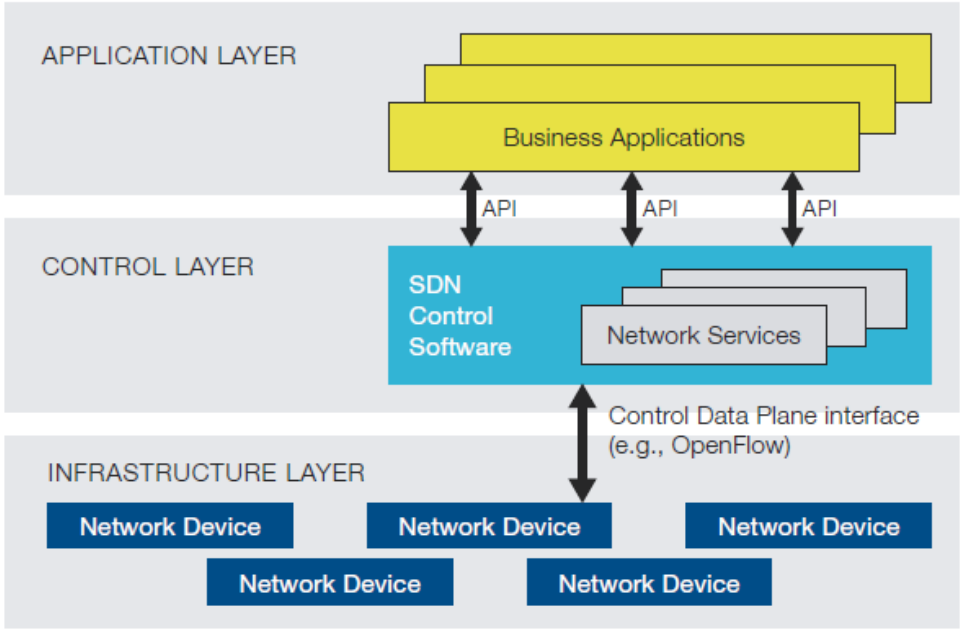
Lecture 7
Software Defined Networking

Design Concept of SDN

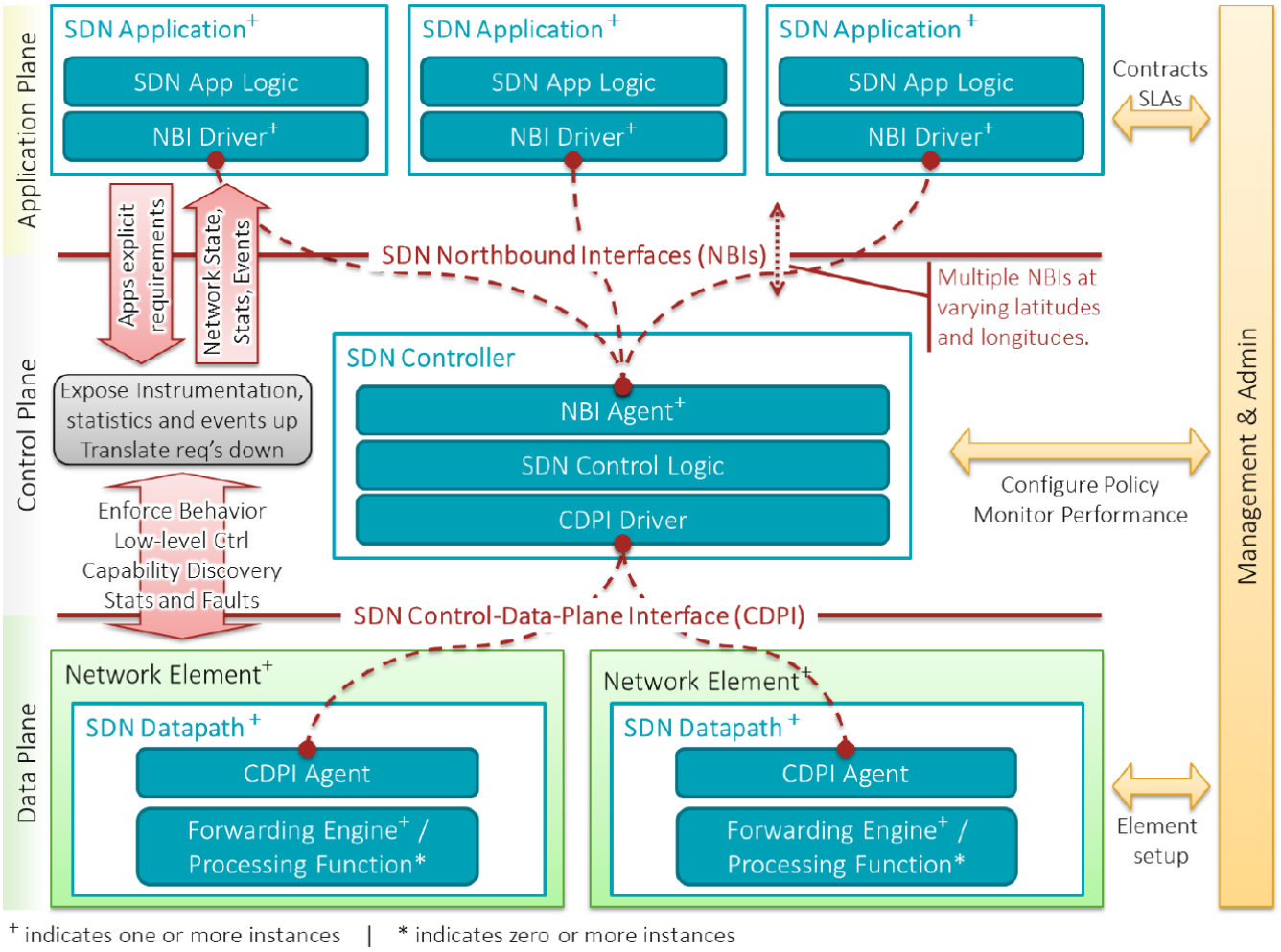
- Network control is decoupled from forwarding (i.e., data plane) and is directly programmable
- “Control” is migrated to an accessible computing device
- Enable the underlying infrastructure to be abstracted for applications and network services
- SDN requires some methods for the control plane to communicate with the data plane. One of such mechanisms is OpenFlow

