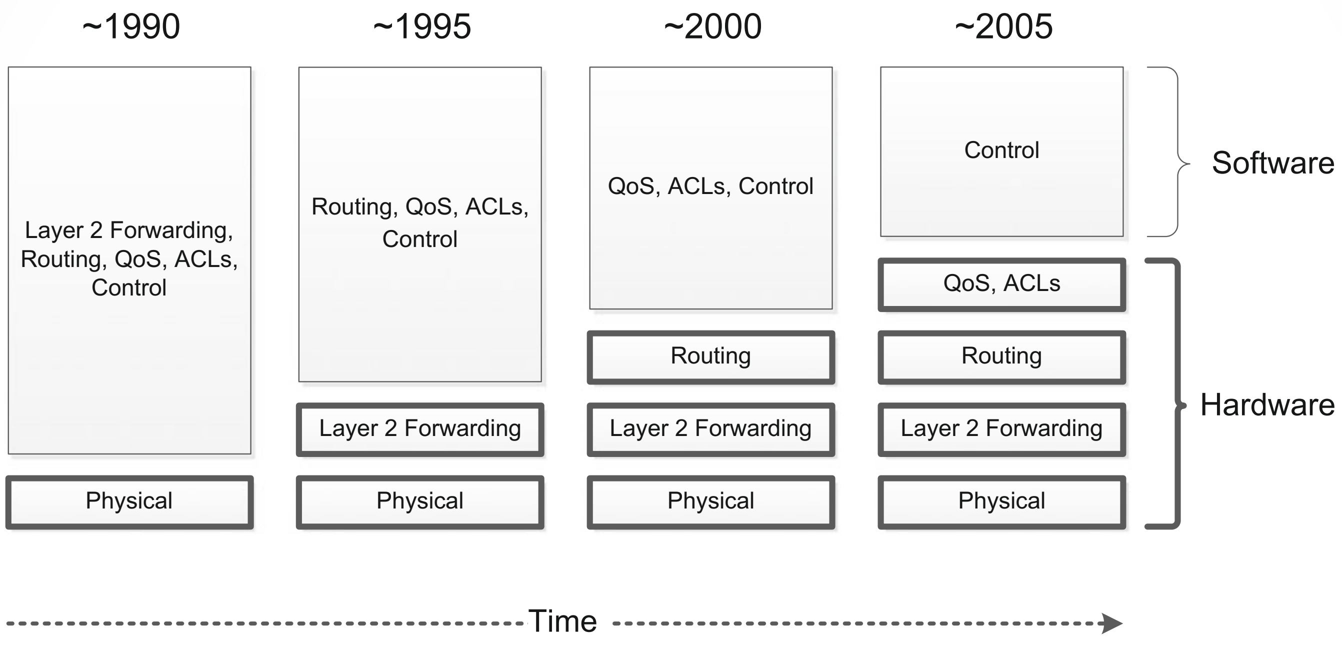
Configuration and Management of Networks

Pedro Amaral

Configuration and Management of Networks

Evolution of Network functionality



Software Moves into silicon

Forwarding in Hardware and Control in Software

Configuration and Management of Networks

Current Networks are expensive and complex:

Complexity and Vendor Lock in

↓

Resistance to change difficulty to innovate

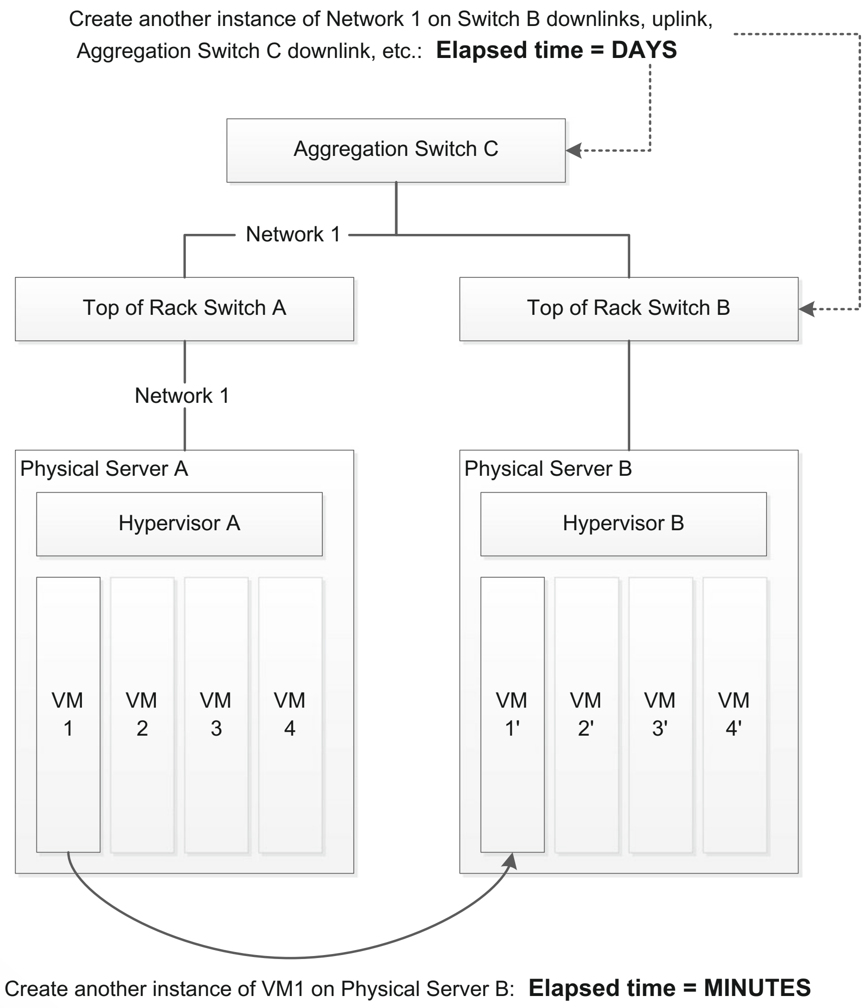
Expensive Hardware
Difficult to configure and operate (large OPEX)

Most important : Networks are inadequate for some modern applications

Ex: Cloud Datacenters

Configuration and Management of Networks

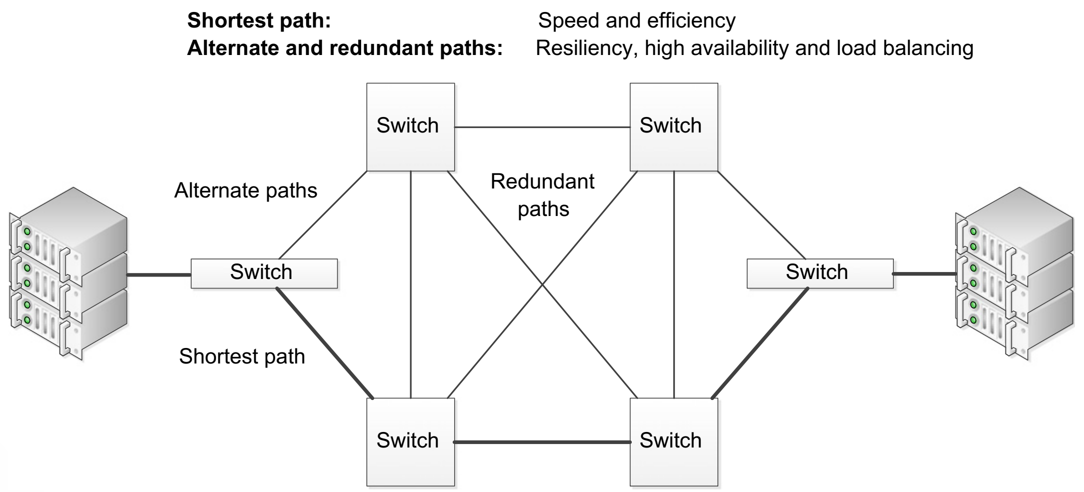
Data Center Network challenges



Configuration and Management of Networks

Data center Network needs:

- Automation
- Scalability (MAC table sizes and VLANs, broadcast control problems)
- Multipathing
- Mutitenancy (virtual Networks)



Configuration and Management of Networks

Software Defined Networks – Control and data separation

Control Plane: Logic for controlling forwarding behavior.
Examples: Routing protocols, network middlebox configuration.

Forwarding Plane: Forwarding traffic according to control plane logic.
Examples: IP forwarding, L2 switching, MPLS label switching.

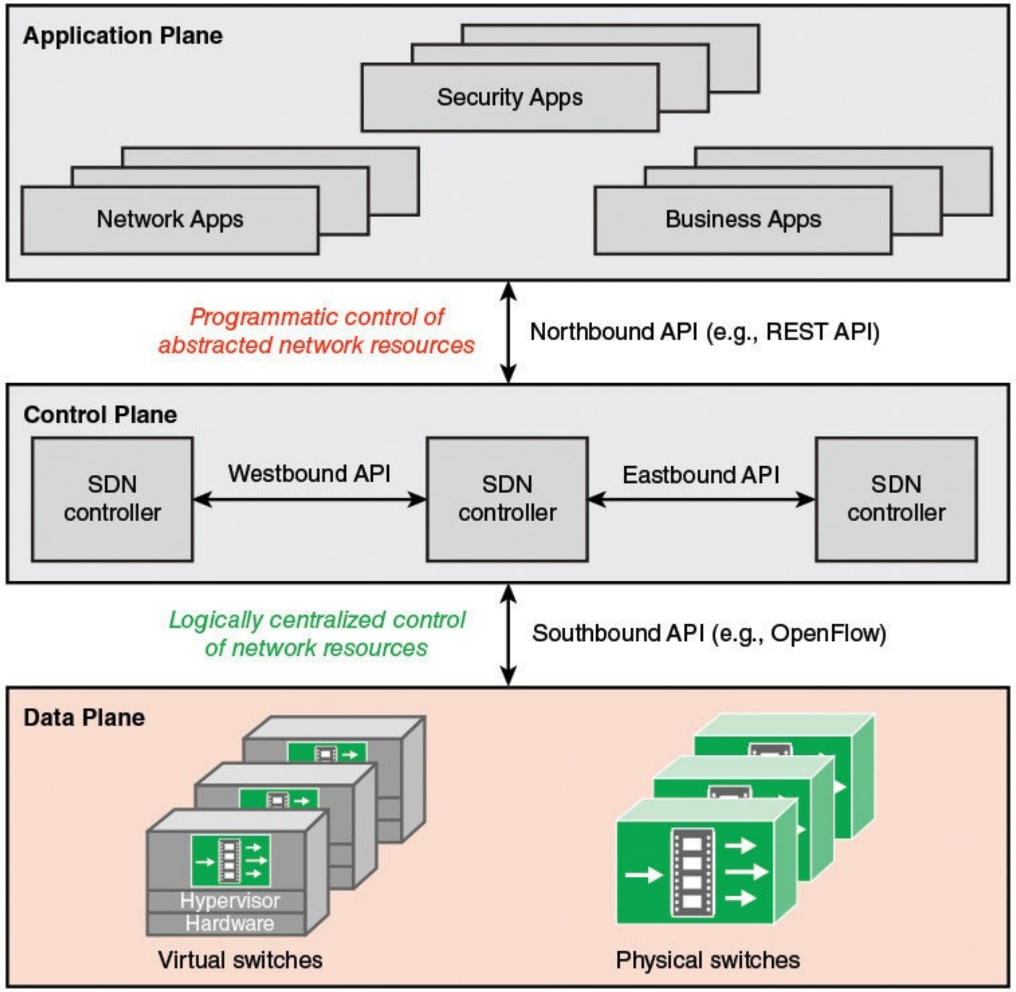
Allows

Independent evolution and development: The software control can evolve independently from the hardware.

Control from high level software program : Control behavior using high-order programs

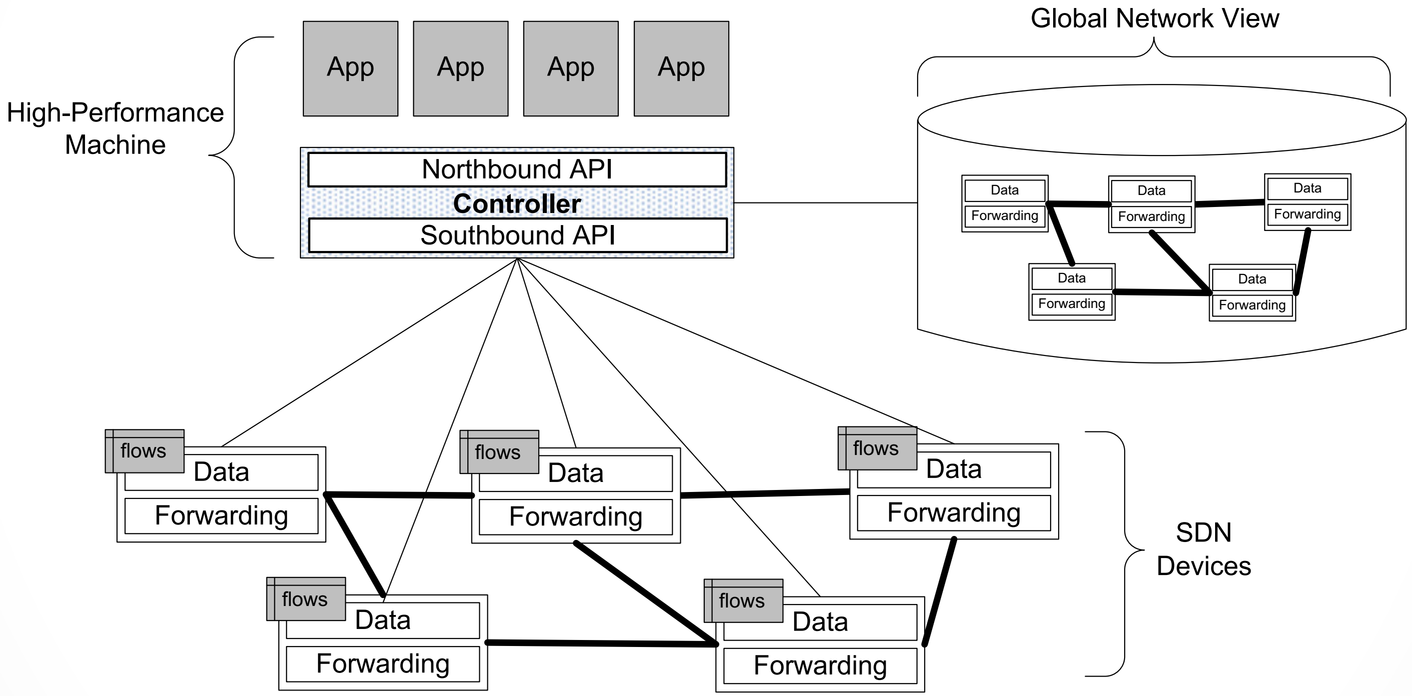
Configuration and Management of Networks

