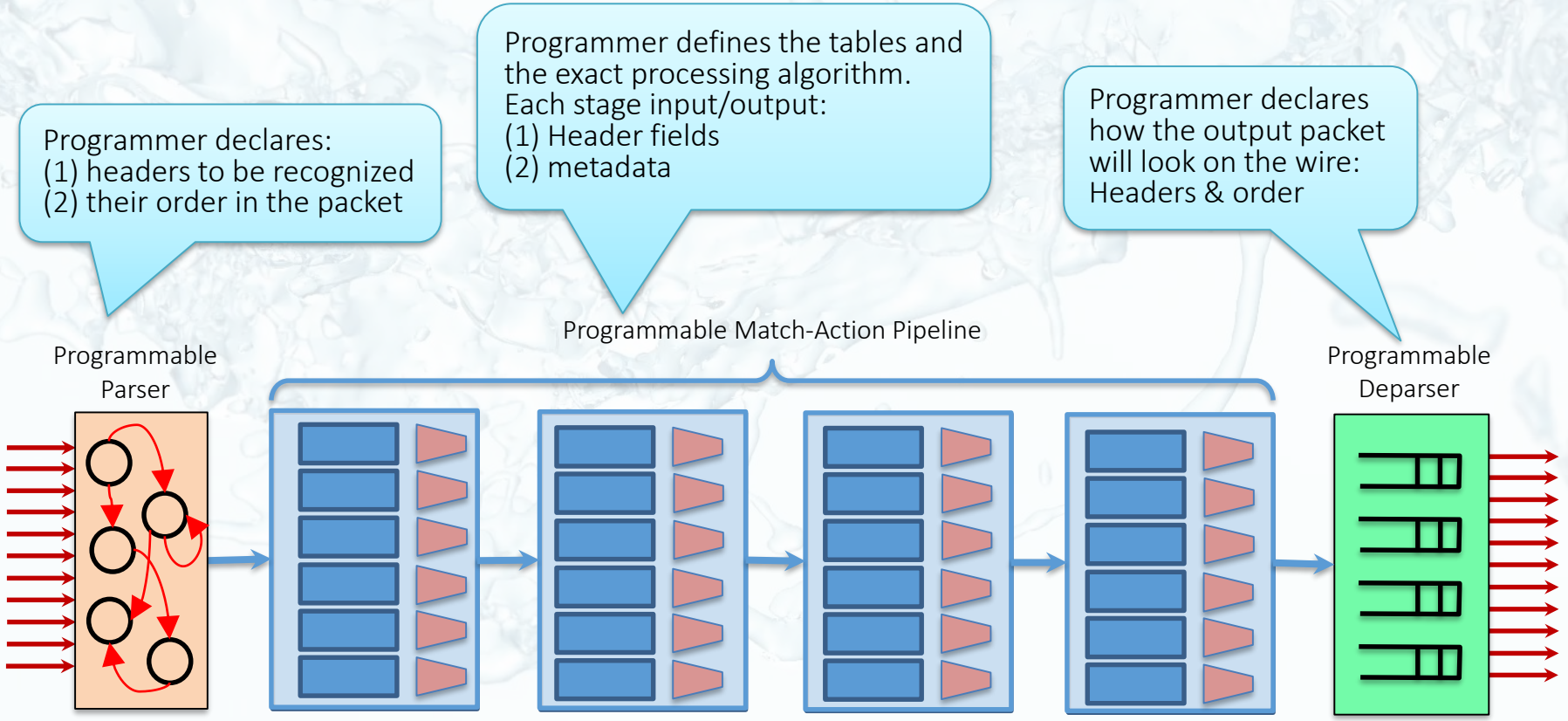


Image: Kim (2017)

How Does PISA Work?

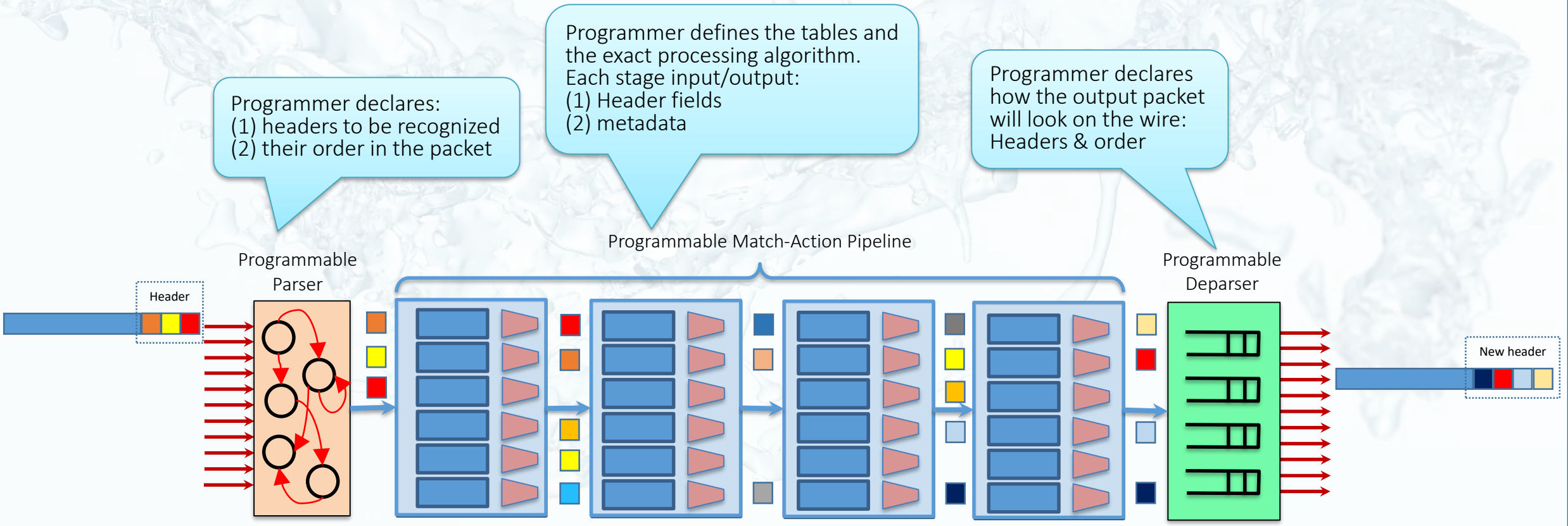
- Components



P4.org (2018)

How Does PISA Work?