SDN Architecture

- Control logic is moved from network elements to software controllers
- Enabling control of data, automation and orchestration of network services through a logical SW entity called **controller**
- SDN architecture
  - **Application layer** consists of end-user business applications
  - **Control layer** provides logically centralized control functionality supervising network forwarding behavior through an open interface
  - **Infrastructure layer** contains network elements (NE) and devices providing packet switching/forwarding

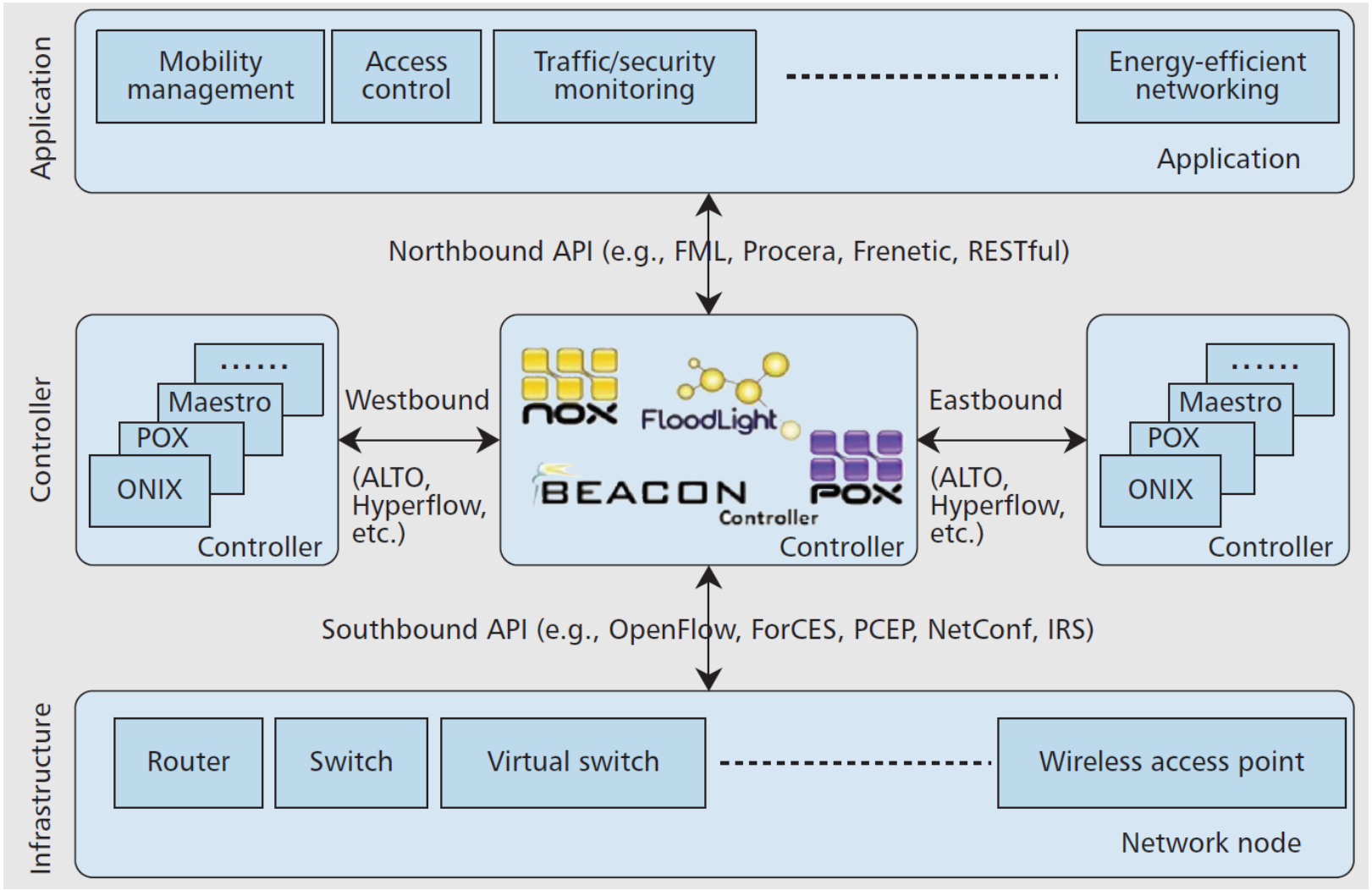


High-Level View of SDN Architecture



