Mobile Communications and Orchestration

PART 2/2 – INTRODUCTION TO THE INTERNET OF THINGS

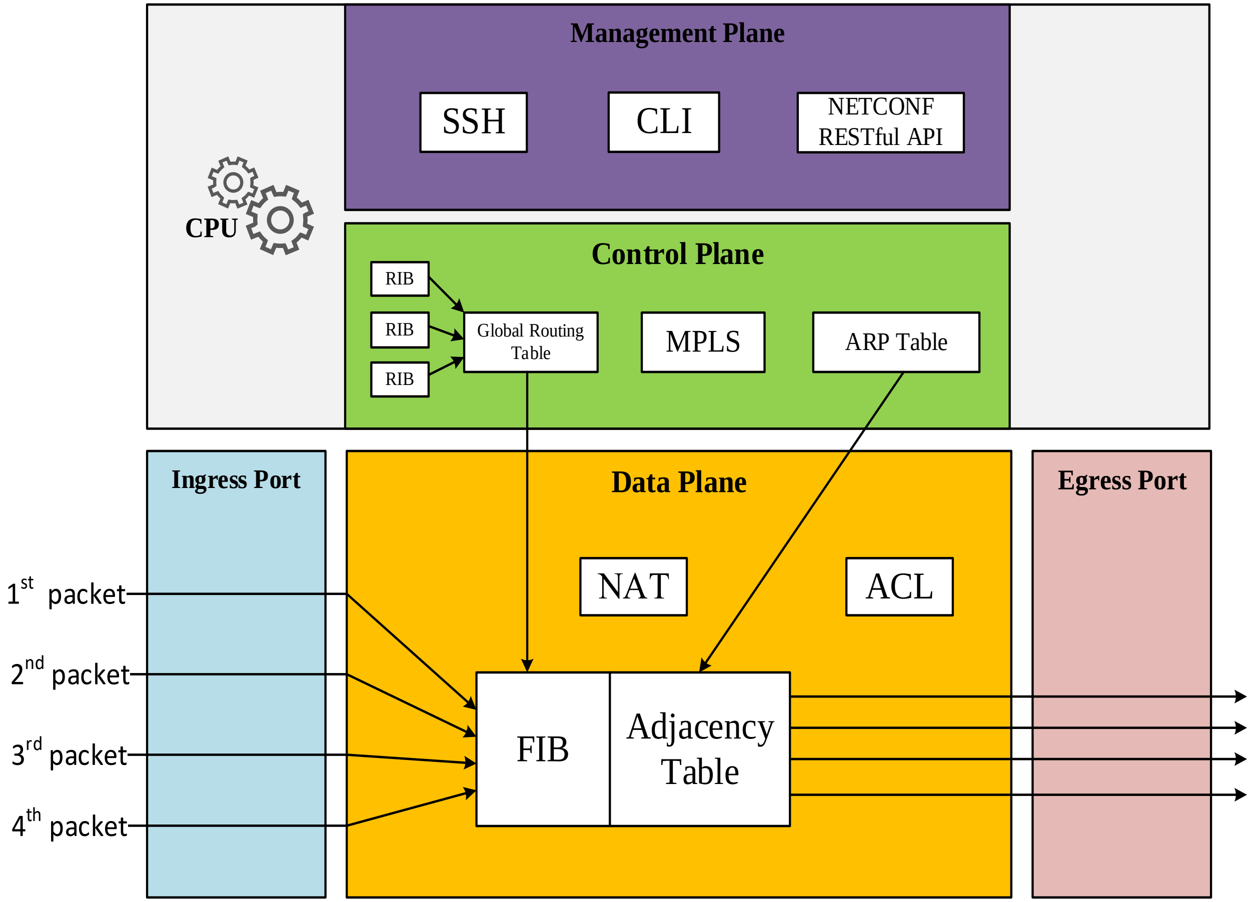
UNIVERSITY OF WESTERN MACEDONIA

CHRISTOS DALAMAGKAS – CDALAMAGKAS@UOWM.GR

PROF. PANAGIOTIS SARIGIANNIDIS – PSARIGIANNIDIS@UOWM.GR

Outline

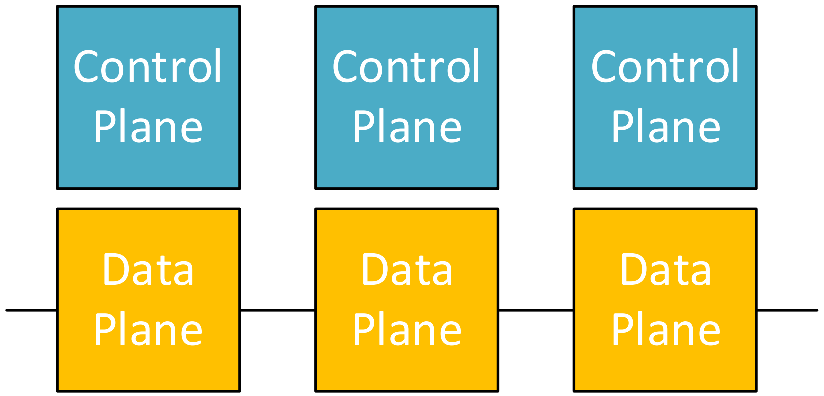
- Introduction to SDN
  - SDN architecture
  - Southbound protocols (CPSI / MPSI)
  - SDN controllers and SDN switches
- Introduction to NFV
  - NFV architecture
  - SDN vs NFV
  - Orchestration
- SDN/NFV-enabled 5G management
- SDN Controller Demo



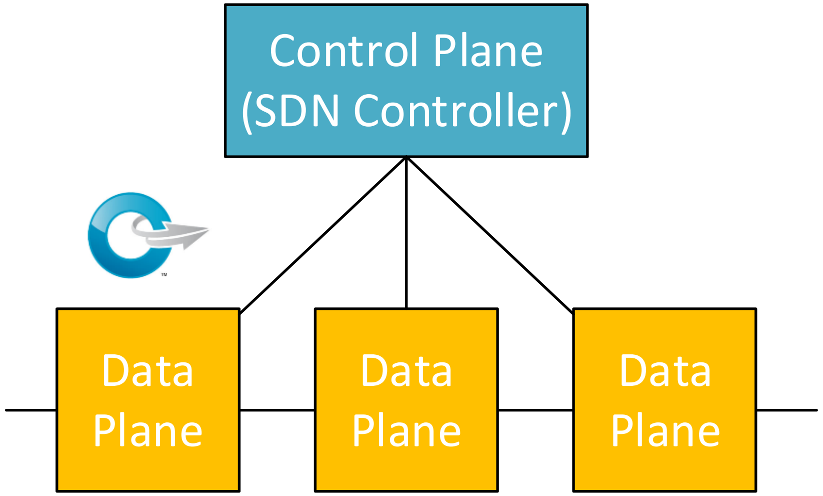
Traditional router architecture

- Data plane
  - Holds the Forwarding Information Base (FIB) and Adjacency Tables – Mirrors of the control plane
  - NAT (Network Address Translation) and ACL (Access Control List) for low-level decisions
  - Takes place in ASICs (Application-specific integrated circuits)
- Control Plane
  - Takes place in CPU
  - Routing protocols that build the Global Routing Table
  - ARP, MPLS and other protocols that manipulate FIB and Adjacency Table
- Management Plane
  - Command Line Interface
  - Secure Shell
  - Vendor APIs (e.g. Aruba REST API) and management protocols (SNMP, NETCONF/RESTCONF etc)
- Each device comes with a complete/independent implementation of all layers
- This causes **scalability** and **flexibility issues**

Traditional Network



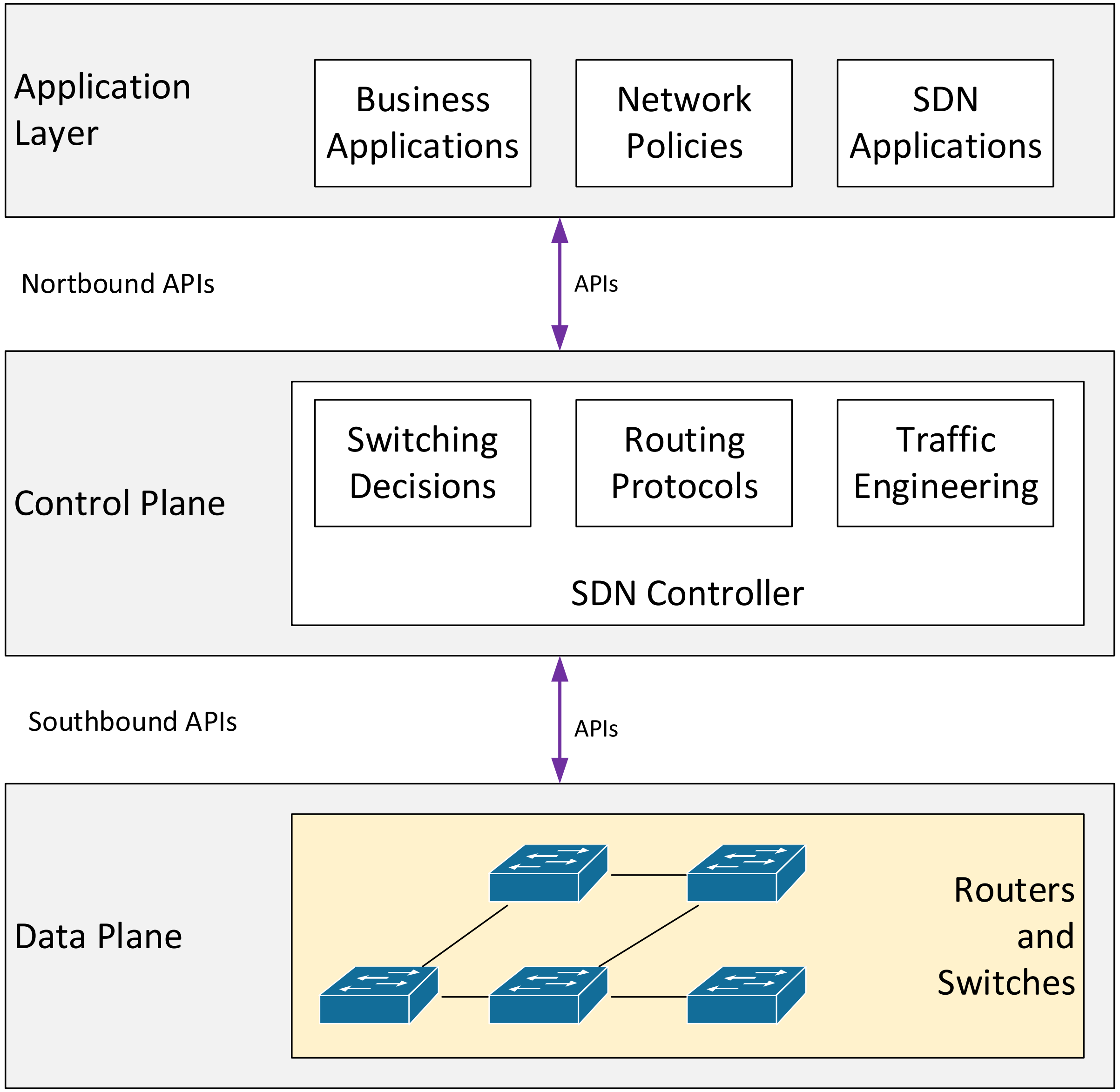
Software-Defined Network



Source: Cisco Networking Academy

SDN Architecture

- Software-Defined Networking changes radically the traditional architecture by decoupling the architectural planes:
  - Control plane has been decoupled – can run on a server as a VM
  - The control plane can be written from scratch
  - Decoupled Control Plane = SDN Controller
  - The Control Plane can instruct the Data Plane what to do with each incoming or outcoming frame