Software Defined Networks – Control and data separation

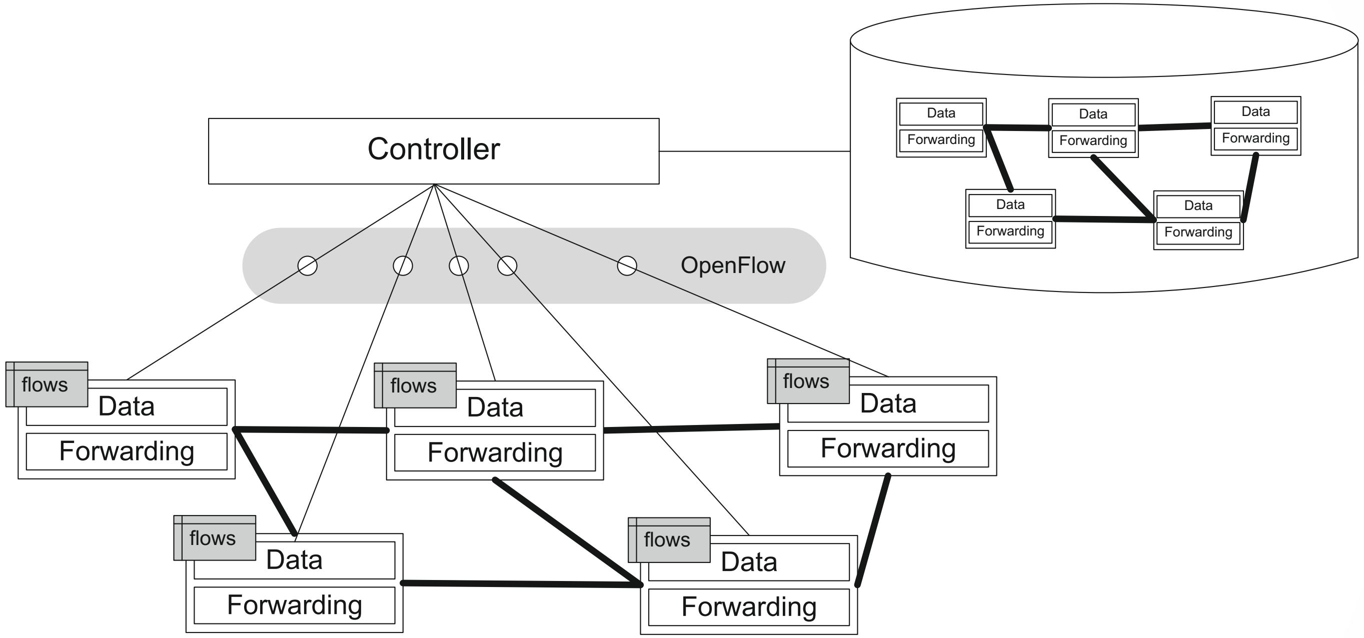


Configuration and Management of Networks

Software Defined Networks – Control and data separation

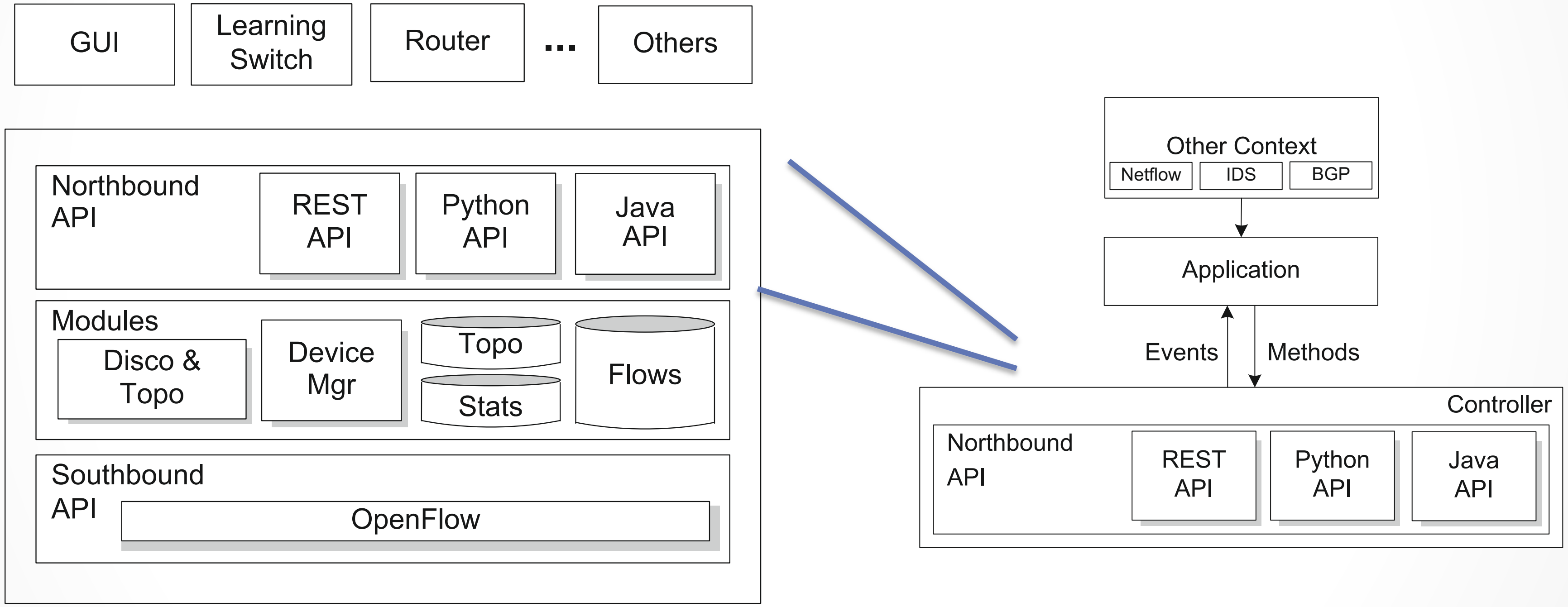


Software Defined Networks – OpenFlow Southbound API

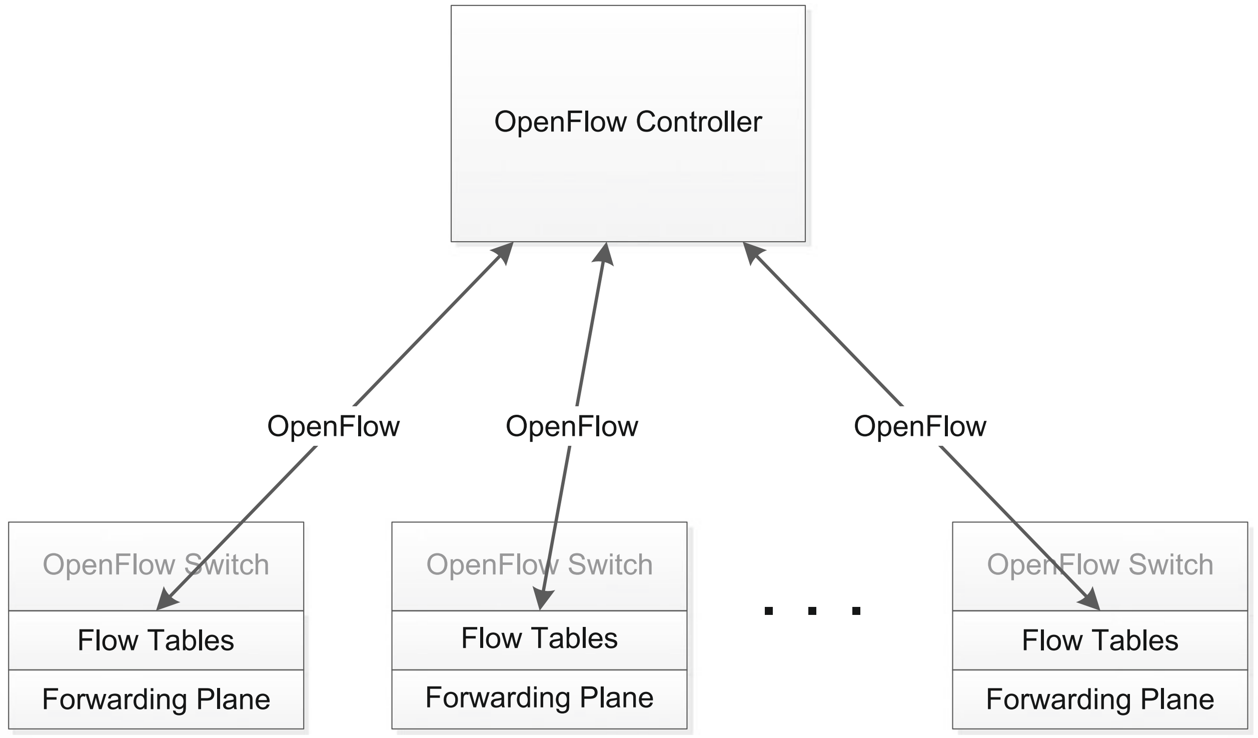


Configuration and Management of Networks

Software Defined Networks – Controller



Software Defined Networks – OpenFlow Southbound API



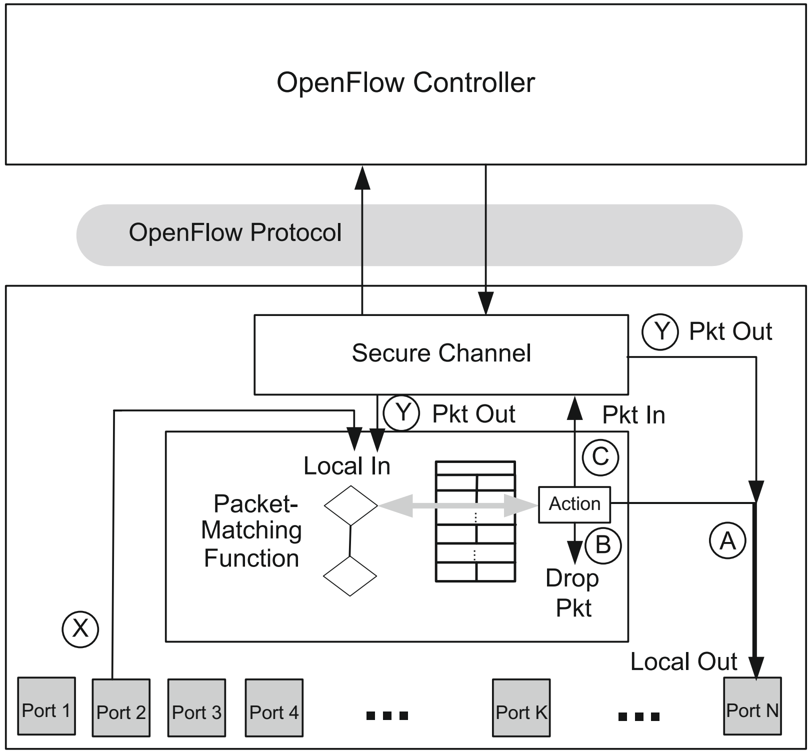
Software Defined Networks – OpenFlow Forwarding Plane

Flow Tables: Perform packet lookup.

- All packets compared to flow table for **match**
- **Instructions** depending on match being found
- Packets that do not match are either sent to the controller (OF 1.0) or discarded (OF 1.3 and after)

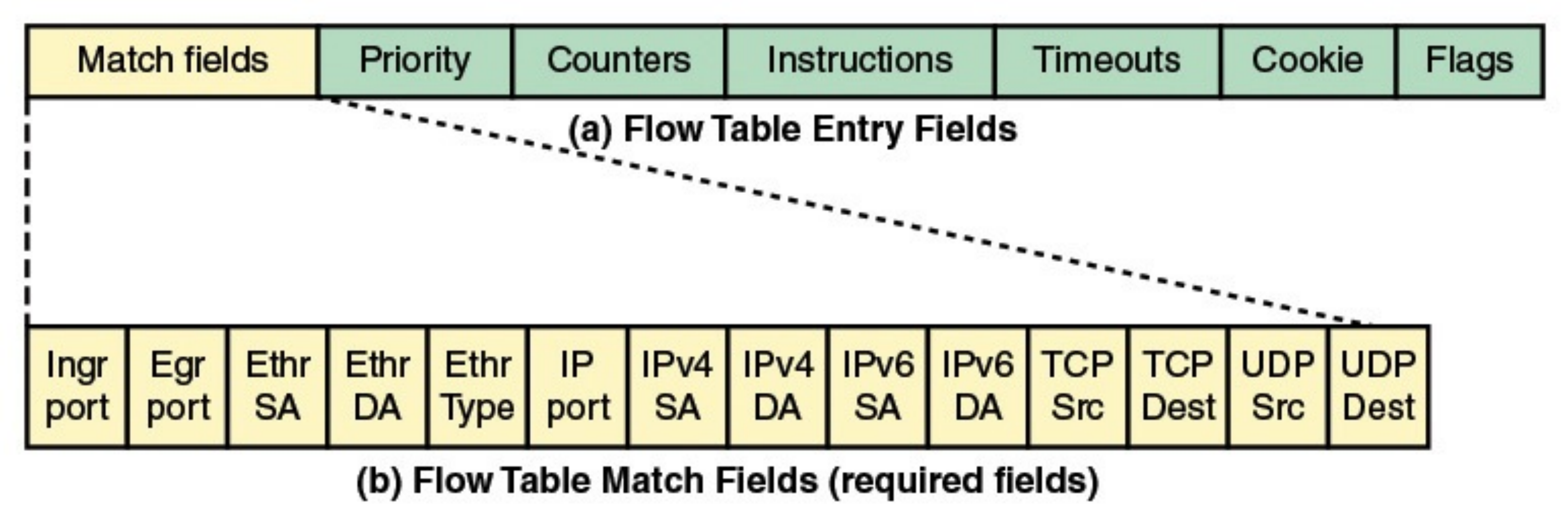
Secure Channel: Communication to the controller (TCP connection or TLS connection).

Software Defined Networks – OpenFlow Forwarding Plane



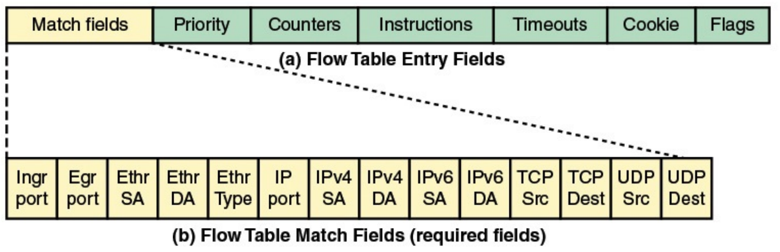
- *A*. Forward the packet out a local port, possibly modifying certain header fields first.
- *B*. Drop the packet.
- *C*. Pass the packet to the controller.

Software Defined Networks – Packet Matching



- **Match fields:** Used to select packets that match the values in the fields.
- **Priority:** Relative priority of table entries. This is a 16-bit field with 0 corresponding to the lowest priority. In principle, there could be 2^{16} = 64k priority levels.

Software Defined Networks – Packet Matching

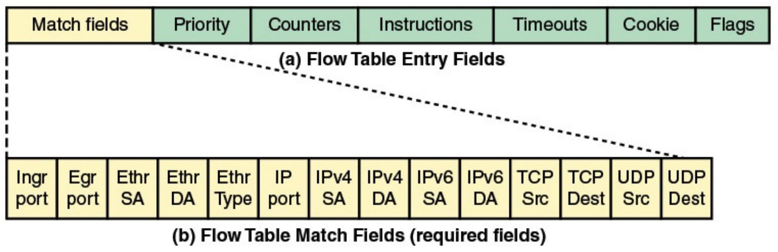


- **Counters:** Updated for matching packets.

Counter	Usage	Bit Length
Reference count (active entries)	Per flow table	32
Duration (seconds)	Per flow entry	32
Received packets	Per port	64
Transmitted packets	Per port	64
Duration (seconds)	Per port	32

- **Instructions:** Instructions to be performed if a match occurs.
- **Timeouts:** Maximum amount of idle time before a flow is expired by the switch. Each flow entry has an idle_timeout and a hard_timeout

Software Defined Networks – Packet Matching



- **Cookie:** 64-bit opaque data value chosen by the controller. May be used by the controller to filter flow statistics, flow modification and flow deletion; not used when processing packets.
- **Flags**: Flags alter the way flow entries are managed; for example, the flag OFPFF_SEND_FLOW_REM triggers flow removed messages for that flow entry.

Software Defined Networks – Instructions

Instructions: Can be grouped in four categories:

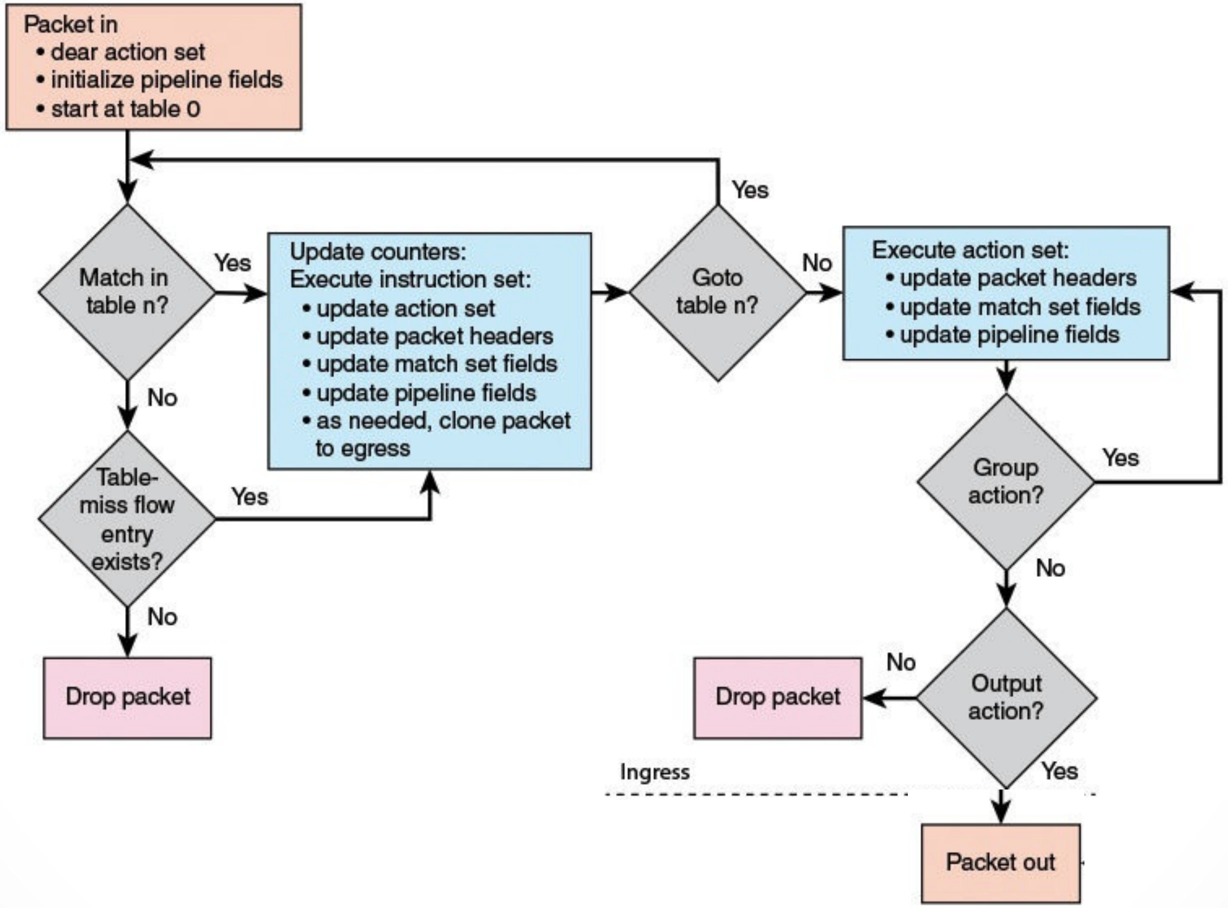
- **Direct packet through pipeline:** The Goto-Table instruction directs the packet to a table farther along in the pipeline.
- **Perform action on packet:** Actions may be performed on the packet when it is matched to a table entry. The Apply-Actions instruction applies the specified actions immediately
- **Update action set:** The Write-Actions instruction merges specified actions into the current action set for this packet.