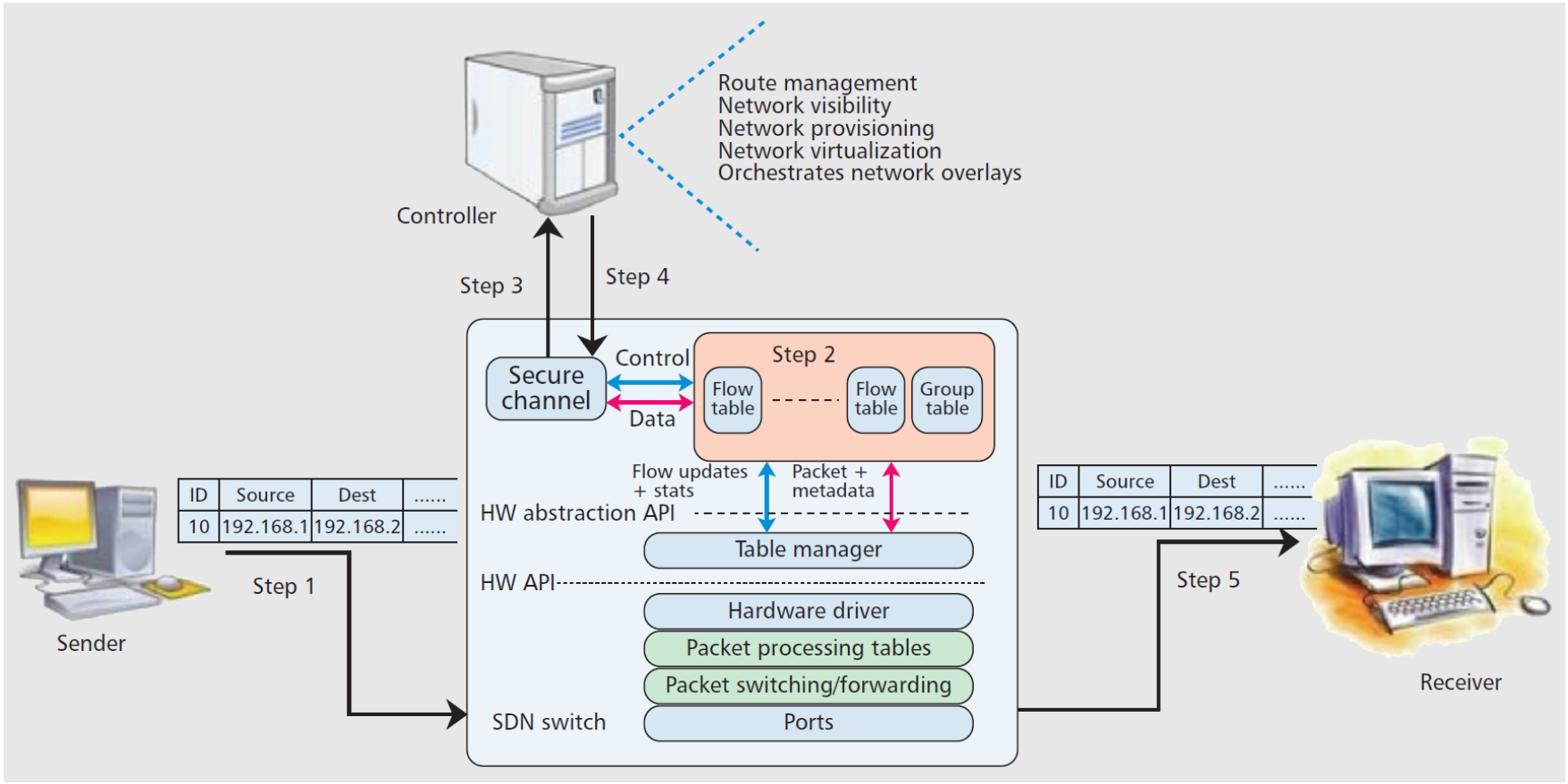
+ indicates one or more instances | * indicates zero or more instances

SDN Functional Architecture



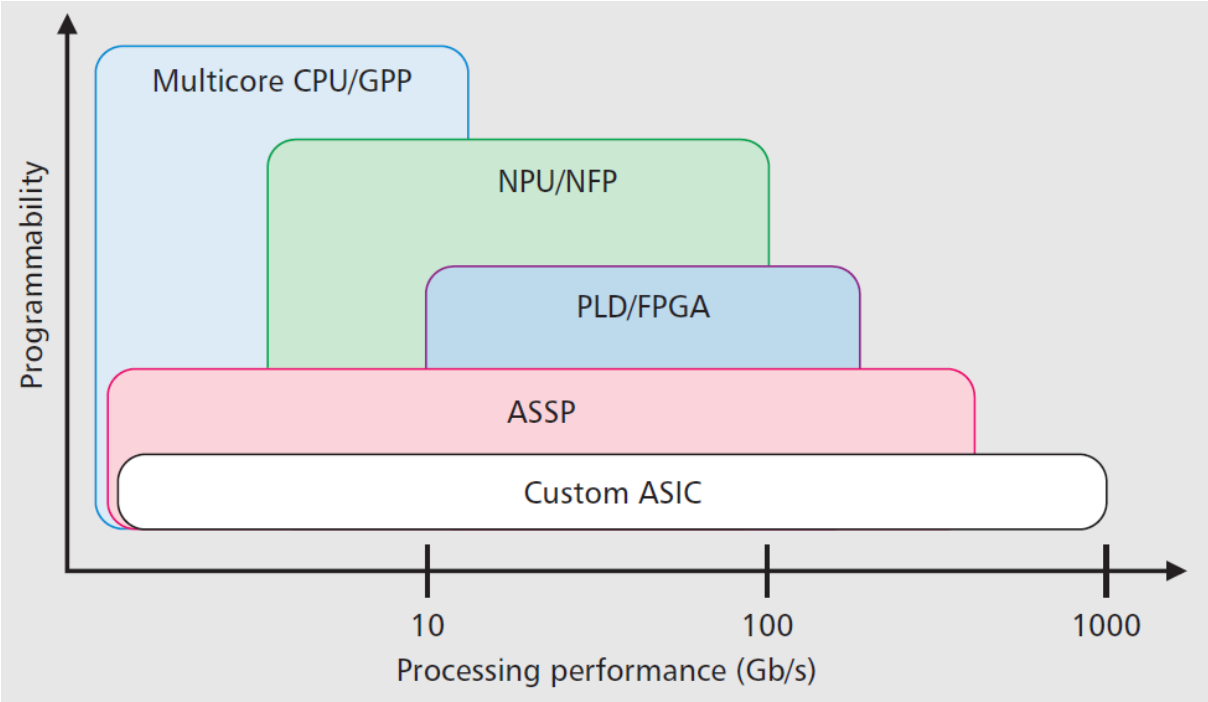
Operations of SDN

- The operations of SDN
  - Table entry hit: perform steps 1 & 5
  - Table entry miss: perform steps 1 ~5



SDN Key Challenges

- Performance vs. flexibility: how can the programmable switch be achieved?
  - A hybrid approach will provide an effective technology solution for SDN
  - e.g., building a platform based on custom-built devices (PLDs/ASSPs) combined with NPUs/NFPs and a CPU/GPP presents a hybrid programmable architecture



