SOFTWARE-DEFINED NETWORKING (SDN)

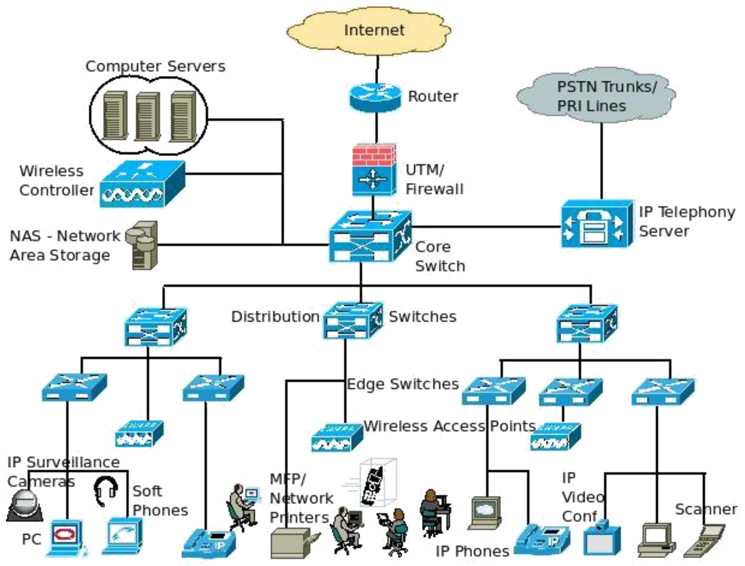
Course: Computer Networks
Student Name: Moustafa Magdy Ahmed
Section: 4

--

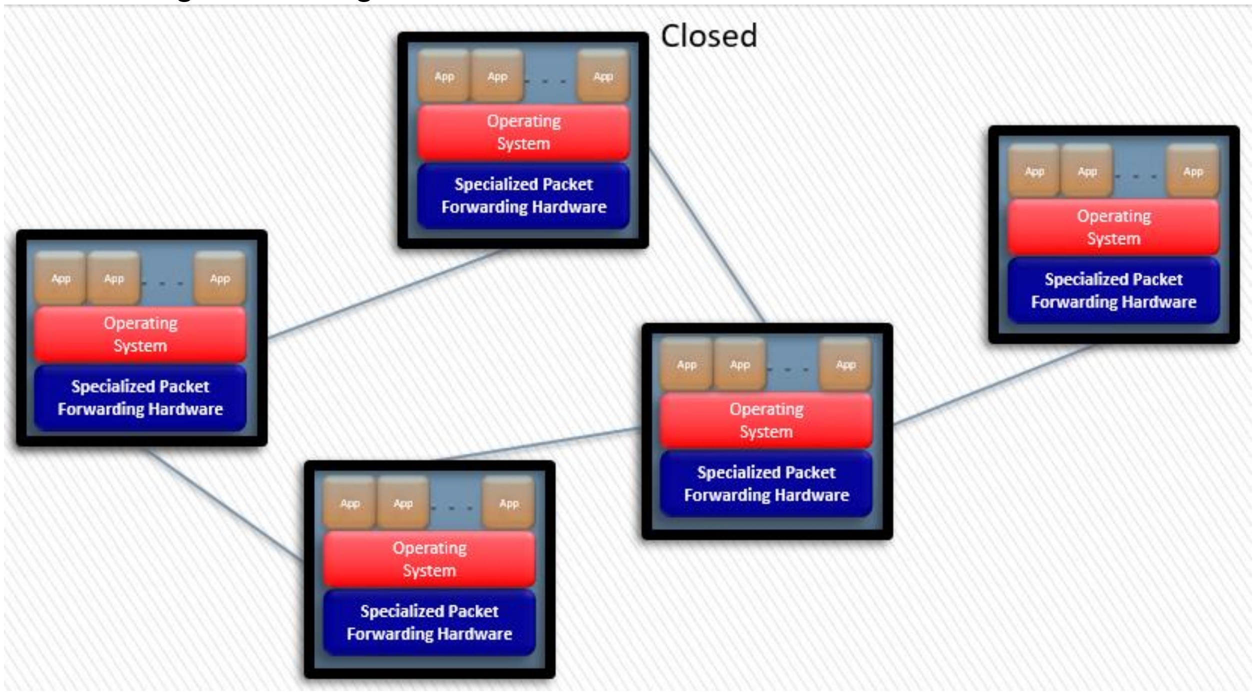
4th Computer Engineering Branch
Electronics, Communication and Computer Department
Faculty of Engineering at Helwan, Helwan University

Software-Defined Networking (SDN)