Source: Cisco Networking Academy

SDN Architecture

- Three Main Layers:
  - **Data Plane (DP)**: Where packet forwarding is implemented, holding the forwarding tables (or flow tables)
  - **Control Plane (CP)**: Manipulation of forwarding tables, Routing protocols, drawing of network topology
  - **Application Plane**: Applications implementing business logic
  - Control plane provides basic network functions, whereas application plane utilises abstracted APIs of the control plane to implement business logic
  - Management Plane is vertical and provides configuration management to CP and DP (e.g. SSH or monitoring via SNMP etc

SDN APIs

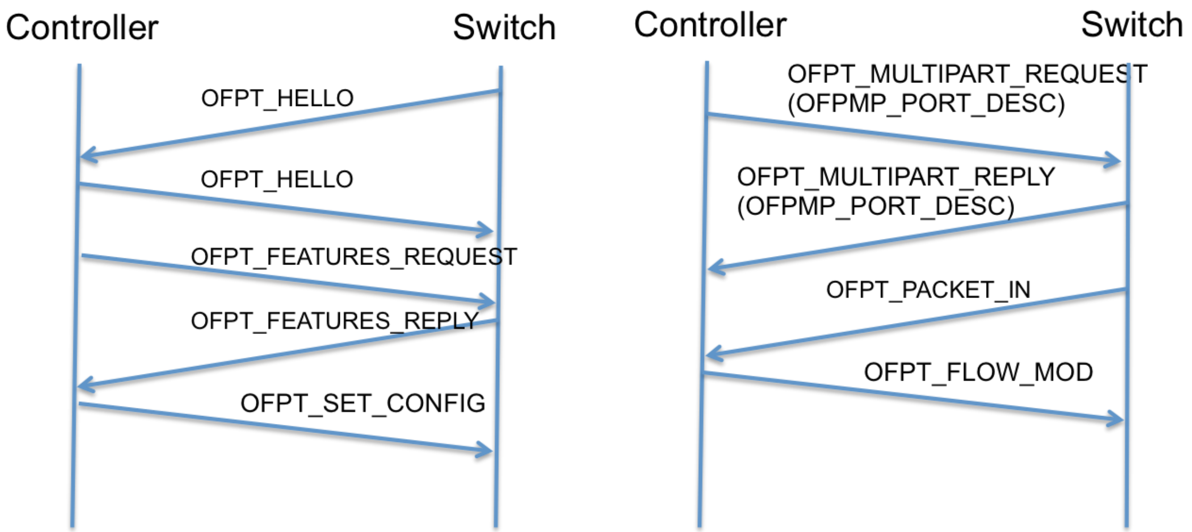
- Planes are communicating using dedicated open APIs
- Southbound API: Communication between Control Plane and Data Plane
  - Control Plane – Southbound Interface (**CPSI**): Directly manipulates the data plane and the forwarding plane
    - OpenFlow
  - Management Plane – Southbound Interface (**MPSI**): Applies configuration management to the network elements
    - NETCONF/RESTCONF
    - SNMP
- Northbound APIs (NBIs): Third-party applications use NBI to communicate with the Control Plane
  - Non standardised REST APIs over HTTP

OpenFlow

- The dominant CPSI protocol – Version 1.3 is the most widely implemented version
- OpenFlow standard is developed by the Open Networking Foundation (ONF)
- SDN switches (almost) replace routers and legacy switches
- Forwarding is done **per flow**
  - Flow = A set of specific matching fields and actions, defined by the SDN Controller
  - Flow rule is a matching condition, based on fields of the incoming packet
    - Matching can be up to TCP/IP Layer 4 fields
    - Each role has an action – what is being done after matching
    - Common actions are DROP or OUTPUT (forward) to a specific port
- QoS and security policies can be enforced per flow
- OpenFlow enables flow-based communication in datacentres and 5G networks

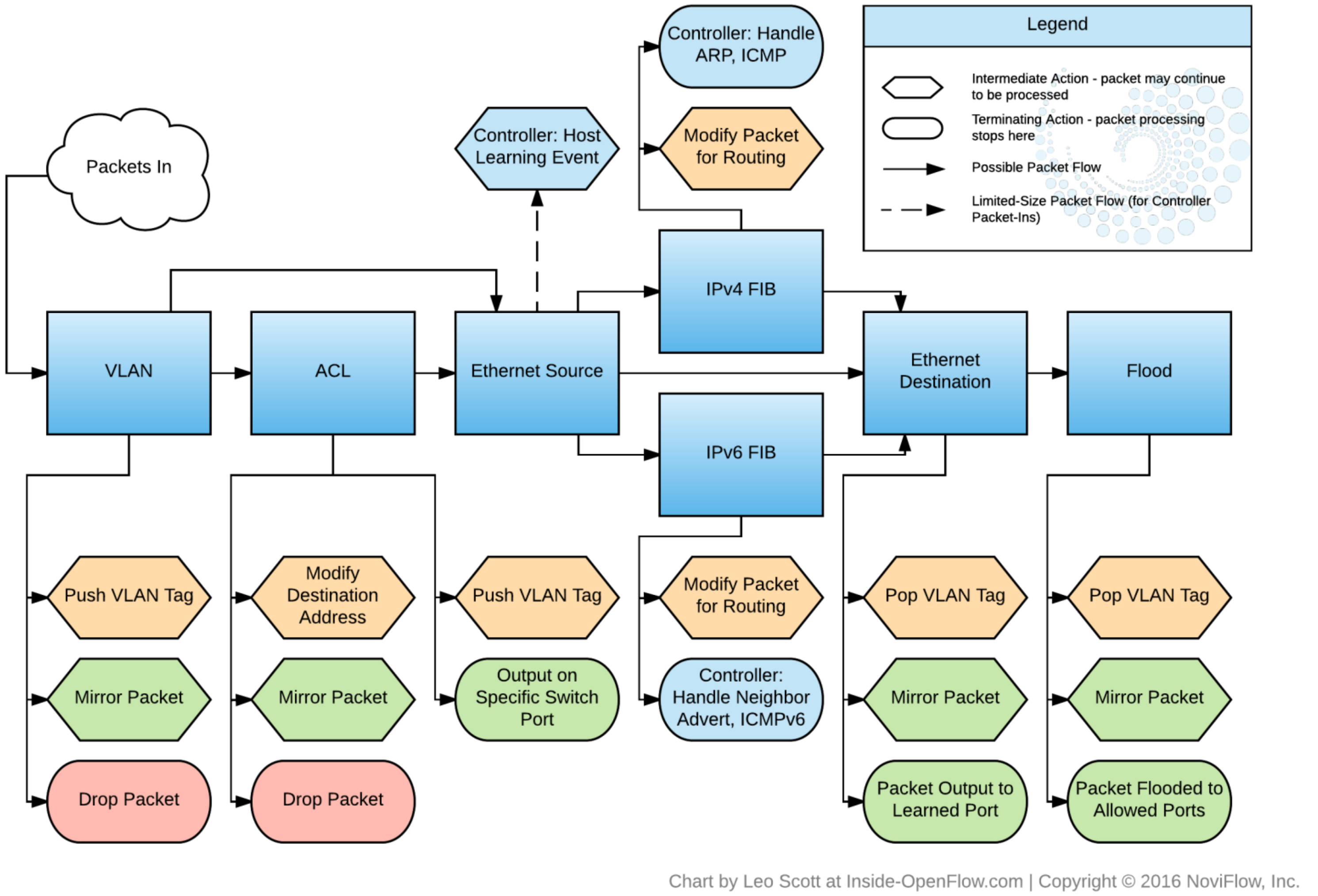
OpenFlow

- Basic OpenFlow messages:
  - **OFPT_HELLO** is a message to initiate an OpenFlow session
  - **OFPT_FEATURES_REQUEST** is used to retrieve basic switch features (e.g. number of tables, buffers)
  - **OFPT_MULTIPART_REQUEST** is used to send one or more requests about switch information
    - OFPMP_PORT_DESC provides information about switch ports
    - OFPMP_TABLE_FEATURES provides information about existing tables and capabilities of each one
  - **OFPT_PACKET_IN** encapsulates an ingress packet that the switch does not know how to handle. The switch can read the packet and decide the appropriate action
  - **OFPT_FLOW_MOD** is a command message that inserts or modifies a network flow
  - **OFPT_PACKET_OUT** encapsulates a complete packet that the controller wants the switch to forward



OpenFlow Session Establishment and Normal Operation

Faucet Packet Flow



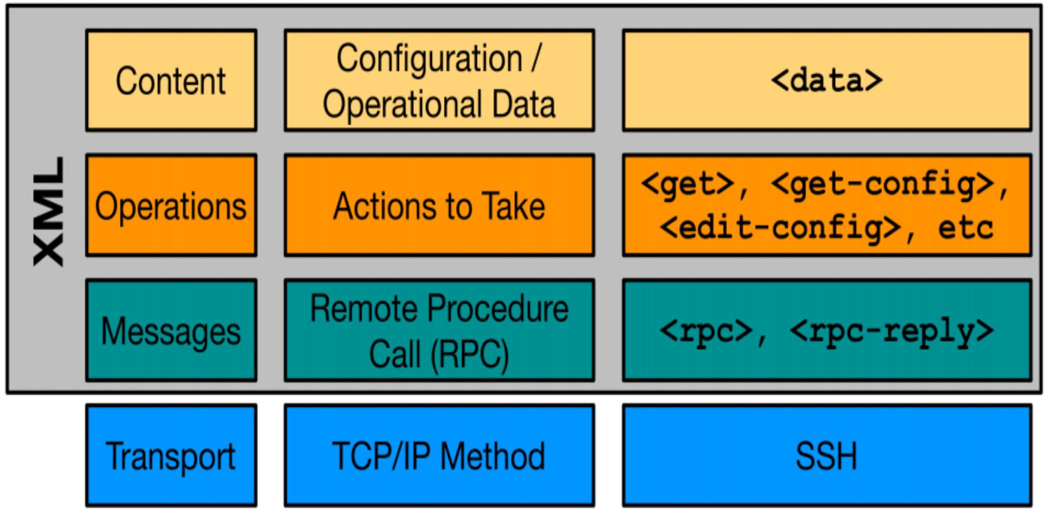
TCAM = Ternary Content-Addressable Memory

