SOFTWARE DEFINED NETWORKS

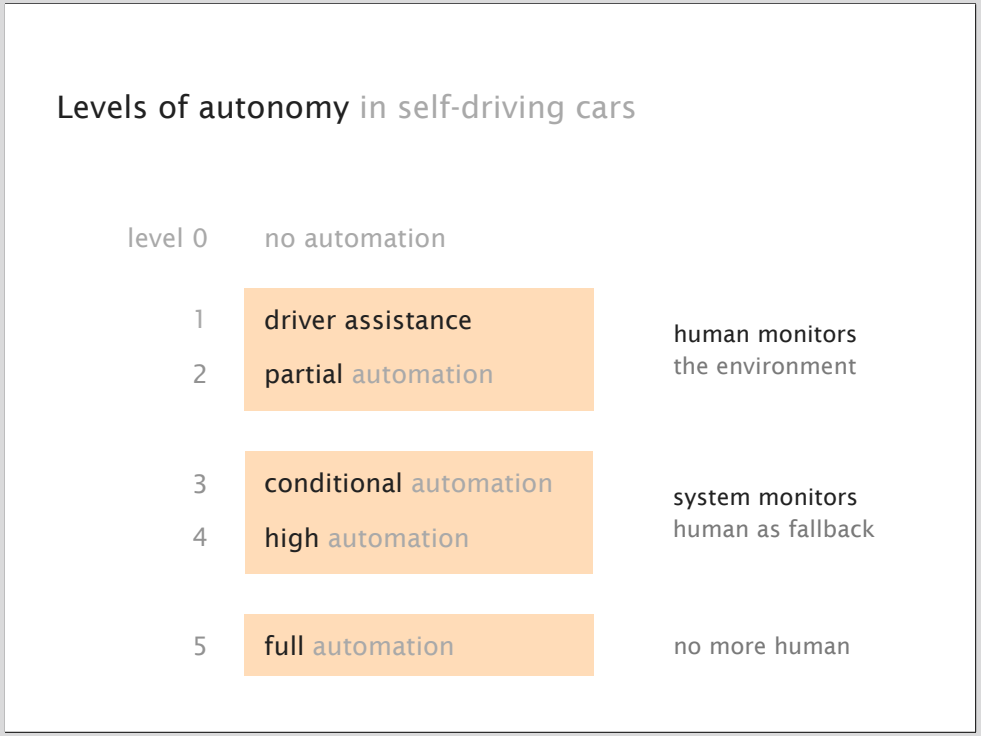
Inspired from the article “D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, ***Software-Defined Networking: A Comprehensive Survey,*** in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015. doi: 10.1109/JPROC.2014.2371999’

ENABLING NEXT GENERATION NETWORKS

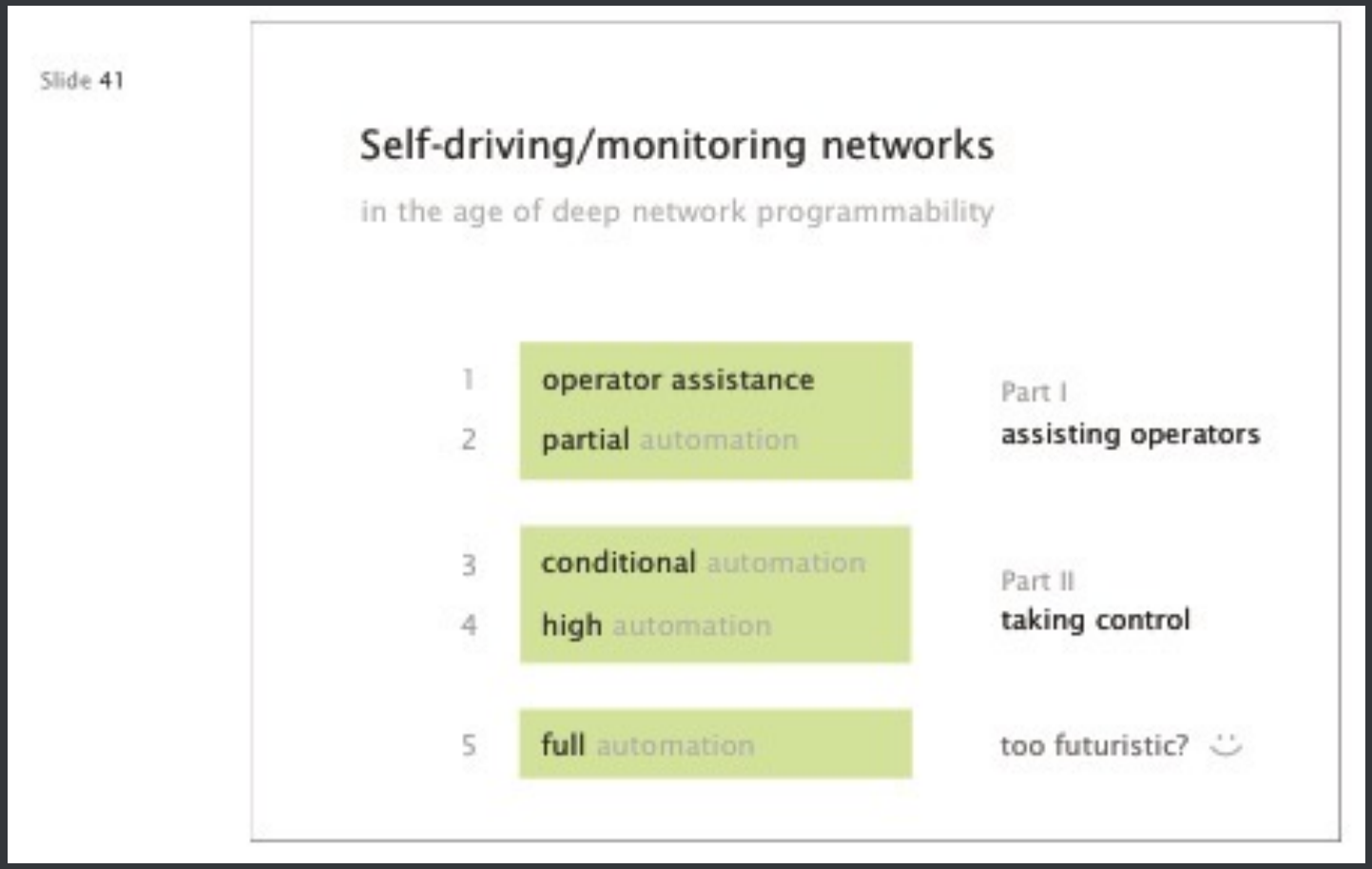
- Internet Service Provides (ISP) Networks
  - CloudRAN
    - Radio Access Network for massive connections
    - Easy handover management
  - Cloud-based ISP networks
- Data Center (DC) Networks
  - Enable *aaS
- ISP and DC
  - Zero-touch network management and configuration
  - On-demand services deployment
    - Virtual Network Function (VNFs)
- And more…

NEXT GENERATION NETWORKS: THE (SELF) DRIVING NETWORK EXAMPLE (1)

Slide 40



NEXT GENERATION NETWORKS: THE (SELF) DRIVING NETWORK EXAMPLE (2)



Laurent Vanbever – TMA Conference 2019 (Keynote)

STATE OF QUO IN NETWORKING

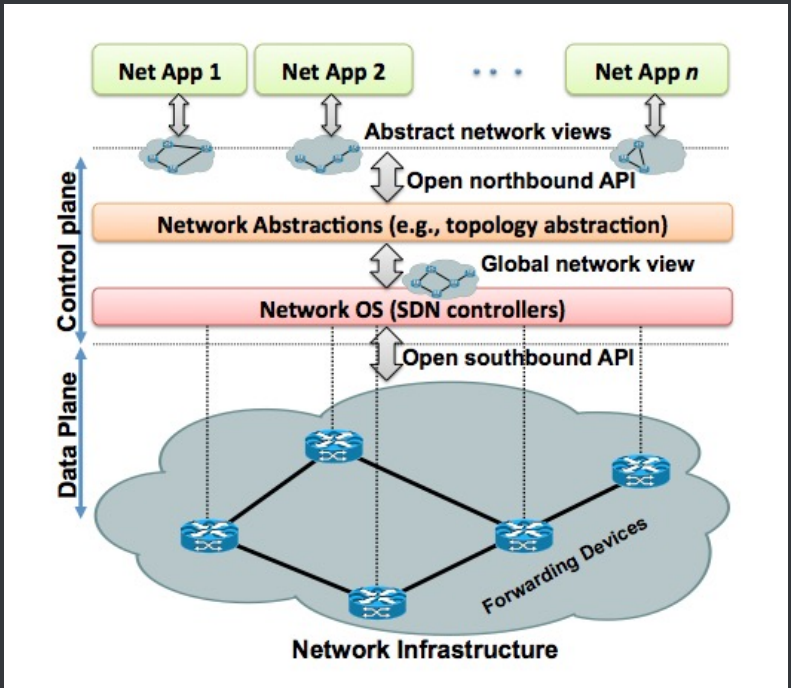
- Computer Networks can be divided in three planes of functionality
  - Data plane → forwarding network devices
  - Control plane → Protocols to build forwarding tables
  - Management plane → monitoring and configuration tools
- In legacy IP networks, the Control Plane and the Data plane are tightly coupled
  - Static architecture
  - The network operations are decentralized → Rigid and complex to administrate
  - Best way to provide network resilience
- To alleviate the lack of in-path network functionalities, several kind of middleboxes have been created: firewalls, NATs, PEPs, etc.
- Hard to innovate

WHAT IS SOFTWARE DEFINED NETWORKING?