Software Defined Networks – Instructions

Types of actions:

- **Output:** Forward packet to specified port. The port could be an output port to another switch or the port to the controller. In the latter case, the packet is encapsulated in a message to the controller.
- **Group:** Process packet through specified group.
- **Push-Tag/Pop-Tag:** Push or pop a tag field for a VLAN
- **Set-Field:** The various Set-Field actions are identified by their field type and modify the values of respective header fields in the packet.
- **Change-TTL:** The various Change-TTL actions modify the values of the IPv4 TTL (time to live), IPv6 hop limit, or MPLS TTL in the packet.
- **Drop**: There is no explicit action to represent drops. Instead, packets whose action sets have no output action should be dropped.

Software Defined Networks –Switch operation



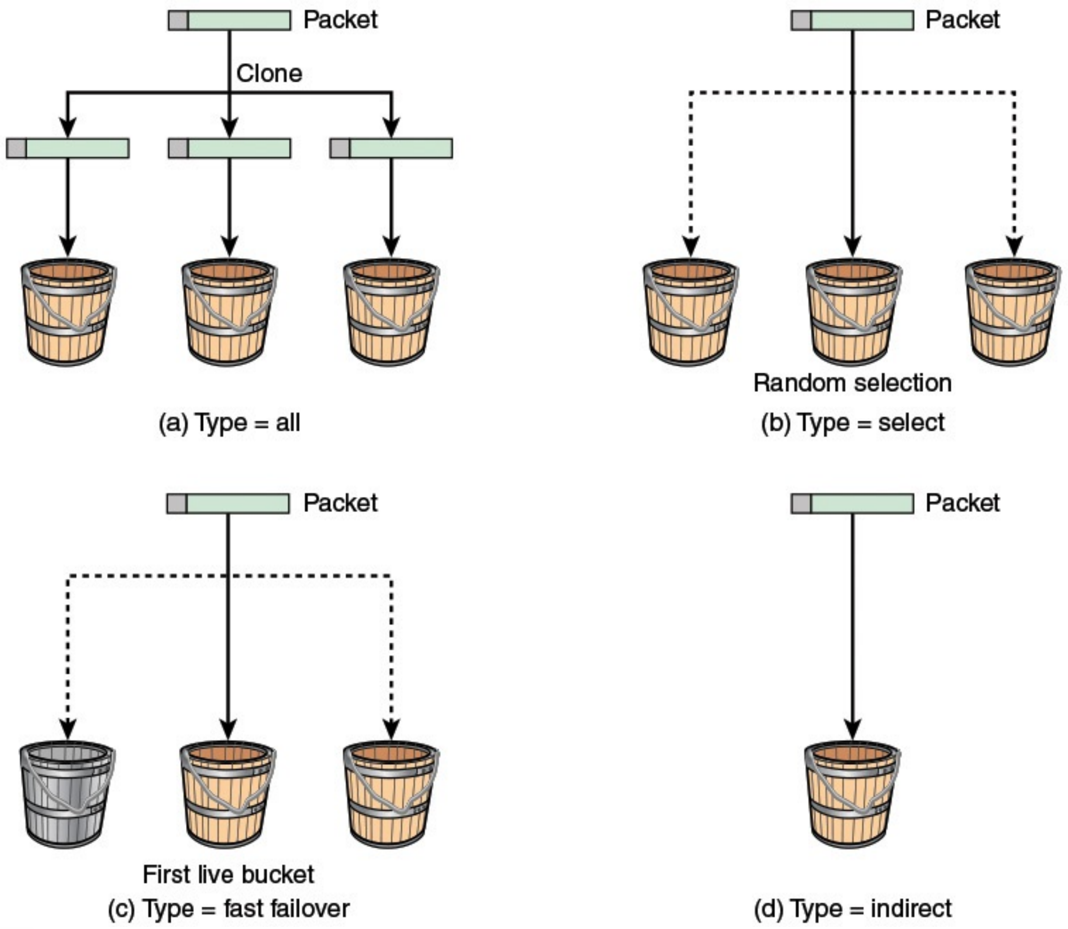
Software Defined Networks – Group Table

Group tables and group actions enable OpenFlow to represent a set of ports as a single entity for forwarding packets.

Group Tables are filled with Group Entries:

- **Group identifier:** A 32-bit unsigned integer uniquely identifying the group. A **group** is defined as an entry in the group table.
- **Group type:** To determine group semantics, as explained subsequently.
- **Counters:** Updated when packets are processed by a group.
- **Action buckets:** An ordered list of action buckets, where each action bucket contains a set of actions to execute and associated parameters.

Software Defined Networks – Group Table



Software Defined Networks – Flow Table Example:

Header Fields	Counters	Actions	Priority
If ingress port == 2		Drop packet	32768
if IP_addr == 129.79.1.1		re-write to 10.0.1.1, forward port 3	32768
if Eth Addr == 00:45:23		add VLAN id 110, forward port 2	32768
if ingress port == 4		forward port 5, 6	32768
if Eth Type == ARP		forward CONTROLLER	32768
If ingress port == 2 && Eth Type == ARP		forward NORMAL	40000

Each Flow Table entry has two timers: **idle_timeout**
seconds of no matching packets after which the flow is removed
zero means never timeout

hard_timeout
seconds after which the flow is removed
zero mean never timeout

Software Defined Networks – OpenFlow Messages:

Message	Description
Controller to Switch	
Features	Request the capabilities of a switch. Switch responds with a features reply that specifies its capabilities.
Configuration	Set and query configuration parameters. Switch responds with parameter settings.
Modify-State	Add, delete, and modify flow/group entries and set switch port properties.
Read-State	Collect information from switch, such as current configuration, statistics, and capabilities.
Packet-out	Direct packet to a specified port on the switch.
Barrier	Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.
Role-Request	Set or query role of the OpenFlow channel. Useful when switch connects to multiple controllers.
Asynchronous-Configuration	Set filter on asynchronous messages or query that filter. Useful when switch connects to multiple controllers.

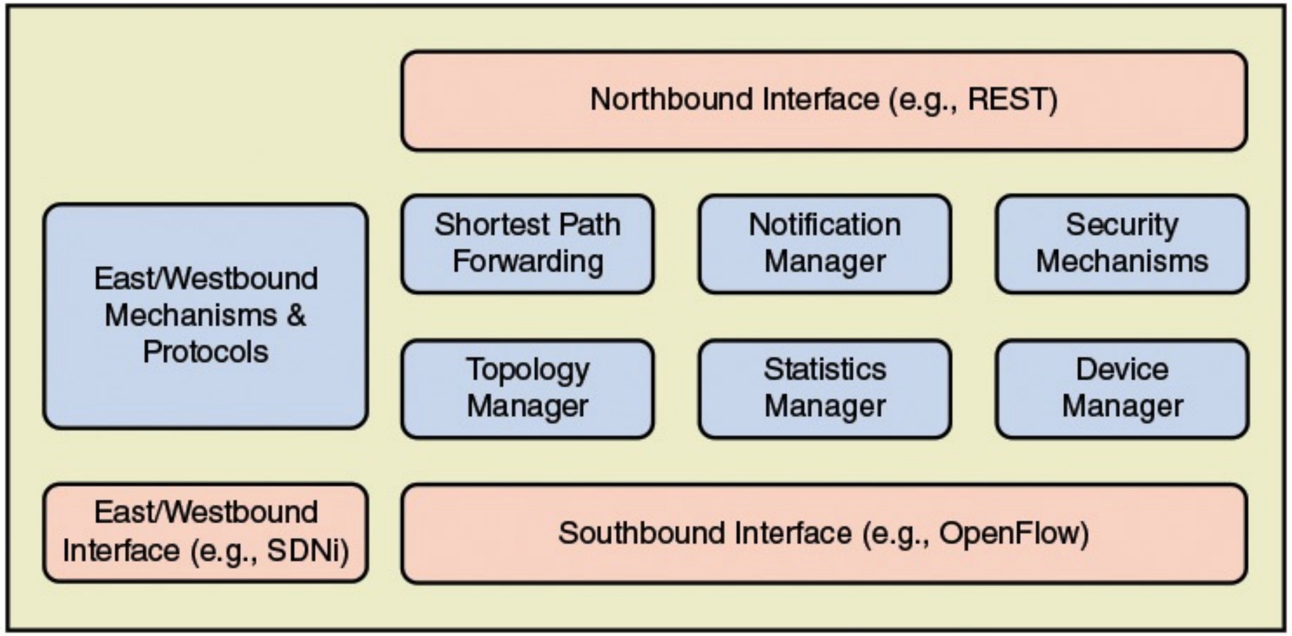
Software Defined Networks – OpenFlow Messages:

Asynchronous	
Packet-in	Transfer packet to controller.
Flow-Removed	Inform the controller about the removal of a flow entry from a flow table.
Port-Status	Inform the controller of a change on a port.
Role-Status	Inform controller of a change of its role for this switch from master controller to slave controller.
Controller-Status	Inform the controller when the status of an OpenFlow channel changes. This can assist failover processing if controllers lose the ability to communicate among themselves.
Flow-monitor	Inform the controller of a change in a flow table. Allows a controller to monitor in real time the changes to any subsets of the flow table done by other controllers.

Symmetric	
Hello	Exchanged between the switch and controller upon connection startup.
Echo	Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply.
Error	Used by the switch or the controller to notify problems to the other side of the connection.
Experimenter	For additional functionality.

Software Defined Networks – Control Plane

Controller typical functions



Software Defined Networks – Control Plane

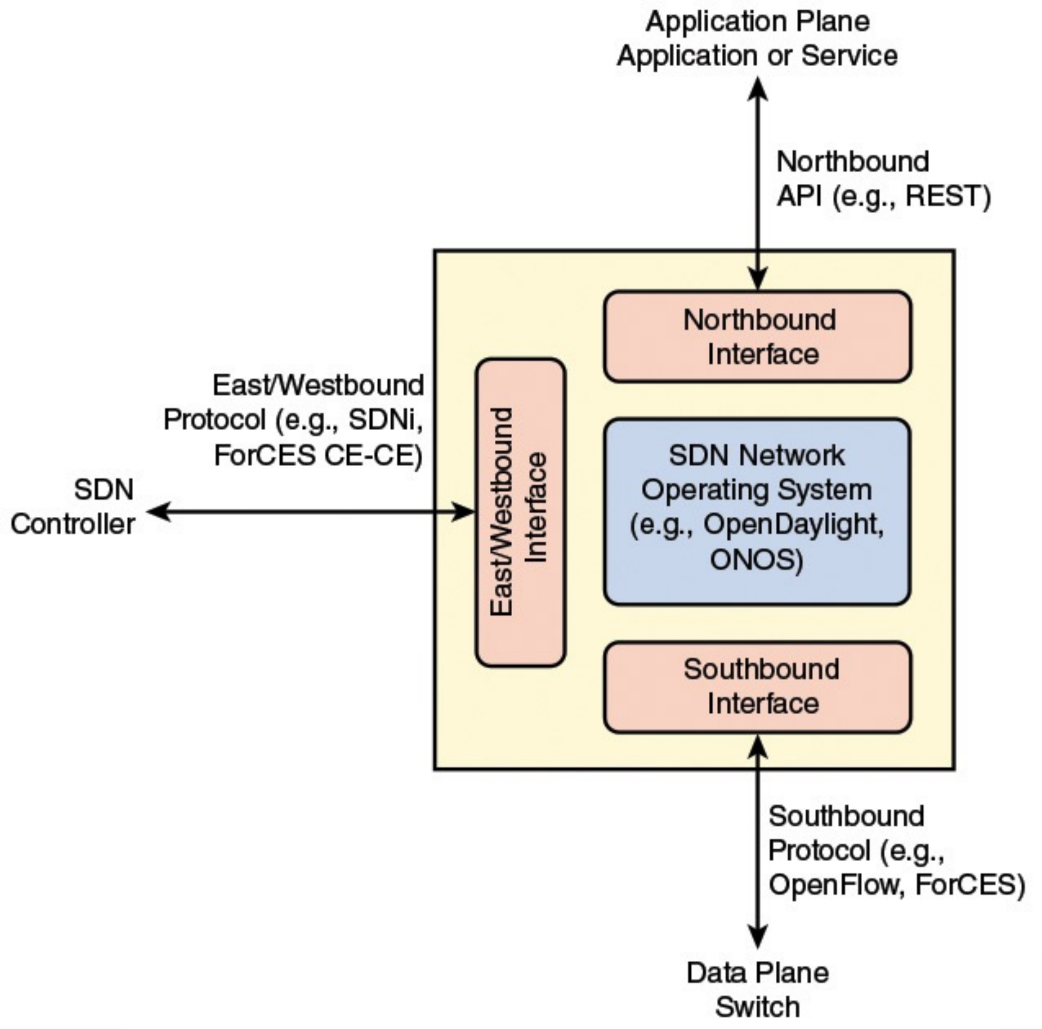
Most prominent Controllers

- **OpenDaylight:** An open source platform for network programmability to enable SDN, written in Java. OpenDaylight was founded by Cisco and IBM, and its membership is heavily weighted toward network vendors.
- **Open Network Operating System (ONOS):** An open source SDN NOS, initially released in 2014. It is a nonprofit effort funded and developed by a number of carriers, such as AT&T and NTT, and other service providers.
- **Ryu:** An open source component-based software defined networking framework supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc.
- **Floodlight:** An open source package developed by Big Switch Networks. Although its beginning was based on Beacon, it was built using Apache Ant, which is a very popular software build tool that makes the development of Floodlight easier and more flexible.