Custom Pipelines with OpenFlow

- OpenFlow allow us to define complex pipelines to accelerate table lookups
- Multiple tables can be created on the SDN switch programmable hardware, using OpenFlow
  - TCAMs: Can match specific fields in any TCP/IP layer
  - HASH: Matches a group of fields in L2
- Depending on the defined capabilities of a table (matchable fields, max number of flow entries), consumes the corresponding resources

NETCONF

- Provides interface to install, manipulate and delete configurations on network devices (MSPI protocol)
- Published in 2011 as RFC6241 - Information is structured according to the YANG data model and communication is based on RPCs over a secure protocol
- 4 Conceptual Layers, depicted bellow



- Content Layer: Includes the payload of a query or a command. The validity of this layer is subject to the YANG standard.
- Operations Layer: Determines the type of query or action performed by NETCONF (get, get-config, edit-config, copy-config, delete-config, lock, unlock, close-session, kill-session)
- Messages: Type of NETCONF message (client rpc, rpc-reply or hello message)
- Secure Transport: Defines the secure protocol over which NETCONF msgs are transmitted (SSH, TLS, HTTPS)

https://www.cbtnuggets.com/blog/certifications/cisco/ccnp-enterprise-what-are-yang-netconf-restconf

NETCONF Operations

Operation	Description
`<get>`	Retrieve running configuration and device state information
`<get-config>`	Retrieve all or part of specified configuration data store
`<edit-config>`	Loads all or part of a configuration to the specified configuration data store
`<copy-config>`	Replace an entire configuration data store with another
`<delete-config>`	Delete a configuration data store
`<commit>`	Copy candidate data store to running data store
`<lock> / <unlock>`	Lock or unlock the entire configuration data store system
`<close-session>`	Graceful termination of NETCONF session
`<kill-session>`	Forced termination of NETCONF session

RESTCONF

- HTTP-based configuration protocol, supporting both JSON and XML – Based too on YANG
- Defined by RFC8040
- RESTCONF is NOT a replacement of NETCONF – Provides a RESTful HTTP interface utilised to query and configure devices with NETCONF configuration datastores
- Often used with Python and Requests library for automation/programmability tasks

RESTCONF	NETCONF
OPTIONS	none
HEAD	<get-config>, <get>
GET	<get-config>, <get>
POST	<edit-config> (nc:operation="create")
POST	invoke an RPC operation
PUT	<copy-config> (PUT on datastore)
PUT	<edit-config> (nc:operation="create/replace")
PATCH	<edit-config> (nc:operation depends on PATCH content)
DELETE	<edit-config> (nc:operation="delete")

RESTCONF Example

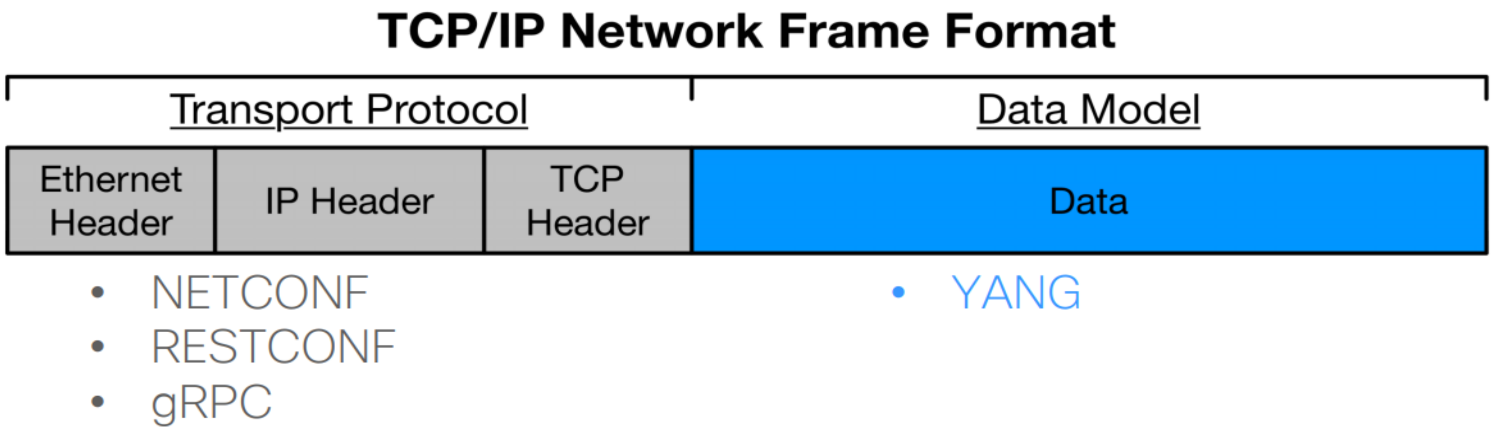
```
def createOSPF(cfg):
        """ Create network statements in OSPF process for the provided IPv4 prefixes
                limiting to interface IP as a best practice e.g.
                        'network 192.168.100.1 mask 0.0.0.0 area 0'
                not 'network 192.168.100.0 mask 0.0.0.255 area 0'
        """

        uri = "/api/running/native/router/ospf"
        networks = list()
        for prefix in cfg['v4prefixes']:
                e = dict(zip(['ip', 'mask', 'area'], [str(prefix.ip), '0.0.0.0', cfg['area-id']]))
                networks.append(e)
        return processRequest(cfg, uri, {'ned:ospf': {'id': cfg['process-id'], 'network': networks}}, 'PATCH')
```

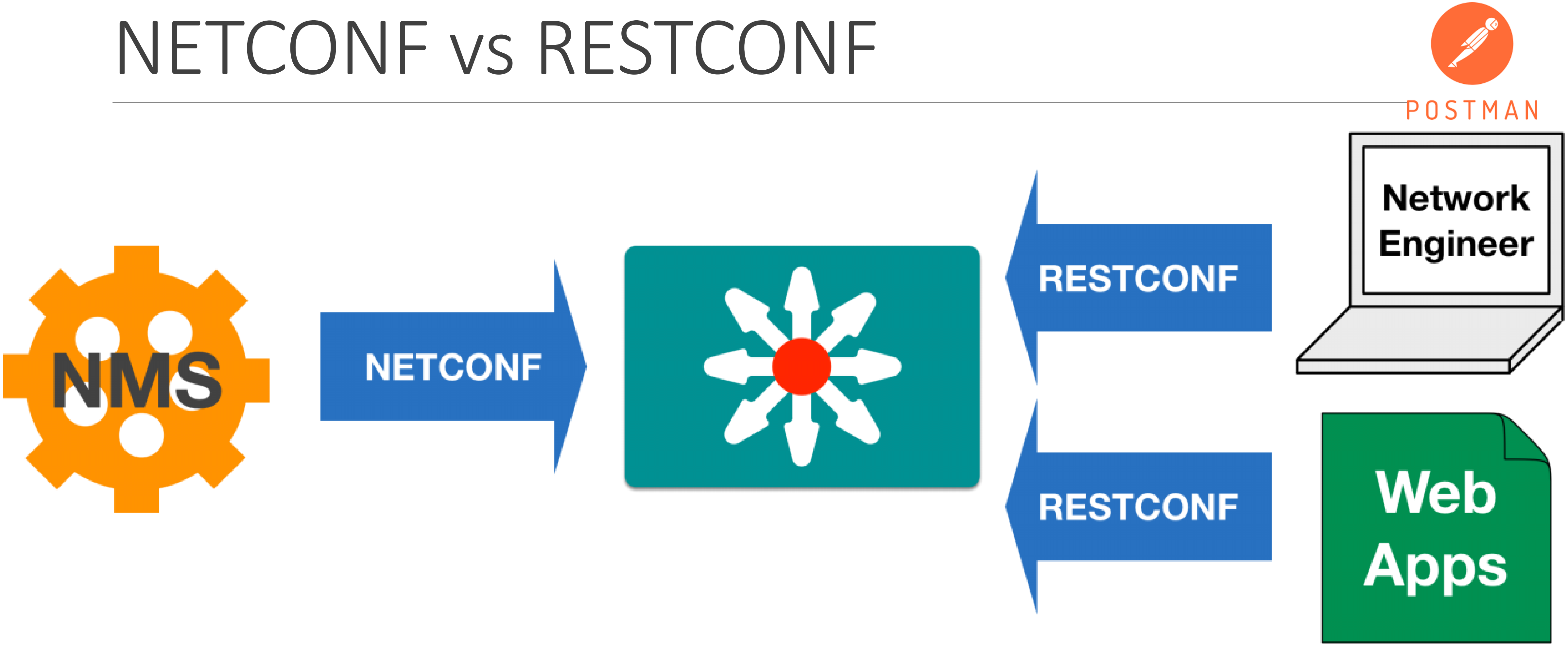
https://github.com/CiscoDevNet/restconf-examples/blob/master/restconf-samplecode/vlans/xe-vlan.py#L102

YANG – Yet Another Next Generation

- Unlike JSON and XML, YANG is a data modelling language → Defines the data schemas that formats like XML and JSON should follow
- What is data model? → The attributes that an item should have. E.g. what are the attributes of an Ethernet interface?
- Configuration protocols like NETCONF and RESTCONF use data schemas based on YANG
- Defined in RFC6020 – Common data types are defined in RFC 6991



NETCONF vs RESTCONF



NMS = Network Management System

SDN Controllers

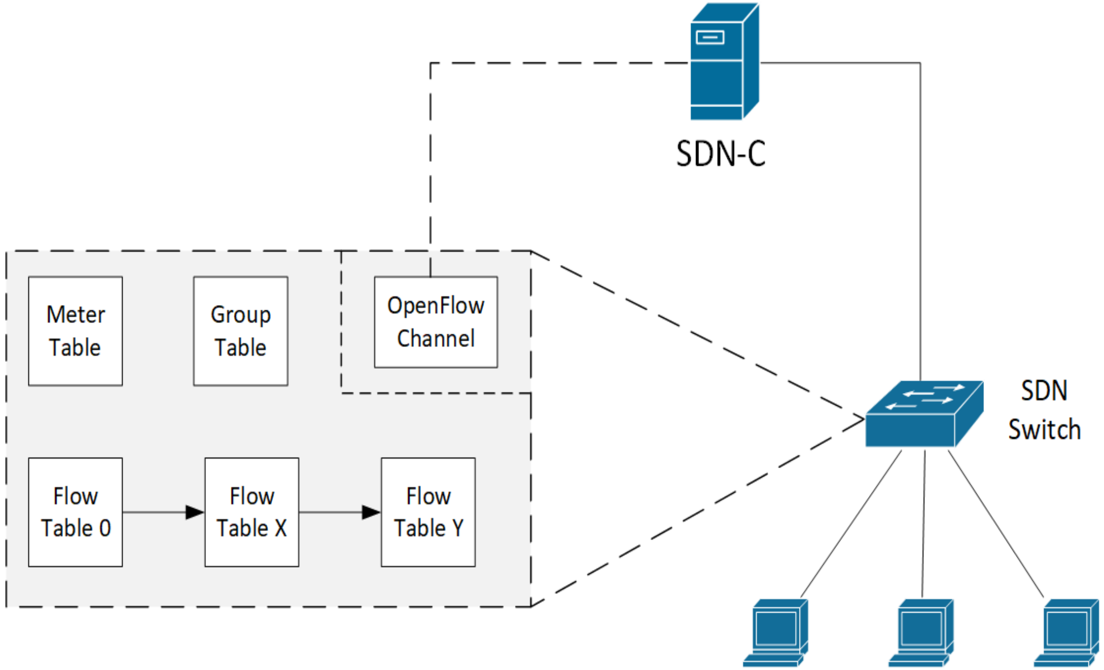
- SDN Controller undertake flow control for improved network management and performance
- SDN Controllers utilize CPSIs to interact with network devices (e.g. OpenFlow)
- Most of them are Python or JAVA based