PLD: 可程式化邏輯
ASSP: 特殊應用標準產品
ASIC: 特殊應用積體電路

SDN Key Challenges

- Scalability: how can the controller be enabled to provide a global network view?
  - Latency introduced by exchanging network information between multiple nodes and a single controller
  - How do SDN controllers communicate with other controllers using east and westbound APIs
  - The size and operation of the controller back-end database
- Interoperability: how can SDN solutions be integrated into existing networks?
  - IETF path computation element (PCE)
  - Backward compatible with existing IP routing and MPLS

OpenFlow has Two Components

- OpenFlow controller
  - Control one or more switches
  - Formulate flows
  - Program switches
  - Applications may run in the same address space as the controller
  - Directive may come from external applications via REST API
- OpenFlow switch
  - Openflow agent runs on switch to communicate with the controller
  - Process commands from controller to populate the flow table

OpenFlow Controllers

- Variety of controllers
  - NOX/POX
  - Ryu
  - Floodlight
  - OpenDaylight
  - Pyretic
  - Frenetic
  - Procera
  - RouteFlow
  - ONOS
- Considerations
  - Programming language
  - Performance
  - Learning curve
  - User base and support
  - Focus
    - Southbound API support
    - Northbound API
    - Production, education, research?

	NOX	POX	Ryu	Floodlight	ODL
Language	C++	Python	Python	Java	Java
Performance	Fast	Slow	Slow	Fast	Fast
Distributed	No	No	Yes	Yes	Yes
OpenFlow	1.0/1/3	1/0	1/0 to 1.4	1.0	1.0/1.3
Learning curve	Moderate	Easy	Moderate	Steep	steep

Best SDN Controller?

- Suggestion 1
  - POX/ Floodlight/Ryu with mininet if you are interested in running quick experiments. (Easy)
  - ODL or ONOS with mininet if you have little bit more time to invest in your research work. (Medium)
  - ODL or ONOS with OpenStack/devstack - if you are looking for product development or industry use. (Tough)
- Suggestion 2
  - For easy prototyping, you can try Ryu
    Ryu: https://osrg.github.io/ryu/ written in Python, very straightforward and easy to program
  - If you want to get closer to telco deployments, you can try ONOS
    ONOS: https://onosproject.org/ which is a bit more complex, but more realistic (up/down of apps during runtime, great CLI, REST and GUI interfaces, etc.). Moreover, ONOS is supported by the ONF
  - OpenDaylight (ODL) is ok, but it's too complex as a first step, as its architecture is more generalized and difficult to model new modules if you're new into SDN

OpenFlow Switch

OpenFlow is a standard managed by Open Network Foundation (ONF)

- An OpenFlow switch consists of three parts
  - A Flow Table: (flow entry + action) to tell the switch how to process the flow
  - A secure channel: connect the switch to a remote controller
  - OpenFlow protocol: an open and standard way for a controller to communicate with this switch

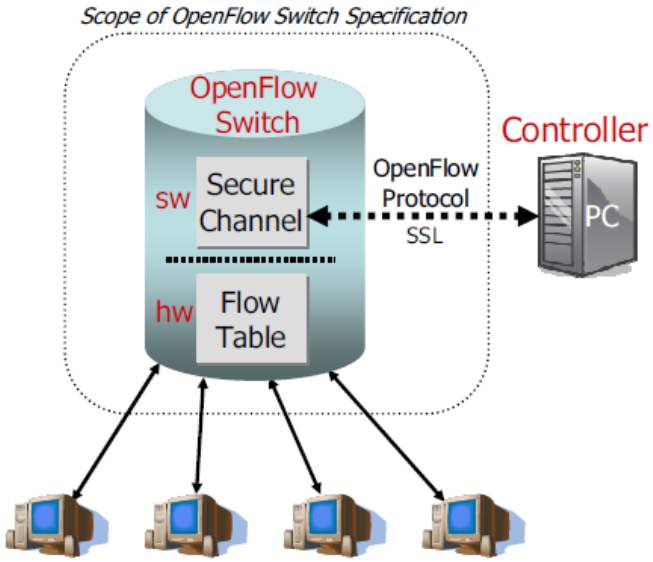
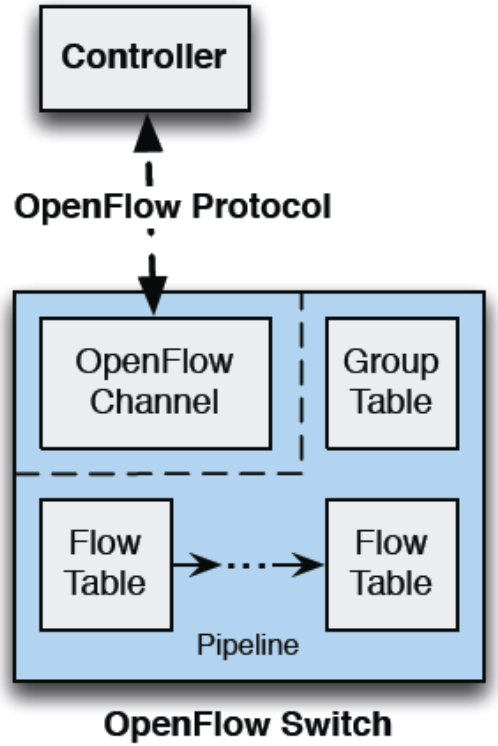


Figure 1: Idealized OpenFlow Switch. The Flow Table is controlled by a remote controller via the Secure Channel.