Configuration and Management of Networks

Software Defined Networks – Control Plane

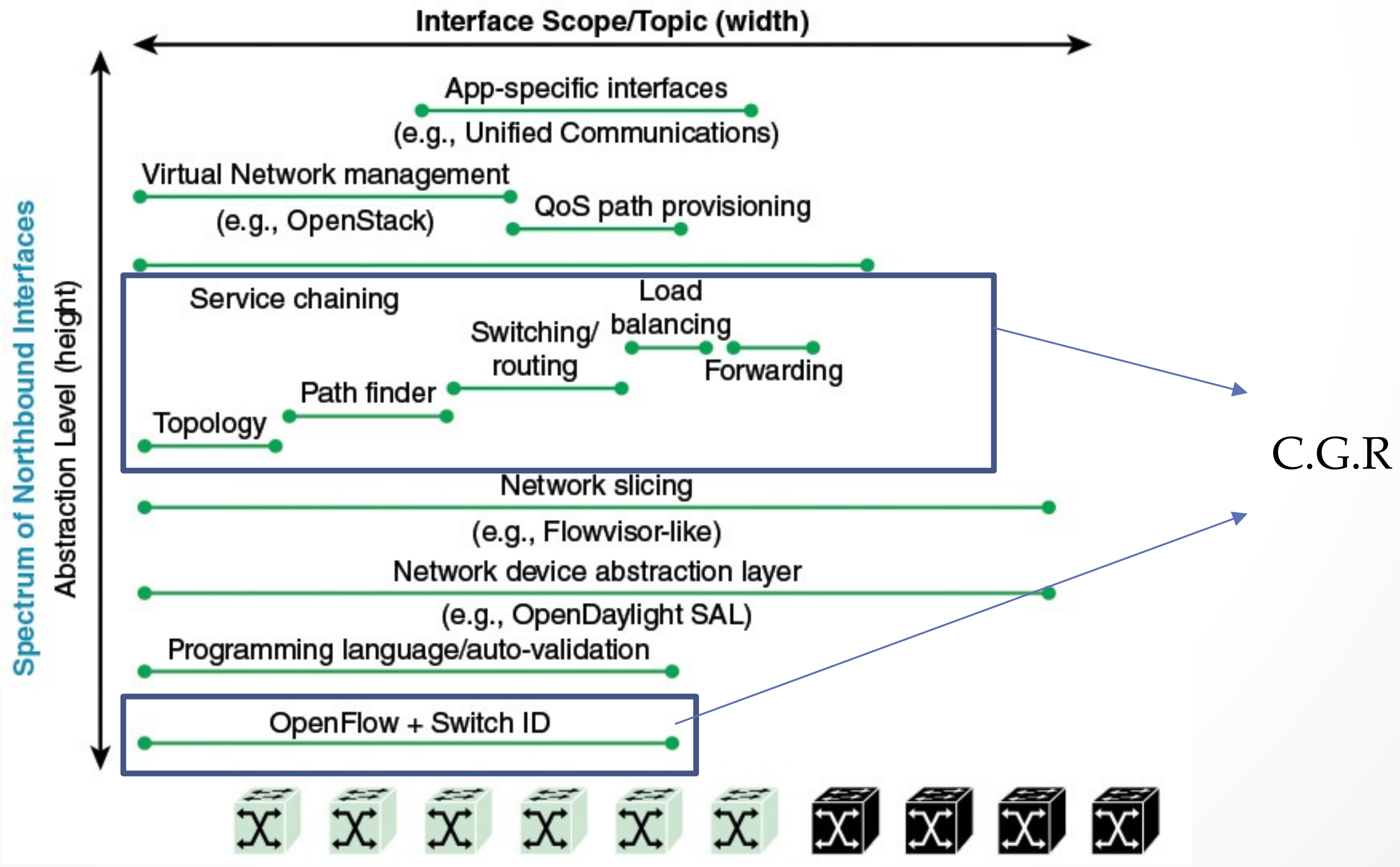
Interfaces



Software Defined Networks – Control plane SDNs

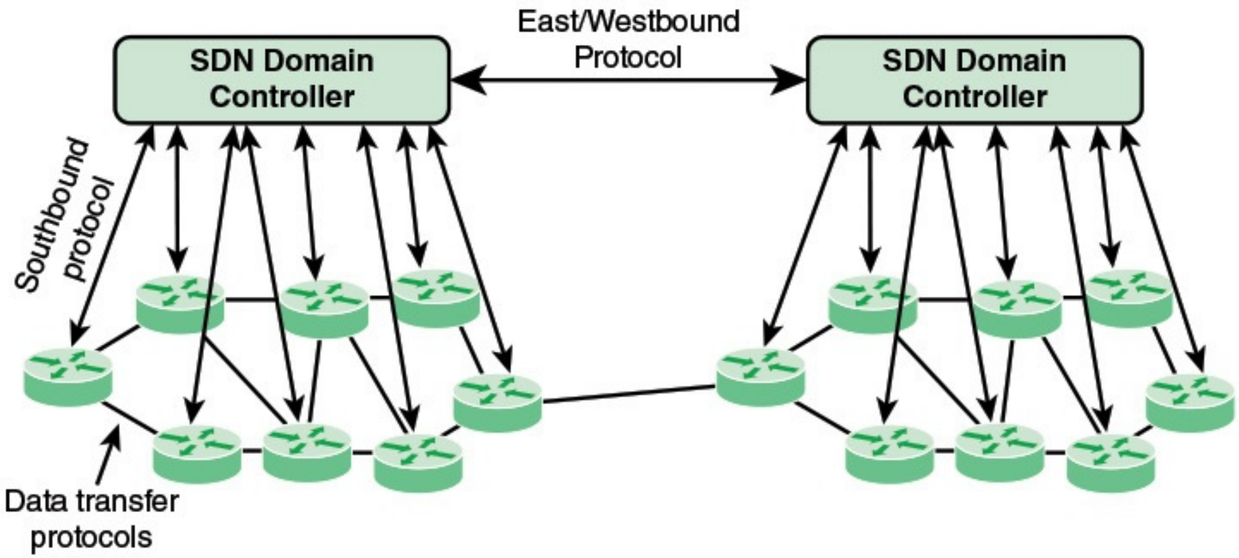
Northbound API

Programming Interface for applications and orchestration system. Several “latitudes” are needed



Software Defined Networks – Control plane SDNs

Logically Distributed Controllers



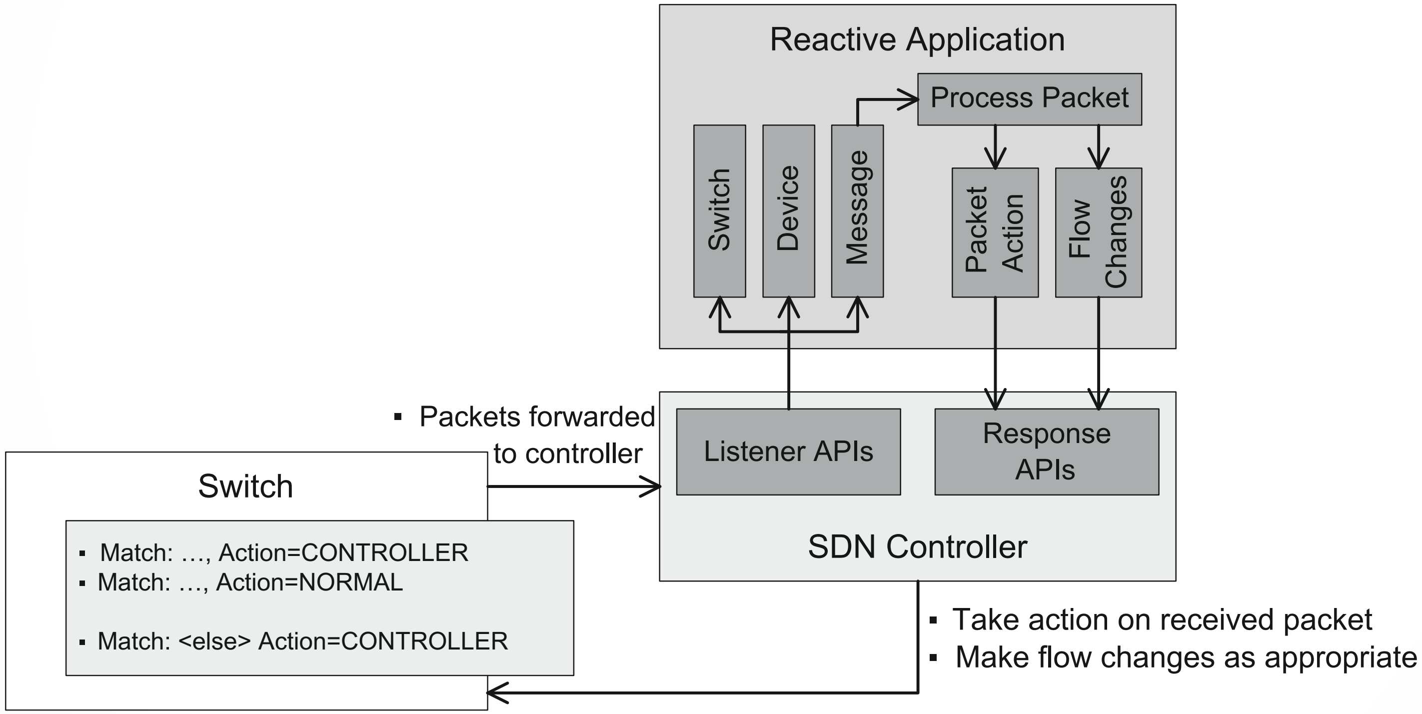
- **Scalability**: The number of devices an SDN controller can feasibly manage is limited. Therefore, a reasonably large network may need to deploy multiple SDN controllers.
- **Reliability**: The use of multiple controllers avoids the risk of a single point of failure.

Software Defined Networks – Programming SDNs

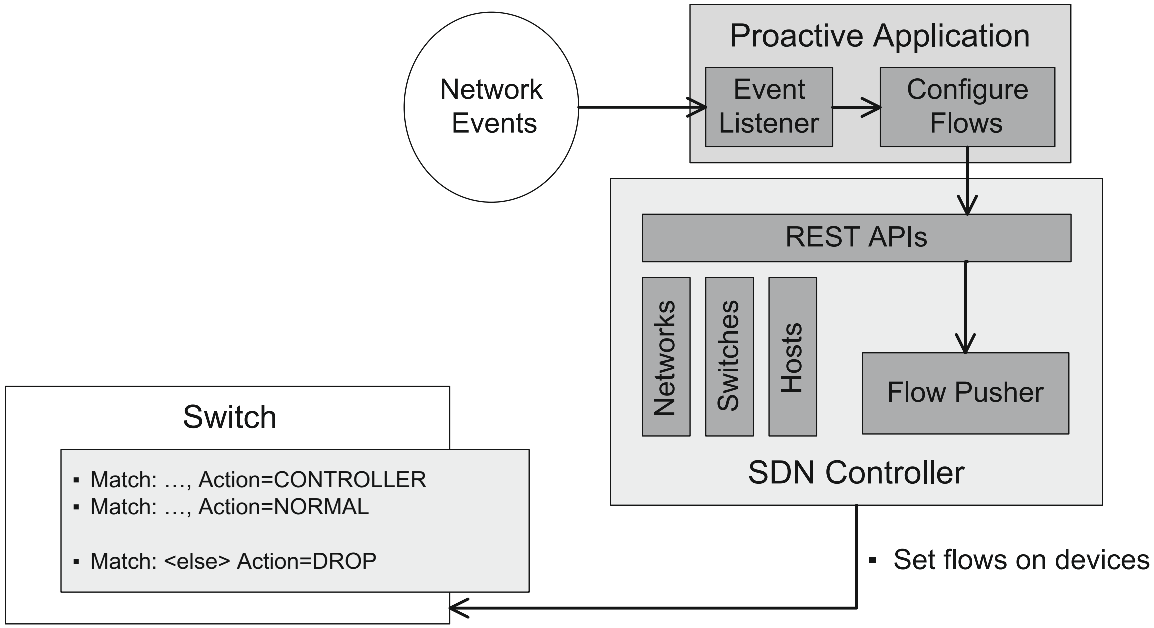
OpenFlow: Programming at this level of abstraction is not easy!

- Difficult to perform multiple independent tasks (e.g. routing, access control)
- OpenFlow is a low level of abstraction
- Race Conditions, if switch-level rules are not installed properly

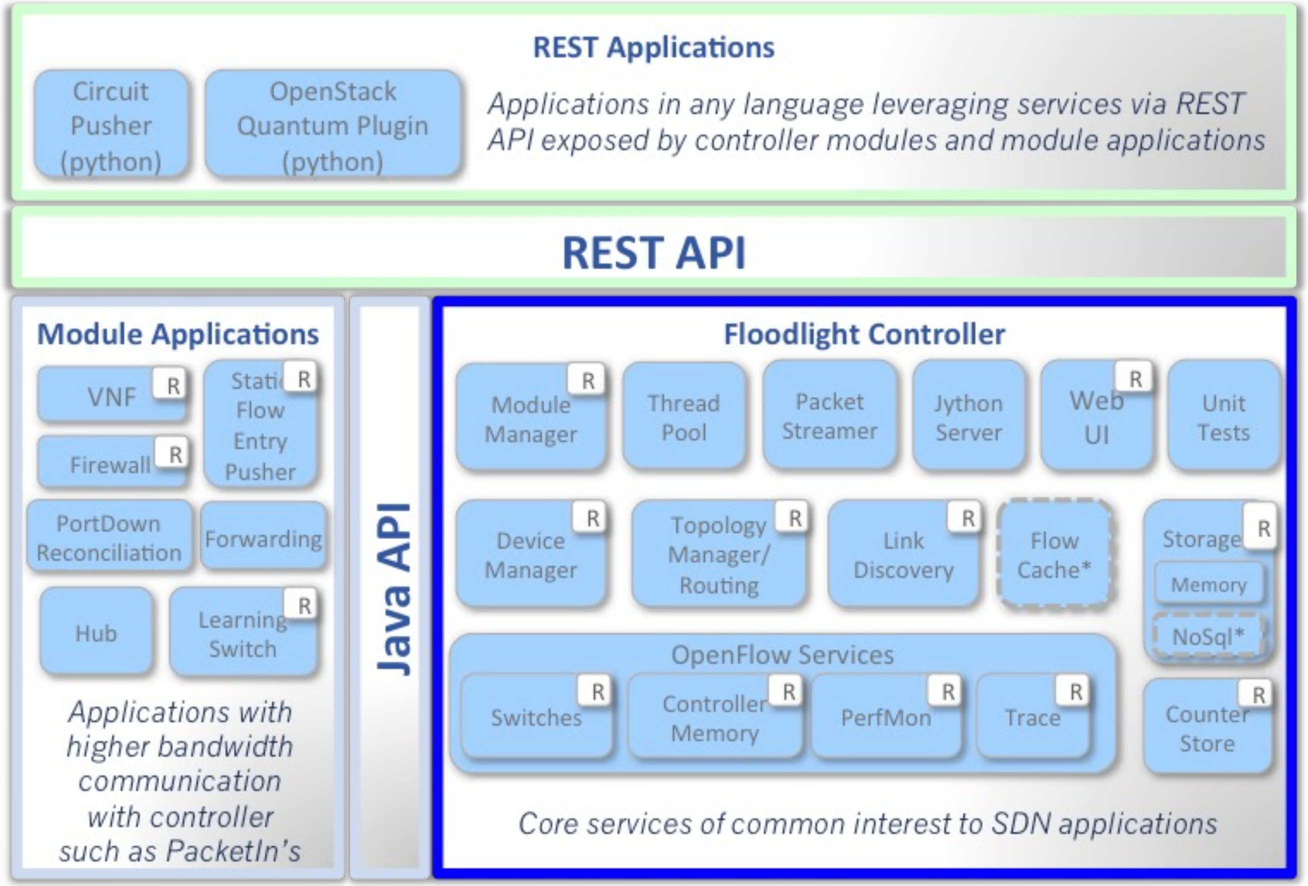
Software Defined Networks – Programming SDNs Application Design



Software Defined Networks – Programming SDNs Application Design



Software Defined Networks – Programming SDNs Floodlight controller



* Interfaces defined only & not implemented: FlowCache, NoSql

Software Defined Networks – Programming SDNs Floodlight controller - examples

Configurations of the modules to run (*.properties file):

```
floodlight.modules=\
net.floodlightcontroller.storage.memory.MemoryStorageSource,\
net.floodlightcontroller.core.internal.FloodlightProvider,\
net.floodlightcontroller.threadpool.ThreadPool,\
net.floodlightcontroller.perfmon.PktInProcessingTime,\
net.floodlightcontroller.staticentry.StaticEntryPusher,\
net.floodlightcontroller.topology.TopologyManager,\
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager,\
net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl,\
net.floodlightcontroller.CGRL2Switch.CGRL2Switch
net.floodlightcontroller.core.internal.FloodlightProvider.openflowPort=6633
net.floodlightcontroller.core.internal.FloodlightProvider.role=ACTIVE
net.floodlightcontroller.core.internal.OFSwitchManager.clearTablesOnInitialHandshakeAsMaster=YES
net.floodlightcontroller.core.internal.OFSwitchManager.clearTablesOnEachTransitionToMaster=YES
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePath=/path/to/your/keystore-file.jks
net.floodlightcontroller.core.internal.OFSwitchManager.keyStorePassword=your-keystore-password
net.floodlightcontroller.core.internal.OFSwitchManager.useSsl=NO
```

Adding modules (net.floodlightcontroller.core.module.IFloodlightModule)

```
net.floodlightcontroller.loadbalancer.LoadBalancer
net.floodlightcontroller.linkdiscovery.internal.LinkDiscoveryManager
net.floodlightcontroller.devicemanager.internal.DeviceManagerImpl
net.floodlightcontroller.firewall.Firewall
net.floodlightcontroller.accesscontrollist.ACL
net.floodlightcontroller.dhcpserver.DHCPServer
net.floodlightcontroller.learningswitch.LearningSwitch
net.floodlightcontroller.statistics.StatisticsCollector
net.floodlightcontroller.routing.RoutingManager
net.floodlightcontroller.CGRL2Switch.CGRL2Switch
```

Software Defined Networks – Programming SDNs Floodlight controller - examples

Treating the reception of a Packet_In message:

```
public class CGRmodule implements IFloodlightModule, IOFMessageListener, IOFSwitchListener {
```

```
// IOFMessageListener

@Override
public Command receive(IOFSwitch sw, OFMessage msg, FloodlightContext cntx) {
    switch (msg.getType()) {
    case PACKET_IN:
        return this.processPacketInMessage(sw, (OFPacketIn) msg, cntx);
    case FLOW_REMOVED:
        return this.processFlowRemovedMessage(sw, (OFFlowRemoved) msg);
    case ERROR:
        log.info("received an error {} from switch {}", msg, sw);
        return Command.CONTINUE;
    default:
        log.error("received an unexpected message {} from switch {}", msg, sw);
        return Command.CONTINUE;
    }
}
```

Software Defined Networks – Programming SDNs Floodlight controller - examples

Treating the reception of a Packet_In message:

```
private Command processPacketInMessage(IOFSwitch sw, OFPacketIn pi, FloodlightContext cntx) {
    OFPort inPort = (pi.getVersion().compareTo(OFVersion.OF_12) < 0 ? pi.getInPort() : pi.getMatch().get(MatchField.IN_PORT));

    /*Read the Packet_In Message Payload (EThernet packet) in to an Ethernet Object*/
    Ethernet eth = IFloodlightProviderService.bcStore.get(cntx, IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
    /* Read packet header attributes into a Match object */
    MacAddress sourceMac = eth.getSourceMACAddress();
    MacAddress destMac = eth.getDestinationMACAddress();

    if (sourceMac == null) {
        sourceMac = MacAddress.NONE;
    }
    if (destMac == null) {
        destMac = MacAddress.NONE;
    }

    if ((destMac.getLong() & 0xfffffffffff0L) == 0x0180c2000000L) {
        if (log.isTraceEnabled()) {
            log.trace("ignoring packet addressed to 802.1D/Q reserved addr: switch {} dest MAC {}",
                    new Object[]{ sw, destMac.toString() });
        }
        return Command.STOP;
    }
    if ((!sourceMac.isBroadcast())&&(!sourceMac.isMulticast())) {
        log.info("Unicast packet received: switch {} Ethertype {}",
                new Object[]{ sw, eth.getEtherType() });
      // If source MAC is a unicast address, learn the port for this MAC/VLAN

    }
    //check if port for destination MAC is known
    // If so output flow-mod and/or packet

    //for now it floods trough all ports like a hub.
    SwitchCommands.sendPacketOutPacketIn(sw, OFPort.FLOOD, pi);
```