- A few µs in the life of a packet in PISA



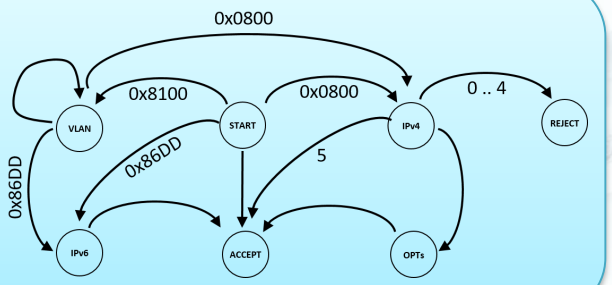
P4.org (2018)

How Does PISA Work?

- Under the hood

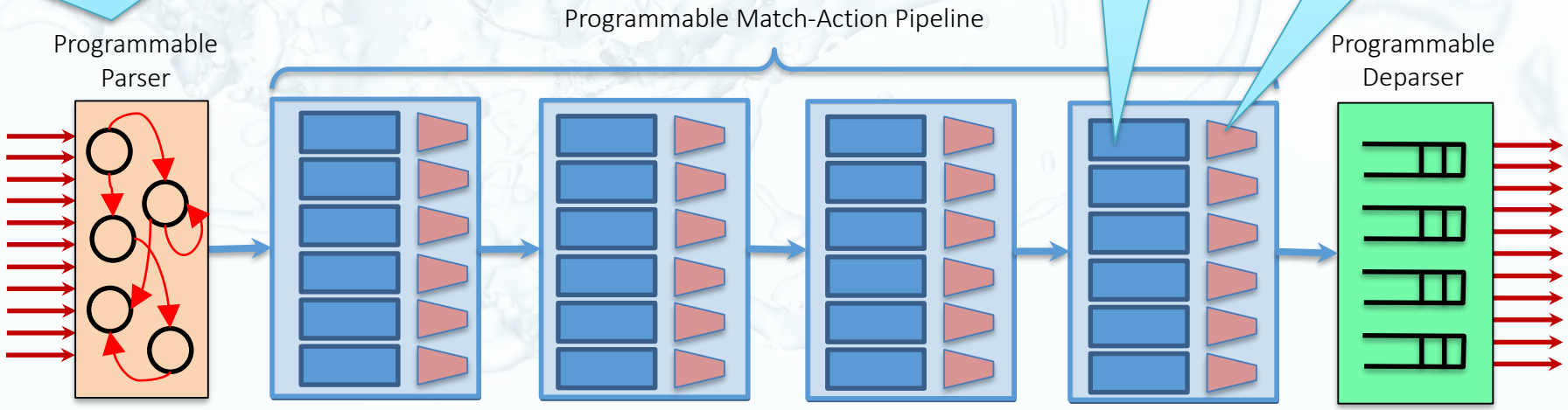
State Machine:

Mandatory states:
START, ACCEPT, REJECT
Other states: user-defined



Match logic:
(1) SRAM/TCAM for LUTs
(2) Hash tables (quick lookup)
(3) Counters, meters

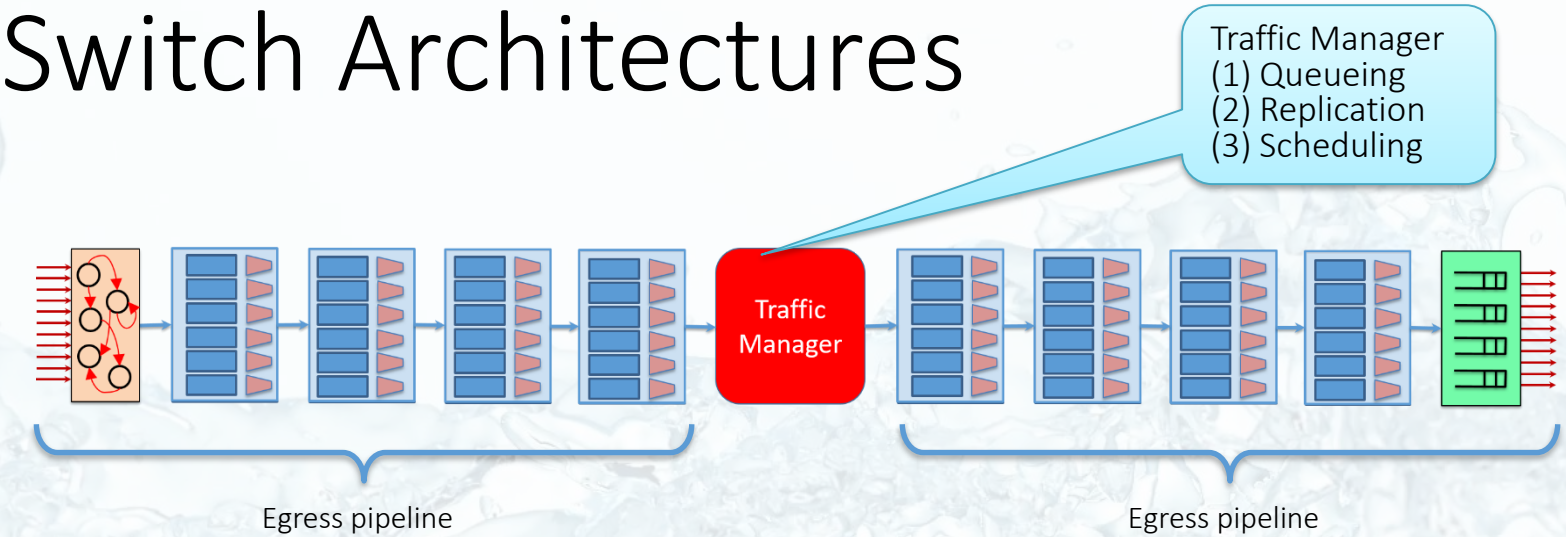
Action logic: ALUs
(1) Boolean / arithmetic operations
(2) Header modifications
(3) Hashing



P4.org (2018)

(Some) PISA Switch Architectures

- Simple switch



- Portable Switch Architecture (symmetric ingress-egress pipelines)

- Also non-standard architecture (with possible proprietary components)

P4.org (2018)