Ryu SDN Framework

HP VAN SDN Controller

Faucet

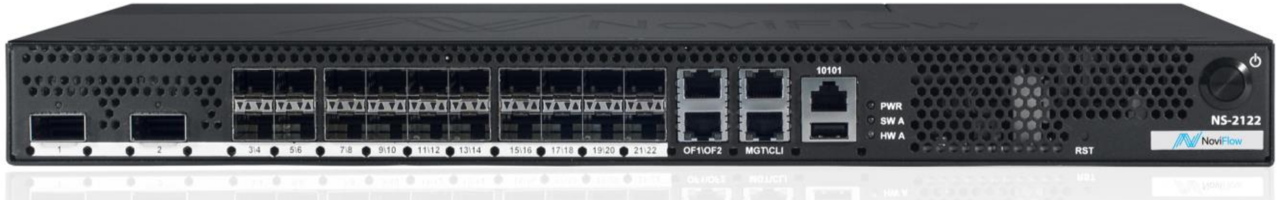
SDN Switches



- A switch controlled by and OpenFlow SDN Controller includes the following structures:
  - Flow tables: Store rules that match incoming frames based on specific criteria
  - Group tables: Apply more complicated rules on matching flows, e.g. load balancing
  - Meter tables: Apply QoS policies on matching flows, including traffic policing or editing DSCP fields
- SDN Switches can be either hardware or software-based
  - Software-based SDN switches can undertake communications between VMs inside a hypervisor
  - Hardware-based SDN switches empower communications between servers in data centres

SDN Switches

NoviFlow



OvS

Open vSwitch

Aruba 2930F Series

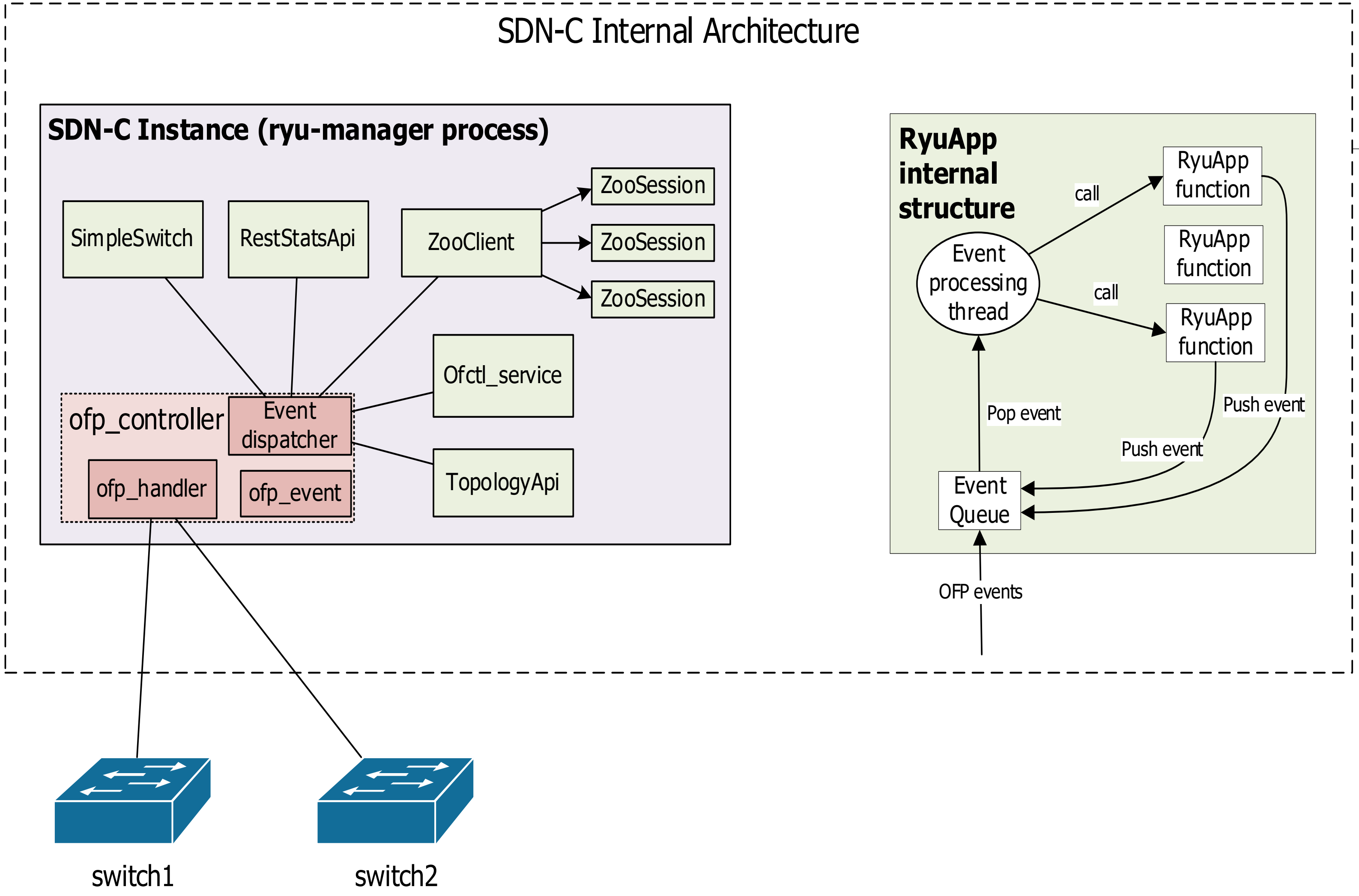
centec

cisco

Cisco Nexus 9000 Series

Case Study: The Ryu SDN Controller

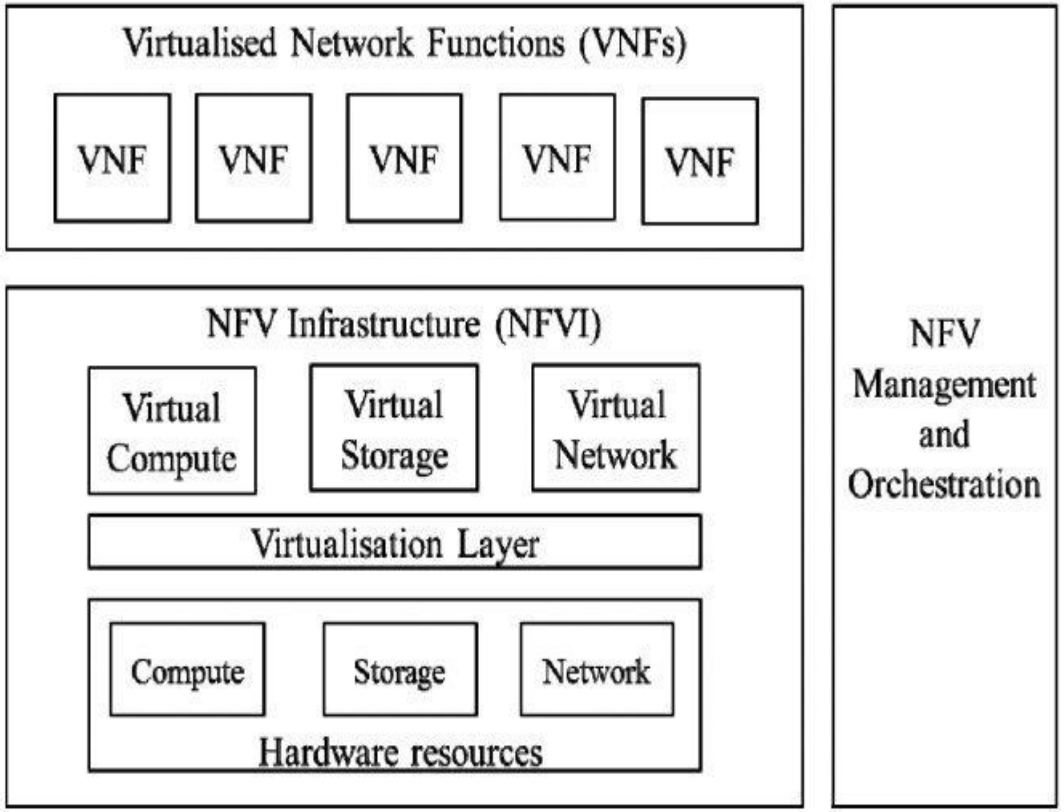
- Ryu is a Python library used to develop SDN Controllers
- Ryu provides a set of core libraries and scripts that implement basic functionalities of an SDN Controller:
  - SimpleSwitch – Decides how to forward network traffic
  - Ofctl_rest – A web service that provides a REST interface to retrieve information about the network topology in JSON format
  - Topology – Leverages Link-Layer Discovery Protocol (LLDP) to discover the network topology
- In the **SDN-microSENSE** project, a Horizon 2020 program funded by EU, we use Apache Zookeeper to provide fault tolerance to the SDN Controller



The Ryu architecture

- The ryu-manager process ‘runs’ multiple python scripts as independent network apps
- Ryu is event driven → Ofp controller is the core app which generates events
- RyuApps listen and act on events (e.g. link dropped, new incoming frame forwarded by a switch, disconnection of an SDN switch)

Network Function Virtualization (NFV)