Software Defined Networks – Programming SDNs Floodlight controller - examples

Creating rules:

```java
public static boolean installRule(IOFSwitch sw, TableId table, short priority,
            Match matchCriteria, List<OFInstruction> instructions,List<OFAction> actions,
            short hardTimeout, short idleTimeout, OFBufferId bufferId, boolean ReceivedRemoved)
{
  OFFlowMod.Builder rule = sw.getOFFactory().buildFlowAdd();
    rule.setHardTimeout(hardTimeout);
    rule.setIdleTimeout(idleTimeout);
    rule.setPriority(priority);
    rule.setTableId(table);
    rule.setBufferId(bufferId);

    rule.setMatch(matchCriteria);
    if (instructions !=null){
      rule.setInstructions(instructions);
    }
    else
      if (actions!=null){
        rule.setActions(actions);
      }
    // if we wan to to receive Flow_Removed Messages for this OpenFlow FlowEntry
Set<OFFlowModFlags> sfmf = new HashSet<OFFlowModFlags>();
if (ReceivedRemoved) {
  sfmf.add(OFFlowModFlags.SEND_FLOW_REM);
  rule.setFlags(sfmf);
}

    try
    {
        sw.write(rule.build());
        log.debug("Installing rule: "+rule);
    }
    catch (Exception e)
    {
        log.error("Failed to install rule: "+rule);
        return false;
    }

    return true;
}
```

Software Defined Networks – Programming SDNs Floodlight controller - examples

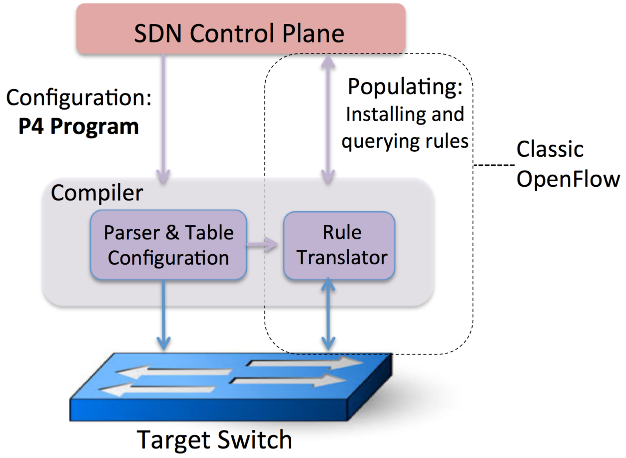
Creating Match clause; Actions and ApplyActions Instruction :

```
protected Match createMatchFromPacket(IOFSwitch sw, OFPort inPort, FloodlightContext cntx) {
    // The packet in match will only contain the port number.
    // We need to add in specifics for the hosts we're routing between.
    Ethernet eth = IFloodlightProviderService.bcStore.get(cntx, IFloodlightProviderService.CONTEXT_PI_PAYLOAD);
    MacAddress srcMac = eth.getSourceMACAddress();
    MacAddress dstMac = eth.getDestinationMACAddress();

    Match.Builder mb = sw.getOFFactory().buildMatch();
    mb.setExact(MatchField.IN_PORT, inPort)
    .setExact(MatchField.ETH_SRC, srcMac)
    .setExact(MatchField.ETH_DST, dstMac);
    return mb.build();
}
```

```
OFActions actions = sw.getOFFactory().actions(); //actions builder
List<OFAction> al = new ArrayList<OFAction>();
OFActionOutput output = actions.buildOutput()
        .setPort(outPort)    // outPort is the port trough which the sw should send the Matching Packets
        .setMaxLen(0xffFFffFF)
        .build();
al.add(output);

if (pi.getVersion().compareTo(OFVersion.OF_13)==0){
        OFInstructions instructions =  sw.getOFFactory().instructions(); //instructions builder
        OFInstructionApplyActions applyActions = instructions.buildApplyActions().setActions(al).build();
        ArrayList<OFInstruction> instructionList = new ArrayList<OFInstruction>();
        instructionList.add(applyActions); //add the applyActions Instruction to the Instruction list
```

Configuration and Management of Networks

SDN – Programmable Data plane (P4)



- P4—used to configure a switch, telling it how packets are to be processed
- OpenFlow - designed to populate the forwarding tables in fixed function switches

⇩

Tell the switch how to operate, rather than be constrained by a fixed switch design

SDN – Programmable Data plane (P4)

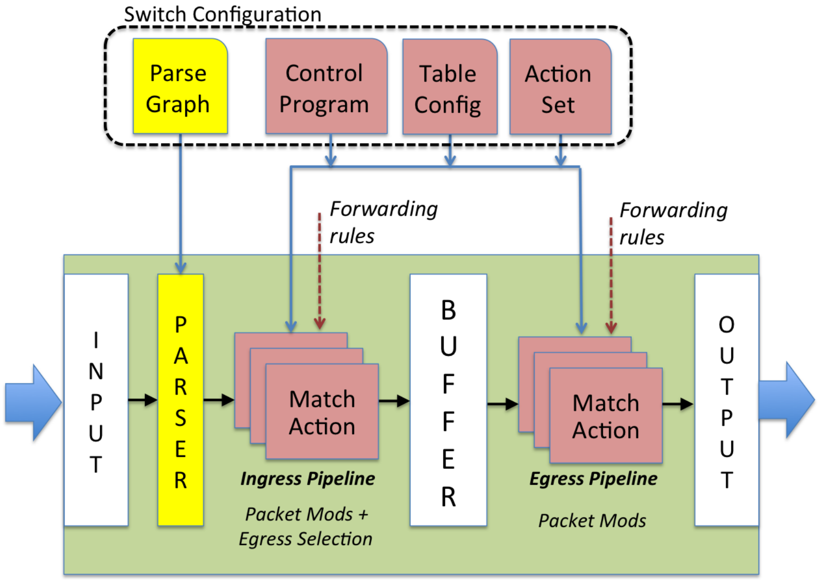
- **PISA (Protocol Independent Switch Architecture) : Flexible Match+Action ASICs**
  - Intel Flexpipe, Cisco Doppler, Cavium (Xpliant), Barefoot Tofino, …
- **NPU (Network processing unit)**
  - EZchip, Netronome, …
- **CPU (Virtual Software Devices)**
  - Open Vswitch, eBPF, DPDK, VPP…
- **FPGA**
  - Xilinx, Altera, …

⇩

These devices let us tell them how to process packets.

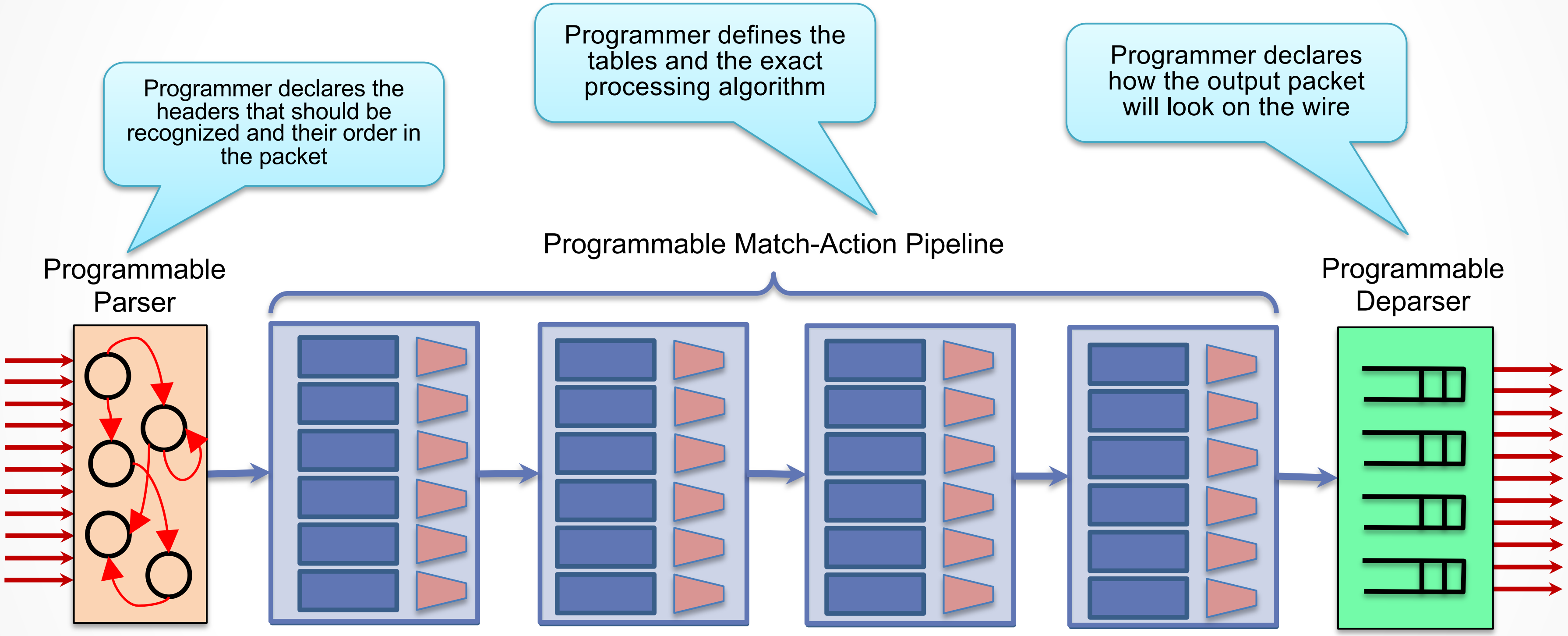
Configuration and Management of Networks

P4 – Example simple switch:



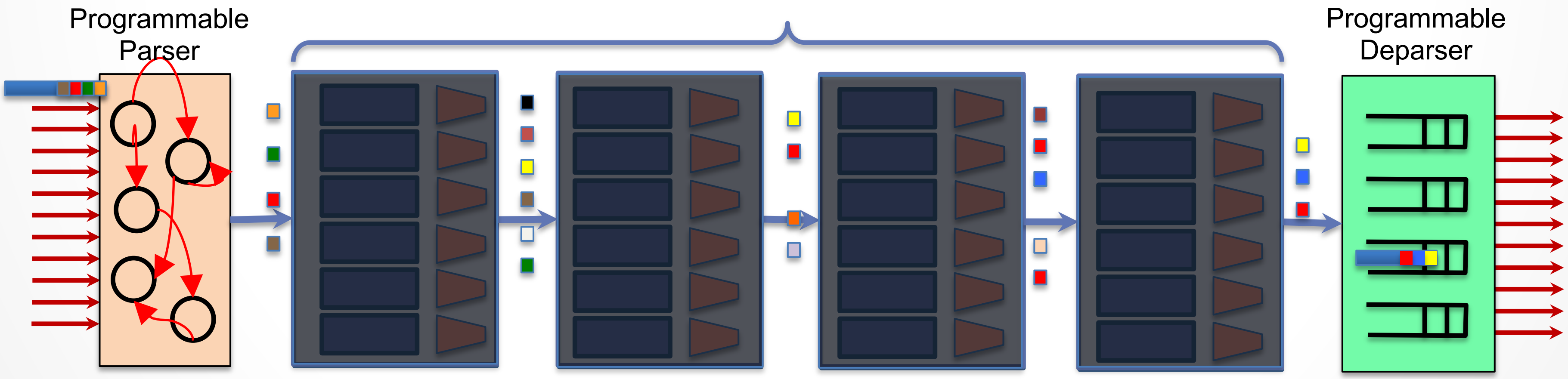
- Programmable parser to allow new headers (Openflow assumes a fixed parser)
- OpenFlow assumes the match+action stages are in series, in P4 they can be in parallel.

P4 : (Based in Protocol-Independent Switch Architecture)



P4 : (Based Protocol-Independent Switch Architecture)

- **Packet is parsed into individual headers (parsed representation)**
- **Headers and intermediate results can be used for matching and actions**
- **Headers can be modified, added or removed**
- **Packet is deparsed (serialized)**



P4 : Program example

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet,
   out headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {

    state start { transition accept; }
}

control MyVerifyChecksum(inout headers hdr, inout metadata
meta) {   apply {  }   }

control MyIngress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
apply {
        if (standard_metadata.ingress_port == 1) {
            standard_metadata.egress_spec = 2;
        } else if (standard_metadata.ingress_port == 2) {
            standard_metadata.egress_spec = 1;
        }
    }
}
```

```
control MyEgress(inout headers hdr,
   inout metadata meta,
   inout standard_metadata_t standard_metadata) {
    apply {  }
}

control MyComputeChecksum(inout headers hdr, inout metadata
meta) {
     apply { }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply { }
}