SDN Switch

- Hardware SDN switch:

- Software SDN switch: Open vSwitch (OVS), https://www.openvswitch.org/
- Simulator: mininet: http://mininet.org/

OpenFlow Switch Data Plane

- Data plane of an OpenFlow switch is composed of:
  - Ports
  - Flow tables
  - Flows
  - Match (classifiers)
  - Modifiers and actions
- Packets are matched to flows in flow tables using the match/classifiers
- Flows contain sets of modifiers and actions which are applied to each packet that it matches

OpenFlow Protocol Overview—Message Types

- Control-to-switch: sent by the controller and may or may not require a response from the switch
  - Features
  - Configuration
  - Modify-state
  - Read-state
  - Packet-out
  - Barrier (controller verifies the switch has processed the flow modifications successfully)
  - Role-request
  - Asynchronous-configuration

OpenFlow Protocol Overview—Message Types

- Symmetric: sent without solicitation, in either direction
  - Hello
  - Echo
- Asynchronous: sent from a switch without a controller's soliciting
  - Packet-in
  - Flow-removed
  - Port-status
  - Error

OpenFlow Protocol Overview—Channel Connection

- One channel established between a controller and an OpenFlow switch
- An OpenFlow controller can have multiple OpenFlow channels
- An OpenFlow switch can have either one channel or multiple channels; however, each channel is to a different controller for reliability consideration

OpenFlow Protocol Overview—Channel Connection

- Connection setup
  - Step 1: OFPT_HELLO
    - Given IP address and port number of the controller so that a SDN switch can connect to
    - Default OpenFlow transport port: 6653
    - Once an OpenFlow connection is established, each side must immediately send OFPT_HELLO message with the highest supported version and version negotiation
  - Step 2: OFPT_FEATURES_REQUEST
- Connection maintenance
  - Handled by the underlying TLS (transport layer security) or TCP connection mechanisms
  - Each connection is maintained separately
  - When main connection is terminated, the auxiliary connections are terminated as well
  - When an auxiliary connection is terminated, the main connection and other auxiliary connections are not be impacted
  - TLS/TCP connection terminated/timeout→ reconnect and reconnection interval increases
  - Flow control is done using the underlying TCP mechanisms

OpenFlow Protocol Overview—Channel Connection

- Connection interruption
  - When? Echo request timeouts→ lost contact with the controller
  - The switch enters immediately either “fail secure mode” or “fail standalone mode”
  - In “fail secure mode”, packets and messages destined to the controllers are dropped; flow entries continue to expire according to their timeouts
  - In “fail standalone mode”, switch acts as a legacy Ethernet switch or router
  - When channel re-established, normal OpenFlow operations resume

OpenFlow Protocol Overview—Multiple Controllers

- Why multiple controllers?
  - Reliability improvement (fail-over)
  - Load balancing
- Switch maintains active connections with all controllers
- Controller role
  - OFPCR_ROLE_EQUAL (default)
    - Full access to the switch
    - Equal to other controller in the same role
    - Asynchronous message reception: yes
    - Controller-to-switch command issuing: yes
  - OFPCR_ROLE_SLAVE
    - Read-only access to switch
  - OFPCR_ROLE_MASTER
    - Full access to the switch
    - Only one controller will serve as the master for an OpenFlow switch
    - Role requested by the controller; other controllers' roles changed to SLAVE done by switch
    - No affect on controllers with role OFPCR_ROLE_EQUAL

OpenFlow Protocol Overview—Multiple Controllers

- Master election
  - Done by controllers themselves
  - Any Slave or Equal controllers can elect itself Master
  - A switch can only have one controller in Master state, multiple controllers in Equal state, multiple controllers in Slave state
  - Dynamic role transition: need generation_id
  - Generation_id is a monotonically increasing counter

OpenFlow Protocol Overview—Auxiliary Connections

- Why auxiliary connections?
  - Improvement of switch processing performance
  - Exploitation parallelism
- Initiated by switch; may be configured by controller
- Switch side: main and auxiliary connections have the same source IP address and same Datapath ID, and have different transport [such as TLS, TCP, DTLS (datagram transport layer security), or UDP].
- Controller side: same destination IP address and port number
- Both parties accept any OpenFlow message types and sub-types on all connections

OpenFlow Protocol Overview—Auxiliary Connections

- Usage guidelines on controller
  - All OpenFlow controller requests which are not Packet-Out should be sent over the main connection
  - Connection maintenance messages (hello, echo request,..) should be sent on the main connection and on each auxiliary connection as needed
  - All Packet-out messages containing a packet from a Packet-In message should be sent on the connection where the Packet-In came from
  - All other Packet-Out messages should be spread across the various auxiliary connections using a mechanism keeping packets of a same flow mapped to the same connection
  - If the desired auxiliary connection is not available, the controller uses the main connection

OpenFlow Protocol Overview—Auxiliary Connections

- Usage guideline on switch
  - All OpenFlow messages which are not Packet-In should be sent over the main connection
  - All Packet-In messages spread across the various auxiliary connection using a mechanism keeping the packets of a same flow mapped to the same connection

Flow Entry in Flow Table

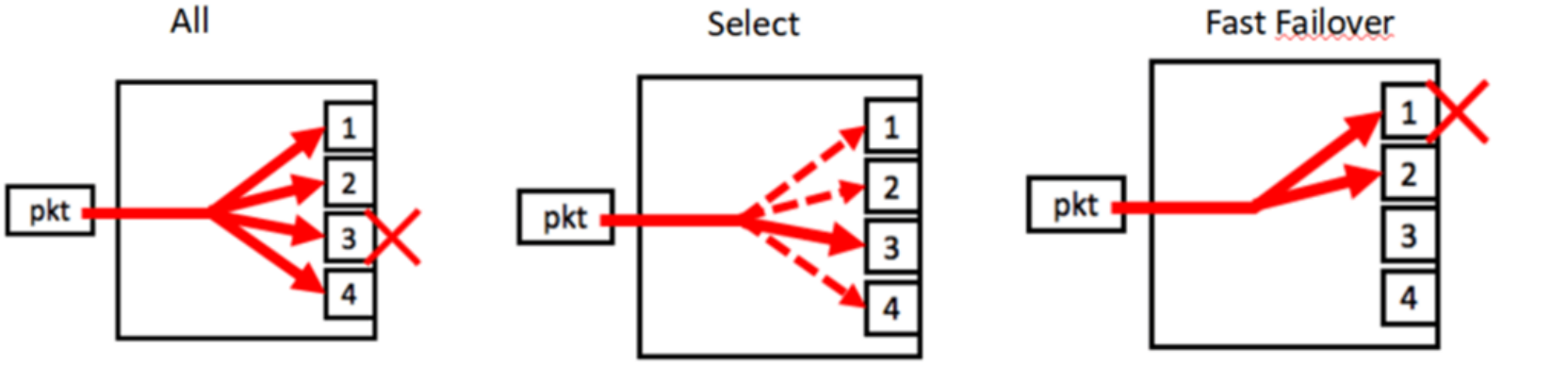
- Packet header
  - 10-tuple in first generation "Type 0" switch
  - Each field can be a wildcard