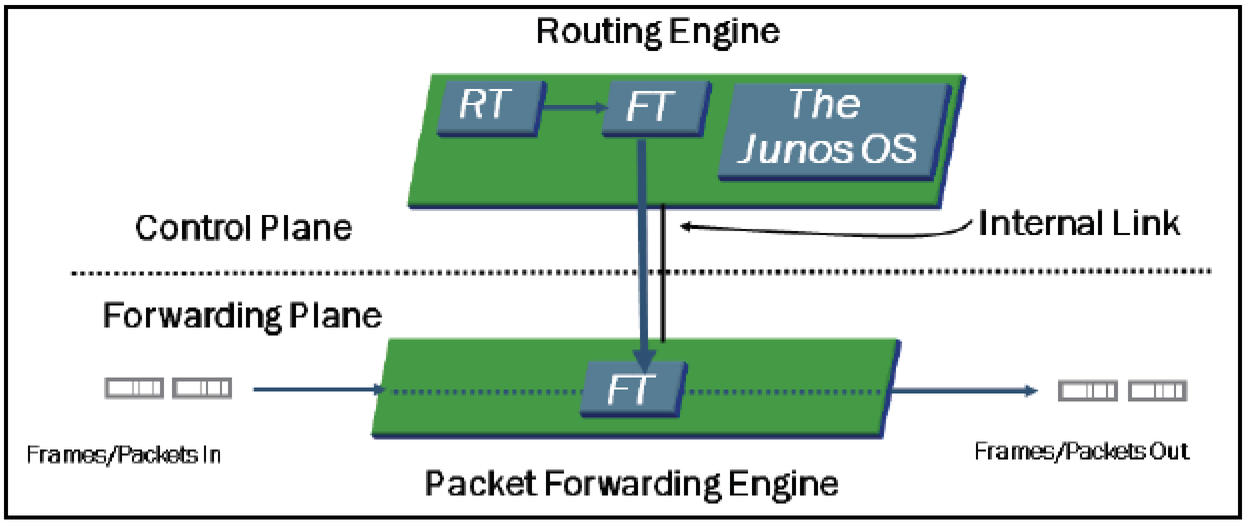
Limitations of Current Networks



- **Enterprise networks are difficult to manage**
- **New control requirements have arisen due to Greater scale and Migration of VMS**
- **Old ways to configure a network**
- **Many complex functions baked into infrastructure Cannot dynamically change according to network conditions**