V1Switch(
   MyParser(),
   MyVerifyChecksum(),
   MyIngress(),
   MyEgress(),
   MyComputeChecksum(),
   MyDeparser()
) main;
```

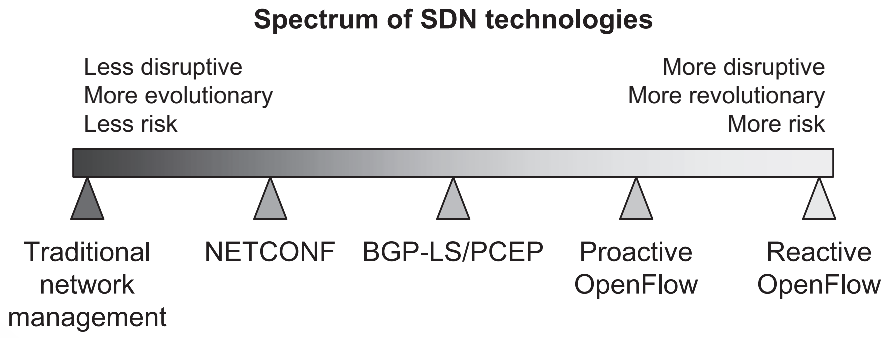
Software Defined Networks – Flavours

So far we have been discussing Open SDN:

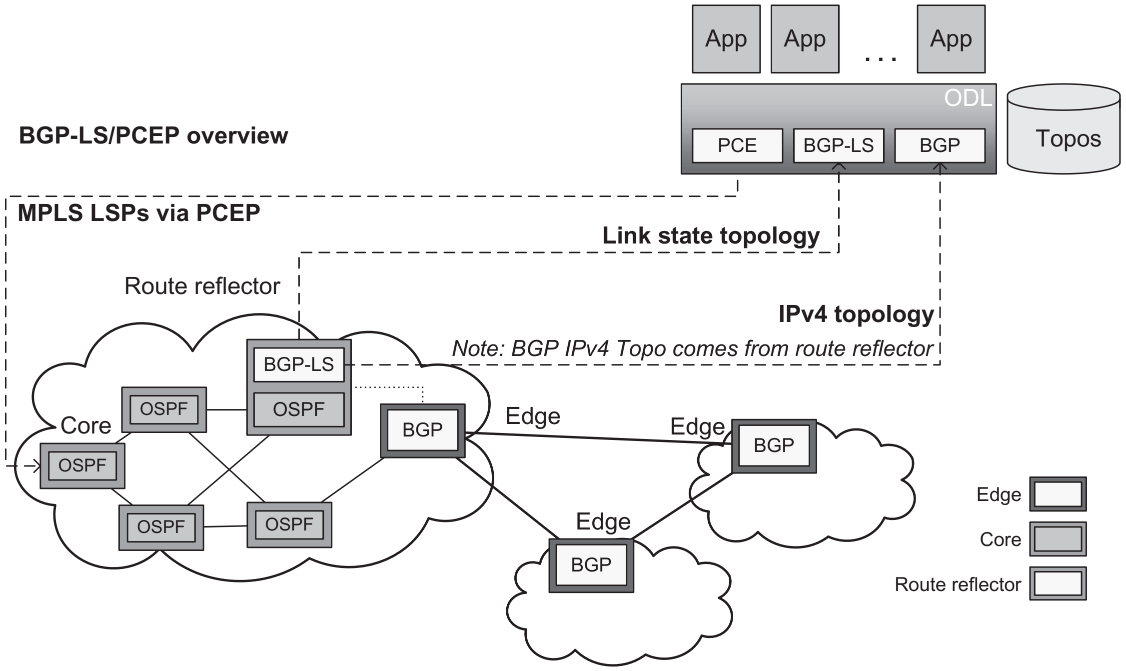
Generic Hardware – no functionality besides forwarding tables

Other flavours include – API based SDN:

This can be seen as Network Management SDN



Software Defined Networks – API based SDN- using existing protocols



- **BGP-LS** is used to pass link-state (OSPF or IS-IS) *Interior Gateway Protocol* (IGP) information about topology to ODL.
- **PCE-P** is used to transmit routing information from the PCE Server to the PCE clients in the network. A PCE client is also more simply known as a *Path Computation Client* (PCC).
- **MPLS** will be used to forward packets throughout the network, using the Label Switched Paths (LSPs) transmitted to head-end nodes via PCE-P.

Software Defined Networks – API based SDN- control points

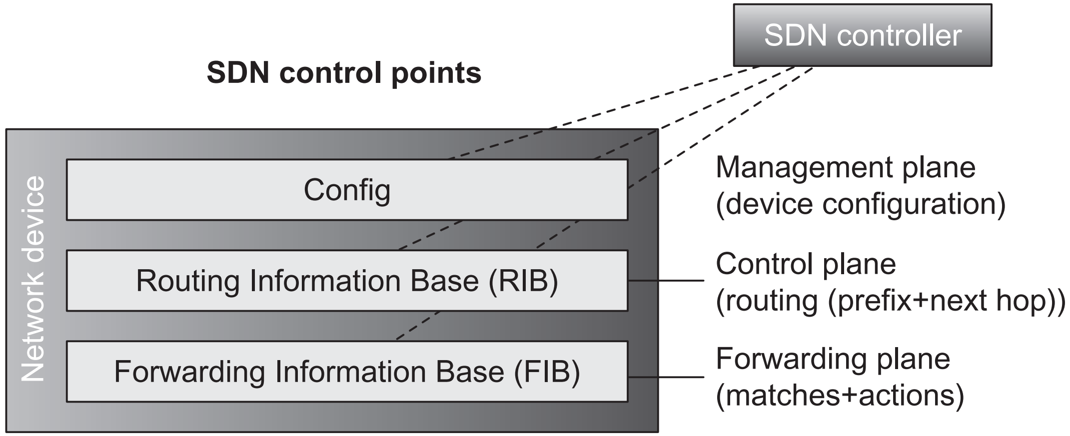
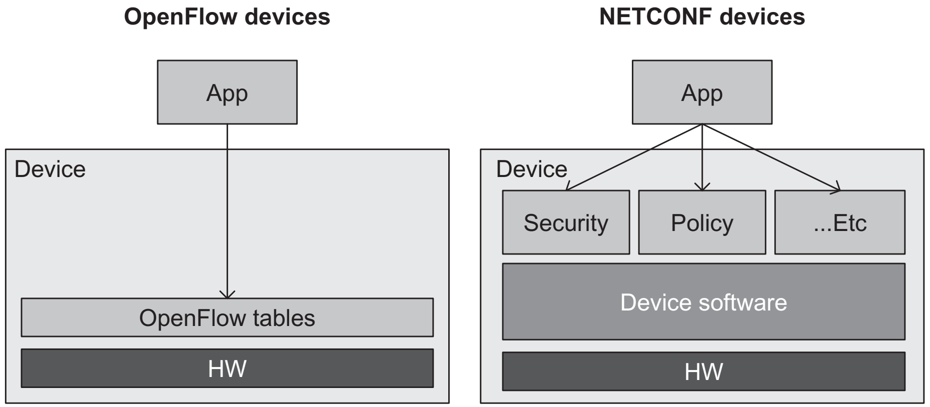


Table 7.1 Comparison of Existing Protocols for SDN

Protocol	Control Point	Details
NETCONF	Config	Interfaces, ACLs, Static routes
BGP-LS	-	Topology discovery is used to pass link-state IGP information about topology to ODL.
BGP	RIB	Topology discovery and setting RIB
PCE-P	MPLS	PCE to set MPLS LSPs. Used to transmit routing information from the PCE Server to the PCE Clients in the network.
BGP-FS	Flows	BGP-FlowSpec to set matches and actions

Configuration and Management of Networks

Software Defined Networks – API based SDN- Netconf



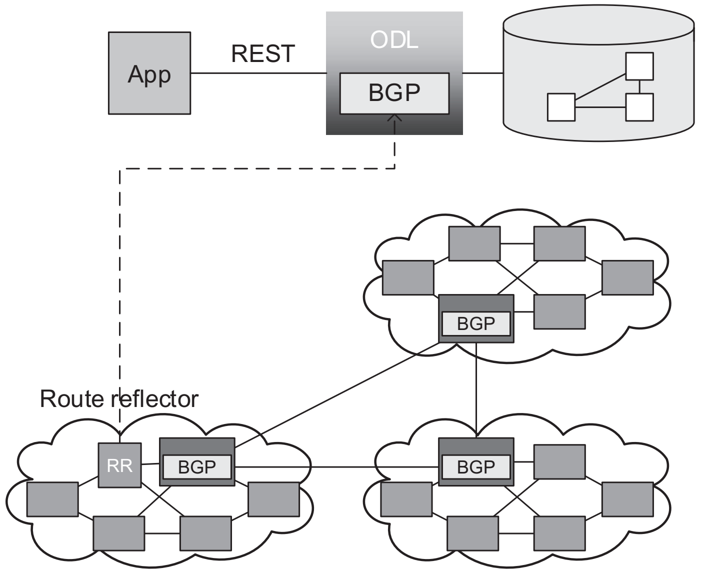
Successor to the SNMP (Simple Network Management) NETCONF has:

- Support for Remote Procedure Calls : Invoke operation in a device.
- Support for Notifications: Managed device can notify management station of events

Only configures the exposed capabilities of the device

Uses XML to communicate with devices, or a REST API (RESTCONF)

Software Defined Networks – API based SDN- BGP



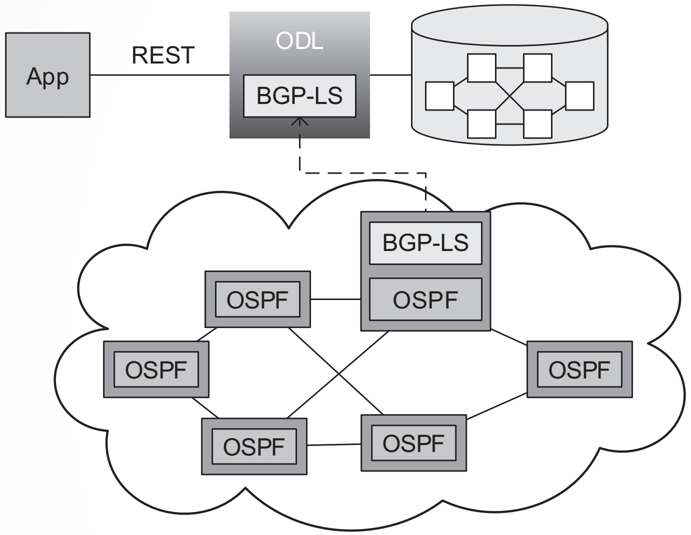
Obtain IPv4 topology:

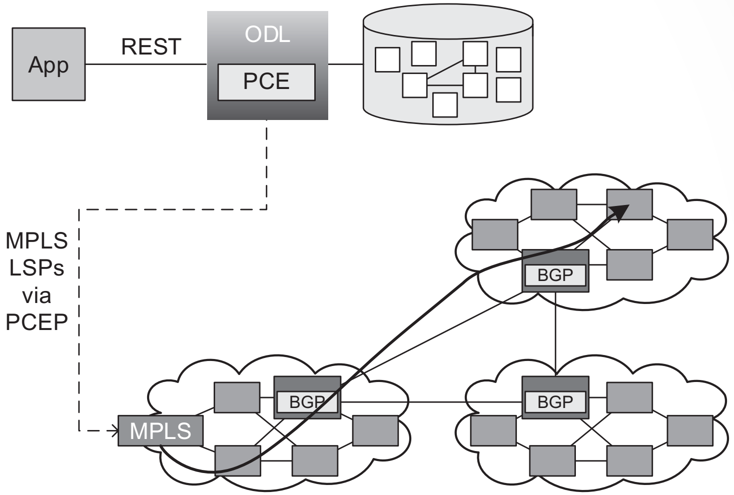
- BGP plugin in controller implements a BGP node

RIB configuration:

- Controller uses BGP plugin to advertise routes (injecting routes)

Software Defined Networks – API based SDN- BGP-LS/PCE-P



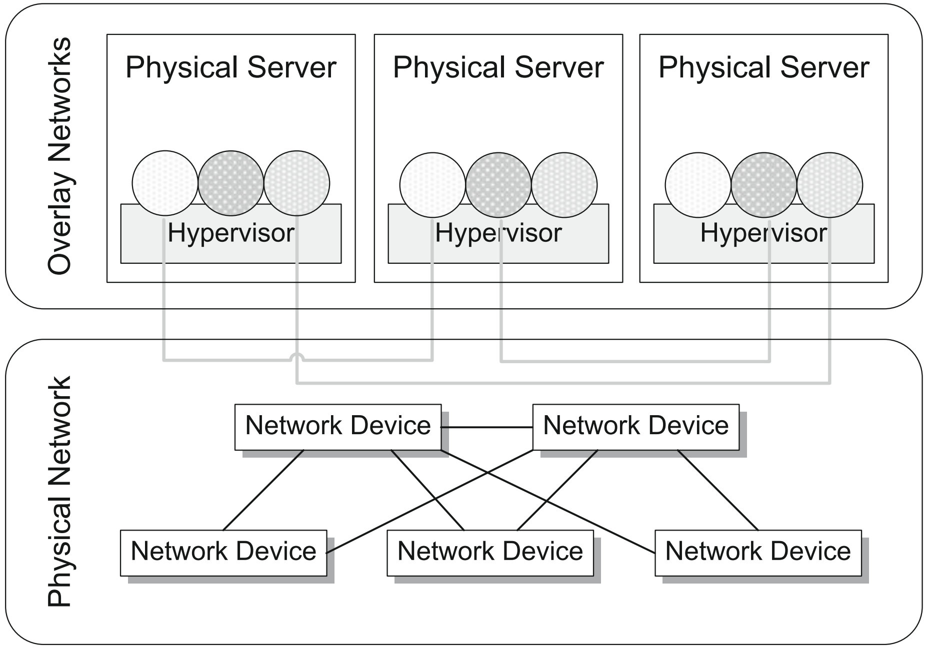


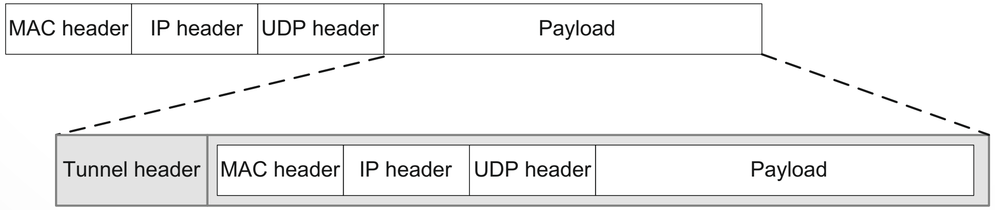
BGP-LS – Used to obtain link state IGP information to controller