- Software Defined Networks (SDN)
  - Coined to represent the OpenFlow project at the Stanford University
  - New network architecture where the data plane is remotely configured by the control plane.
- In SDN
  - The control and data planes are decoupled
  - Forwarding decisions are flow-based, instead of destination-based
    - Flow programming enables a very high flexibility
  - Control logic is hosted in an external entity
  - The network can be programmed by mean of software applications

THE ABSTRACTIONS LAYERS

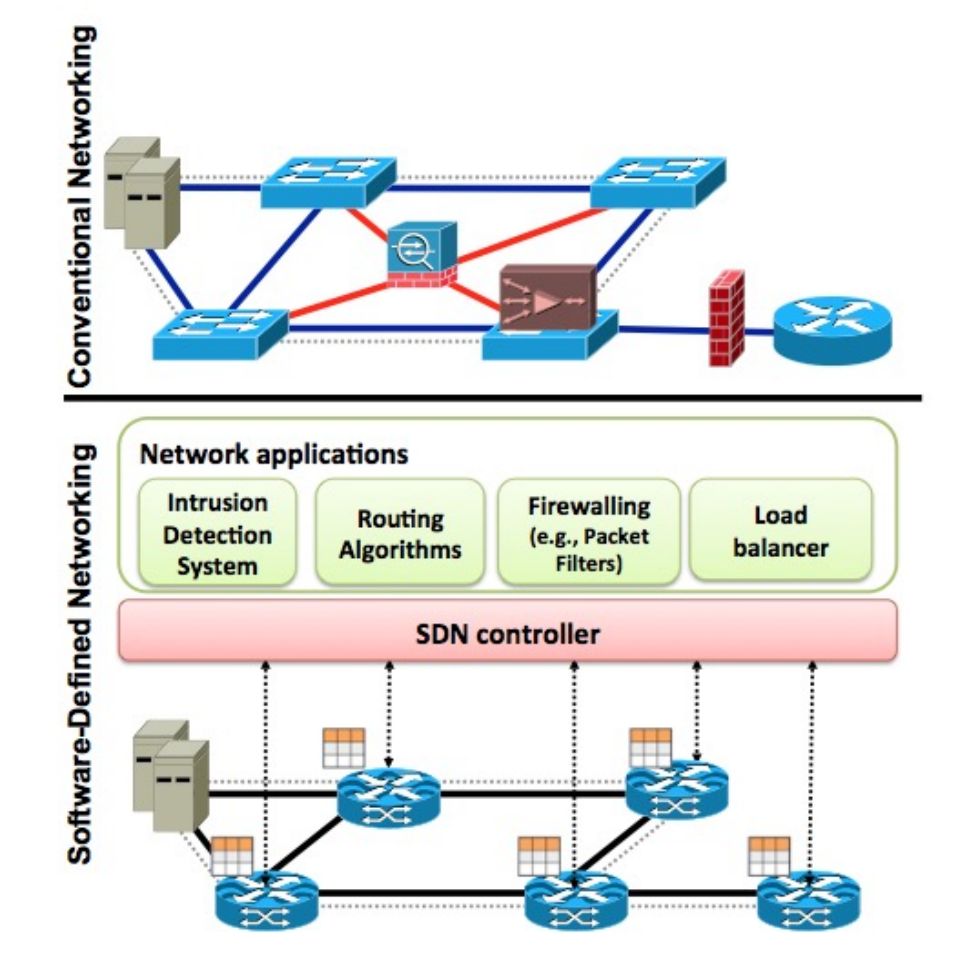
- Specification abstraction: provide any desired network behavior without being responsible of its implementation at the forwarding device
- Distribution abstraction – NOS: install the forwarding behavior in devices and collect status information
- Forwarding abstraction: should allow any forwarding behavior required by the network application



ADVANTAGES OF THE SDN ABSTRACTION

- Easier to program the networks
- Each network device can benefit from the whole view of the network
- Integrating new capabilities into a network device is easier
- Updating all or a part of the network devices is easier

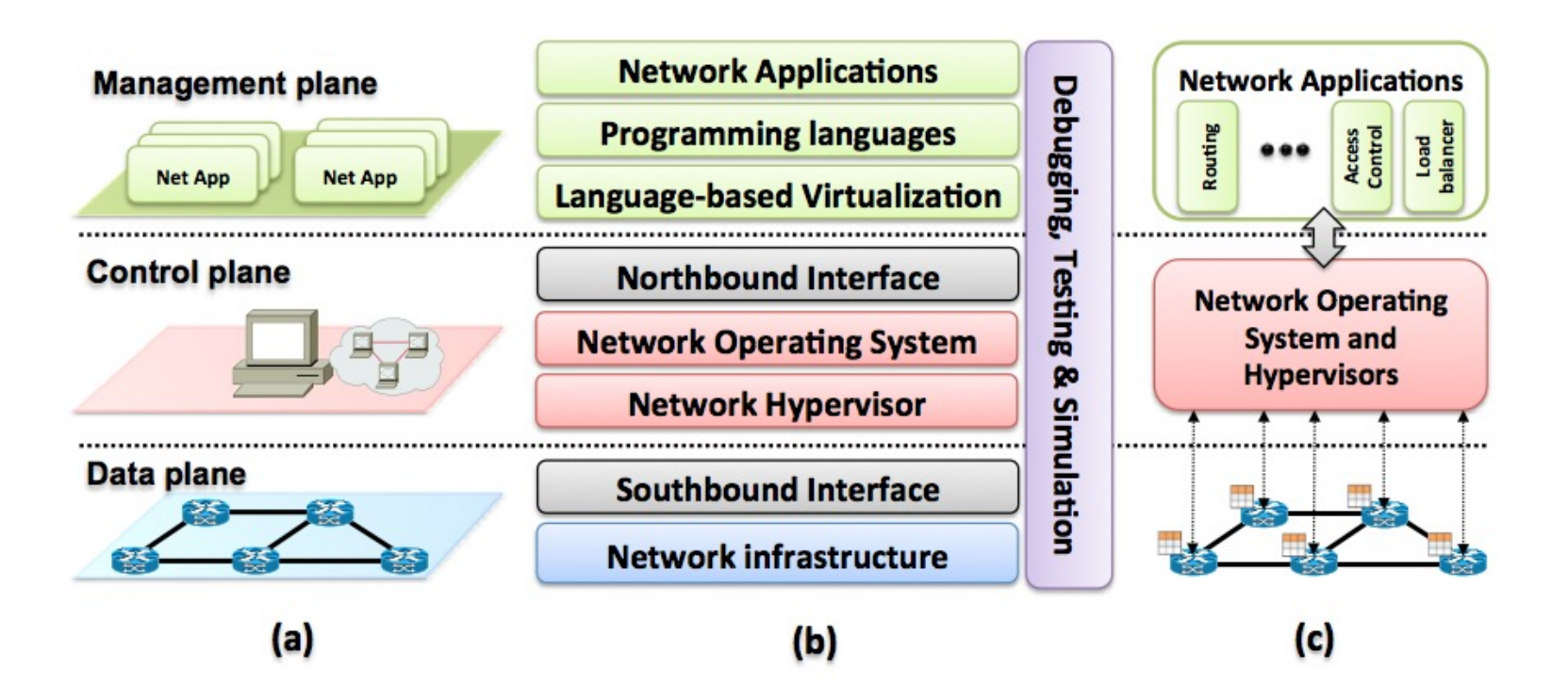
SDN VS TRADITIONAL NETWORKS



TERMINOLOGY

- Forwarding device (FD): hardware or software-based data plane equipments.
  - Forwarding devices receive the rules from the Control Plane through the “Southbound Interface”
- Data Plane (DP): the interconnection of the forwarding devices materialize the DP
- Southbound Interface (SI): define the instruction set supported by the FD and the protocol of communications with the Control Plane
- Control Plane (CP): FD are programmed at the Control Plane, through the SI.
- Northbound Interface: The API that can be provided by the NOS to developers, providing an abstraction of the low level instructions set understood by the FD.
- Management Plane: Applications that provide the functionalities required by the network (e.g. Routing, Load balancing, etc.)