Separate Control and Forwarding planes inside device

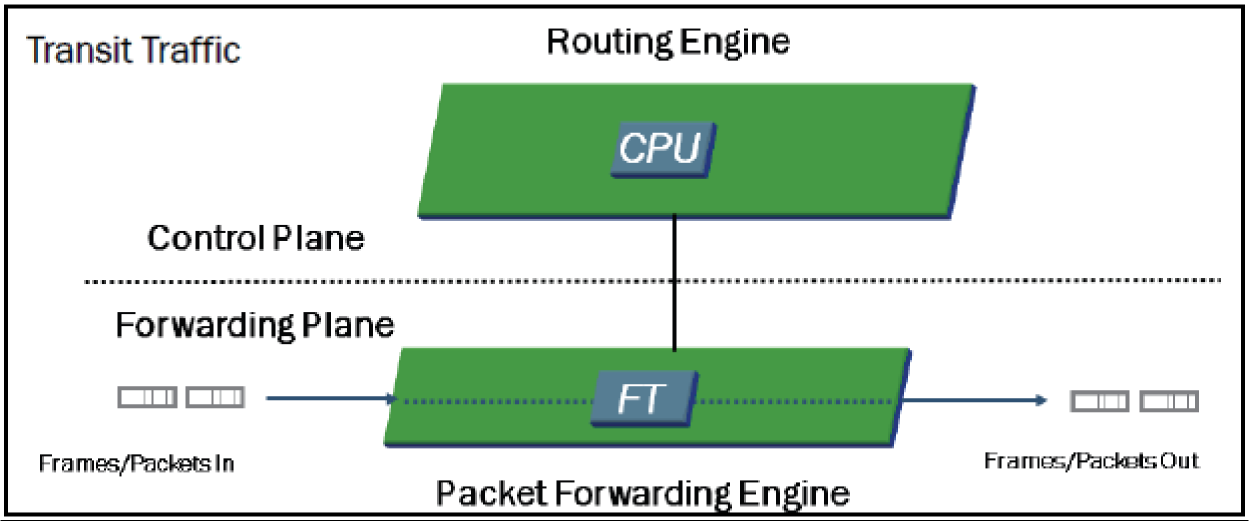


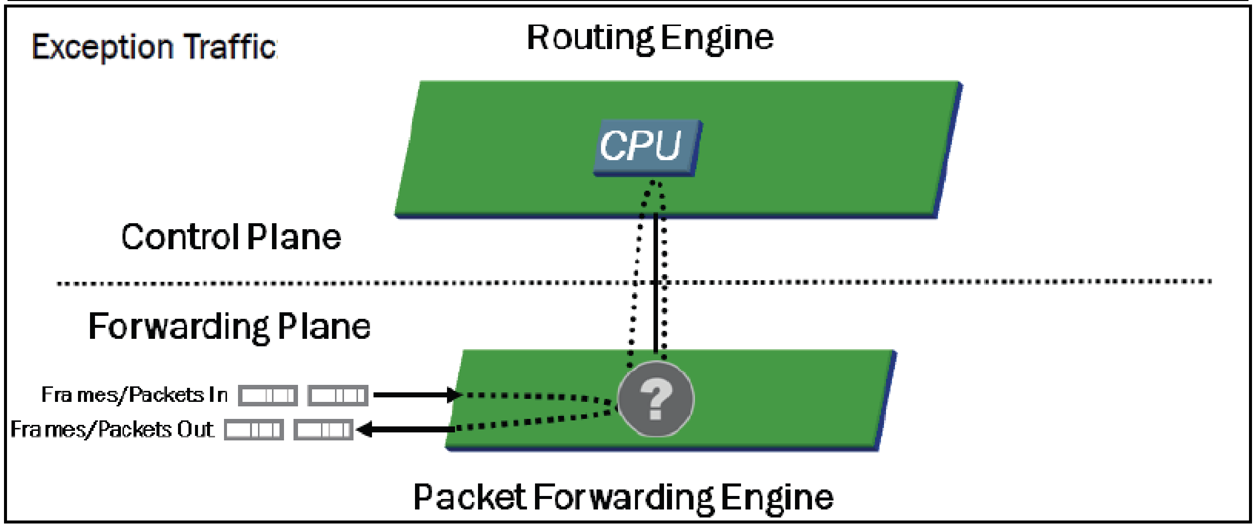
Some network devices (Like: Juniper-Junos devices) have aspect of modularity is the separation of the control plane and the forwarding or data plane. The processes that control routing and switching protocols are cleanly separated from the processes that forward frames, packets, or both through the device. This design allows you to tune each process for maximum performance and reliability. The separation of the control and forwarding.

The control plane runs on the Routing Engine (RE). The RE is the brain of the platform, it is responsible for performing protocol updates and system management. The RE runs various protocol and maintains the routing tables, bridging table, and primary forwarding table and connects to the Packet Forwarding Engine (PFE) through an internal link.

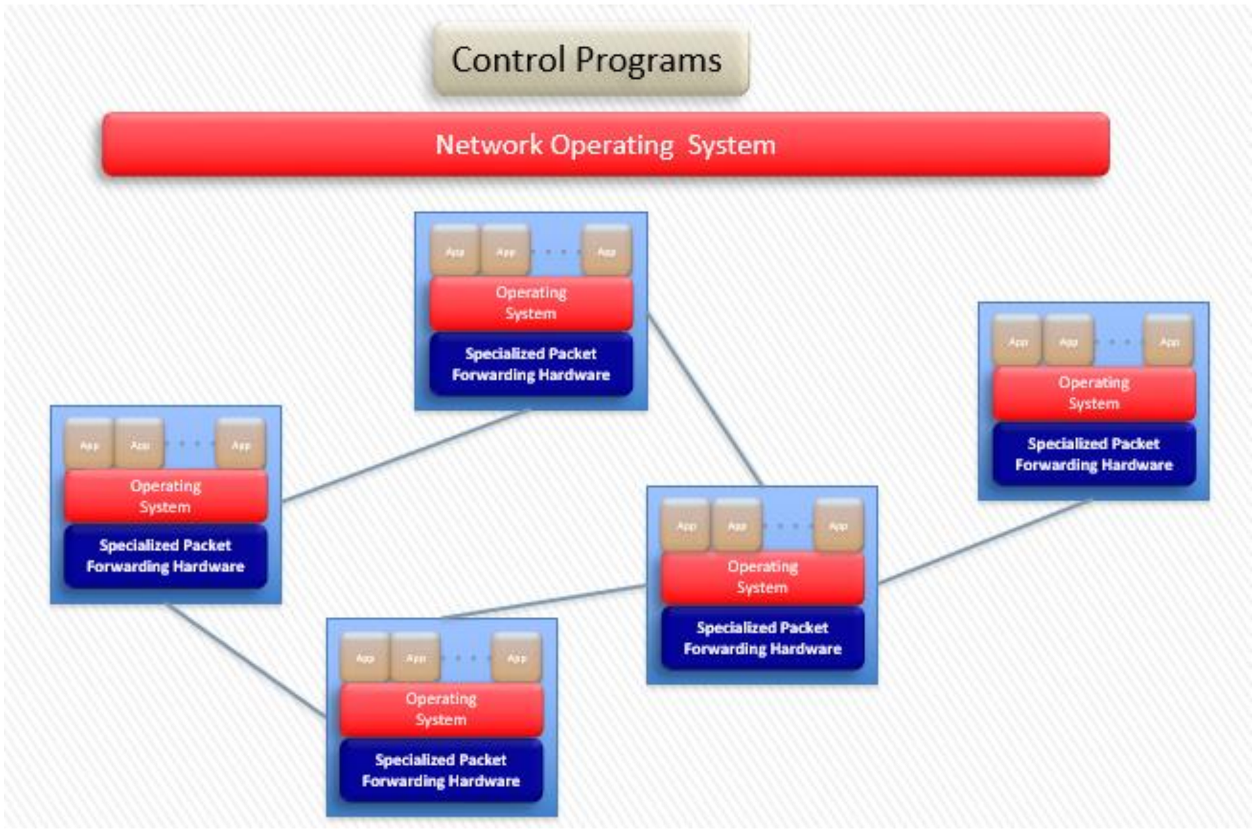
The PFE, usually runs on separate hardware and is responsible for forwarding transit traffic through the device. In many platforms, the PFE uses application-specific integrated circuits (ASICs) for increased performance. Because this architecture separates control operations—such as protocol updates and system management—from forwarding operations, so platforms can deliver superior performance and highly reliable deterministic operation.