P4 Programming: Types

- Basic types
  - `bit<n>` (unsigned), `int<n>` (signed), …
  - `typedef`
- `header` types
  - Ordered tuple of elements, byte aligned
  - Valid/invalid
    - Built in isValid(), setValid(), setInvalid() operations
- `struct` types
  - Unordered set of elements
    - no alignment restriction
  - E.g., for metadata

```
#include <core.p4>
#include <v1model.p4>

typedef bit<48> EthernetAddress;
typedef bit<32> IPv4Address;

header ethernet_t {
    EthernetAddress dst_addr;
    EthernetAddress src_addr;
    bit<16>         ether_type;
}

header ipv4_t {
    bit<4>      version;
    bit<4>      ihl;
    bit<8>      diffserv;
    bit<16>     total_len;
    bit<16>     identification;
    bit<3>      flags;
    bit<13>     frag_offset;
    bit<8>      ttl;
    bit<8>      protocol;
    bit<16>     hdr_checksum;
    IPv4Address src_addr;
    IPv4Address dst_addr;
}

struct headers_t {
    ethernet_t ethernet;
    ipv4_t     ipv4;
}
```

P4.org (2018)

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<1>  drop;
    bit<32> packet_length;
...
}

struct my_metadata_t {
...
}
```

P4 Programming: Parser

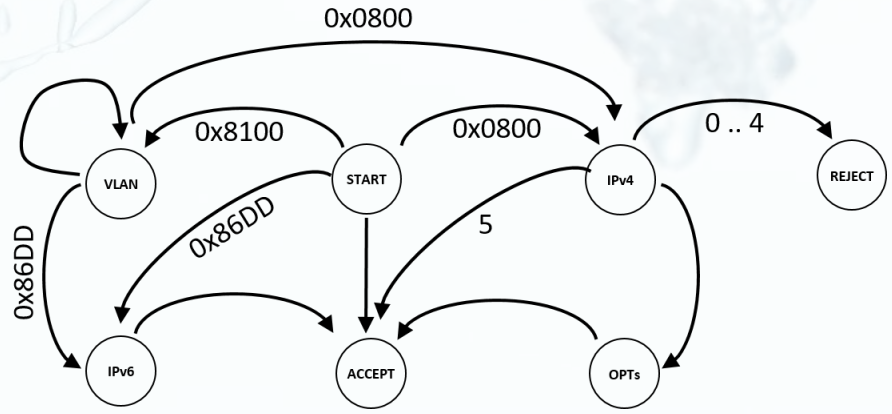
system-provided inputs

Gurevitch (2017)

```
parser MyParser(packet_in packet,
                out my_headers_t hdr,
                inout my_metadata_t meta,
                in standard_metadata_t standard_metadata)
{
     state start {
          packet. extract (hdr.ethernet);
          transition select (hdr.ethernet.etherType) {
              0x8100 &&& 0xEFFF : parse_vlan_tag;
              0x0800 : parse_ipv4;
              0x86DD : parse_ipv6;
              ...
              default : accept ;
          }
     }

     state parse_vlan_tag {
          packet. extract (hdr.vlan_tag .next );
          transition select (hdr.vlan_tag .last .etherType) {
               0x8100 : parse_vlan_tag;
               0x0800 : parse_ipv4;
               0x86DD : parse_ipv6;
               0x0806 : parse_arp;
               default : accept ;
          }
```

```
     state parse_ipv4 {
          packet. extract (hdr.ipv4);
          transition select(hdr.ipv4.ihl) {
               0 .. 4: reject ;
               5: accept ;
               default: parse_ipv4_options;
     }
     state parse_ipv4_options {
          packet. extract (hdr.ipv4.options,
                           (hdr.ipv4.ihl - 5) << 2);
          transition accept ;
     }
     state parse_ipv6 {
          packet. extract (hdr.ipv6);
          transition accept ;
     }
}
```



P4 Programming: Action

- Like C-functions, without loops
  - Algorithms on DAGs
- Support for operations on header stacks
  - Next, last, push, pop, …
- Bit manipulation
  - Concat, slice

```
control MyIngress( inout my_headers_t hdr,
                   inout my_metadata_t meta,
                   inout standard_metadata_t standard_metadata)
{
     /* Local Declarations */
     action swap_mac( inout bit<48> dst, inout bit<48> src) {
          bit<48> tmp;
          tmp = dst; dst = src; src = tmp;
     }

     action reflect_back() {
          standard_metadata.egress_spec =
                                  standard_metadata.ingress_port;
     }

     /* The body of the control */
     apply {
          if (hdr.ethernet.dstAddr[40:40] == 0x1) {
               mark_to_drop();
          } else {
               swap_mac(hdr.ethernet.dstAddr,
                        hdr.ethernet.srcAddr);
               reflect_back();
          }
     }
}
```

Gurevitch (2017)

P4 Programming: Match-Action

- Tables:
  - Key to match + match type
    - E.g., exact, LPM, Ternary (mask/wildcards)
  - Action (+ optional data)

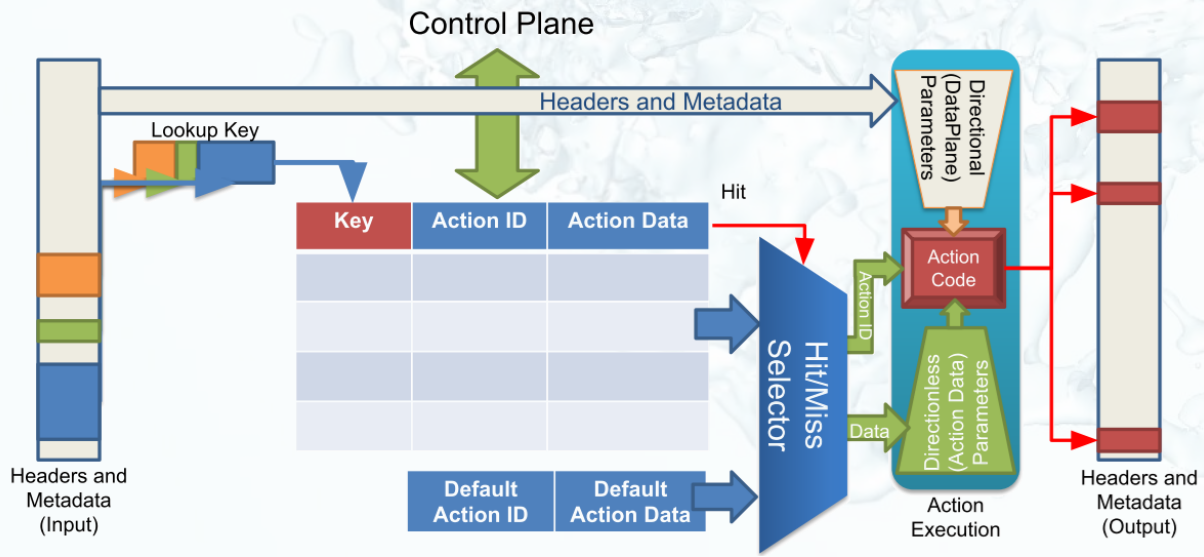


table actions

actual content populated by control plane at runtime (e.g., OF)