THE GRAPHICAL VIEW OF SDN ARCHITECTURE AND ELEMENTS

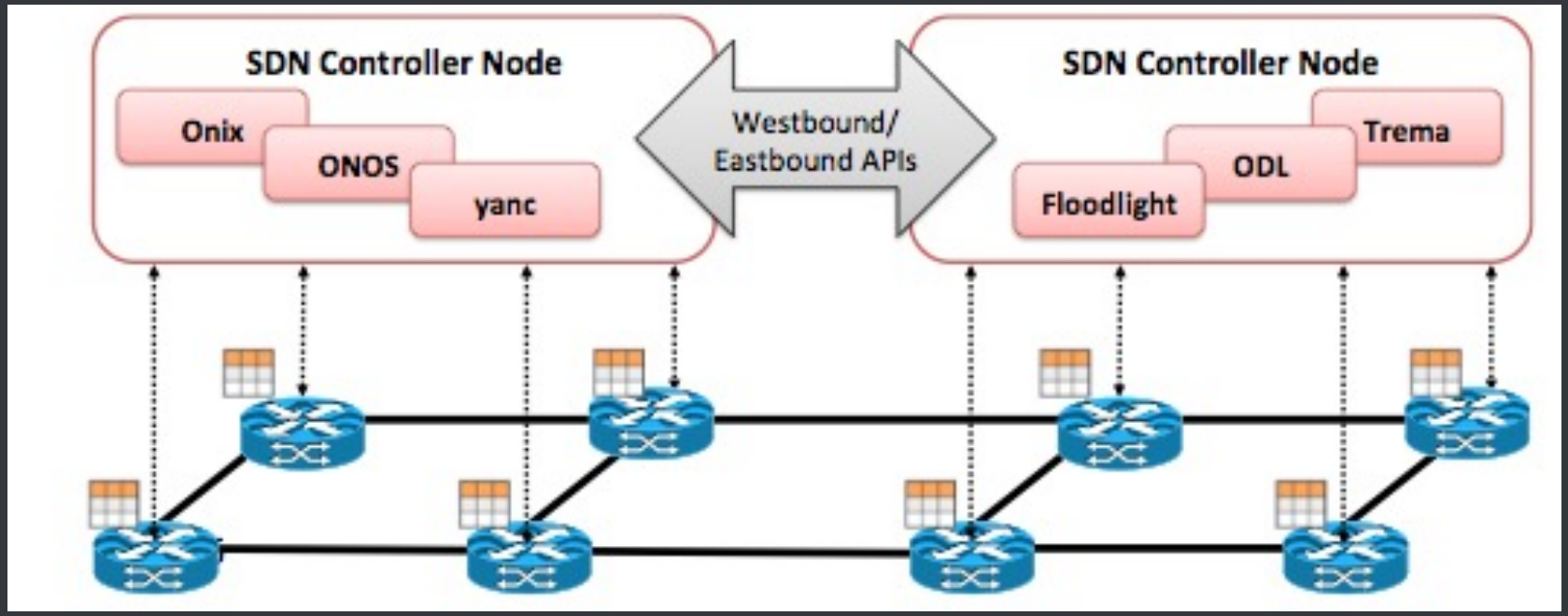


Plane, Layers and System design architecture

THE SDN CONTROLLERS

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. language	Version
Beacon [127]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	Java	v1.0
DISCO [123]	distributed	REST	—	yes	—	Java	v1.1
Floodlight [128]	centralized multi-threaded	RESTful API	no	no	Apache	Java	v1.1
HP VAN SDN [122]	distributed	RESTful API	weak	yes	—	Java	v1.0
HyperFlow [133]	distributed	—	weak	yes	—	C++	v1.0
Kandoo [158]	hierarchically distributed	—	no	no	—	C, C++, Python	v1.0
Onix [7]	distributed	NVP NBAPI	weak, strong	yes	commercial	Python, C	v1.0
Maestro [126]	centralized multi-threaded	ad-hoc API	no	no	LGPLv2.1	Java	v1.0
Meridian [131]	centralized multi-threaded	extensible API layer	no	no	—	Java	v1.0
MobileFlow [154]	—	SDMN API	—	—	—	—	v1.2
MuL [159]	centralized multi-threaded	multi-level interface	no	no	GPLv2	C	v1.0
NOX [53]	centralized	ad-hoc API	no	no	GPLv3	C++	v1.0
NOX-MT [125]	centralized multi-threaded	ad-hoc API	no	no	GPLv3	C++	v1.0
NVP Controller [64]	distributed	—	—	—	commercial	—	—
OpenContrail [121]	—	REST API	no	no	Apache 2.0	Python, C++, Java	v1.0
OpenDaylight [13]	distributed	REST, RESTCONF	weak	no	EPL v1.0	Java	v1.{0,3}
ONOS [69]	distributed	RESTful API	weak, strong	yes	—	Java	v1.0
POX [160]	centralized	ad-hoc API	no	no	GPLv3	Python	v1.0
ProgrammableFlow [161]	centralized	—	—	—	—	C	v1.3
Ryu NOS [130]	centralized multi-threaded	ad-hoc API	no	no	Apache 2.0	Python	v1.{0,2,3}
SNAC [162]	centralized	ad-hoc API	no	no	GPL	C++	v1.0
Trema [129]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	C, Ruby	v1.0
Unified Controller [117]	—	REST API	—	—	commercial	—	v1.0
yanc [134]	distributed	file system	—	—	—	—	—

THE WESTBOUND/EASTBOUND API



SDN-BASED PRODUCTS

Group	Product	Type	Maker/Developer	Version	Short description
Hardware	8200zl and 5400zl [77]	chassis	Hewlett-Packard	v1.0	Data center class chassis (switch modules).
	Arista 7150 Series [78]	switch	Arista Networks	v1.0	Data centers hybrid Ethernet/OpenFlow switches.
	BlackDiamond X8 [79]	switch	Extreme Networks	v1.0	Cloud-scale hybrid Ethernet/OpenFlow switches.
	CX600 Series [80]	router	Huawei	v1.0	Carrier class MAN routers.
	EX9200 Ethernet [81]	chassis	Juniper	v1.0	Chassis based switches for cloud data centers.
	EZchip NP-4 [82]	chip	EZchip Technologies	v1.1	High performance 100-Gigabit network processors.
	MLX Series [83]	router	Brocade	v1.0	Service providers and enterprise class routers.
	NoviSwitch 1248 [76]	switch	NoviFlow	v1.3	High performance OpenFlow switch.
	NetFPGA [33]	card	NetFPGA	v1.0	1G and 10G OpenFlow implementations.
	RackSwitch G8264 [84]	switch	IBM	v1.0	Data center switches supporting Virtual Fabric and OpenFlow.
	PF5240 and PF5820 [85]	switch	NEC	v1.0	Enterprise class hybrid Ethernet/OpenFlow switches.
	Pica8 3920 [86]	switch	Pica8	v1.0	Hybrid Ethernet/OpenFlow switches.
	Plexxi Switch 1 [87]	switch	Plexxi	v1.0	Optical multiplexing interconnect for data centers.
	V330 Series [88]	switch	Centec Networks	v1.0	Hybrid Ethernet/OpenFlow switches.
	Z-Series [89]	switch	Cyan	v1.0	Family of packet-optical transport platforms.
Software	contrail-vrouter [90]	vrouter	Juniper Networks	v1.0	Data-plane function to interface with a VRF.
	LINC [91], [92]	switch	FlowForwarding	v1.3	Erlang-based soft switch with OF-Config 1.1 support.
	ofsoftswitch13 [93]	switch	Ericsson, CPqD	v1.3	OF 1.3 compatible user-space software switch implementation.
	Open vSwitch [94], [61]	switch	Open Community	v1.0	Switch platform designed for virtualized server environments.
	OpenFlow Reference [95]	switch	Stanford	v1.0	OF Switching capability to a Linux PC with multiple NICs.
	OpenFlowClick [96]	vrouter	Yogesh Mundada	v1.0	OpenFlow switching element for Click software routers.
	Switch Light [97]	switch	Big Switch	v1.0	Thin switching software platform for physical/virtual switches.
	Pantou/OpenWRT [98]	switch	Stanford	v1.0	Turns a wireless router into an OF-enabled switch.
	XorPlus [31]	switch	Pica8	v1.0	Switching software for high performance ASICs.