PCE-P – Used to set LSP paths that unlike traditional LSPs can be inter-domain

Software Defined Networks – Via Overlay Virtual Networks

One of the prevailing solutions for Data centre Networks





Software Defined Networks – Via Overlay Virtual Networks

SDN is done with a controller interacting with virtual switches:

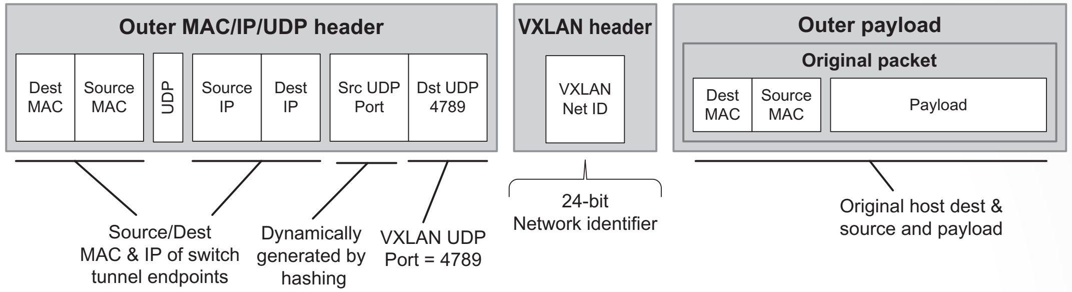
- Virtual Switches reside in the Hypervisors and connect VMs
- Traffic is forwarded between VSwitches via tunnels
- Controller knows end hosts macs, and mappings to tunnels
- Controller can use OpenFlow to configure VSwitchs and create the overlay networks.

Several Tunneling mechanisms

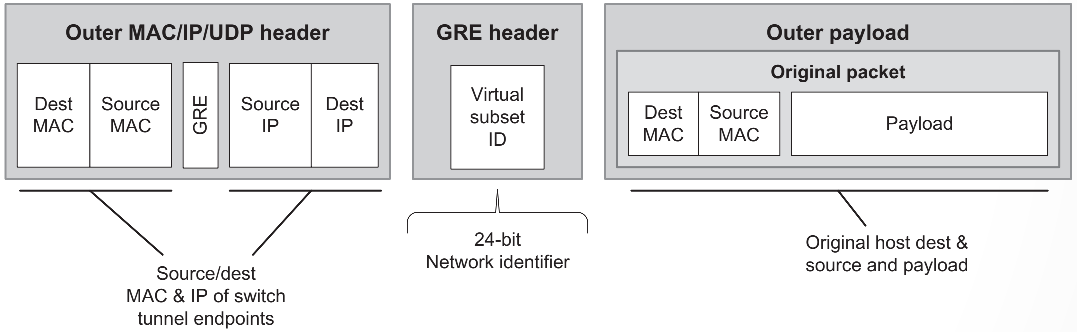
- VXLAN (cisco)
- NVGRE (Microsoft)
- STT (Nicira VMware)

Software Defined Networks – Via Overlay Virtual Networks

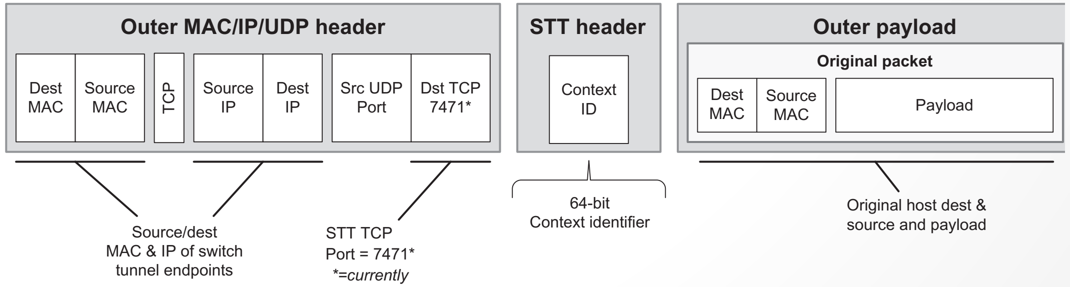
- VXLAN (cisco)



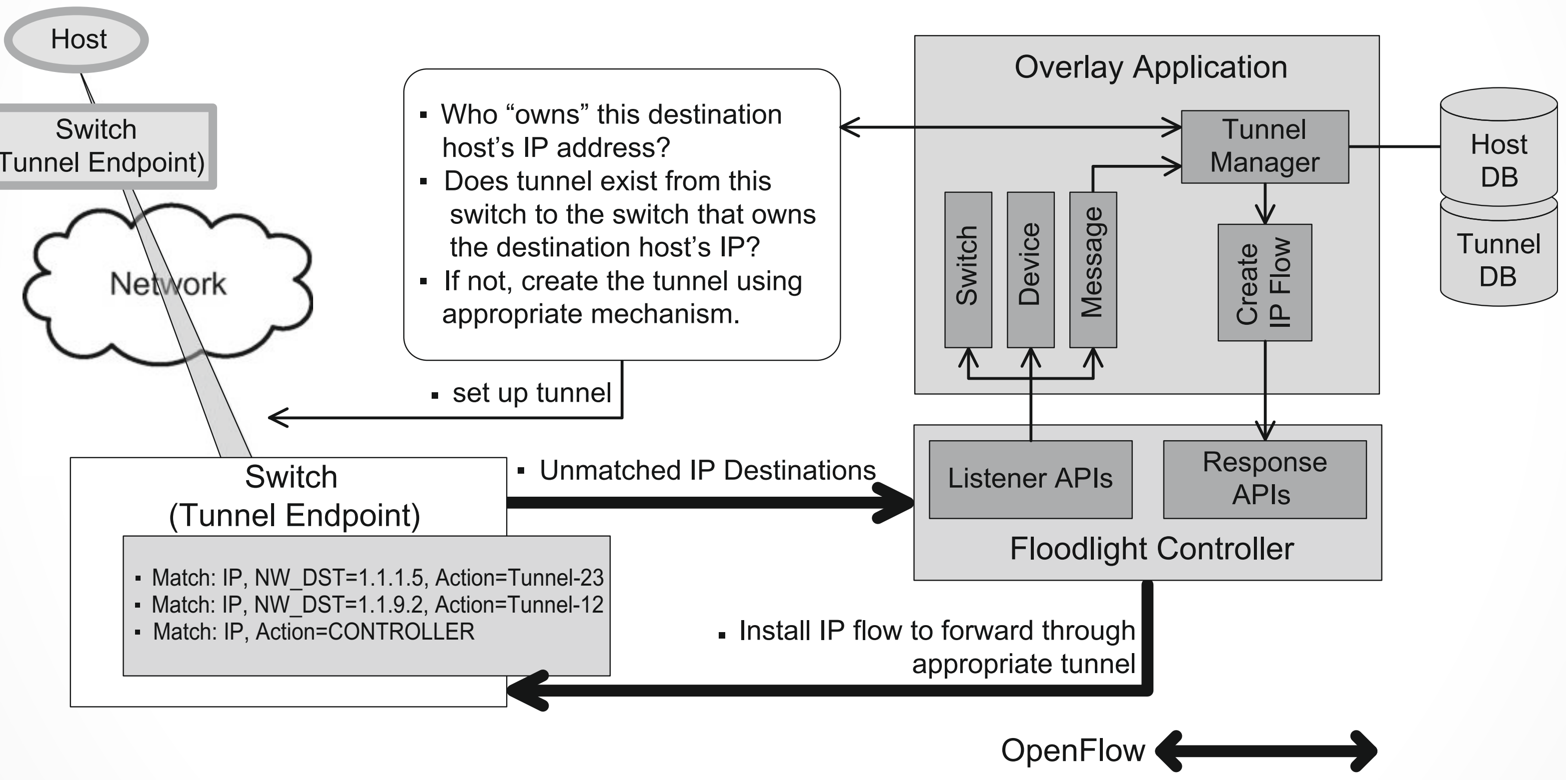
- NVGRE (Microsoft)



- STT (Nicira VMware)



Software Defined Networks – Via Overlay Virtual Networks



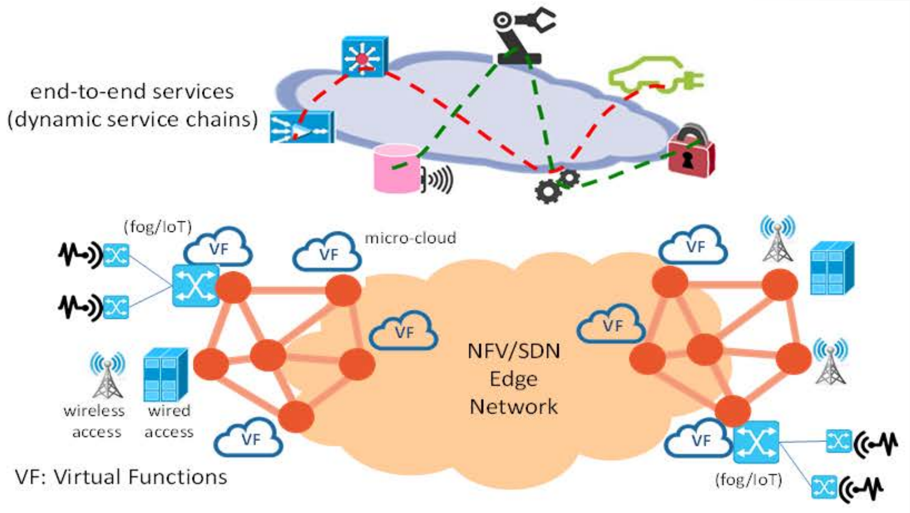
5 G networks and SDN-NFV

5G Networks - Requirements

5 G networks and SDN-NFV

5G Networks - Convergence between computing (Cloud) and communication systems (Network).

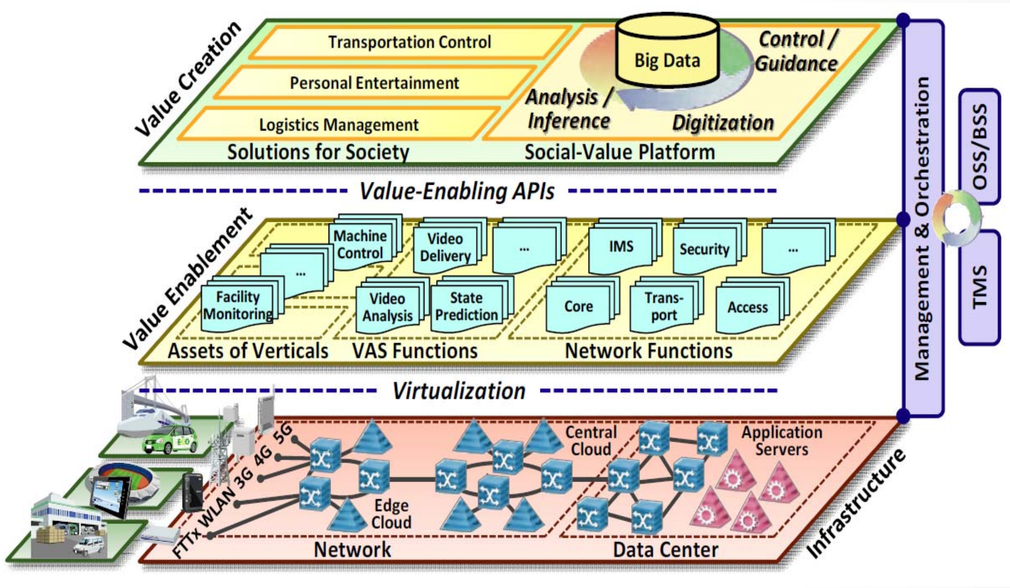
- Service platforms deployed at clouds in the core or micro-clouds at the edge:
  - Fog computing (computing along the network)
  - MEC (edge computing e.g. in base Stations)
- Composed of generalized Virtual Functions (VFs) providing Applications and Network services



Configuration and Management of Networks

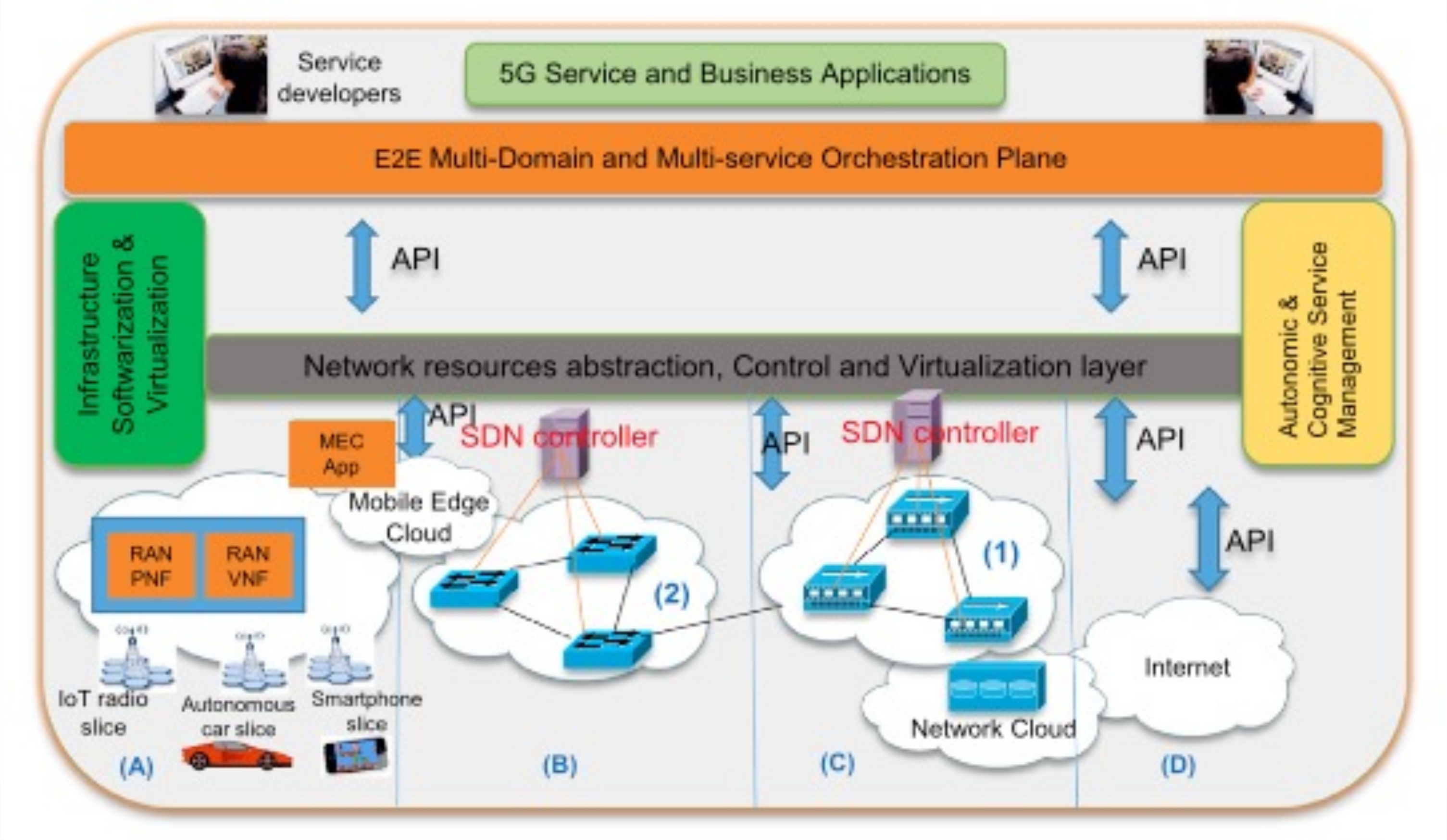
5 G networks and NFV

- Physical – Compute, storage, Network (Back-end-DCs, MEC and Fog; Core Network and RAN).
- Virtual - Application functions and Network functions as virtualized instances or entities (provide Services in isolation)
- Value – Top Level consuming APIs from virtual layer (With functional service and operational requirements)



Configuration and Management of Networks

5 G networks and NFV

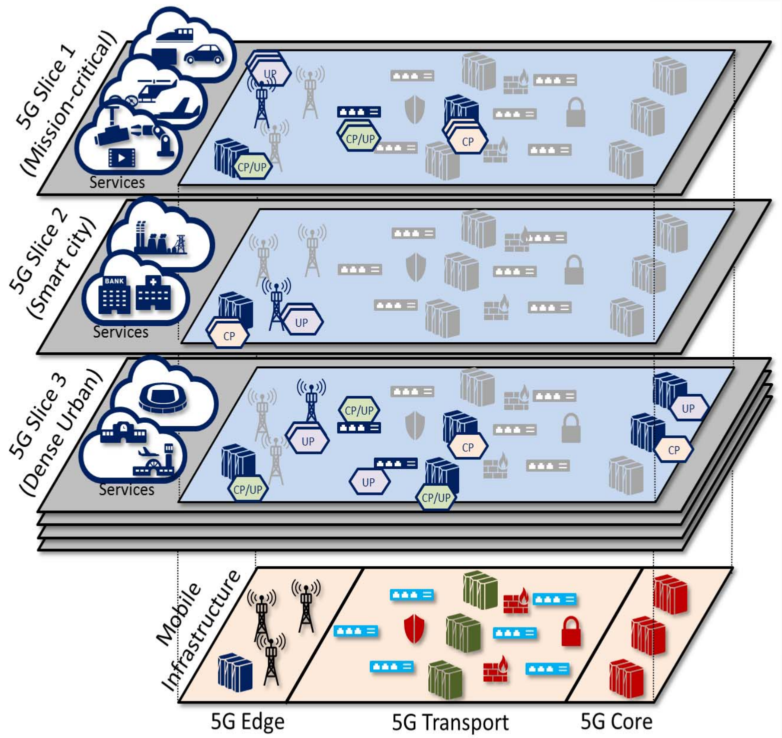


Software network technologies in 5G architecture. A indicates RAN; B = transport networks; C = core networks and D represents the Internet.

Configuration and Management of Networks

5 G networks and NFV

- 5G Slicing Concept:
  - Multiple logical self-contained networks, orchestrated in different ways according to their specific service requirements.
  - Temporarily owned by Tenants (A slice includes Physical, Virtualization and Service Layer – also called a Vertical)