The PFE receives the forwarding table (FT) from the RE by means of an internal link. FT updates are a high priority for the OS kernel and are performed incrementally.





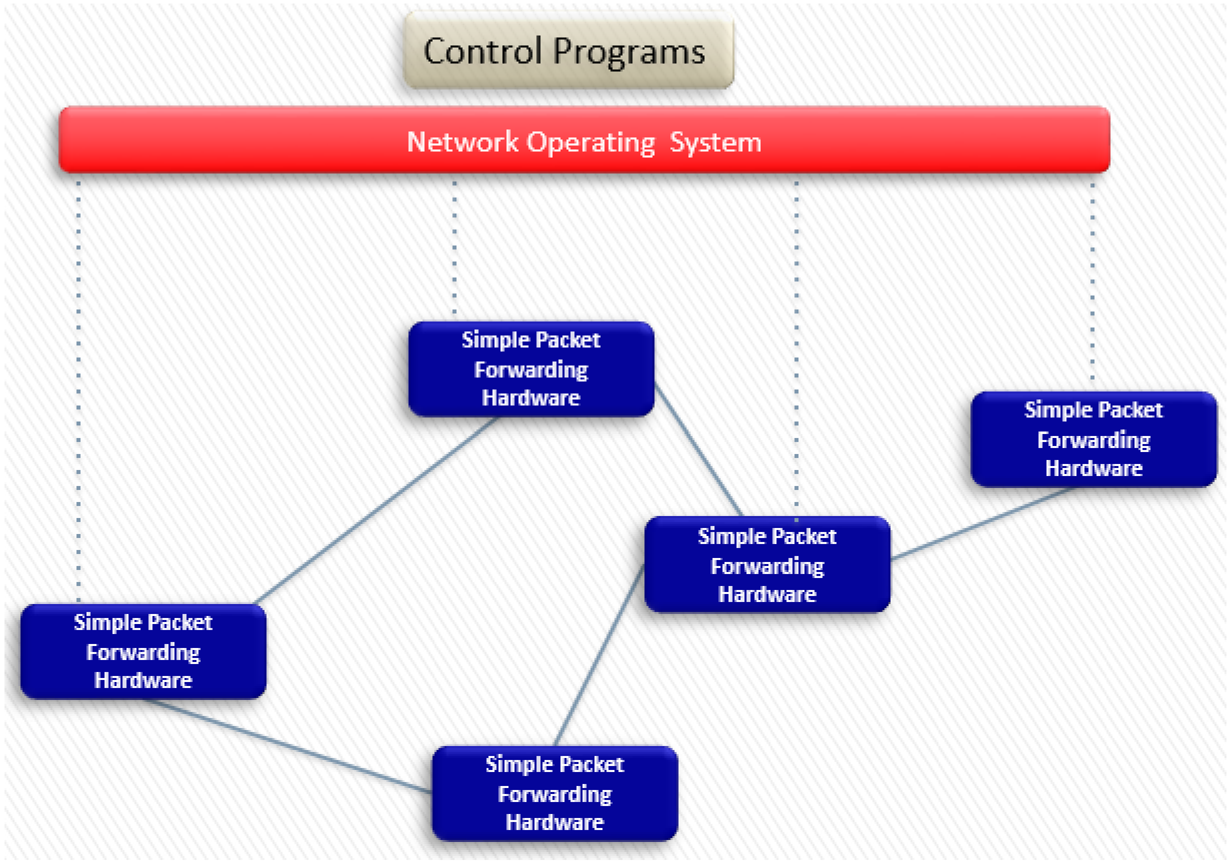
This Separation only improve the performance of network bus still their other problems



This may help to improve some weakness of current networking like:

- **Management**
- **Control**
- **Configuration ways**

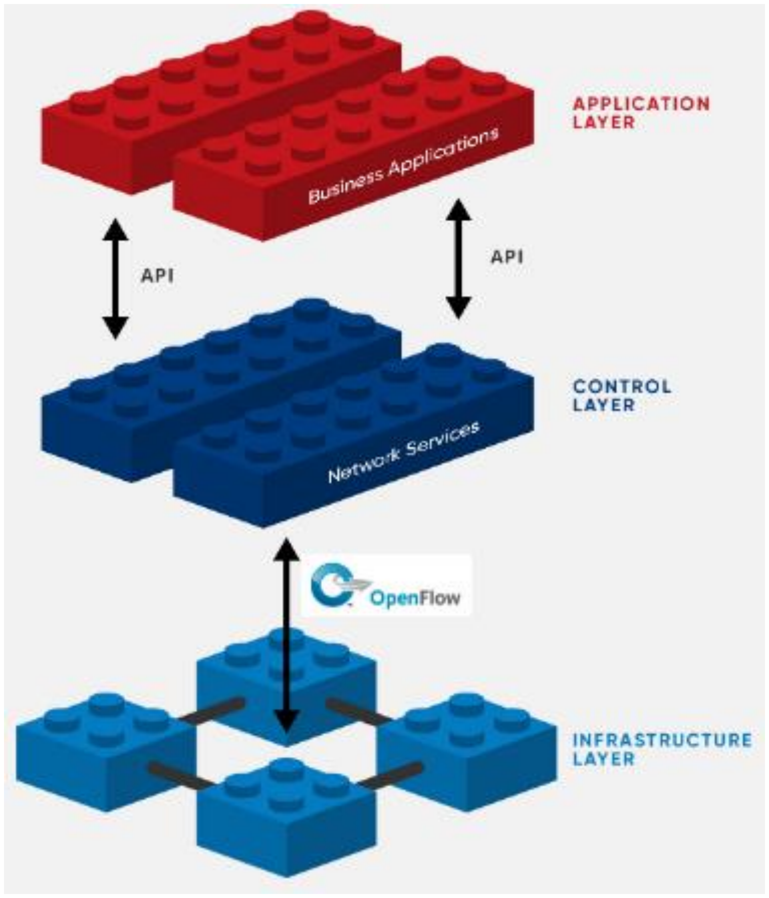
But still appear that's cannot dynamically change according to network conditions as each device still has his nature to be (Router, Switch, Firewall, ..)


Control Programs
Network Operating System
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware

Software-Defined Networking (SDN)

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow® protocol is a foundational element for building SDN solutions.

- **DIRECTLY PROGRAMMABLE**
- **AGILE**
- **CENTRALLY MANAGED**
- **PROGRAMMATICALLY CONFIGURED**
- **OPEN STANDARDS-BASED AND VENDOR-NEUTRAL**



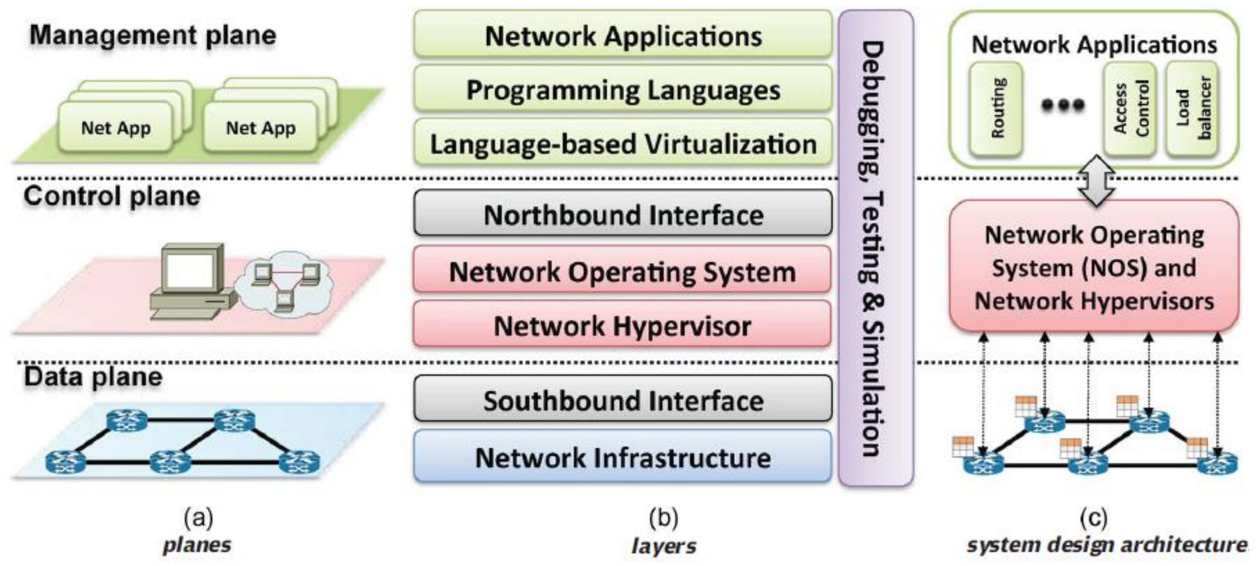
- **DIRECTLY PROGRAMMABLE**
  Network control is directly programmable because it is decoupled from forwarding functions.