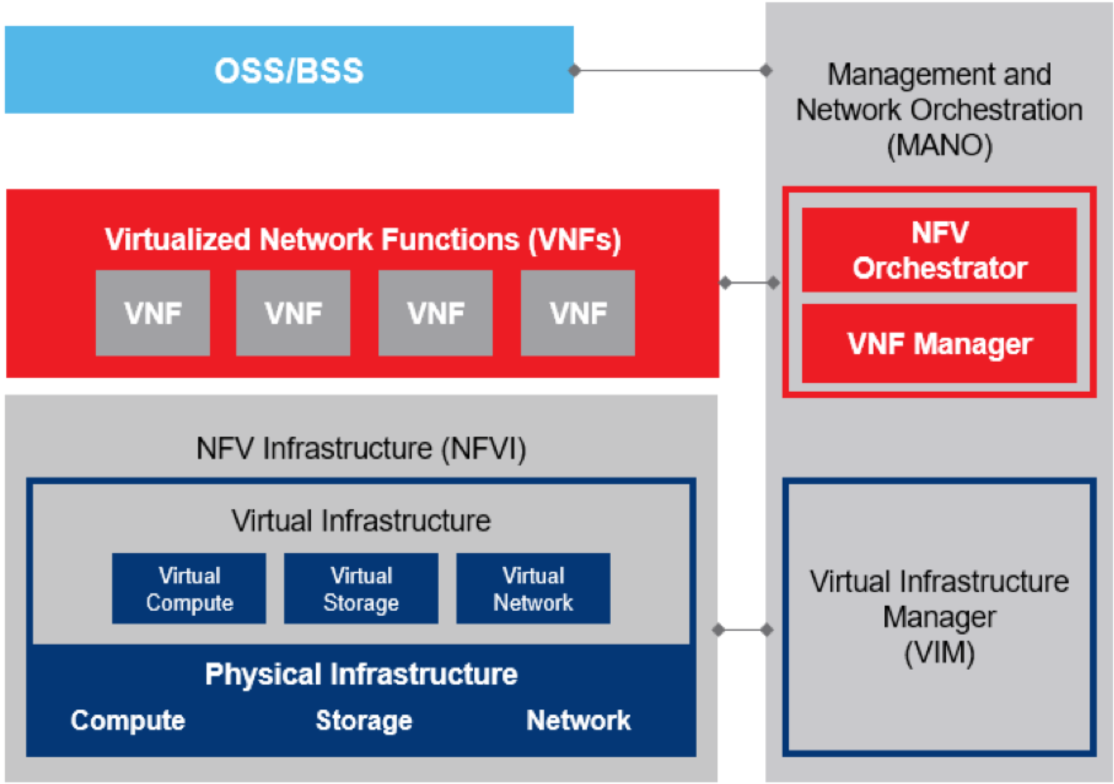


- NFV is the decoupling of network functions from proprietary hardware appliances and running them as software in virtual machines (or containers)
- Various networks functions, like firewalls, traffic control and routing, are called virtual network functions (VNFs)
- The European Telecommunications Standards Institute (ETSI) defined a high-level architectural framework and design philosophy to foster the adoption of virtualization

Objectives of NFV (official)

- Improved capital efficiency compared with dedicated hardware
  - Commercial-off-the-shelf (COTS) hardware provides NFs through software virtualisation techniques
  - Hardware is abstracted and shared with multiple NFs
- **Scalability** according to usage and **flexibility** for deploying VNFs in general-use hardware
  - Software can be easily relocated to different sites: Nearer to the customer, in central offices or datacentres
- Rapid service innovation through software-based **agile** service deployment
  - CI/CD – Continuous Integration / Continuous Testing / Continuous Deployment
- Improved operational **efficiency** resulting from common automation procedures
- **Reduced power usage** by migrating workloads and powering down unused hardware
- Standardised and open interfaces between VNFs and the infrastructure and management entities so that any of the decoupled elements can be provided by different vendors (**Interoperability**)

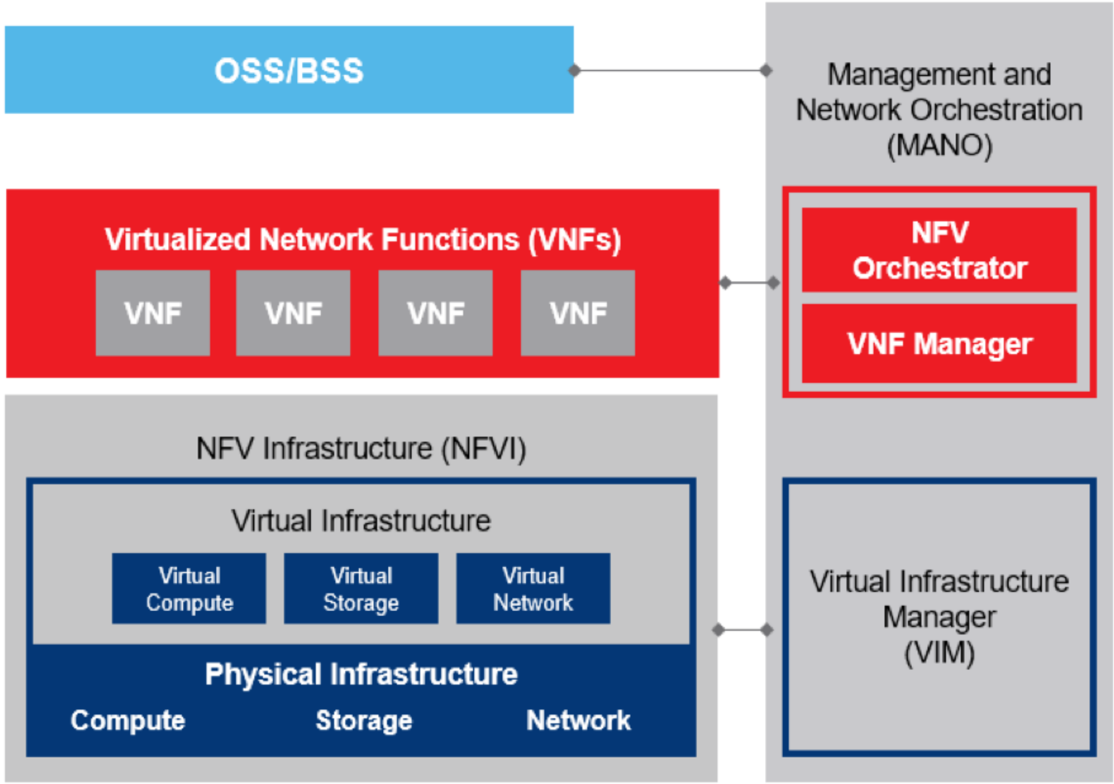
NFV Components / NFV Infrastructure



- Standardized x86 computing hardware
- Physical storage and network devices
- Provides the virtualization layer, which abstracts hardware resources into virtual ones in order to be allocated to VNFs
- Provides virtual storage and network resources
- Provides the hypervisor and the management of the virtual infrastructure via the Virtual Infrastructure Manager (VIM)

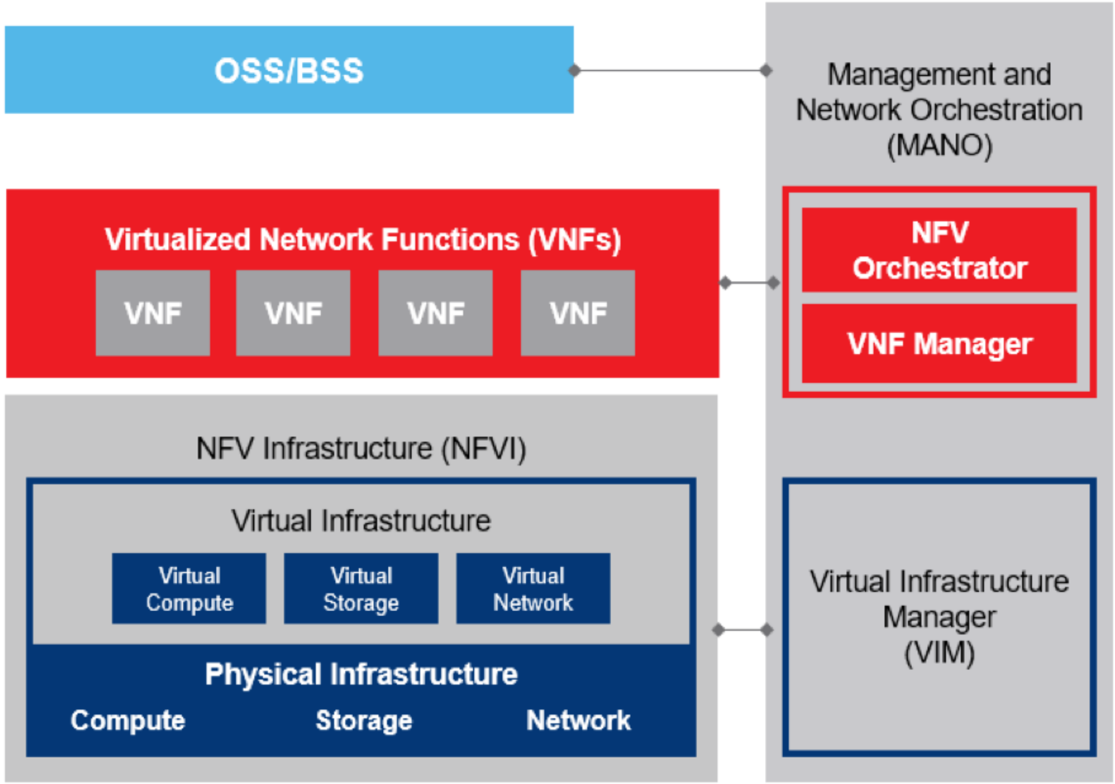
https://blog.equinix.com/blog/2019/10/17/networking-for-nerds-defining-the-elements-of-nfv-architectures/

NFV Components / Virtual Network Functions (VNF)



- Run on top of the NFVI
- VNFs can be:
  - Virtual Routers (Cisco CSR 1000V)
  - Virtual Switches (Open vSwitch)
  - SD-WAN
  - Firewalls (pfSense)
  - QoS engines
- VNFs are deployed remotely on-demand
- No need for field engineers, cabling etc
- Multiple VNFs can be aggregated to a single Network Service (NS)

NFV Components / Network Services (NS)



- VNFs connect with one another to create Network Services (NS)
- NS are orchestrated by the NFV Orchestrator
- An NS is comprised of the following:
  - Individual VNFs that implement distinct functionalities
  - Virtual Links: Layer 2 connections between VNFs
  - VNF Forwarding Graphs (VNF-FG), traffic flows between VNFs
- Network Service Descriptors (NSDs) are templates (JSON or TOSCA) describing the deployment of a NS, including
  - Service topology (VNFs + VNF-FGs + Virtual Links)
  - Service characteristics, e.g. SLA

https://telcocloudbridge.com/blog/beginners-guide-to-network-service-and-vnf-forwarding-graph-in-nfv/