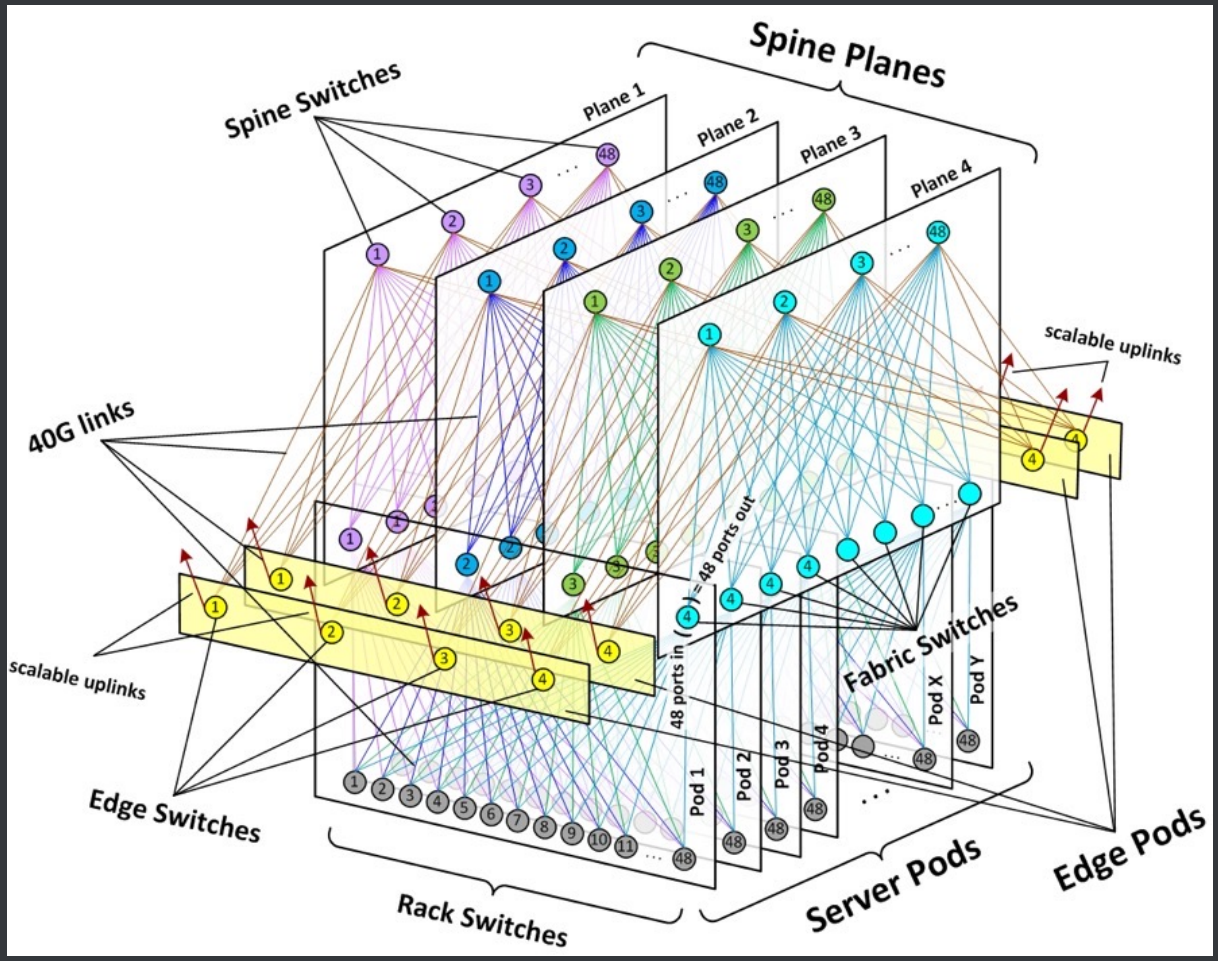
RELATED WORK: PAST & PRESENT OF SDN

Category	Pre-SDN initiatives	More recent SDN developments
Data plane programmability	Tennenhouse Wetherall [38], smart packets [39], ANTS [40], SwitchWare [41], Calvert [42], high performance router [43], NetScript [44], IEEE P1520 [45]	ForCES [22], OpenFlow [9], POF [23]
Control and data plane decoupling	NCP [36], Tempest [46], ForCES [22], RCP [37], SoftRouter [47], PCE [48], 4D [49], IRSCP [50]	SANE [51], Ethane [52], OpenFlow [9], NOX [53], POF [23]
Network virtualization	Tempest [46], MBone [54], 6Bone [55], RON [56], Planet Lab [57], Impasse [58], GENI [59], VINI [60]	Open vSwitch [61], Mininet [62], FlowVisor [63], NVP [64]
Network operating systems	Cisco IOS [65], JUNOS [66]. ExtremeXOS [67], SR OS [68]	NOX [53], Onix [7], ONOS [69]
Technology pull initiatives	Open Signaling [70]	ONF [10]