- **AGILE**
  Abstracting control from forwarding lets administrators dynamically adjust network-wide traffic flow to meet changing needs.

- **CENTRALLY MANAGED**
  Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.

- **PROGRAMMATICALLY CONFIGURED**
  SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.

- **OPEN STANDARDS-BASED AND VENDOR-NEUTRAL**
  When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.



Layer I: Network Infrastructure

An SDN infrastructure, is composed of a set of networking equipment (switches, routers, and middlebox appliances) as simple forwarding elements without embedded control or software to take autonomous decisions. The network intelligence is removed from the data plane devices to a logically centralized control system.

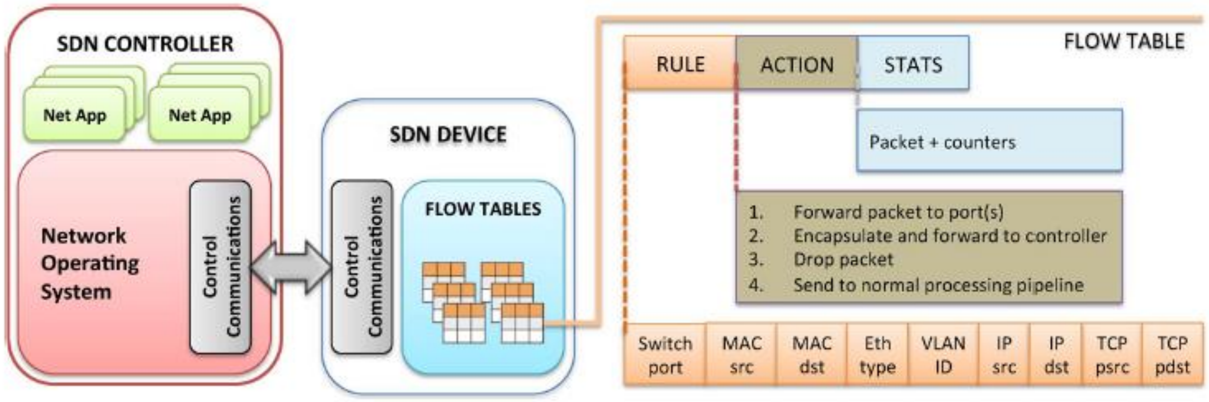
More importantly, these new networks are built (conceptually) on top of open and standard interfaces (e.g., OpenFlow), a crucial approach for ensuring configuration and communication compatibility and interoperability among different data and control plane devices.

An OpenFlow enabled forwarding device is based on a pipeline of flow tables where each entry of a flow table has three parts:

1) **a matching rule**
2) **actions to be executed on matching packets**
3) **counters that keep statistics of matching packets**

- **Inside an OpenFlow device, a path through a sequence of flow tables defines how packets should be handled.**
- **When a new packet arrives, the lookup process starts in the first table and ends either with a match in one of the tables of the pipeline or with a miss.**

- If there is no default rule, the packet will be discarded. However, the common case is to install a default rule which tells the switch to send the packet to the controller (or to the normal non-OpenFlow pipeline of the switch).



Layer 2: Southbound Interface

The southbound interface (or southbound APIs) is the OpenFlow (or alternative) protocol specification. Its main function is to enable communication between the SDN controller and the network nodes (both physical and virtual switches and routers) so that the router can discover network topology, define network flows and implement requests relayed to it via northbound APIs.

OpenFlow is the most widely accepted and deployed open southbound standard for SDN. It provides a common specification to implement Open-Flow-enabled forwarding devices, and for the communication channel between data and control plane devices.

The OpenFlow protocol provides three information sources for NOSs:

1) **event based messages are sent by forwarding devices to the controller when a link or port change is triggered.**
2) **flow statistics are generated by the forwarding devices and collected by the controller**
3) **packet-in messages are sent by forwarding devices to the controller when they do not know what to do with a new incoming flow or because there is an explicit ‘‘send to controller’’ action in the matched entry of the flow table.**

Other examples on southbound APIs Like:

- **OVSDB**
- **SNMP**
- **BGP**
- **L2/L3 Agent**

Layer 3: Network Hypervisor

Hypervisors enable distinct virtual machines to share the same hardware resources at relative low cost.

virtual machines can be easily migrated from one physical server to another and can be created and/or destroyed on demand, enabling the provisioning of elastic services with flexible and easy management.

The main network requirements can be captured along two dimensions: network topology and address space.

virtualized workloads must operate in the same address of the physical infrastructure.