```
/* From core.p4 */
action NoAction() {
}

/* From basic.p4 */
action drop() {
    mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                    bit<9> port) {
...
}

/* User Program */
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

Key	Action	Action Data
10.0.1.1/32	ipv4_forward	dstAddr=00:00:00:00:01:01 port=1
10.0.1.2/32	drop	
*`	NoAction	

P4.org (2017)

P4 Programming: Deparser

- Assemble packet as bitstream
  - Serialize header

```
/* From core.p4 */
extern packet_out {
    void emit<T>( in T hdr);
}

/* User Program */
control DeparserImpl(packet_out packet,
                     in headers hdr) {
    apply {
    ...
    packet.emit(hdr.ethernet);
    ...
    }
}
```

P4.org (20187

Who Uses P4, How, and Why?

- Fast prototyping of new network designs
  - Able to tell the network elements exactly what to do
  - New protocols
- Academia
- De-facto standard in industry

partial list from p4.org

Open Networking Foundation (ONF)

https://www.opennetworking.org/

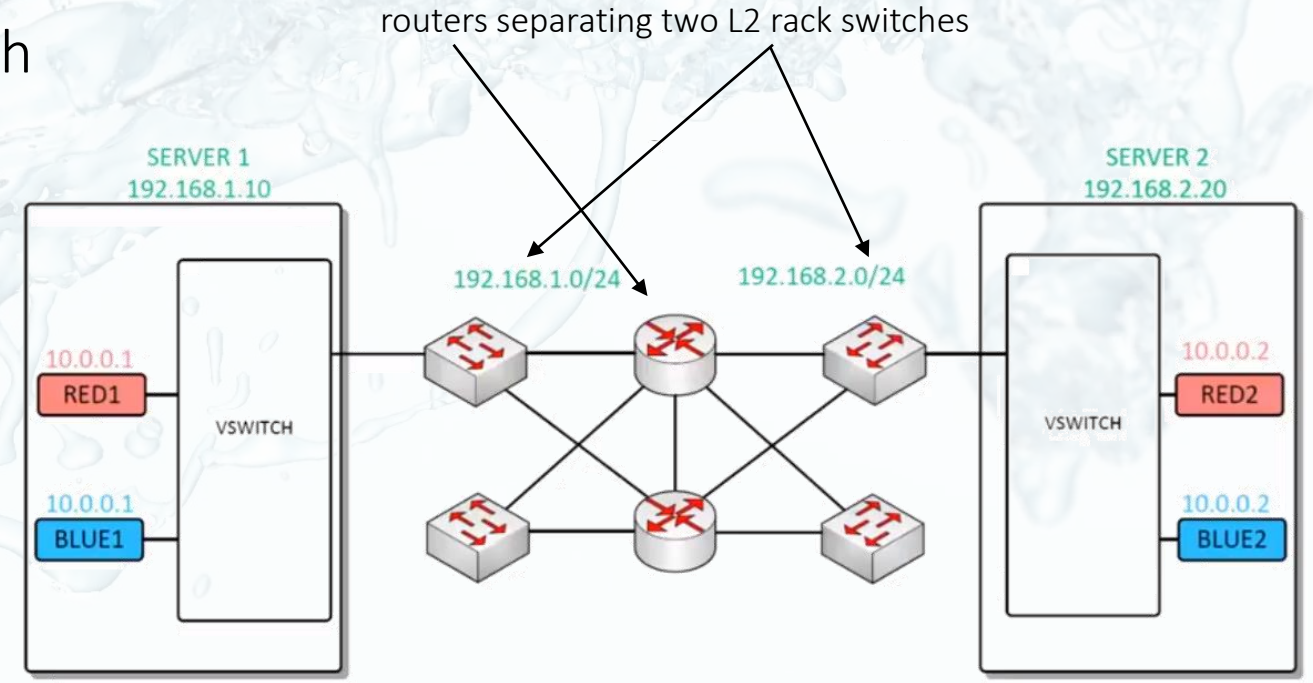
- Consortium advocating open-source networking
  - Operators
  - Vendors
  - …
- Standardization
  - OpenFlow
  - Mininet
  - P4
  - …

Outline

- Software Defined Networks (SDN)
- OpenFlow (OF)
- Open vSwitch (OVS)
- Mininet
- P4
- Virtual Extensible LAN (VXLAN)