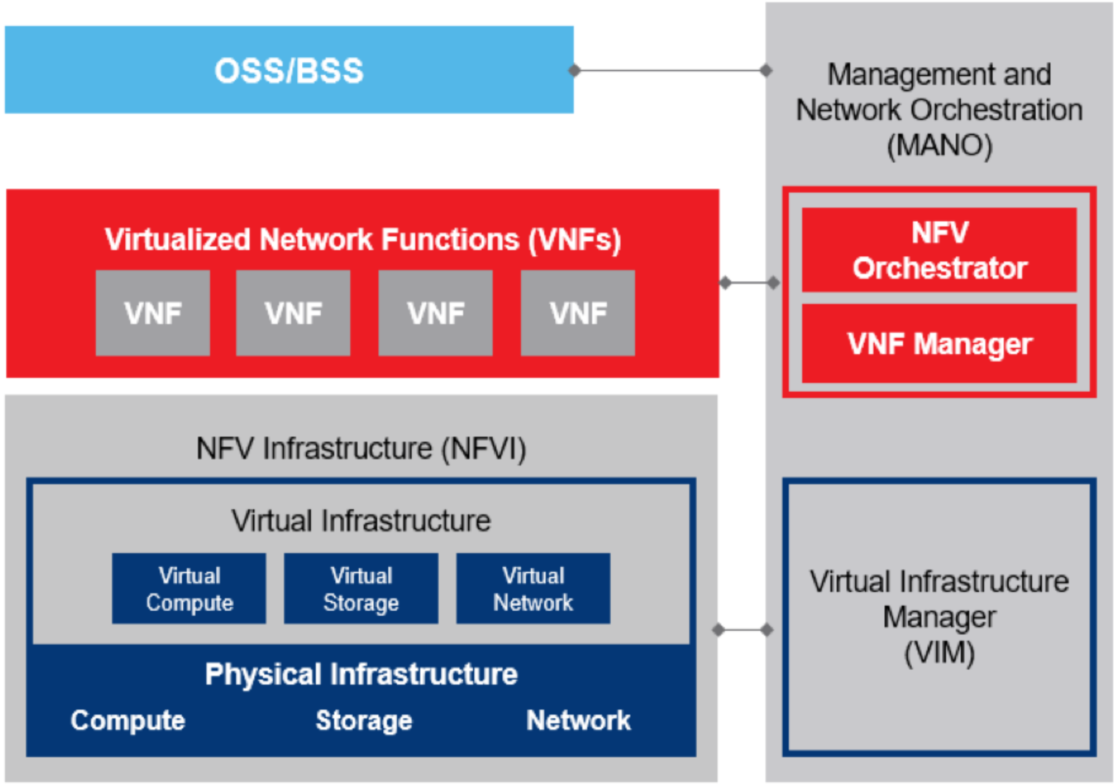
```json
{
    "name":"iperf-NSD",
    "vendor":"fokus",
    "version":"0.1-ALPHA",
    "vnfd":[  ...  ],
    "vld":[
        {
            "name":"private"
        }
    ],
    "vnf_dependency":[
        {
            "source" : {
                "name": "iperf-server"
            },
            "target":{
                "name": "iperf-client"
            },
            "parameters":[
                "private"
            ]
        }
    ]
}
```

NSD Example

- NS name
- NS vendor
- NS version
- vnfd: VNFs that consist the NSD (iperf3 Server and iperf3 client – Omitted from this example)
- vld: Virtual links (a single L2 broadcast domain named "private")
- vnf_dependency: (Optional) Requirements and dependencies between the VNFs
  - The source VNF provides information (e.g. configuration parameters) to the target VNF

https://openbaton.github.io/documentation/tutorial-docker/#iperf-nsd

NFV Components / NFV MANO

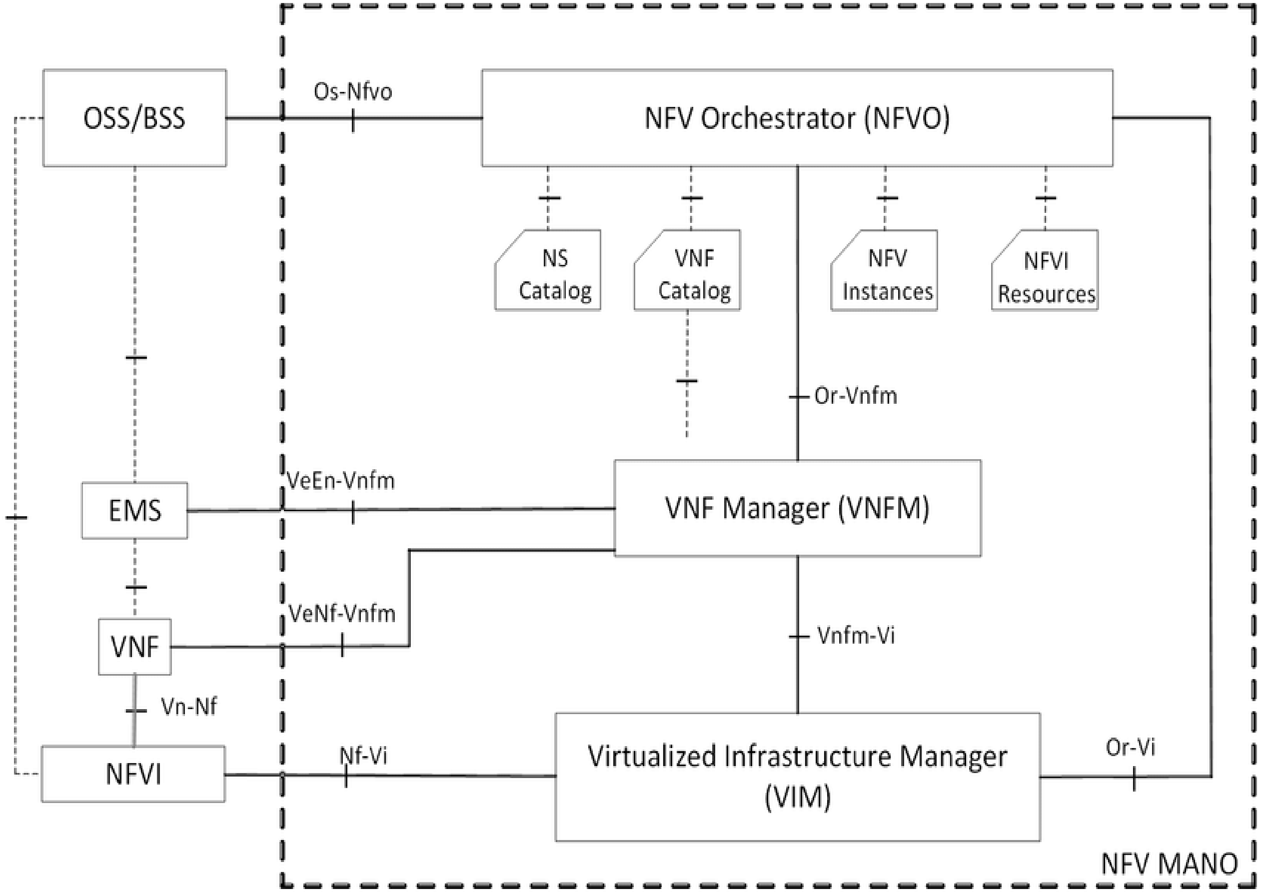


- NFV Management and Network Orchestration (MANO) is a framework developed by ETSI to coordinate NFVI, VNFs and VMs → **ETSI MANO**
- NFV MANO uses templates that allow admins to deploy a variety of VNFs and NS
- NFV MANO is comprised of three functional areas:
  - NFV Orchestrator → Focuses on NS and resource orchestration
  - VNF Manager → Focuses on management of VNFs
  - Virtual Infrastructure Manager → Controls and manages the NFVI compute, storage and network resources

NFV Orchestrator vs VNF Manager

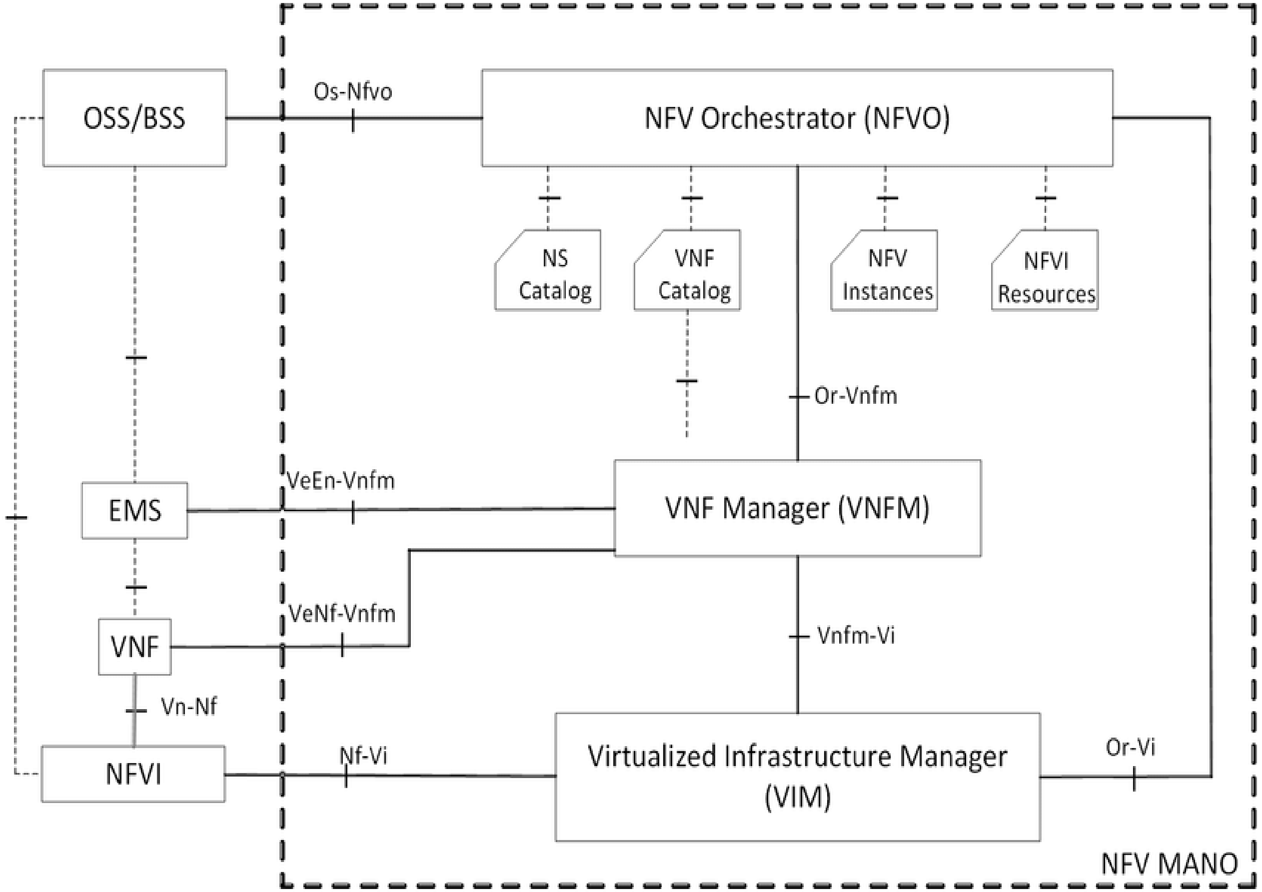
NFV Orchestrator	**VNF Manager**
Orchestration and management of NFV infrastructure, resources and NS - Resource Orchestration - Service Orchestration	Management of VNFs
Onboarding of NS and VNFs, topology management of NS (VNF-FG), Instantiation - scaling in/out – termination – updating NS and VNFs, monitoring and auto-healing of VNFs (**Service Orchestrator**) Coordination, authorisation, release and engagement of NFVI resources amongst different or within a single Datacentre (**Resource Orchestrator**)	Instantiation, configuration, start/stop, scaling, updating and termination of VNFs
A single NFV Orchestrator	Multiple VNF Managers can be deployed - A single VNF manager for one or more VNFs