- **FlowVisor:**

FlowVisor is an experimental software-defined networking (SDN) controller that enables network virtualization by dividing a physical network into multiple logical networks. FlowVisor ensures that each controller touches only the switches and resources assigned to it. It also partitions bandwidth and flow table resources on each switch and assigns those partitions to individual controllers.

FlowVisor slices a physical network into abstracted units of bandwidth, topology, traffic and network device central processing units (CPUs). It operates as a transparent proxy controller between the physical switches of an OpenFlow network and other OpenFlow controllers and enables multiple controllers to operate the same physical infrastructure, much like a server hypervisor allows multiple operating systems to use the same x86-based hardware

Other examples on Network Slicing hypervisor like:

- **OpenVirteX**
- **AutoSlice**
- **AutoVFlow**
- **FlowN**

Layer 4: Network Operating System/Controller

The idea of operating systems abstracting device-specific characteristics and providing, in a transparent way, common functionalities is still mostly absent in networks.

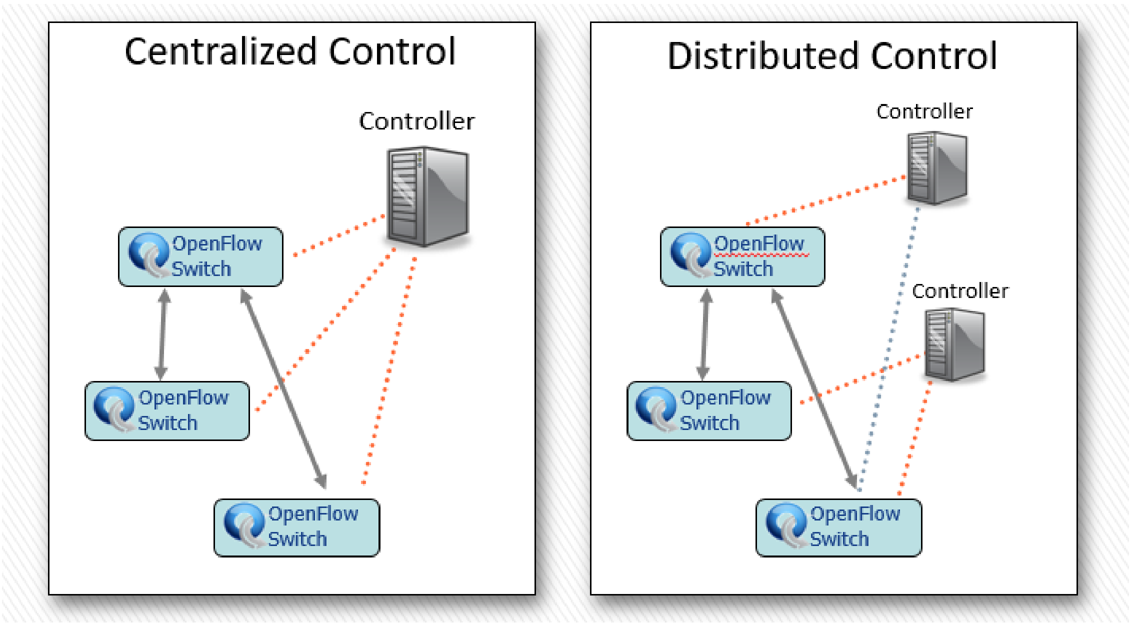
A NOS (or a controller) is a critical element in an SDN architecture as it is the key supporting piece for the control logic (applications) to generate the network configuration based on the policies defined by the network operator. Network operating systems (NOS) provide abstractions, essential services, and common APIs to developers.

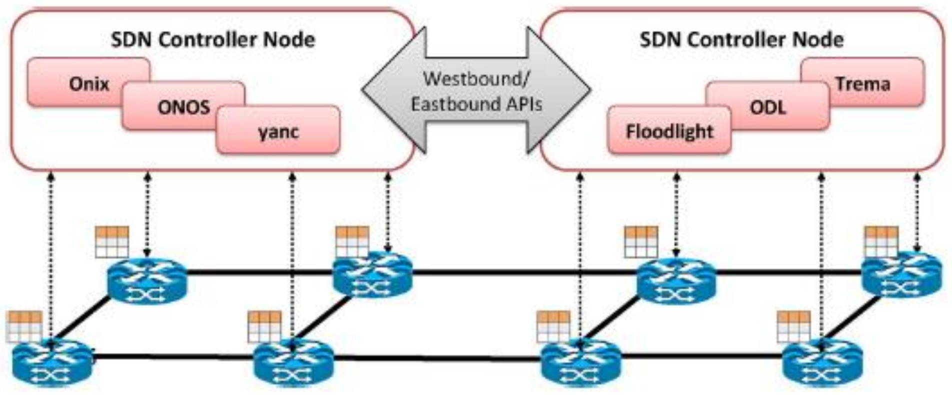
Generic functionality as network state and network topology information, device discovery, and distribution of network configuration can be provided as services of the NOS.