VXLAN: Enhancing Virtualization

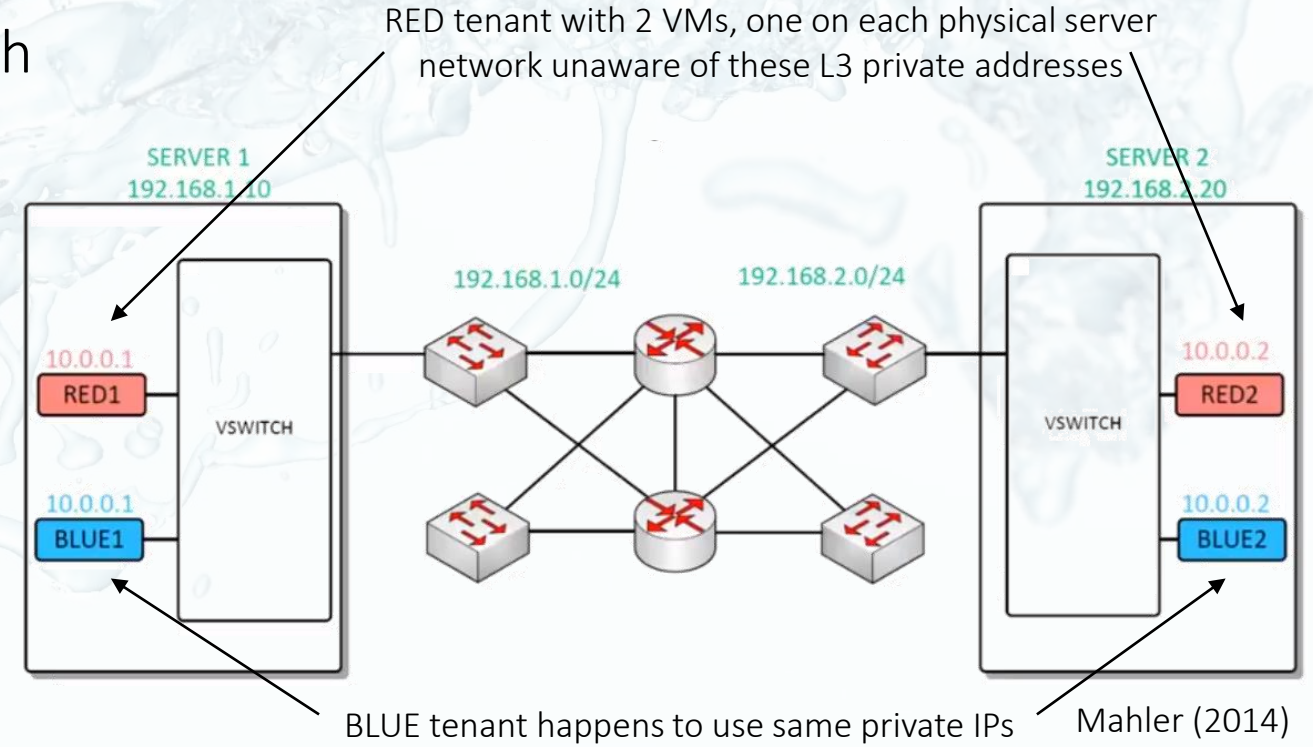
- A quick recap of Virtual LAN (VLAN)
  - 802.1Q VLAN IDs are limited
    - 12 bits --> ~4000 IDs
  - (Very!) insufficient for large datacenters
  - VLAN IDs added/removed by switch
- What's missing?
  - "VLAN"s across L3-boundaries
  - (Many) more distinct VLANs
  - Logical separation(multi-tenancy)
    - Overlapping addressing
    - Security



Mahler (2014)

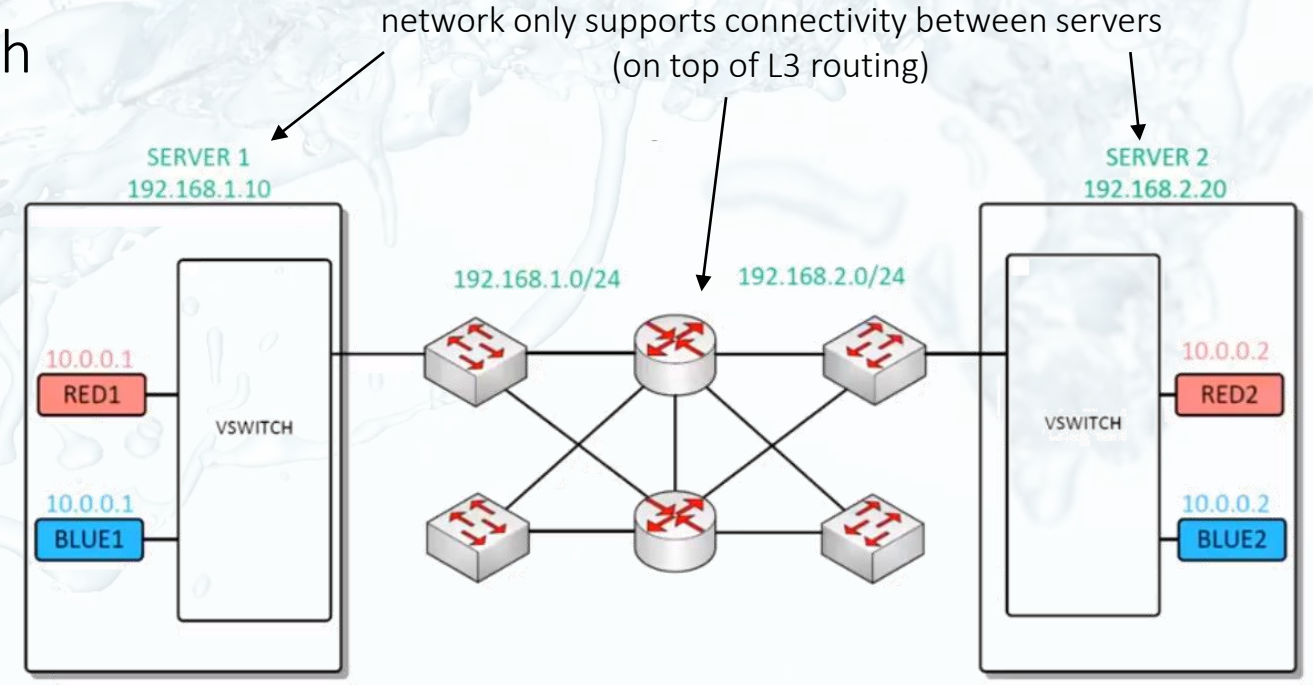
VXLAN: Enhancing Virtualization

- A quick recap of Virtual LAN (VLAN)
  - 802.1Q VLAN IDs are limited
    - 12 bits --> ~4000 IDs
  - (Very!) insufficient for large datacenters
  - VLAN IDs added/removed by switch
- What's missing?
  - "VLAN"s across L3-boundaries
  - (Many) more distinct VLANs
  - Logical separation(multi-tenancy)
    - Overlapping addressing
    - Security