In	VLAN	Ethernet			IP			TCP	
Port	ID	SA	DA	Type	SA	DA	Proto	Src	Dst

- Associated action
  - Forward packet to port(s)
  - Encapsulate and forward to the controller (typically for the first packet in a flow through the secure channel)
  - Drop the packet
  - Send to the normal processing pipeline (group table)
- Statistics
  - # of packet, bytes for each flow,…etc.

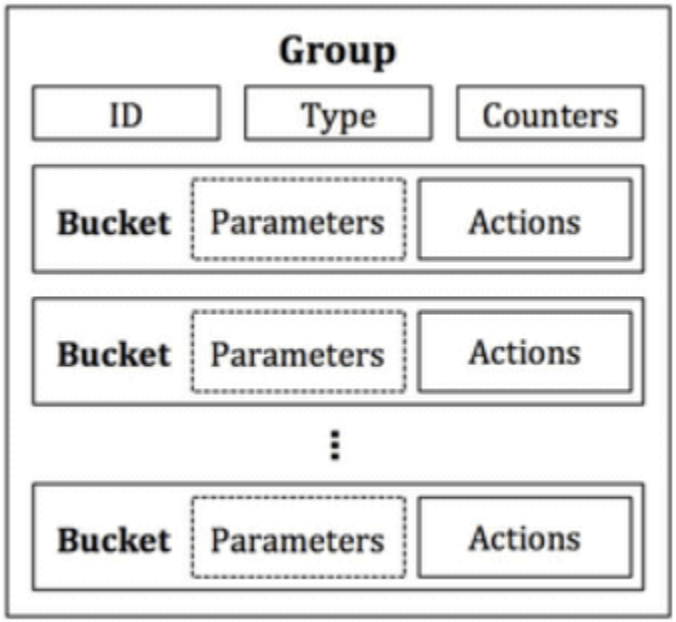
Group Table (1)

- The group table supports the following 4 group types
  - All: executes all the buckets in the group; mostly used for flooding and multicasting
  - Indirect: executes one defined bucket in the group. The action taken by this group type is sending packets to the next hop
  - Select: executes one bucket in the group. The action bucket is chosen by a switch-defined algorithm, such as round robin or hashing (for example, load sharing)
  - Fast failover: executes the first live bucket, used in cases such as redundancy



Group Table (2)

- A group table consists of group entries. The counters in the following table are available in a group entry
  - Group Identifier: A 32-bit unsigned integer uniquely identifying the group
  - Group type: Determines group semantics
  - Counter: Number of packets processed by a group
  - Action bucket: Ordered list of action buckets, where each action bucket contains a set of actions to execute and associated parameters

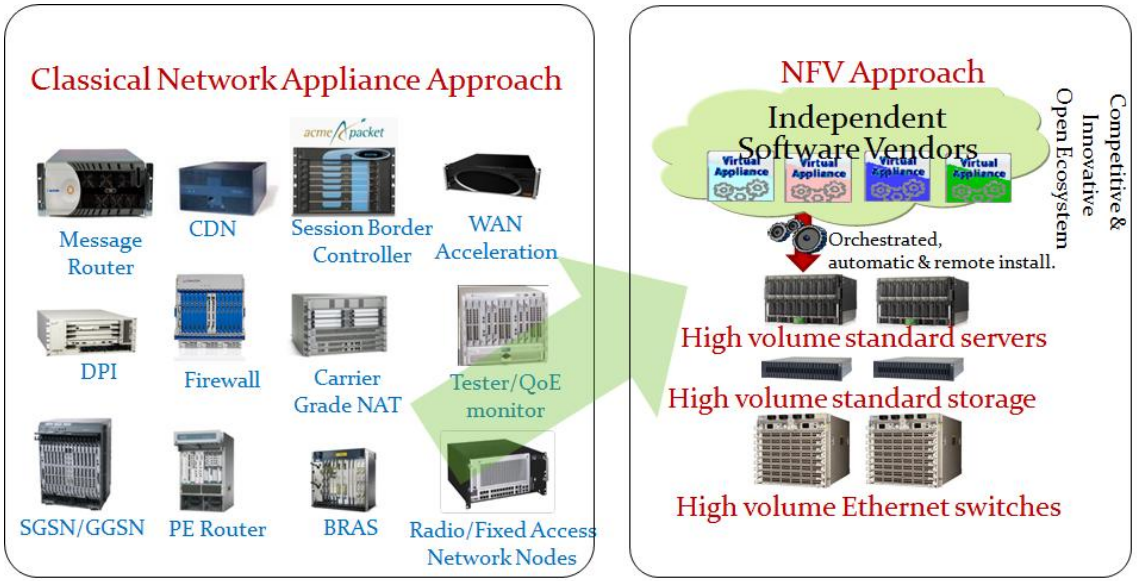


Using Routing As An Example

- Reference link: https://sdn-lab.com/2014/12/25/shortest-path-forwarding-with-openflow-on-ryu/
- The network application will be organized
  - Topology discovery
  - Network view construction
  - Forwarding