NFV MANO Components



- Fundamental elements:
  - VNFs – Virtualised tasks
  - PNFs – Dedicated hardware to perform specific network functions, e.g. firewall
  - Virtual Deployment Unit (VDU): The VM that hosts the VNF (considered component of VNF)
  - Virtual Links (VL)
  - Network Services (NS)
  - VNF Forwarding Graphs (VNF-FG)
- Descriptor Files: For each of the above, there is a descriptor file that describes attributes

NFV MANO Components



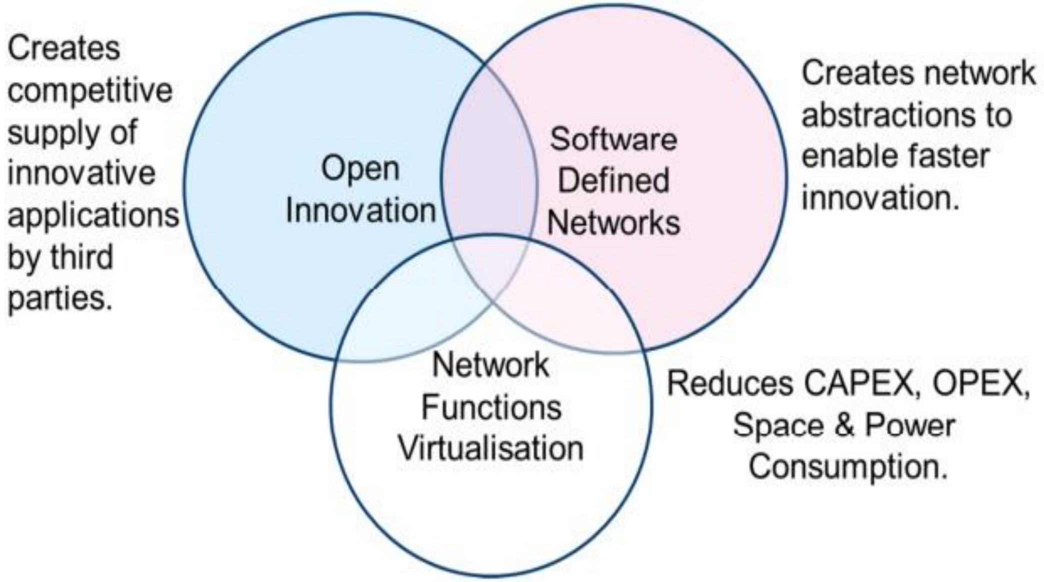
- Repositories: Hold information in ETSI MANO
  - NS Catalog: A repository of all usable NSD
  - VNF Catalogue: A repository of all usable VNF Descriptors (VNFD)
  - VNF Instances: Holds information of all VNF and NS Instances
  - NFVI Resources – A repository of utilised NFVI resources for running VNFs or NS

Outside of MANO - OSS/BSS and EMS

- Operation Support Systems / Business Support Systems (OSS/BSS)
  - External services that interact with the NFV MANO
  - OSS/BSS are utilized by Communications Service Providers (CSP)
  - Usable for Order Management, Billing, Customer Relationship Management, Network Inventory Management and Network Operations
  - Example: openslice
- Element Management System (EMS)
  - Similar role that EMS have on physical networks
  - Provide FCAPS (Fault, Configuration, Accounting, Performance and Security) management
  - Management is applied to VNFs
  - One EMS per VNF or a single EMS for multiple VNFs
  - EMS can also be VNFs
  - Example: Dhyan's Netman

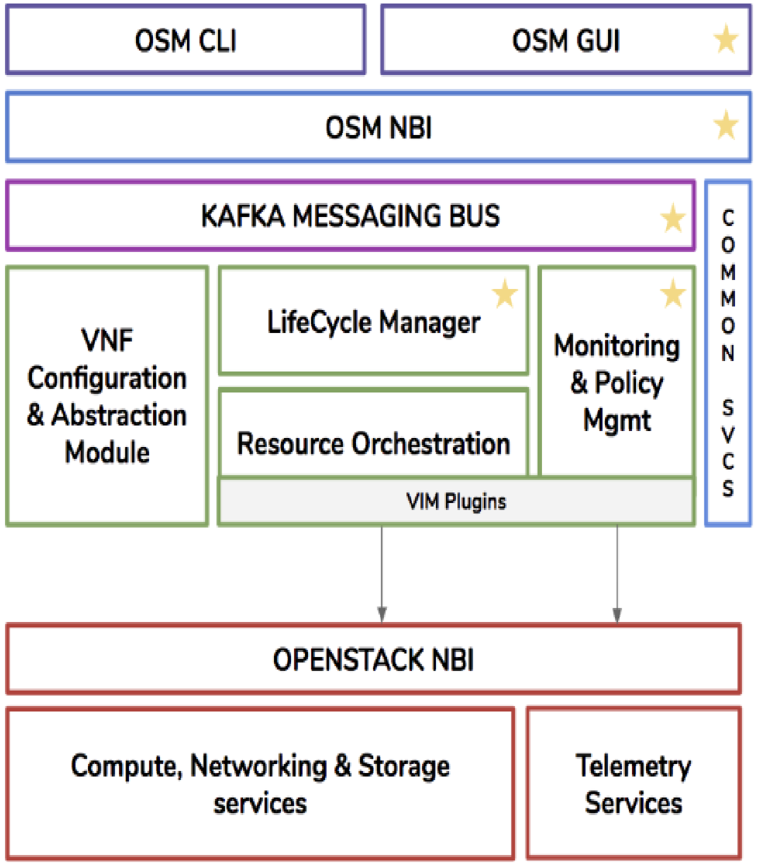
https://opennetworking.org/wp-content/uploads/2014/10/sb-OSS-BSS.pdf

NFV vs SDN



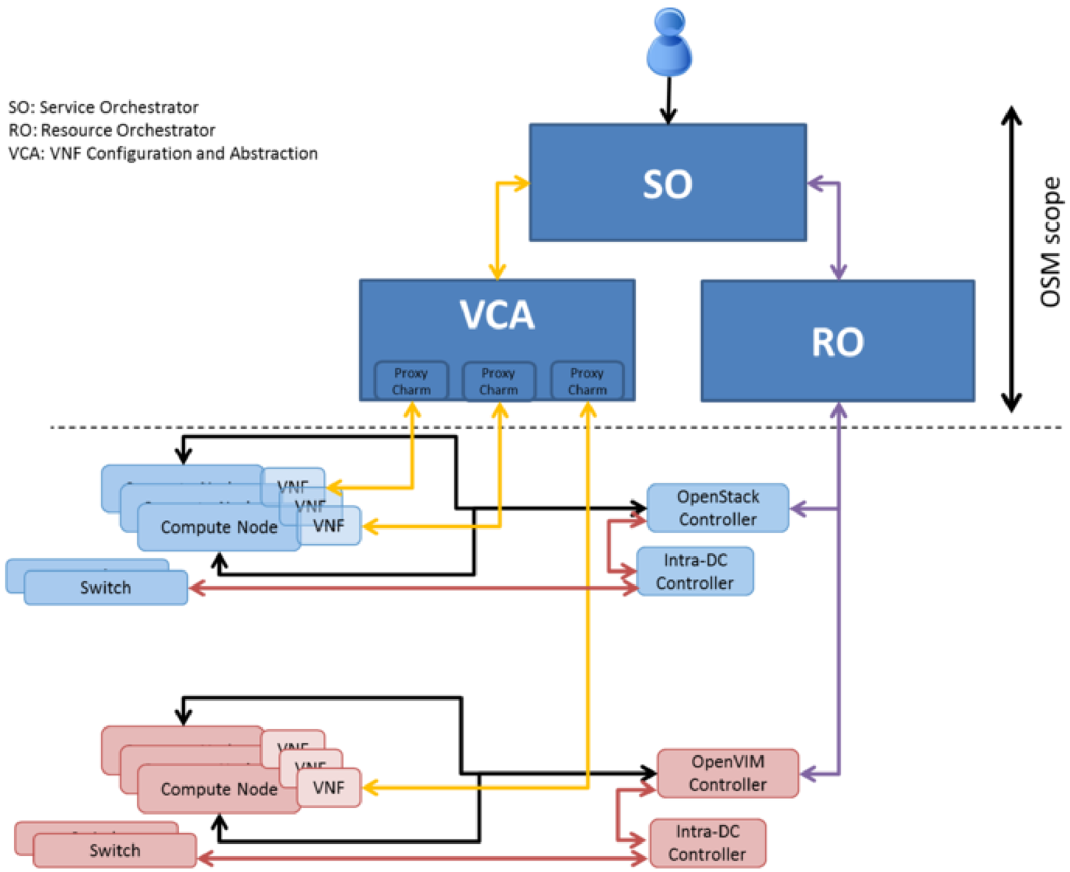
- SDN is NOT an element of the NFV architecture
  - However, they are both contributing to a software-centric and automated approach to network management
- NFV separates hardware and software, while SDN separates control and user plane
- SDN abstracts network functionalities whereas NFV abstracts network resources
- SDN is NOT a mandatory or dependent technology in order to implement NFV / NFV can be implemented without SDN
  - However, SDN can provide added value and enhanced performance
- NFV can help SDN deployments by providing the infrastructure, services and VNFs, upon which the SDN software can be run

Open-Source MANO (OSM)