VXLAN: Enhancing Virtualization

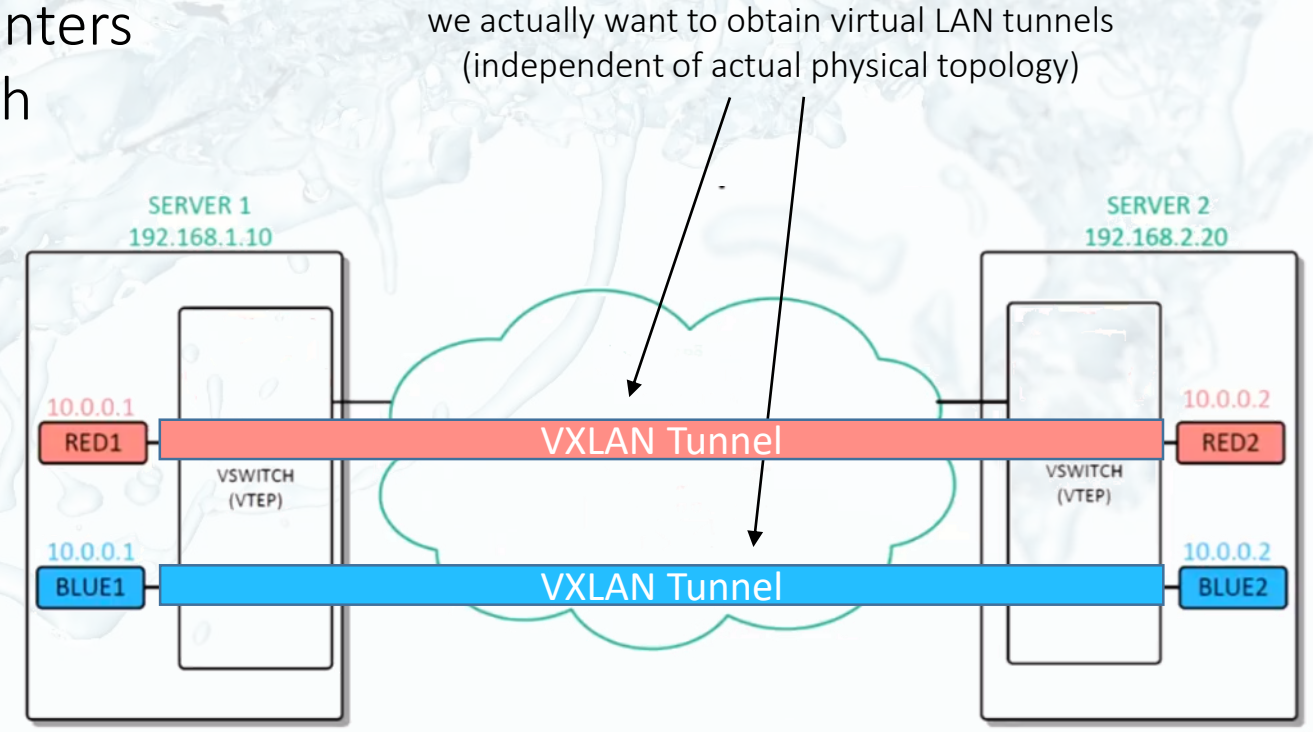
- A quick recap of Virtual LAN (VLAN)
  - 802.1Q VLAN IDs are limited
    - 12 bits --> ~4000 IDs
  - (Very!) insufficient for large datacenters
  - VLAN IDs added/removed by switch
- What's missing?
  - "VLAN"s across L3-boundaries
  - (Many) more distinct VLANs
  - Logical separation(multi-tenancy)
    - Overlapping addressing
    - Security



Mahler (2014)

VXLAN: Enhancing Virtualization

- A quick recap of Virtual LAN (VLAN)
  - 802.1Q VLAN IDs are limited
    - 12 bits --> ~4000 IDs
  - (Very!) insufficient for large datacenters
  - VLAN IDs added/removed by switch
- What's missing?
  - "VLAN"s across L3-boundaries
  - (Many) more distinct VLANs
  - Logical separation(multi-tenancy)
    - Overlapping addressing
    - Security
  - Be efficient about it
    - Small forwarding tables



Mahler (2014)

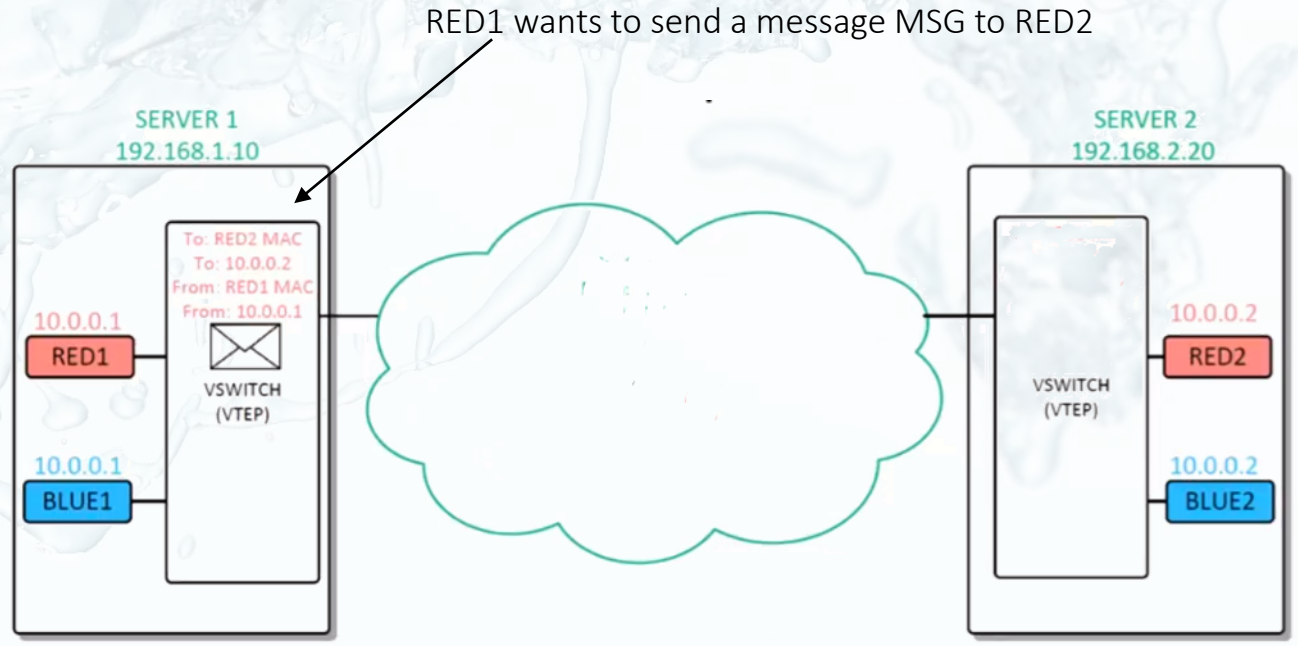
VXLAN: Yet Another Encapsulation

- Virtual switch serves as a Virtual Tunnel Endpoint (VTEP)
  - Requires mapping of L2 address to VTEP IP address
    - So that it knows via which (external) IP to reach the (internal) MAC
    - Done, e.g., via an SDN controller, or “ARP-learning”
      - Send ARP messages with VNI

Mahler (2014)