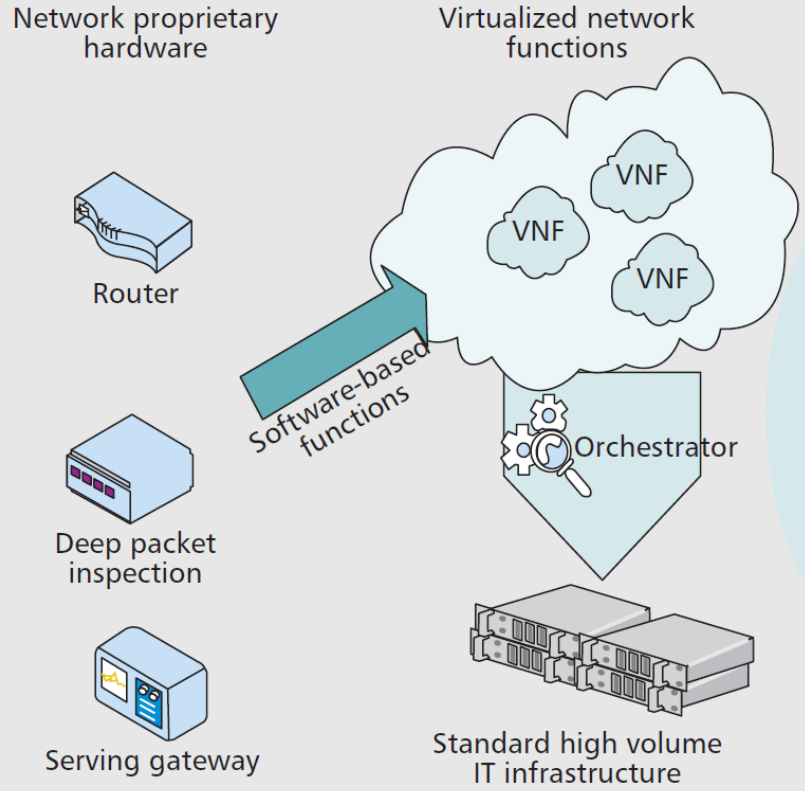
Network Function Virtualization (NFV)

- Transferring network functions from dedicated hardware appliances to software-based applications running on commercial off-the-shelf (COTS) equipment
- These applications are executed and consolidated on standard IT platforms like high-volume servers, switches, and storage



[3] Hassan Hawilo, et al., "NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)," IEEE Network, Nov./Dec. 2014, pp. 18-26

NFV v.s. SDN

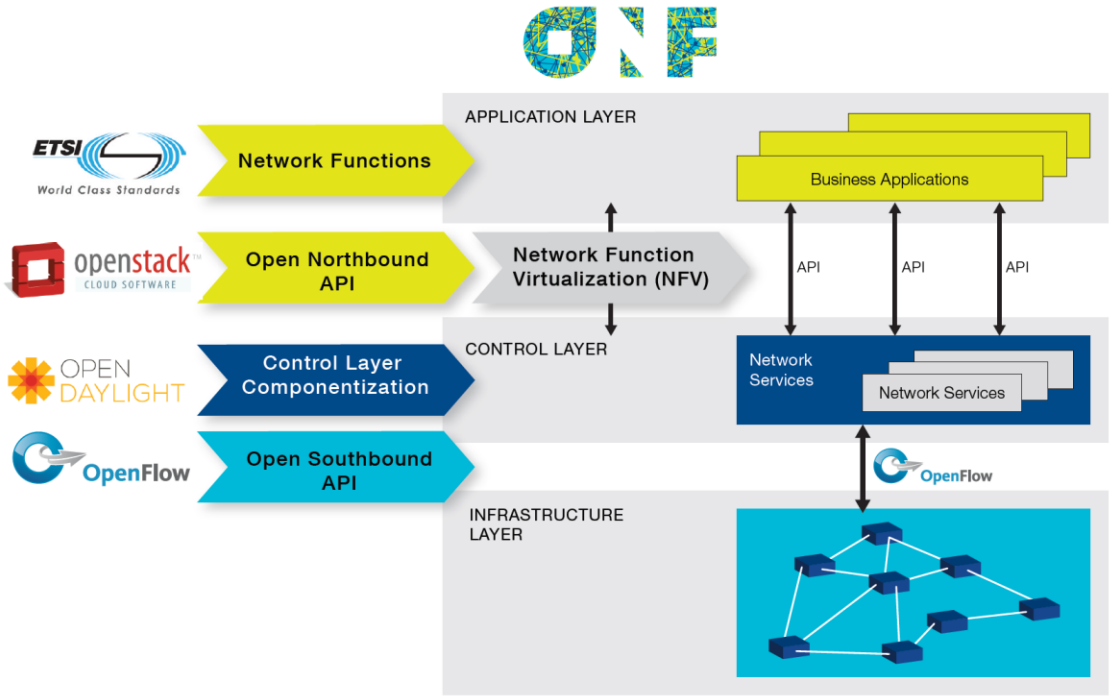


Software-defined networking (SDN) and NFV are two different independent technologies; however, they are complementary to each other.

NFV		SDN
1) NFV is the concept of transferring the network functions from dedicated hardware appliances to software-based applications. 2) NFV decouples the network functions from the proprietary hardware without affecting the functionality.	1) SDN serve NFV by providing the programmable connectivity between VNFs; these connections can be managed by the orchestrator of the VNFs which will mimic the role of the SDN controller [6]. 2) NFV serve SDN by implementing its network functions in a software manner on a COTSs servers. It can virtualize the SDN controller to run on cloud, which could be migrated to best fit location according to the network needs.	1) SDN is concerned in network functionalities it decouples the control and data planes [6]. 2) SDN provides centralized controller and network programmability [6].