A centralized controller is a single entity that manages all forwarding devices of the network. Naturally, it represents a single point of failure and may have scaling limitations

A Distributed NOS can be scaled up to meet the requirements of potentially any environment, another common property of distributed controllers is fault tolerance. When one node fails, another neighbor node should take over the duties and devices of the failed node.

Eastbound and Westbound: East/westbound APIs are a special case of interfaces required by distributed controllers. Currently, each controller implements its own east/westbound API. The functions of these interfaces include import/export data between controllers, algorithms for data consistency models, and monitoring/notification capabilities (e.g., check if a controller is up or notify a take over on a set of forwarding devices).





Layer 5: Northbound Interface

The northbound application program interfaces (APIs) are usually SDN Rest APIs used to communicate between the SDN Controller and the services and applications running over the network. The northbound APIs can be used to

facilitate innovation and enable efficient orchestration and automation of the network to align with the needs of different applications via SDN network programmability.

The northbound interface is mostly a software ecosystem, not a hardware one, as is the case of the southbound APIs.

Layer 6: Language-based Virtualization

Specific Programing language is necessary for virtualization.

Two essential characteristics of virtualization solutions are the capability of expressing modularity and of allowing different levels of abstractions while still guaranteeing desired properties. Provides type of high-level abstraction of network topology.

Another form of language-based virtualization is static slicing. This a scheme where the network is sliced by a compiler, based on application layer definitions. The output of the compiler is a monolithic control program that has already slicing definitions and configuration commands for the network. In such a case, there is no need for a hypervisor to dynamically manage the network slices

Layer 7: Programing Languages

Programmability in networks is starting to move from low-level machine languages such as OpenFlow to high-level programming languages.

Assembly-like machine languages such as OpenFlow and POF, essentially mimic the behavior of forwarding devices, forcing developers to spend too much time on low-level details rather than on the problem solve.

The use of these low-level languages makes it difficult to reuse software, to create modular and extensive code, and leads to a more error-prone development process.

Abstractions provided by high-level programming languages can significantly help address many of the challenges of these lower level instruction sets and provides the capability of creating and writing programs for virtual network topologies

In SDNs, high-level programming languages can be designed and used to:

1) **create higher level abstractions for simplifying the task of programming forwarding devices**
2) **providing abstractions for different important properties and functions**
3) **enable more productive and problem-focused environments for network software programmers speeding up development and innovation**
4) **promote software modularization and code reusability in the network control plane**
5) **foster the development of network virtualization**

Low-level instruction sets suffer from several problems. To address some of these challenges, higher level programming languages have been proposed, with diverse goals, such as:

- **avoiding low-level and device-specific configurations and dependencies spread across the network**
- **providing abstractions that allow different management tasks to be accomplished through easy to understand and maintain network policies**
- **decoupling of multiple tasks (e.g., routing, access control, traffic engineering)**
- **implementing higher level programming interfaces to avoid low-level instruction sets**
- **solving forwarding rules problems, e.g., conflicting or incomplete rules that can prevent a switch event to be triggered, in an automated way**
- **addressing different race condition issues which are inherent to distributed systems**
- **enhancing conflict-resolution techniques on environments with distributed decision makers;**
- **providing native fault-tolerance capabilities on data plane path setup**
- **reducing the latency in the processing of new flows**
- **easing the creation of stateful applications (e.g., stateful firewall).**

Layer 8: Network Applications

An SDN application is a software program designed to perform a task in a software-defined networking (SDN) environment that will be translated into commands to be installed in the data plane, dictating the behavior of the forwarding devices.

SDN applications can replace and expand upon functions that are implemented through firmware in the hardware devices of a conventional network.

SDNs can be deployed on any traditional network environment, from home and enterprise networks to data centers and Internet exchange points. perform traditional functionality such as:

- **routing**
- **load balancing**
- **security policy enforcement**

and novel approaches such as:

- **reducing power consumption**

- **fail-over and reliability**
- **end-to-end QoS enforcement**
- **network virtualization**
- **mobility management**

Resources

- JNCIA-Junos - Study Guide Part 1
- https://www.opennetworking.org/sdn-definition/
- https://searchnetworking.techtarget.com/definition/software-defined-networking-SDN/
- https://www.juniper.net/us/en/solutions/sdn/what-is-sdn/
- https://www.juniper.net/us/en/products-services/sdn/
- https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html
- https://www.cisco.com/c/en/us/solutions/service-provider/software-defined-networks-sdn-service-providers/index.html
- Software-Defined Networking: A Comprehensive Survey
- https://whatis.techtarget.com/definition/northbound-interface-southbound-interface/
- https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/
- https://www.sdxcentral.com/sdn/definitions/southbound-interface-api/
- https://searchnetworking.techtarget.com/definition/FlowVisor/
- https://searchnetworking.techtarget.com/definition/network-hypervisor/
- https://www.sdxcentral.com/sdn/definitions/north-bound-interfaces-api/