SDN IN THE CLOUD

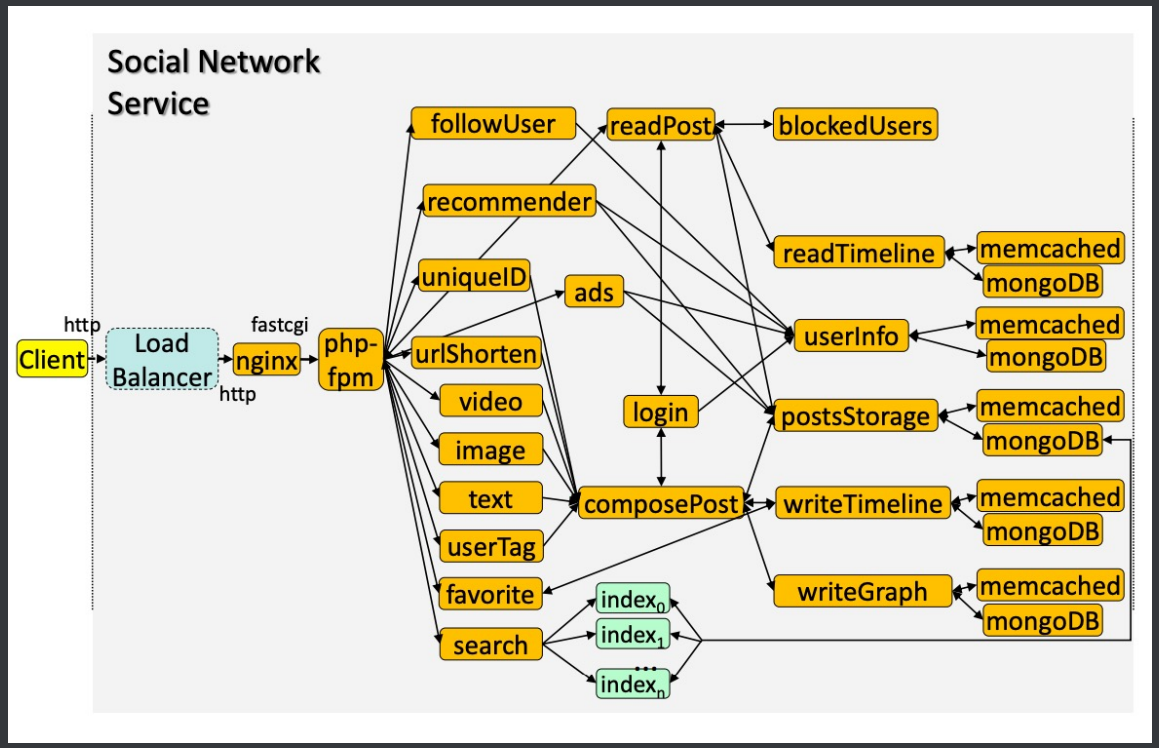
REAL CABLING

Facebook Data Center Network Topology



From https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

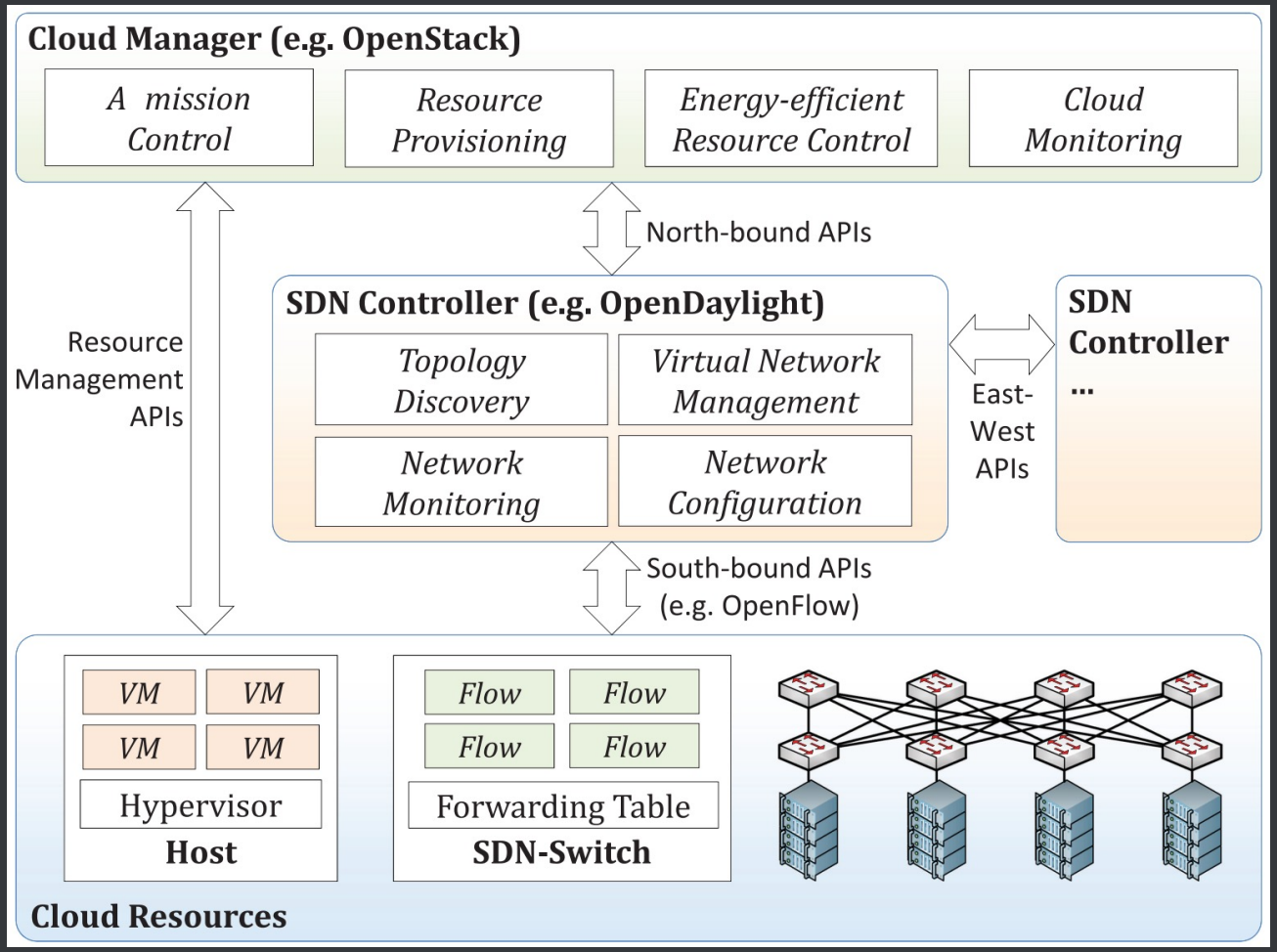
COMPONENTS INTERCONNECTION OF APP (MICROSERVICES)



- One application, several functions and servers
- Functions may be moved across physical servers (e.g. maintenance)
- Needs network connectivity
  - Pure virtual lines
  - Virtual and physical lines

From "*Yu Gan et al. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). Association for Computing Machinery, New York, NY, USA, 3–18. DOI:https://doi.org/10.1145/3297858.3304013*"

SDN IN THE CLOUD



From "Jungmin Son and Rajkumar Buyya. 2018. A Taxonomy of Software-Defined Networking (SDN)-Enabled Cloud Computing. ACM Comput. Surv. 51, 3, Article 59 (May 2019), 36 pages. DOI:https://doi.org/10.1145/3190617"

OPENFLOW

OPENFLOW-BASED SND NETWORKS

- OpenFlow is currently the most known design of SDN-enabled FD (other specification are also available, like the Negotiable Datapath Models -NDMs- from the ONF Forwarding Abstraction Working Group -FAWG-)
  - Only 2 elements are mandatory: the controller and the forwarding device
- From OpenFlow 1.2, new matching fields can be added
- OpenFlow version 1.2 and later OpenFlow 1.3 was defined by the Open Networking Foundation (ONF) standards organization

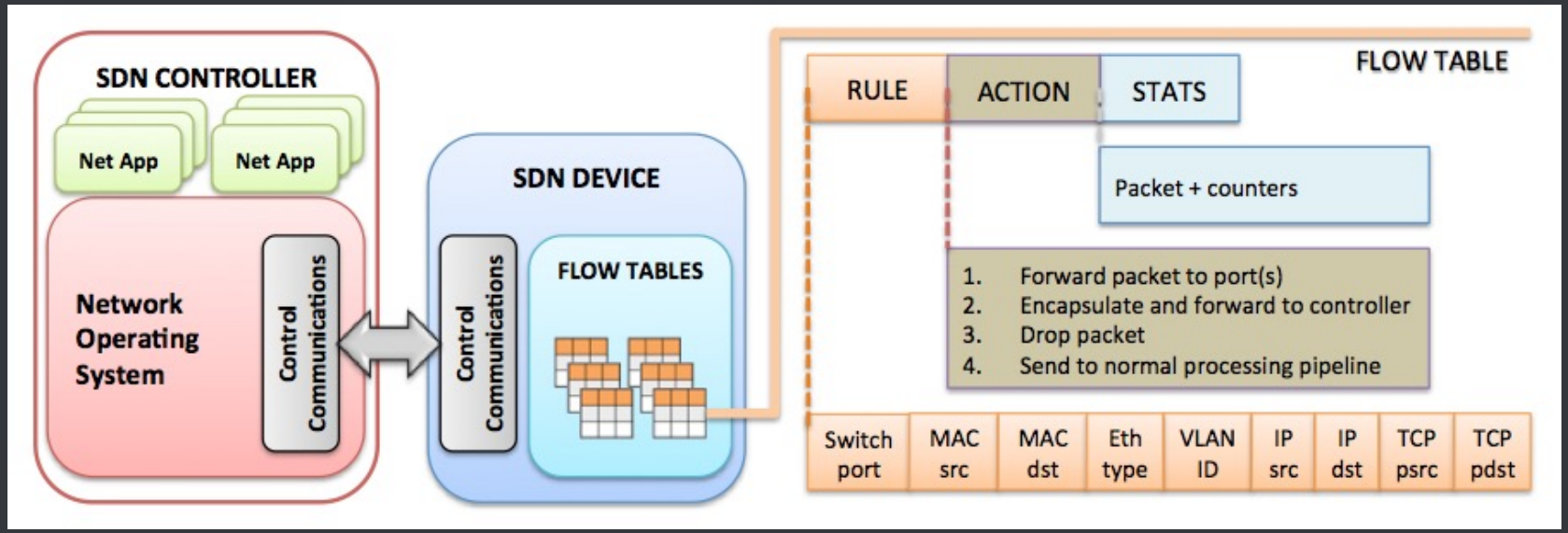
OPENFLOW BASICS

- The controller – FD communications is done through the OpenFlow protocol
  - The Forwarding Information Base (FIB) is materialized by a FlowTable at the FD
  - OpenFlow protocol provides the API to add, delete or modify an entry at the FlowTable
- A Flow Entry is composed by 3 parts
  - Fields to match with incoming packets
  - Actions to be taken in cases of a match
    - Forward, Encapsulate & Froward to the controller, drop
    - Set fields, push/pop tags, set queue, …
  - Counters
    - Received/transmitted packets, duration and several others

OPENFLOW PROCESSING PIPELINE

- When a packet arrives to a OpenFlow-based switch
  - If the packet matches one of the entries at the FlowTable, the action indicated on that entry is executed
  - Otherwise, the packet (a part) is forwarded to the controller (*packet-in*)
- The controller can provide
  - A port where the packet must be forwarded (*packet-out*)
  - A new flow entry (*flow-mod*) that will match future packet with the same characteristics