NFV v.s. SDN

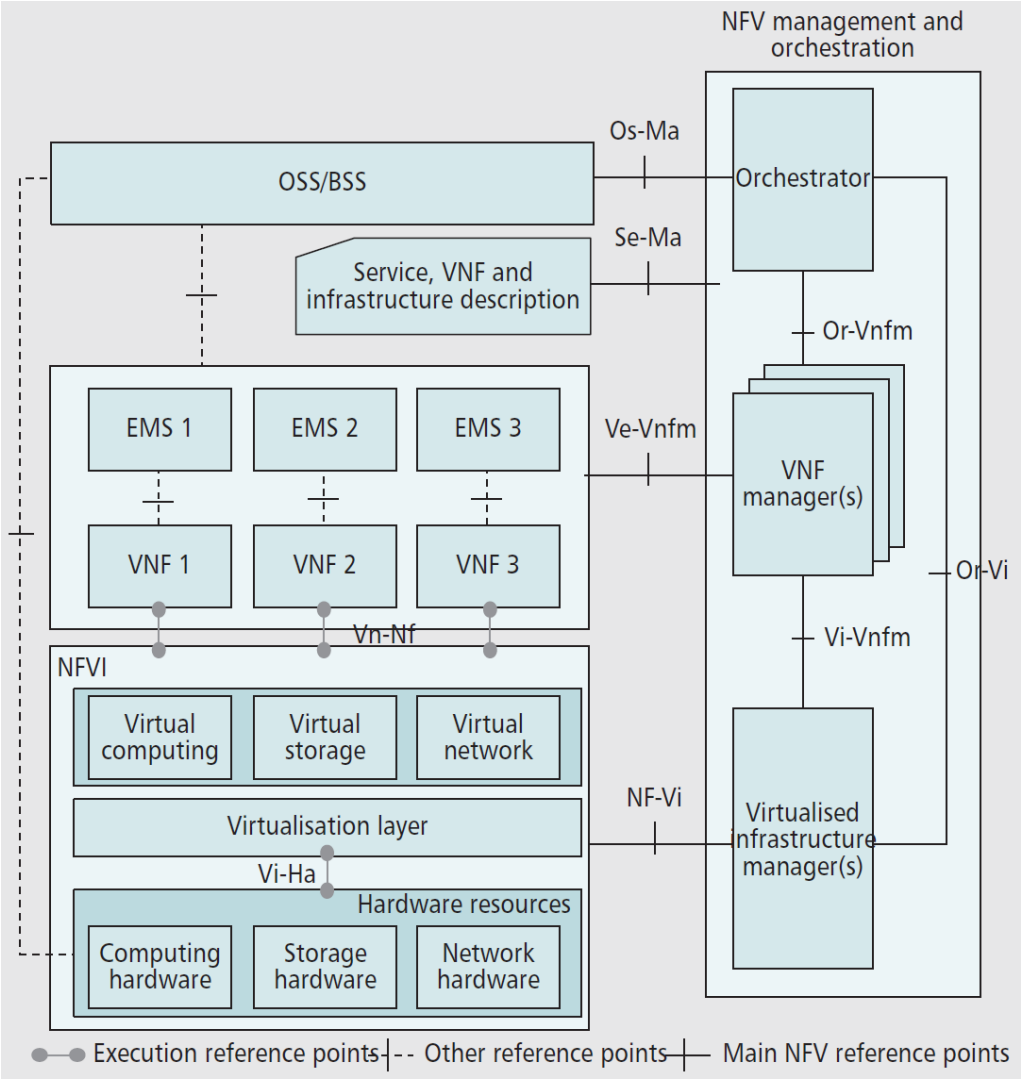
- Both NFV and SDN architectures are optimized for the dynamic cloud environment at carrier scale
- Both NFV and SDN seek to leverage automation and virtualization to achieve greater agility while reducing both OpEX and CapEX
- NFV is intended to optimize the deployment of network functions (e.g., firewalls, load balancers)
- SDN is focused on optimizing the underlying networks



NFV Framework

EMS: element management system
OSS: operation support system
BSS: business support system

- The basic components of virtualized platforms where NFV is deployed are:
  - Physical server: the bare metal machine that has all the physical resources such as CPU, storage, and RAM
  - Hypervisor: the software that runs and manages physical resources
  - Guest virtual machine: a software that emulates the architecture and functionalities of a physical platform on which the desired application is executed



NFV Challenges

- Security: should obtain a security level close to that of a proprietary hosting environment
  - Virtualization environment domain (hypervisor)
    - Solution: isolation of served VM space with access provided only with authentication controls; data communications and VM migration run in a secure environment
  - Computing domain (CPU, memory, etc)
    - Solutions: secured threads, erasure of private and shared memory allocations before reallocation, and data encryption with exclusive access to the NFV
  - Infrastructure domain (networking, e.g., vSwitches, and shared physical NICs))
    - Solution: usage of secured networking techniques (TLS, IPSec, or SSH)
  - Application domain

NFV Challenges

- Computing performance
  - Hardware aspect: CPU clock rate, cache memory size, memory bandwidth, etc
  - Software aspect
    - Multithreading to be executed over multiple cores or possibly scaled over different hosts
    - Independent memory structure to avoid OS deadlocks
    - Own network stack implementation per VNF
    - Avoidance of implementing network stacks in OS
    - Direct access to input/output interfaces
    - Processor affinity techniques to take advantage of cache memories

NFV Challenges

- Interconnection of VNFs
  - Through Vswitch: suitable for the case that two VNFs are on the same physical server and the same LAN
  - Through Vswitch and NIC: suitable for the case that two VNFs are on the same physical server while different LANs
    - The connection passes through the 1st vSwitch to the NIC, then to the external switch, and back again to the same NIC
    - This NIC forwards the connection to the vSwitch of the 2nd LAN and then to the VNF
  - Through two NICs: suitable for the case that two VNFs are on different physical servers
    - The connection passes through the 1st vSwitch to the NIC, then to the external switch
    - The switch forwards the connection to the NIC of the desired server
    - Finally, this NIC forwards it to its internal vSwitch and then to the destination VNF

NFV Challenges

- Operation and management
  - VNFs can be implemented as simple drag-and-drop operations in the orchestration management system
  - Orchestration management system provides north-bound interactions to interact with VNFs for managing and providing accesses
  - Orchestration management system provides south-bound interactions to interact with NFVI, request information about VNF software images, etc
- Carrier-grade service assurance
  - Resilience to failure, service continuity, and service assurance
  - Ways to achieve the goals include VNF automation, performance monitoring, and real-time resource scaling