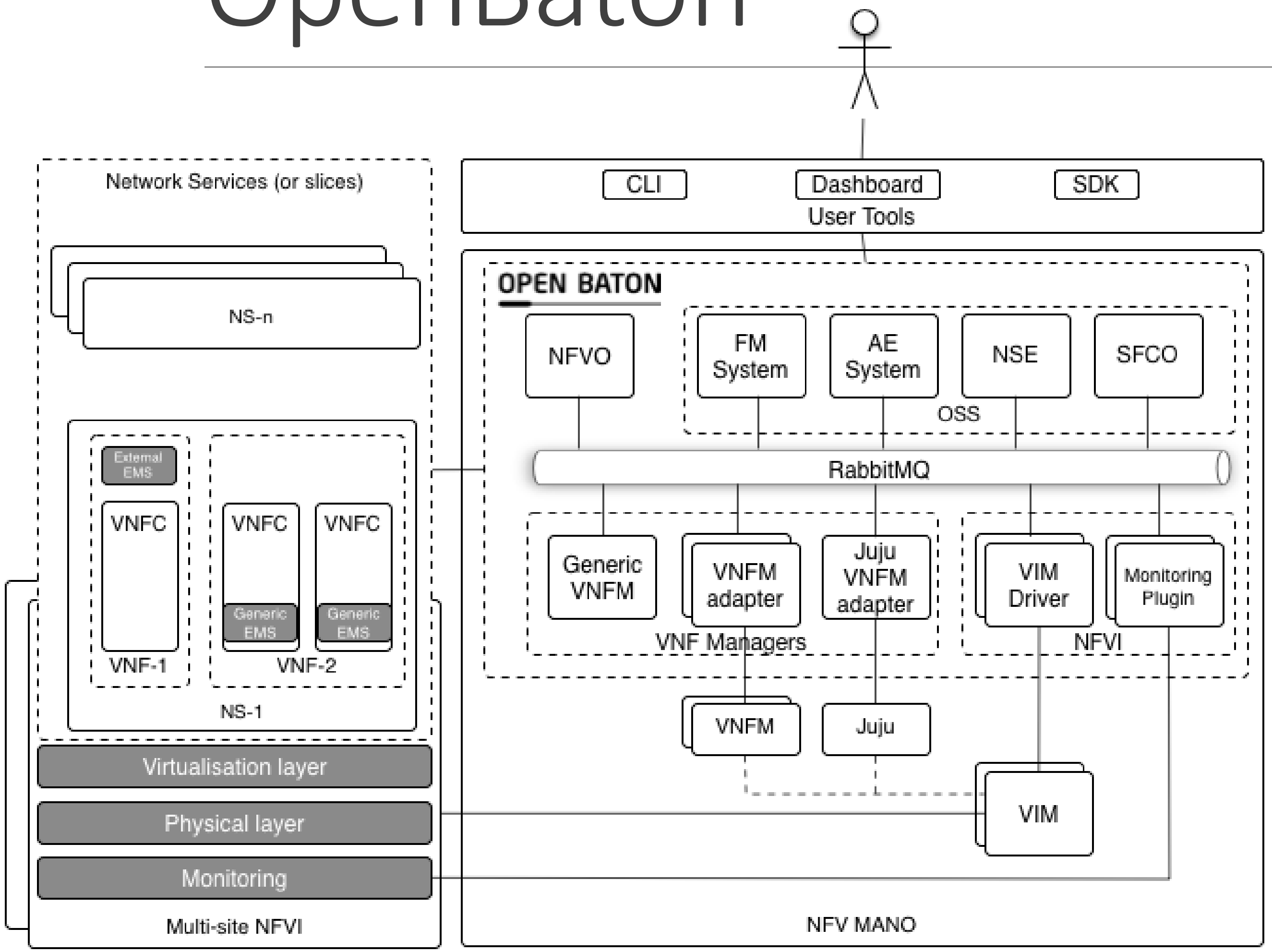
- OSM is the “official” implementation of ETSI NFV framework, community-led and provided by ETSI
- OSM is closely aligned with the evolution of ETSI NFV standard and is committed to regular updates in-line with the reference implementation of NFV MANO
- Founding members: BT, Canonical, Intel, Mirantis, RIFT.io, Telefonica, ++
- OSM can be integrated with **Openstack** as a Virtual Infrastructure Manager

OSM Components



- Service Orchestrator (SO): Based on the rift.io software, provides end-to-end service orchestration and provisioning | Most development on JavaScript and NodeJS
- Resource Orchestrator (RO): Provisions resources over an IaaS provider (OpenStack, Vmware, OpenVIM) | Most development in Python
- VNF Configuration and Abstraction (VCA): Performs initial configuration using **Juju** charms.

OpenBaton



- Open-source implementation of a NFV MANO compliant framework, alternative to OSM
- VIM plugins for **Juju**, Docker Engine, Docker Swarm and **Kubernetes**
- **Zabbix** for network monitoring
- **RabbitMQ** as the message bus

Other alternatives: OPEN-O, T-NOVA/TeNor, SONATA, Cloudify, MAESTRO (Ubitech)

Accompanying software components (just examples)

- Message Bus for asynchronous message delivery between components. **RabbitMQ** and **Apache Kafka** are some of the most reliable solutions
- The ELK stack can be used for data management and storage: (Elasticsearch license)
  - **Elasticsearch** for data store, runtime data and logs
  - **Logstash** performs pre-processing of logs, converting them to common formats
  - **Kibana** can be used for visualisations of Elasticsearch data
- **Prometheus** is an alternative to Elastisearch for time-series data and **Grafana** alternative to Kibana for visualisations (Apache 2.0 license)
- **Riemann.io** can be used as policy engine to process runtime decisions about availability, SLA and overall monitoring
- **Kubernetes** for orchestration and scaling of docker containers
- **Keycloak** or **WSO2** can be used as identity server, providing authorisation, authentication and accounting for VNFs and users

https://www.mirantis.com/blog/which-nfv-orchestration-platform-best-review-osm-open-o-cord-cloudify/

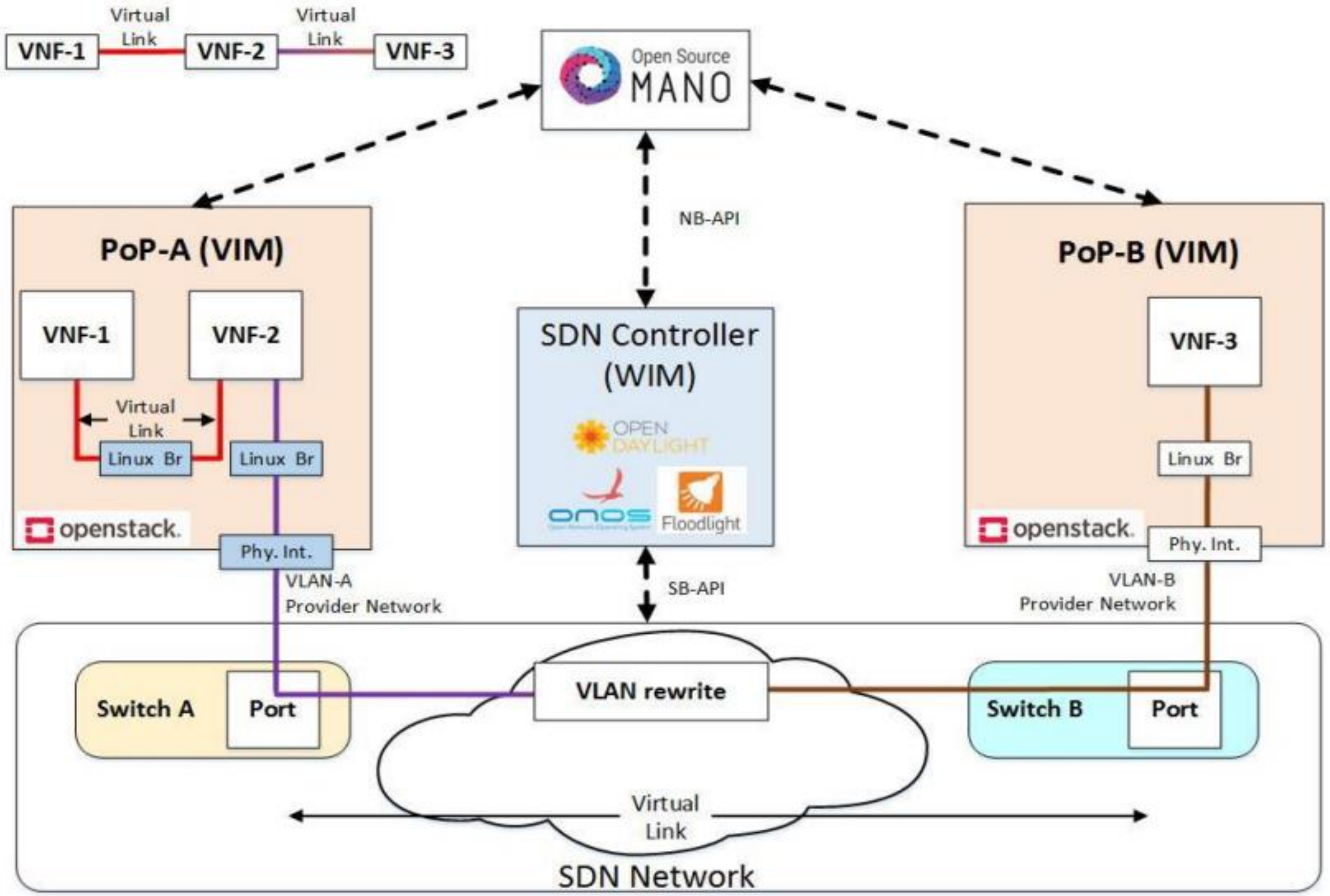
NFV, SDN and 5G

- 5G relies on NFV to implement a compliant and standard-based mobile network architecture that is based on virtualisation
- SDN provides additional capabilities of automation and QoS assurance
- Specialised VNFs (e.g. mobility management VNF or Slicing Management VNF) are developed to satisfy specific functional requirements of 5G
- 5G software is abstracted and can run in common x86 hardware, on the cloud or on the edge
- Development of innovative services is ensured by adopting common open-source tools and common interfaces → Focus is now given on intelligence

5G Research and OSM

- Numerous research projects in EU leverage OSM to implement NFV MANO for 5G implementations. Research is oriented into providing new features to OSM and satisfying new emerging use cases
  - **Metro-Haul** → Aims to provide a smart optical metro infrastructure that supports traffic originating from heterogenous 5G access networks → Contributed to the WAN Infrastructure Management (OSM release 5)
  - **Matilda** → Holistic 5G end-to-end services operational framework, tackling the life cycle of design, development and orchestration of 5G-ready applications and network services.
  - **5G-Tango** → Incorporates flexible programmability of 5G networks by collaborating with different open-source initiatives to accelerate the NFV uptake in industry
  - **5G-City** → Provides a distributed cloud and radio platform for municipalities and infrastructure owners acting as 5G neutral hosts
  - **5G-Media** → Provides an orchestration and DevOps platform for network media services and applications running on 5G networks
  - **5G-Transformer** → Aims to bring network slicing into mobile transport networks tailored to verticals

- OSM is the orchestrator in Metro-Haul
  - Deploy, manage and orchestrate the Network services across disaggregated datacentres.
  - Design and development of Wide Area Network Infrastructure Manager (WIM) in OSM
    - ✓ Deploy VNFs across multiple datacentres in a Network Service.
    - ✓ Create L2 VLANs over the underlying network infrastructure.
    - ✓ Protocol-Agnostic systems allowing variety of SDN technologies to work with.



5GCity Orchestration Platform