OPENFLOW ARCHITECTURE



OPENFLOW COMPONENTS

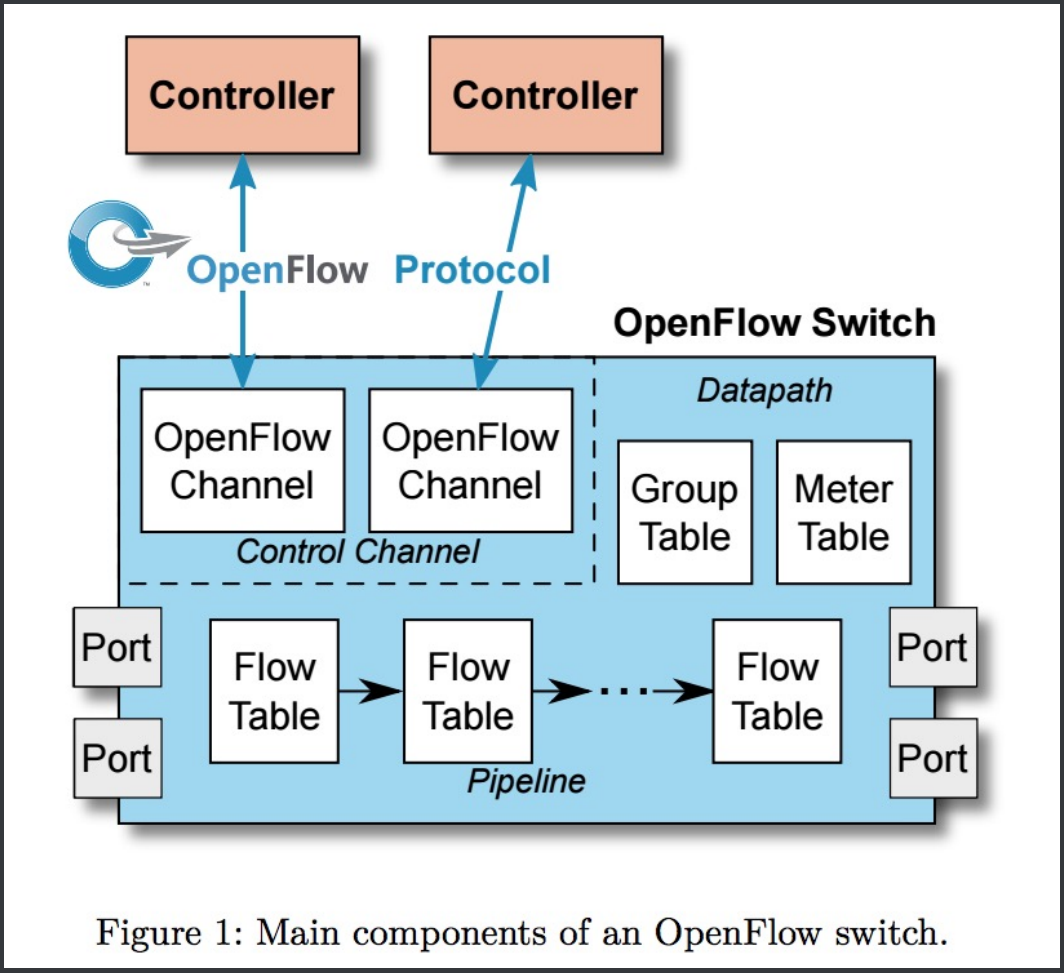


Figure 1: Main components of an OpenFlow switch.

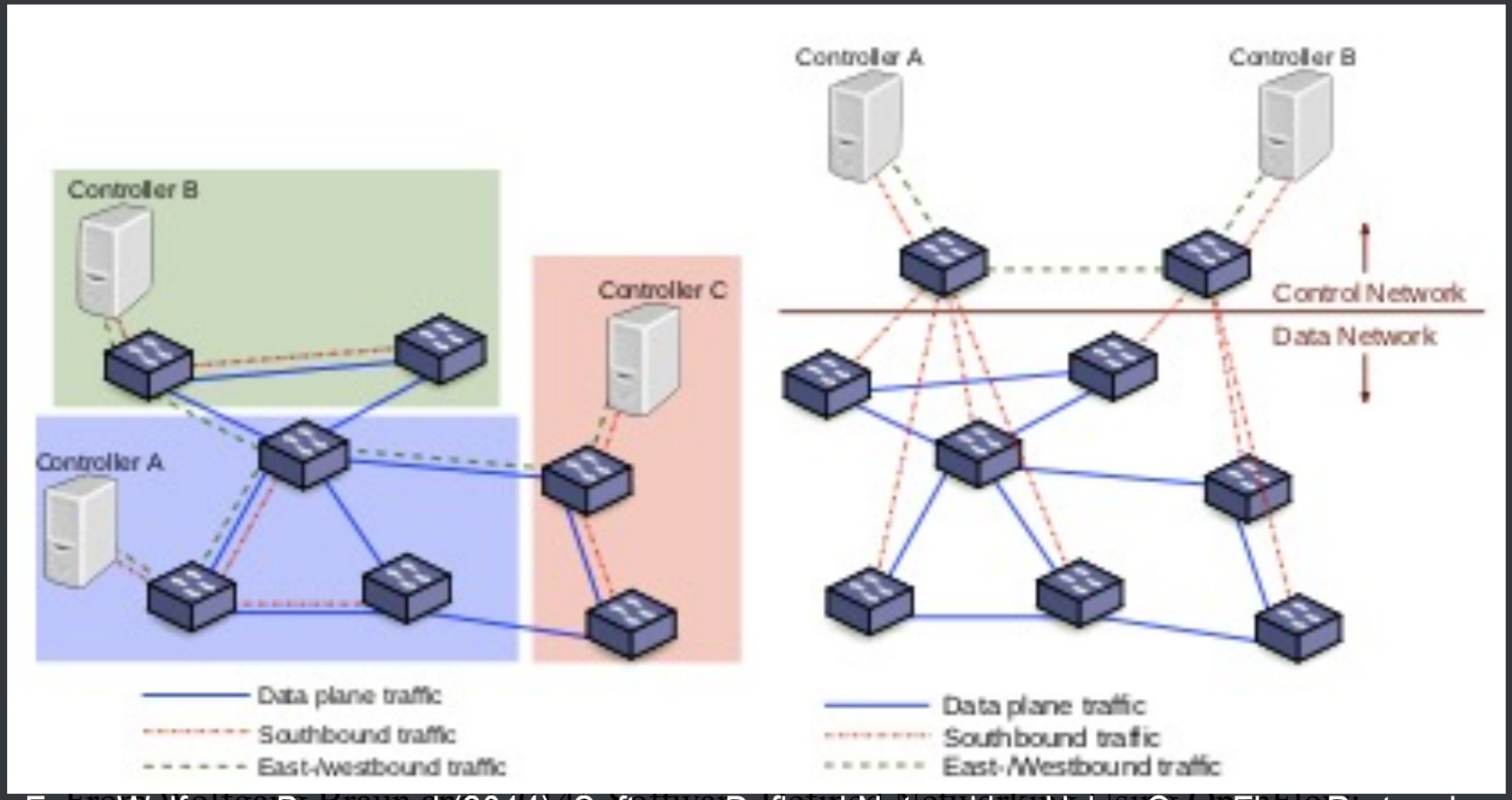
From https://overlaid.net/2017/02/15/openflow-basic-concepts-and-theory/

CONTROL PLANE - DATA PLANE COMMUNICATION

IN-BAND VS OUT-OF-BAND

- In legacy networks, forwarding devices exchange both control and data packets through the same links
  - In-band mode
  - E.g. TCP packets and OSPF Hello Packets
- SDN decouples control plane from data plane
  - How to reach the controller while the data plane is not correctly configured?
  - Out-of-band mode
    - There is one dedicated port per switch to reach the controller
    - Data packets and control packets do not share the same channel
    - How to be sure which port of the switch has a path to the controller?

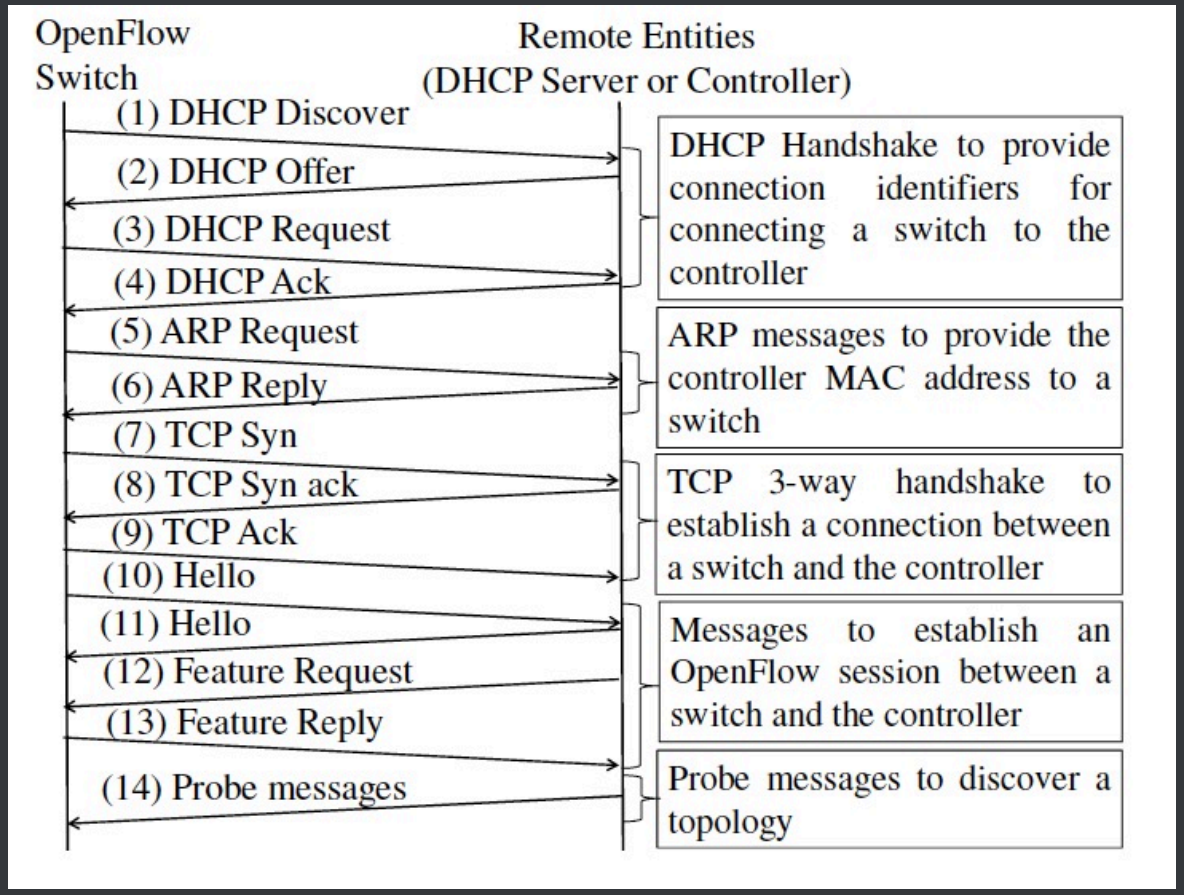
IN-BAND VS OUT-OF-BAND (2)



From Wolfgang Braun and (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications. Future Internet, 6, 302-336.