VXLAN: Yet Another Encapsulation

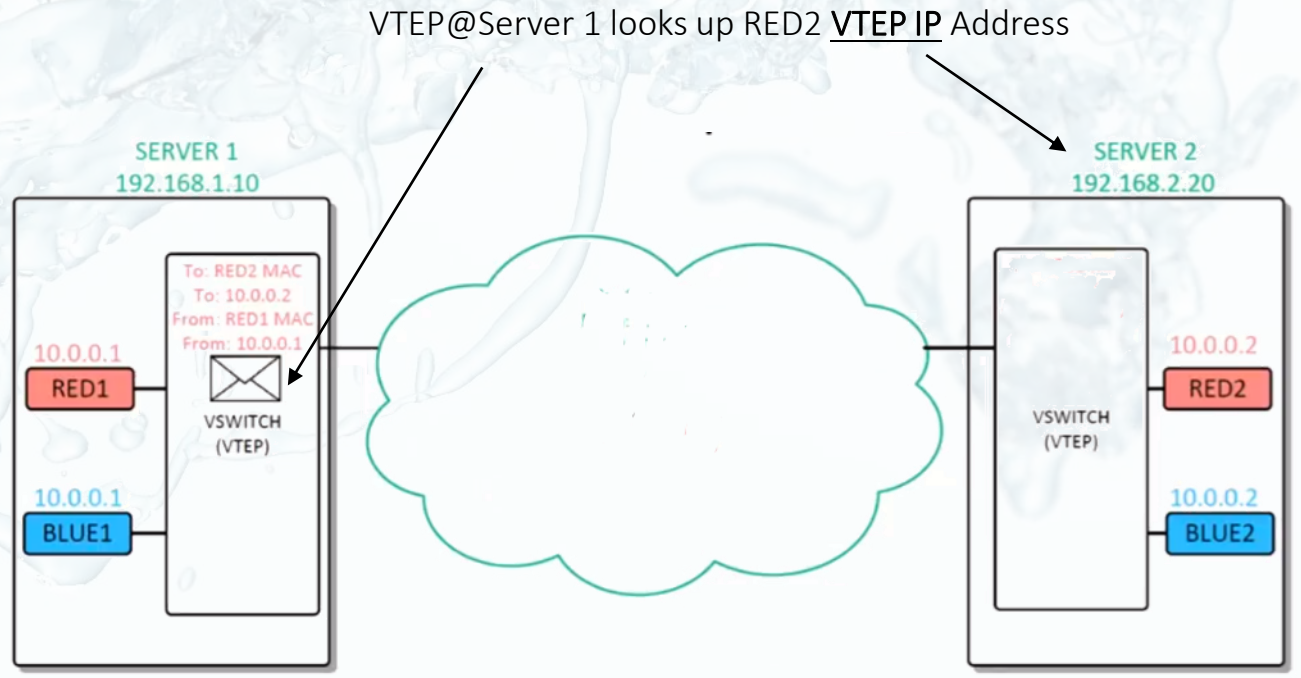
- Virtual switch serves as a Virtual Tunnel Endpoint (VTEP)
  - Requires mapping of L2 address to VTEP IP address



Mahler (2014)

VXLAN: Yet Another Encapsulation

- Virtual switch serves as a Virtual Tunnel Endpoint (VTEP)
  - Requires mapping of L2 address to VTEP IP address



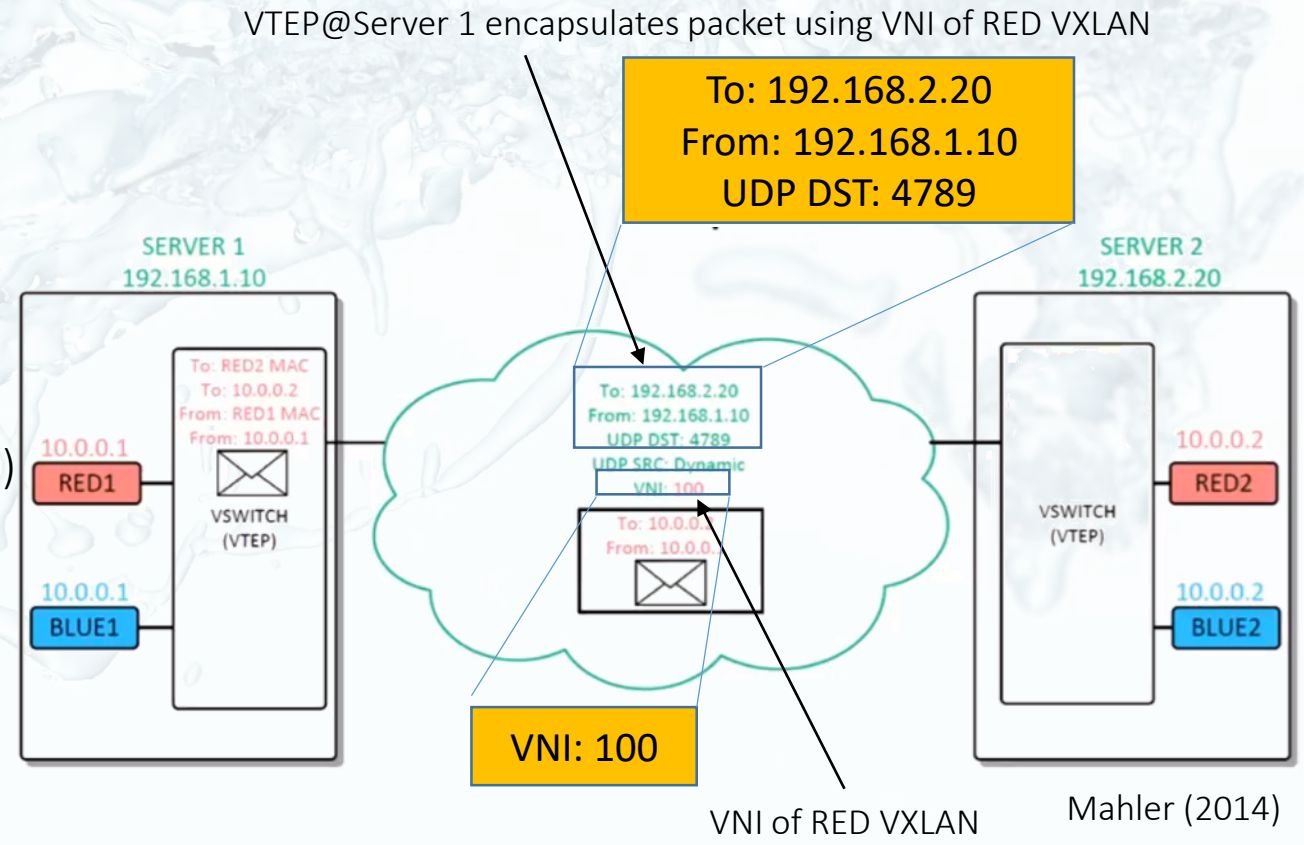
Mahler (2014)