  - Set of virtual network functions that run on the same infrastructure with a tailored orchestration.



Configuration and Management of Networks

5 G networks and NFV

Challenges:

- Seamless and flexible management of physical and virtualized resources across the three tiers.
- Agile end-to-end service orchestration for each respective service vertical, where each vertical may have multiple service instances.
- Enabling end-to-end connectivity services to each service instance, which is also programmable.

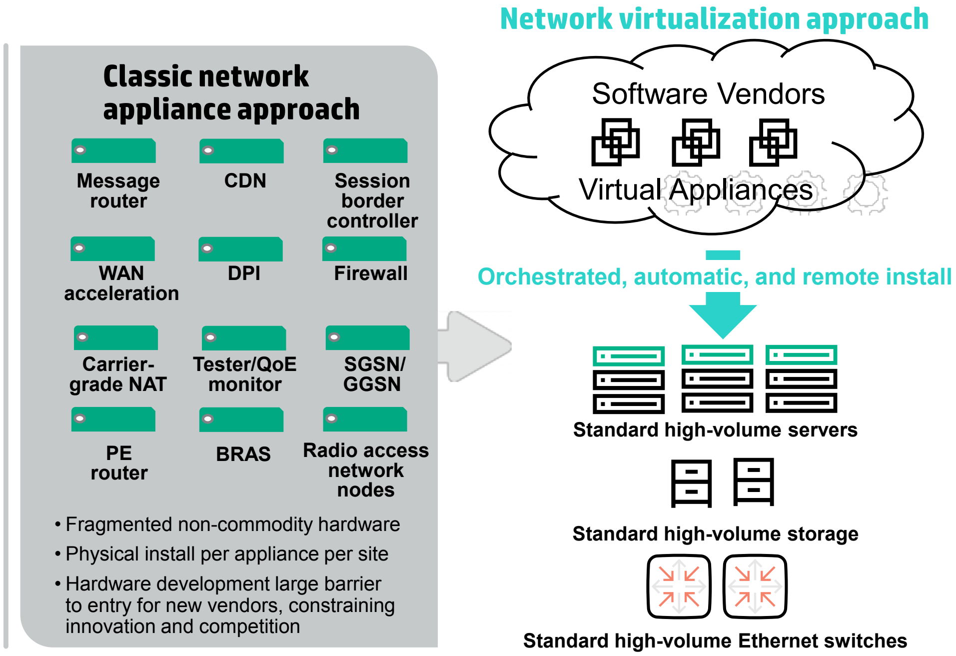
SDN and NFV –Key technologies:

- NFV - Virtualized Services (Cloud)
  - Flexibility, Agility and Scalability.
- SDN – Programmable connectivity
  - Dynamic steering of traffic.

Configuration and Management of Networks

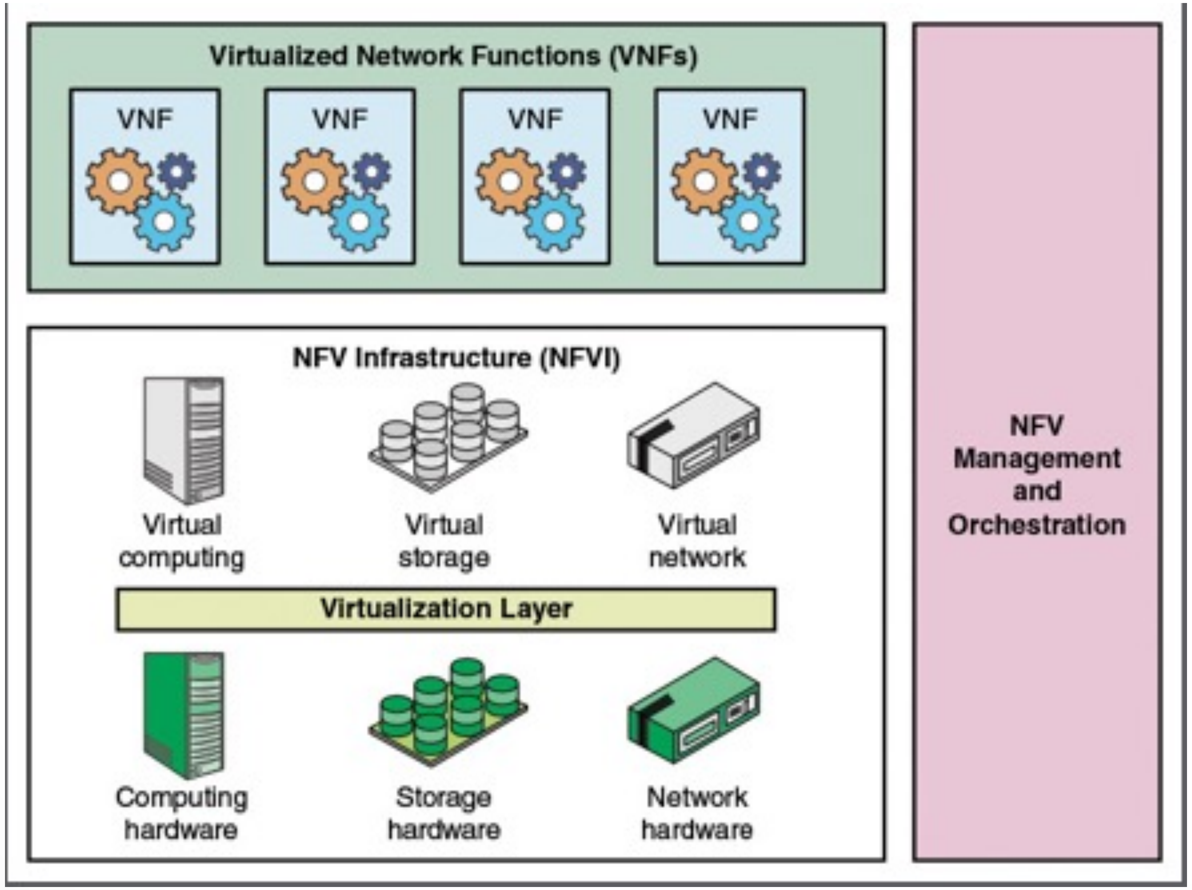
Network Function Virtualization (NFV)

NFV – Virtualizing classic Network functions (e.g. routers, firewalls, DPI, Load balancers and Evolved Packet Core nodes)



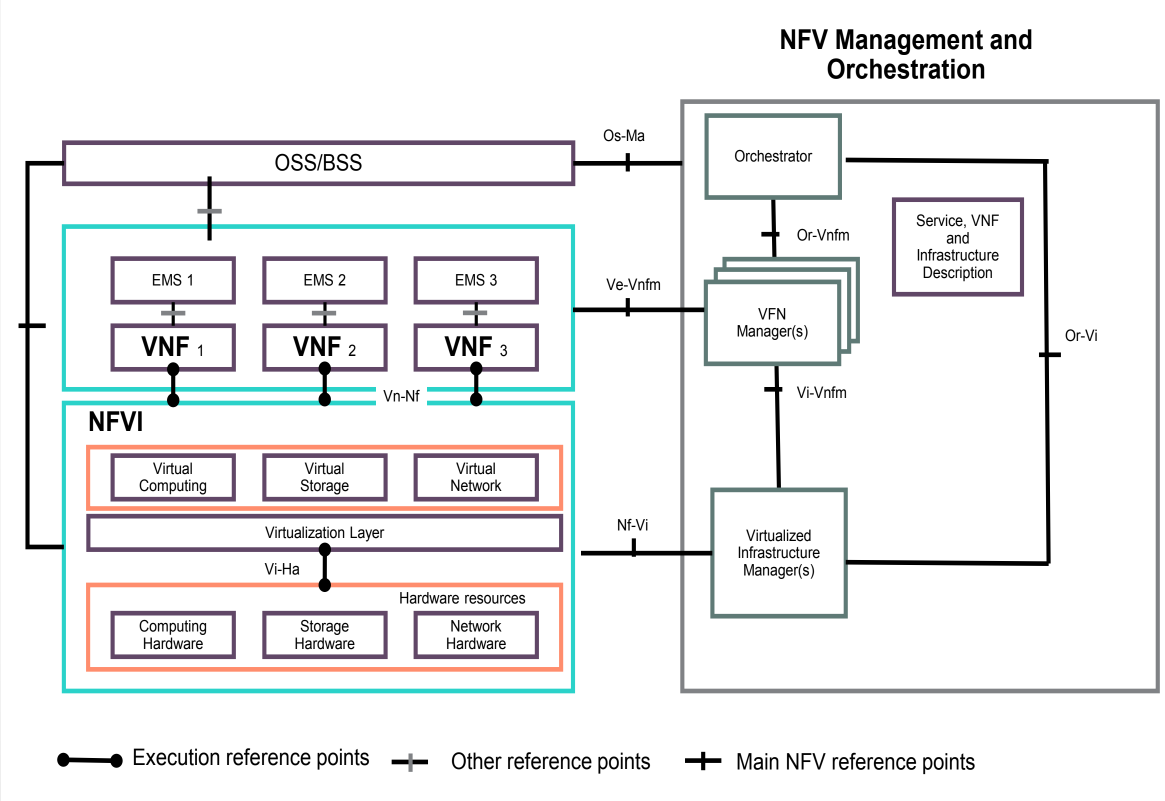
Network Function Virtualization (NFV)

High Level NFV framework



NFV - MANO

- NFV Management And Orchestration (MANO)

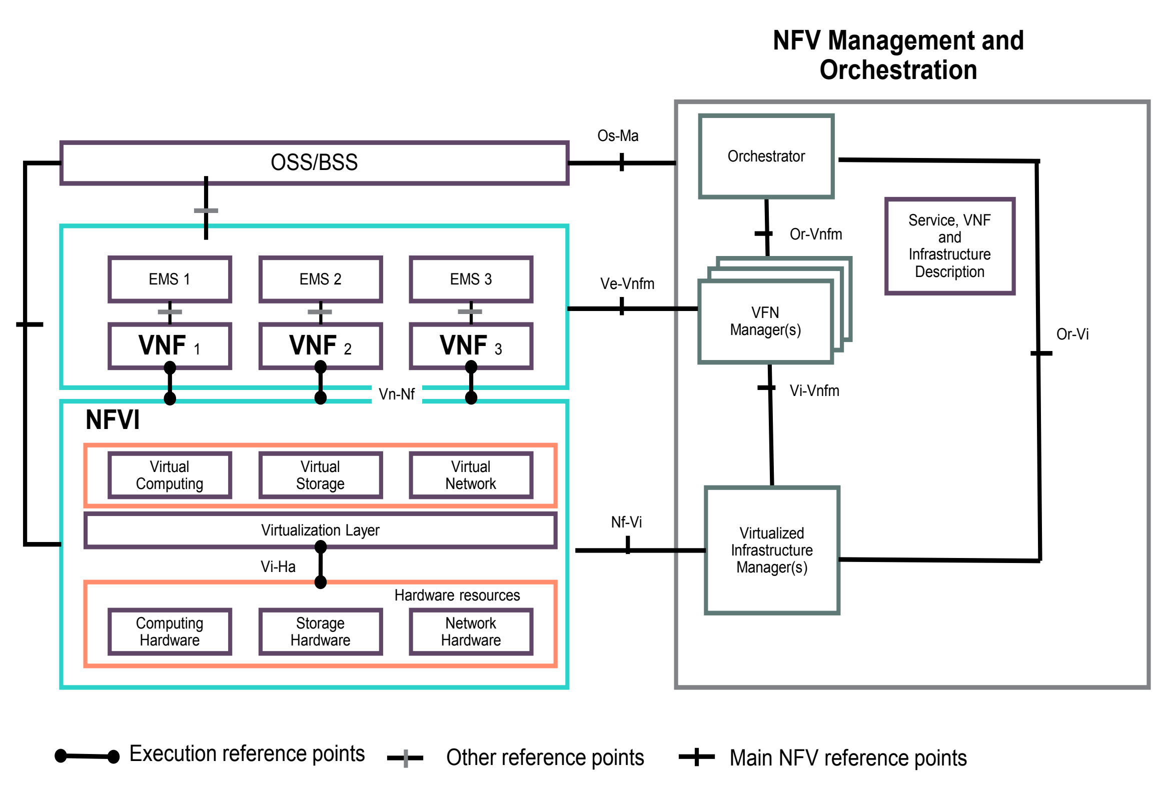


- Network Function Virtualization Orchestrator (NFVO).
  - Manages network services. On-boarding of network service descriptions.
- Virtual Network Function Manager (VFNM).
  - Life-cycle of a VNF. Connects to VNFs or their Element Managers

Configuration and Management of Networks

NFV - MANO

- NFV Management And Orchestration (MANO)

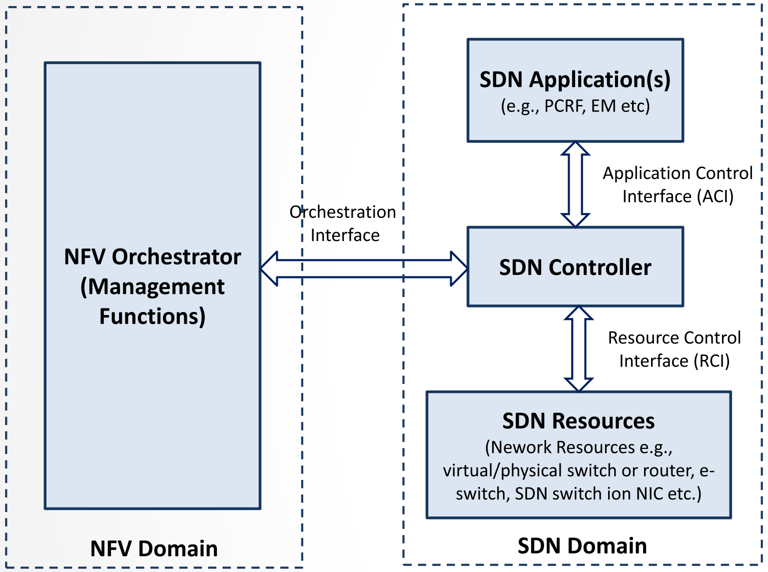


- Virtualized Infrastructure Manager (VIM)
  - VNF management at VM and container level
  - Providing connectivity between the various VNFs of a network service

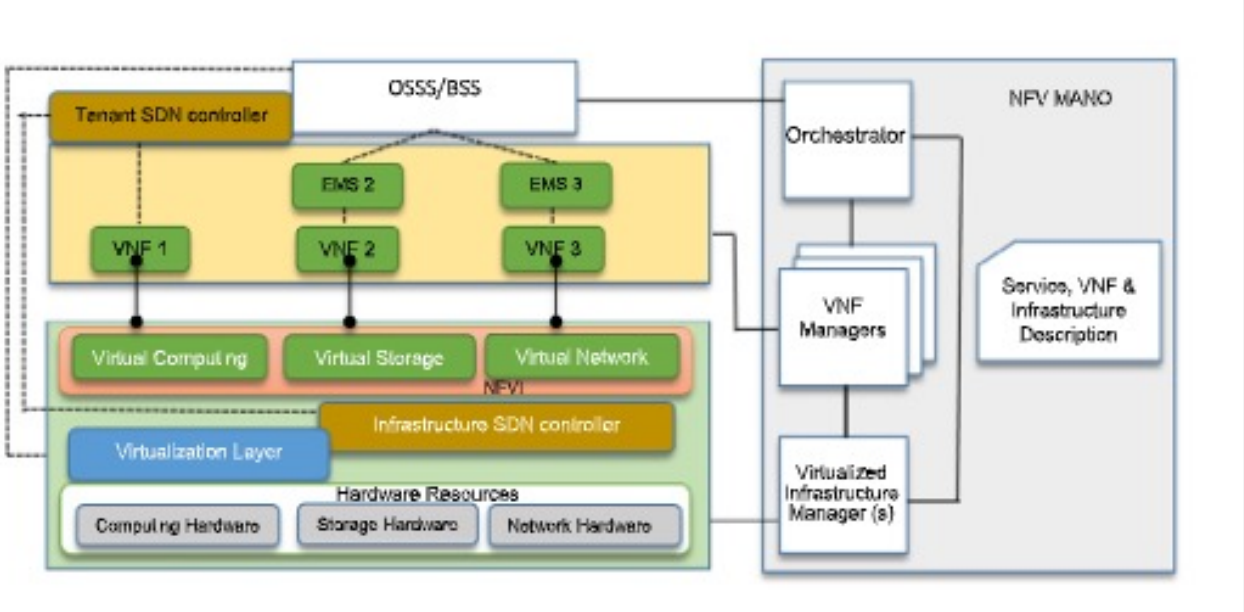
Configuration and Management of Networks

NFV & SDN

Interaction



Possible SDN positioning



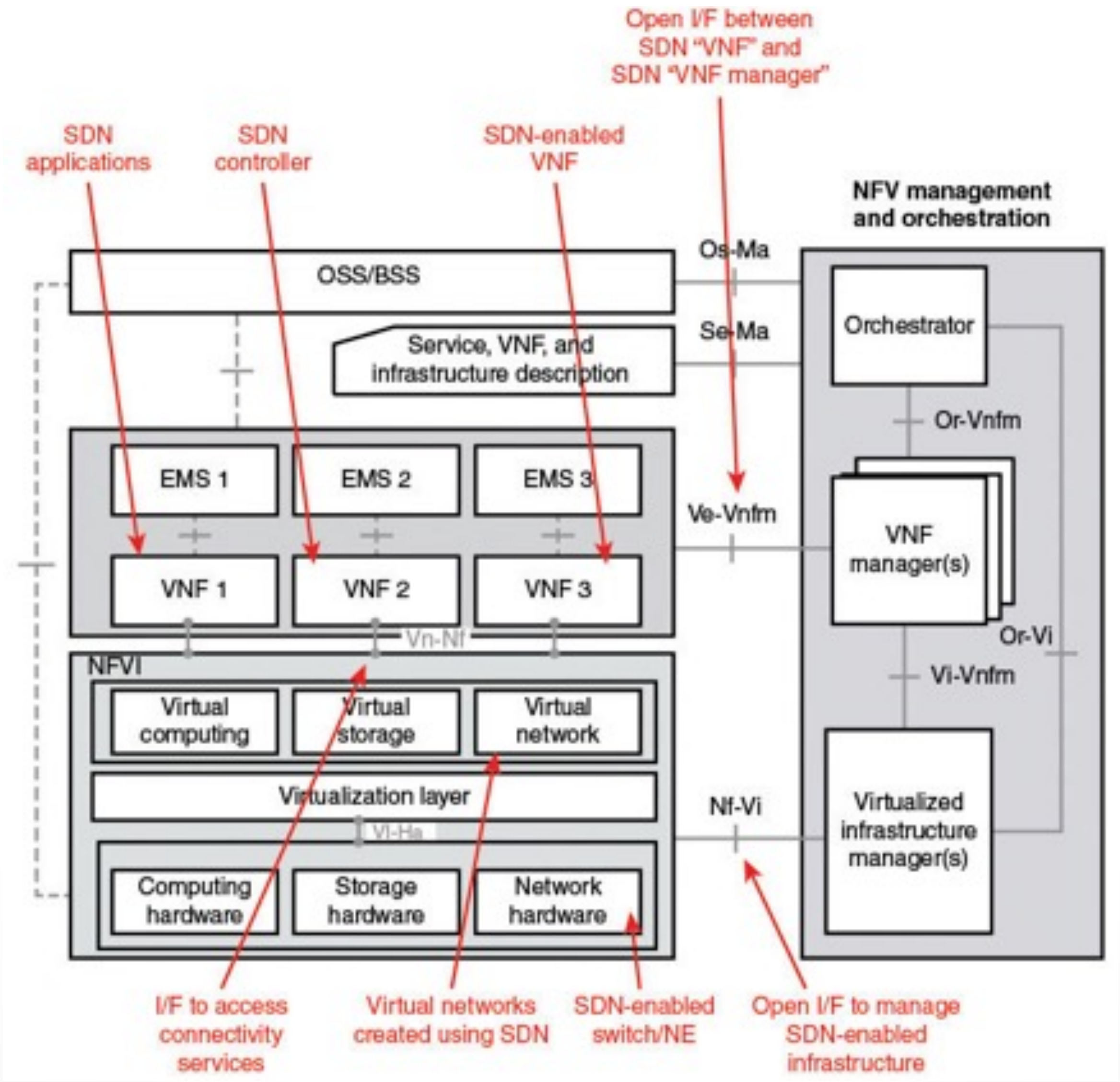
Infrastructure SDN Controller : Provides the required connectivity with the VNFs as managed by the VIM to support the deployment and connectivity of VNFs.

Tenant SDN Controler: provides an overlay comprising tenant VNFs that define the network service(s).

Configuration and Management of Networks

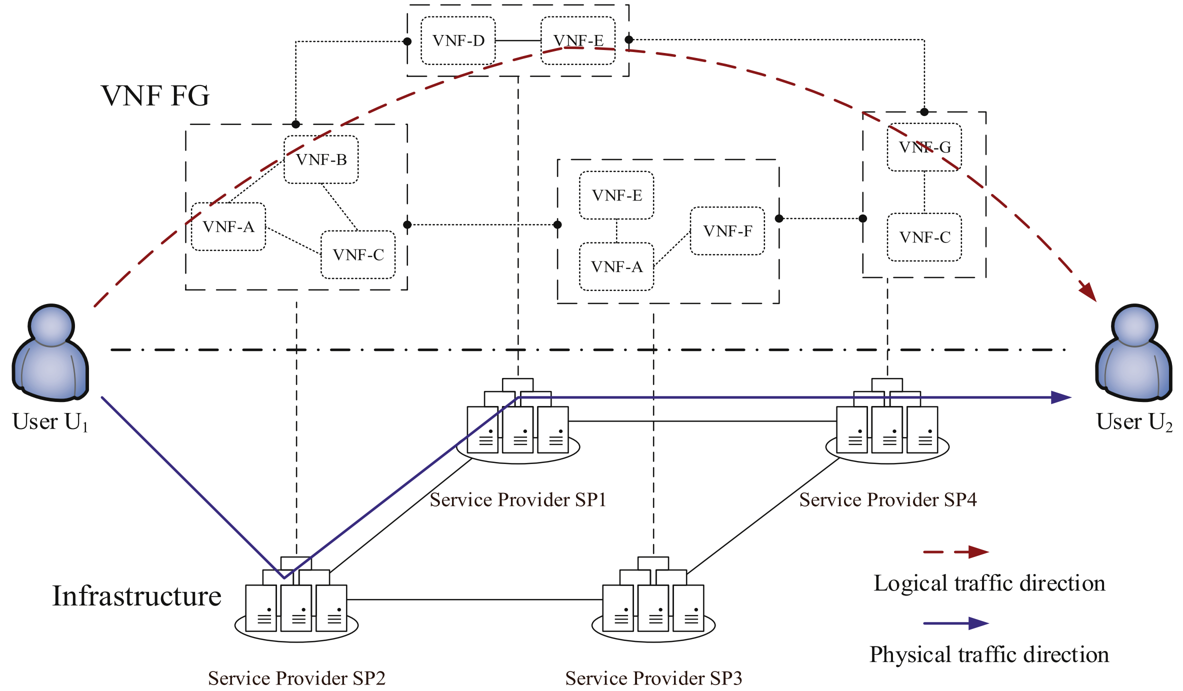
NFV & SDN

Possible SDN positioning



Configuration and Management of Networks

The VNF placement and Service Function Chaining problems



Service Request: U_1 → VNF-A → VNF-B → VNF-E → VNF-G → U_2

Configuration and Management of Networks

NFV & SDN

Example