Wolfgang Braun and (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications. *Future Internet, 6*, 302-336.

OPENFLOW BOOTSTRAPPING



From: S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Automatic bootstrapping of OpenFlow networks," 2013 19th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN), Brussels, 2013, pp. 1-6.doi: 10.1109/LANMAN.2013.6528283

BOOTSTRAPPING CONFIGURATION

```
# ovs-appctl bridge/dump-flows br0

duration=71s, n_packets=32, n_bytes=2446, priority=180008, tcp, nw_src=10.1.0.1, tp_src=6633,
actions=NORMAL

duration=71s, n_packets=40, n_bytes=4060, priority=180007, tcp, nw_dst=10.1.0.1, tp_dst=6633,
actions=NORMAL

duration=71s, n_packets=1, n_bytes=42,
priority=180006,arp,arp_spa=10.1.0.1,arp_op=1,actions=NORMAL

duration=71s, n_packets=1, n_bytes=42,
priority=180005,arp,arp_tpa=10.1.0.1,arp_op=2,actions=NORMAL

duration=71s, n_packets=0, n_bytes=0, priority=180004, arp, dl_src=52:54:00:12:35:02, arp_op=1,
actions=NORMAL

duration=71s, n_packets=1, n_bytes=42, priority=180002,arp,dl_src=2a:5f:66:60:e7:4e, arp_op=1,
actions=NORMAL

duration=71s, n_packets=1, n_bytes=42, priority=180001,arp,dl_dst=2a:5f:66:60:e7:4e, arp_op=2,
actions=NORMAL

duration=71s, n_packets=0, n_bytes=0, priority=180003, arp, dl_dst=52:54:00:12:35:02, arp_op=2,
actions=NORMAL

duration=71s, n_packets=0, n_bytes=0,
priority=180000,udp,in_port=LOCAL,dl_src=2a:5f:66:60:e7:4e, tp_src=68, tp_dst=67, actions=NORMAL

table_id=254, duration=354s, n_packets=0, n_bytes=0, priority=2,recirc_id=0,actions=drop

table_id=254, duration=354s, n_packets=1, n_bytes=70,
priority=0,reg0=0x1,actions=controller(reason=)

table_id=254, duration=354s, n_packets=0, n_bytes=0, priority=0,reg0=0x2,actions=drop

table_id=254, duration=354s, n_packets=0, n_bytes=0, priority=0,reg0=0x3,actions=drop
```

CONTROLLER STATES

- A switch can connect to multiple controllers
- The controller can be in one of the following states
  - Equal
    - Full access to the switch (receive Packet-In and send Flow-Mod messages)
    - Multiple controllers can be in this state
    - This is the default status
  - Master
    - Full access to the switch (receive Packet-In and send Flow-Mod messages)
    - Only one controller can be in the master state
  - Slave
    - Read-only access -> Port-status messages
- The controller can request a status modification at any time

CONTROLLER CONNECTION MODE

- Single
  - The switch will connect to only one of the controllers
  - In case of disconnection, the switch will try to connect to a different controller
- Multiple
  - The OpenFlow instance will connect to multiple controllers
  - In case of disconnection to one of the controller, the switch will try to reconnect to that controller after a connection interval

CONNECTION INTERRUPTION MODE

- When a switch loss the connection with all the controllers, the switch will enter one of the following modes
  - Secure
    - The switch will avoid removing unexpired flow entries
    - If case of a hit, the packet is processes according to that (those) rule(s)
    - In case of a miss, the packet is dropped
  - Smart
    - The switch will avoid removing unexpired flow entries
    - If case of a hit, the packet is processes according to that (those) rule(s)
    - In case of a miss, the packet is forwarded through the normal process
  - Standalone
    - Normal forwarding process

COMMAND LINE AND THE POX CONTROLLER

MANUAL INTERACTION WITH THE OPENVSWITCH FLOWTABLE (1)

- Currently, the controller listen by default on TCP port number 6653
- Mininet's openvswitches listen by default on the controller port + switch id
  - S1 → 6654, S2 -> 6655, and so on
- Display the FlowTable
  - ovs-ofctl dump-flows tcp:192.168.0.15:6654
- Add an entry on the FlowTable
  - ovs-ofctl add-flow tcp:127.0.0.1:6654 dl_src=00:00:00:00:00:01,eth_type=0x0800,nw_src=10.0.0.1,nw_dst=10.0.0.2,actions=output:"s1-eth2"
- Delete an entry from the FlowTable
  - ovs-ofctl del-flow tcp:127.0.0.1:6654 dl_src=00:00:00:00:00:01,eth_type=0x0800,nw_src=10.0.0.1,nw_dst=10.0.0.2
- Modify an entry at the FlowTable
  - ovs-ofctl mod-flows tcp:127.0.0.1:6654 dl_src=00:00:00:00:00:01,eth_type=0x0800,nw_src=10.0.0.1,nw_dst=10.0.0.2,actions=output:"s1-eth3"

MANUAL INTERACTION WITH THE OPENVSWITCH FLOWTABLE (2)

- Resubmit to a table
  - ovs-ofctl add-flow tcp:127.0.0.1:6654 table=0,priority=0,udp,actions=drop
  - ovs-ofctl add-flow tcp:127.0.0.1:6654 "table=0,priority=1,udp,udp_dst=67,actions=resubmit(,1)"
  - ovs-ofctl add-flow tcp:127.0.0.1:6654 "table=0,priority=1,udp,udp_src=67,actions=resubmit(,1)"
  - ovs-ofctl add-flow tcp:127.0.0.1:6654 table=1,ip,nw_dst=10.0.0.2,actions=output:"s1-eth2”
- Groups
  - ovs-ofctl -O OpenFlow13 add-group tcp:127.0.0.1:6654 group_id=1,type=select,bucket=actions=output:2,bucket=actions=output:3
  - ovs-ofctl add-flow tcp:127.0.0.1:6654 in_port=1,actions=group:1

RULE INSTALLATION FROM A POX-BASED APP

```
msg = of.ofp_flow_mod()
msg.priority = 5
msg.flags |= of.OFPFF_SEND_FLOW_REM
msg.idle_timeout = 60
msg.hard_timeout = 0
msg.match.dl_src = packet.src
msg.match.dl_type = pkt.ethernet.ARP_TYPE
msg.actions.append(of.ofp_action_output(p
    ort = oport))
event.connection.send(msg)
```

PACKET-OUT FROM A POX-BASED APP

```
msg = of.ofp_packet_out()
msg.buffer_id = event.ofp.buffer_id
msg.in_port = event.ofp.in_port
msg.data = event.ofp
msg.actions.append(of.ofp_action_output(p
    ort = oport))
event.connection.send(msg)
```

EVENT HANDLERS

```
from pox.core import core
…
core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
core.openflow.addListenerByName("ConnectionDown", _handle_ConnectionDown)
core.openflow.addListenerByName("FlowRemoved", _handle_FlowRemoved)

def _handle_ConnectionUp (event):

def _handle_ConnectionDown (event):

def _handle_PacketIn (event):

def _handle_FlowRemoved (event):
```

PACKET ANALYSIS WITH POX

```
packet = event.parsed
ofp = event.ofp
if packet.type == packet.ARP_TYPE:
  # do some actions
elif packet.type == packet.IP_TYPE:
  ipp = packet.payload
  if ipp.protocol == ipp.ICMP_PROTOCOL:
    # Got an ICMP packet
  elif ipp.protocol == ipp.TCP_PROTOCOL:
    # Got a TCP packet
    tp = ipp.payload
    if tp.dstport == 5000:
        # do something
```