VXLAN: Yet Another Encapsulation

- Virtual switch serves as a Virtual Tunnel Endpoint (VTEP)
  - Requires mapping of L2 address to VTEP IP address
- Encapsulation-based:

IP	UDP	VNI	MSG

  - Uses reserved UDP DST port 4789
  - UDP SRC is dynamically allocated
    - Hash function, for load balancing
      - We've seen this (and will see more…)
  - VXLAN Network Identifier (VNI) identifies the VXLAN
    - Similar to traditional VLAN tag
    - 24 bits --> ~17,000,000 VNIs

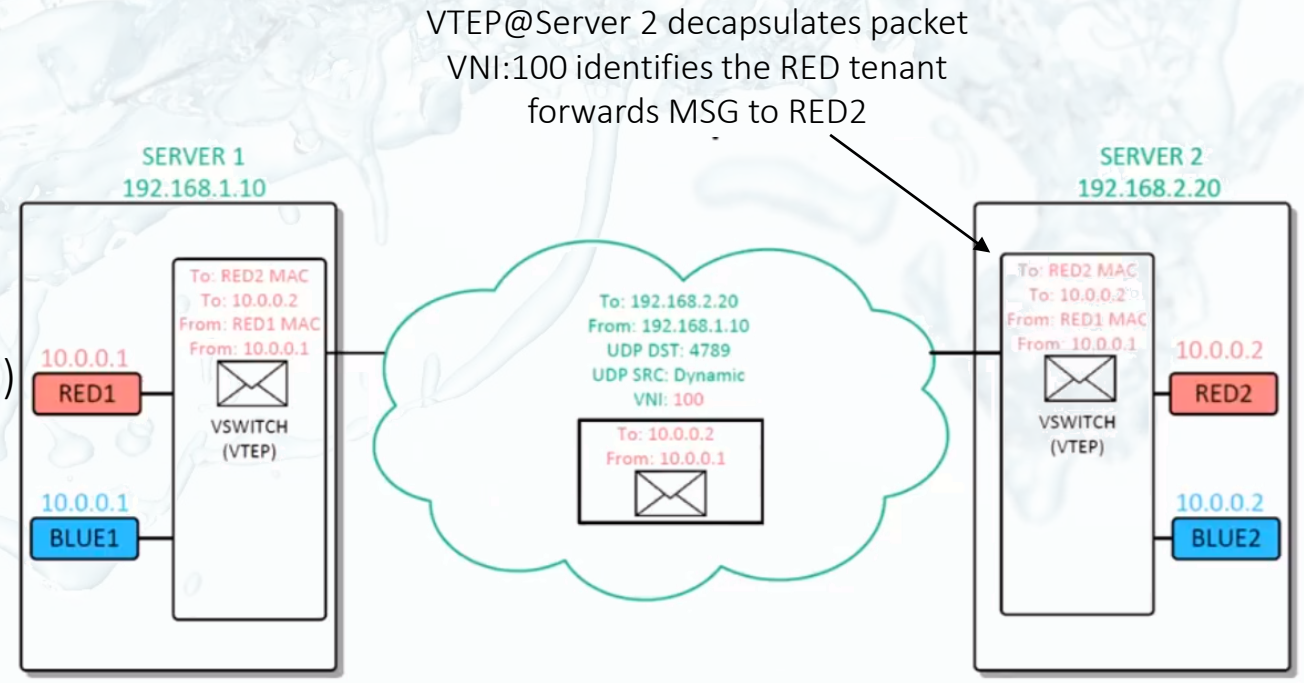


VXLAN: Yet Another Encapsulation

- Virtual switch serves as a Virtual Tunnel Endpoint (VTEP)
  - Requires mapping of L2 address to VTEP IP address
- Encapsulation-based:

IP	UDP	VNI	MSG

  - Uses reserved UDP DST port 4789
  - UDP SRC is dynamically allocated
    - Hash function, for load balancing
      - We've seen this (and will see more…)
  - VXLAN Network Identifier (VNI) identifies the VXLAN
    - Similar to traditional VLAN tag
    - 24 bits --> ~17,000,000 VNIs



Mahler (2014)

(Partial) Bibliography

- Stallings. “Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud”, Pearson (2016)
- Hedlund, “Architecting Data Center Networks in the era of Big Data and Cloud” (2012)
- McKeown et al. “OpenFlow: enabling innovation in campus networks.” Computer Communication Review 38(2): 69-74 (2008)
- Seetharaman, “OpenFlow/SDN tutorial”, OFC/NFOEC (2012)
- Nadeau and Gray, “SDN: Software Defined Networks”, O’Reilly (2013)
- Nunes et al. “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks.” IEEE Communications Surveys and Tutorials 16(3): 1617-1634 (2014)
- Kreutz et al. “Software-Defined Networking: A Comprehensive Survey.” Proceedings of the IEEE 103(1): 14-76 (2015)
- Coker & Azodolmolky, “Software Defined Networking with OpenFlow”, Packt (2013)
- https://www.opennetworking.org/ (SDN Definitions, OpenFlow Specifications)
- https://www.openvswitch.org/
- Pettit and Gross, “Open vSwitch”, LinuxSummit (2011)
- Graf, “Underneath OpenStack Quantum: Software Defined Networking with Open vSwitch”, Swiss OpenStack User Group (2013)
- http://mininet.org/
- Aggarwal, “Learning OVS (open vswitch) Using Mininet”, https://ervikrant06.wordpress.com/ (2015)
- Bosshart et al. “P4: programming protocol-independent packet processors”. Computer Communication Review 44(3): 87-95 (2014)
- Kim, “Programming the Network Data Plane In P4”, GENI Webinar (2017)
- P4.org, “P4 Language Tutorial”, https://bit.ly/p4d2-2018-spring (2018)
- Gurevich, “P4_16 Introduction”, P4.org (2017)
- VXLAN: IETF RFC 7348
- Mahler, “Introduction to Cloud Overlay Networks – VXLAN”, YouTube (2014)