SOME SDN CHALLENGES

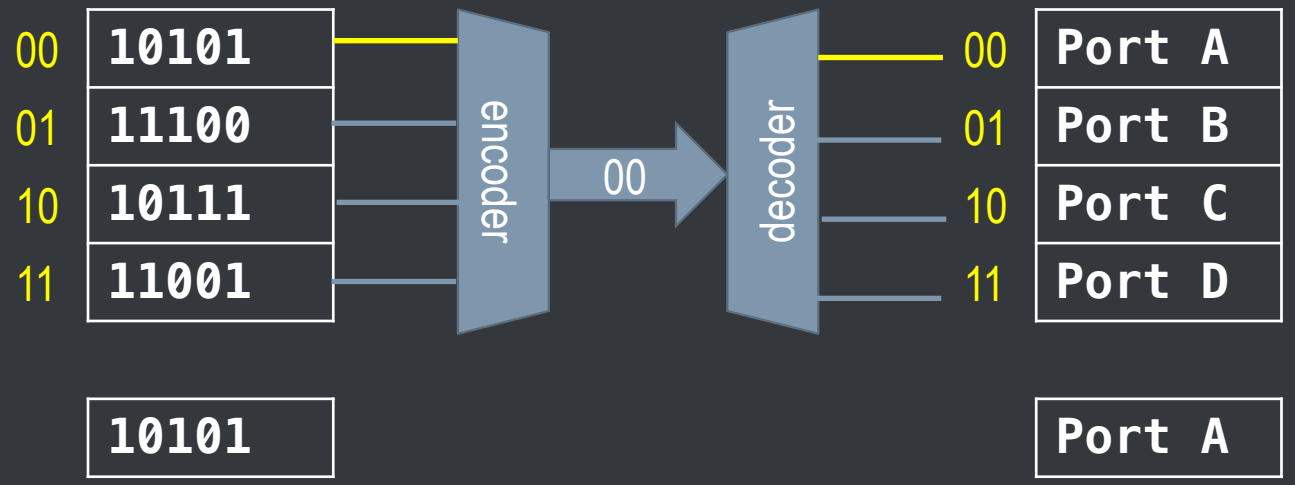
CHALLENGES

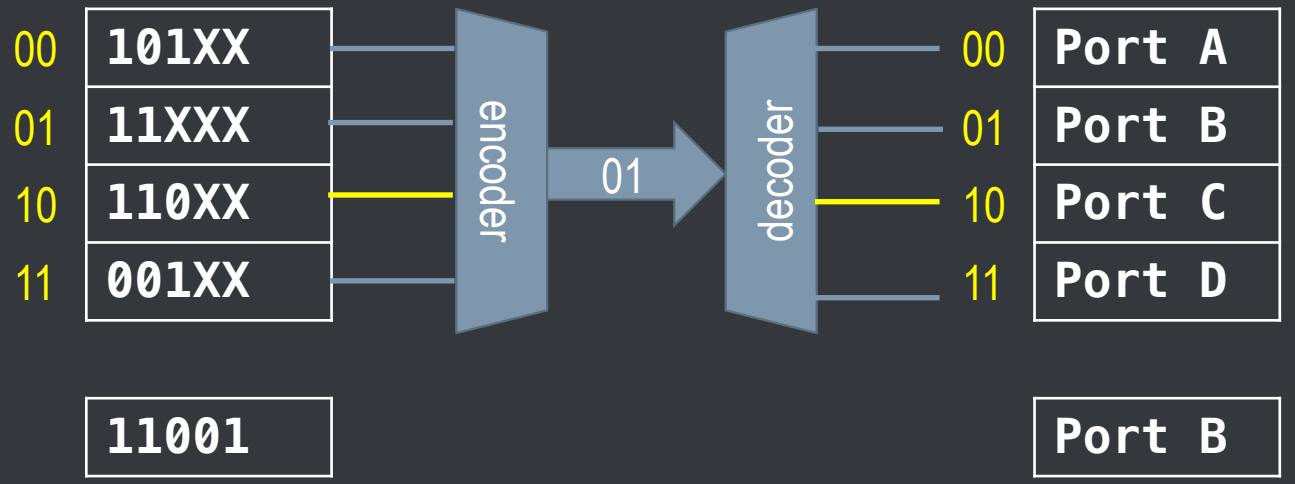
- There are several challenges that must still be faced by SDN-based networks and at different levels
  - Switch design
  - Controllers platforms
  - Scalability
  - Security
  - Performance
  - ...

THE CONTENT-ADDRESSABLE MEMORIES

- The Content-Addressable Memory (CAM) is a kind of memory used in high speed searching applications
- In RAM memories, the user provides a memory address and the RAM returns the data stored at that address. In CAMs, the user provides the word to search for, and the memory returns the address(es) where such a word is (are) found.
- CAMs are very useful for switches: a switch reads the destination MAC address, submit the address to the CAM memory and retrieve the port number where it should be forwarded
- For routers operations, CAM memories are not flexible enough.
  - Routers use the network address and a network mask in order to determine the outgoing port.

ROUTING

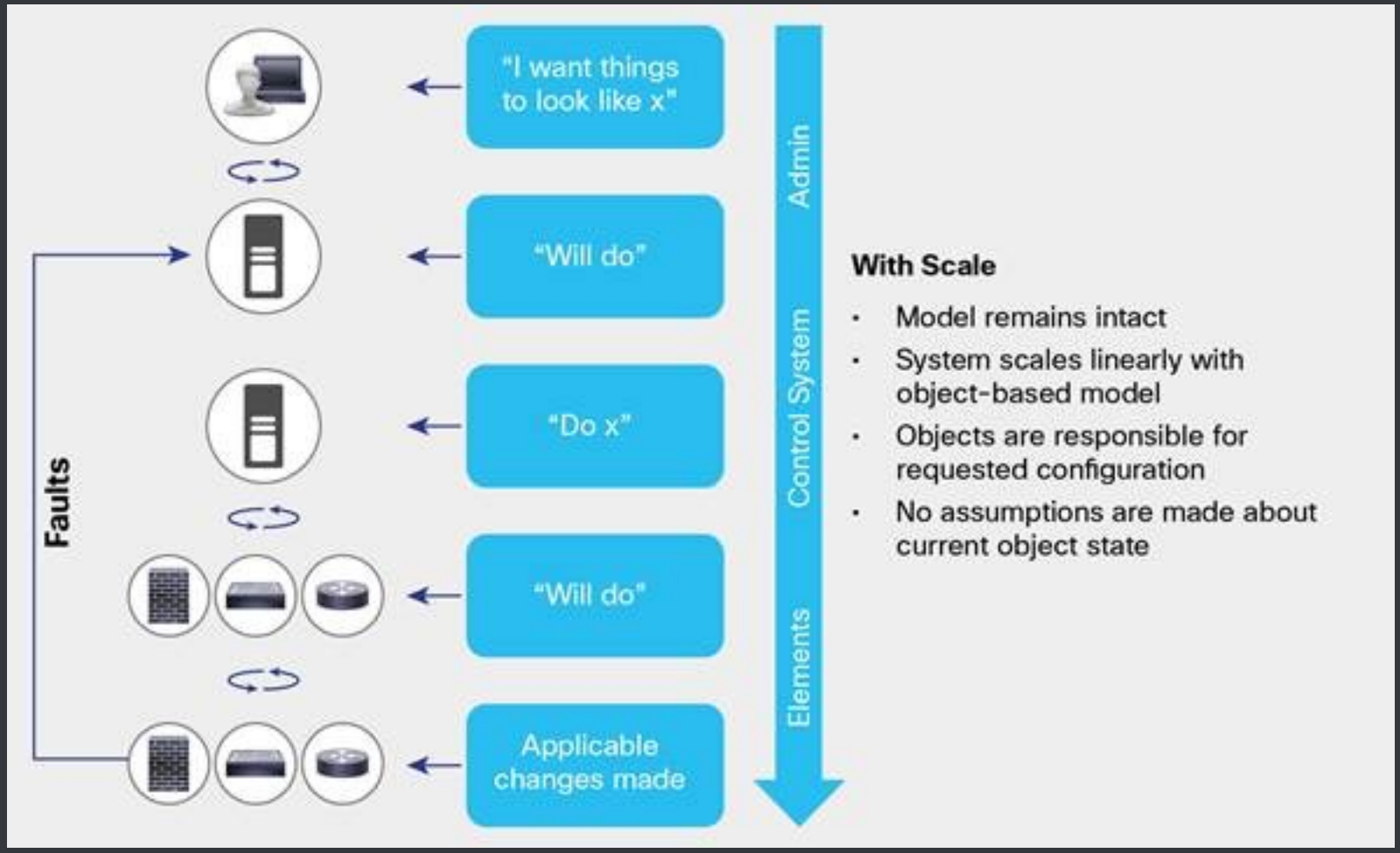

00
01
10
11
10101
11100
10111
11001
encoder
00
decoder
00
01
10
11
Port A
Port B
Port C
Port D
10101
Port A


00
01
10
11
101XX
11XXX
110XX
001XX
encoder
01
decoder
00
01
10
11
Port A
Port B
Port C
Port D
11001
Port B

THE TCAM PROBLEM

- Binary CAMs. A word is compared bit per bit with the content of the CAM
- Ternary CAMs. It implements a “don't care” bit. E.g. it is possible to store the word 1XX, that will match 100, 101, 110, 111.
  - With TCAMs, the router only needs to store the bits related to the network address and insert “don't care” bits in the host address part
  - TCAMs are bigger, expensive and consume more energy than CAMs
  - Size of TCAMs in routers is limited
  - SDN needs TCAMs

OPFLEX



From https://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.doc/_jcr_content/renditions/white-paper-c11-731302_0.jpg