MODULE 3 :

CONTROL PLANE

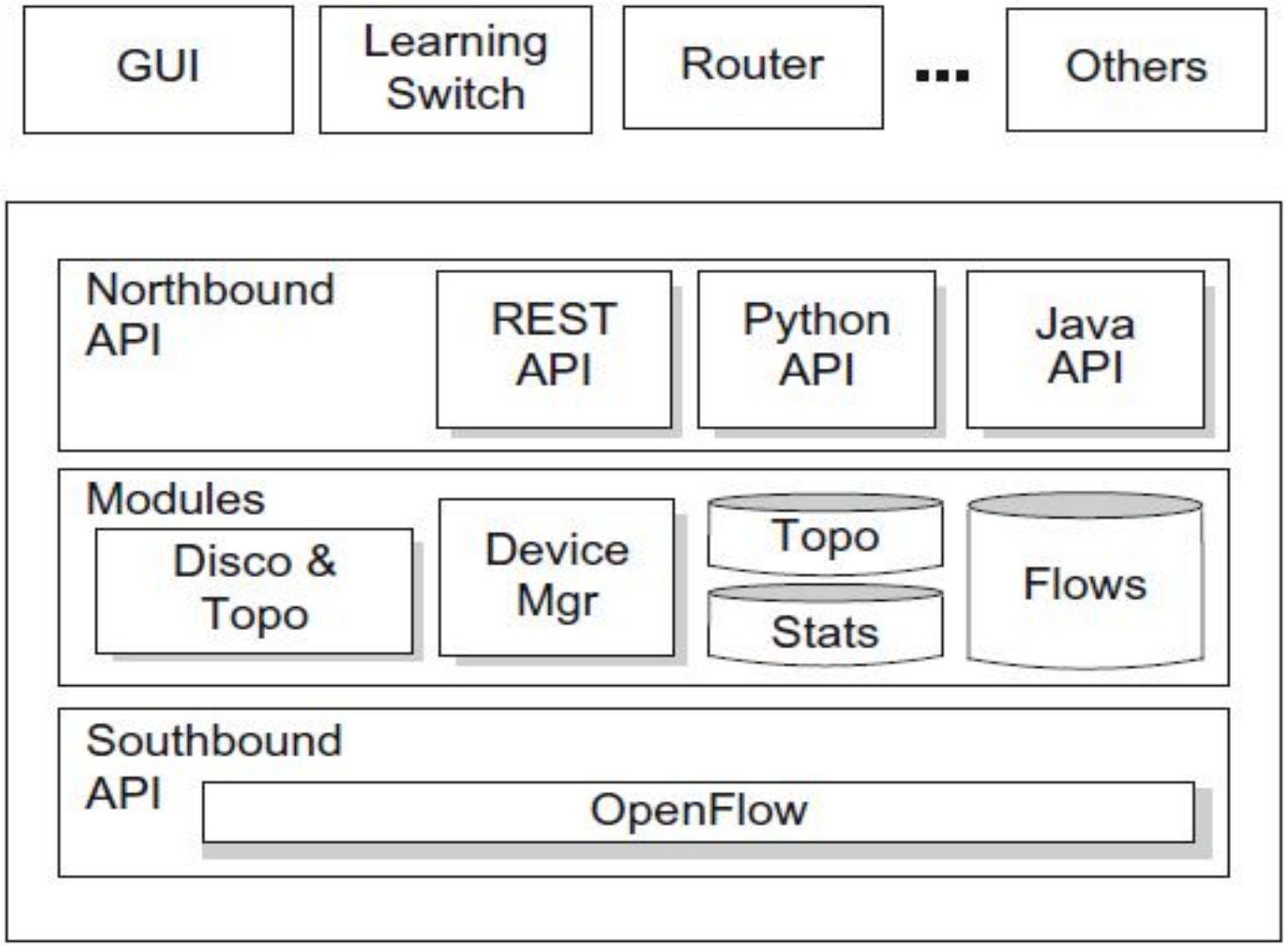
Overview:

- In **routing**, the **control plane** is the part of the router architecture.
- Is concerned with drawing the network topology, or the information in a (possibly augmented) routing table that defines what to do with incoming packets.
- Control plane functions, such as participating in routing protocols, run in the architectural control element.
- In most cases, the routing table contains a list of destination addresses and the outgoing interface(s) associated with them.
- Control plane logic also can define certain packets to be discarded, as well as preferential treatment of certain packets for which a high quality of service is defined by such mechanisms as differentiated services.

SDN Controller:

- In general, SDN controller is a software system or collection of systems that together provides:

1. Management of network state, and in some cases, the management and distribution of this state, may involve a database.
2. A high-level data model that captures the relationships between managed resources, policies and other services provided by the controller.
3. RESTful (representational state transfer) API is provided that exposes the controller services to an application.
4. A secure TCP control session between controller and the associated agents in the network elements.
5. A device, topology, and service discovery mechanism; a path computation system; and potentially other network-centric or resource-centric information services.


GUI
Learning Switch
Router
...
Others
Northbound API
REST API
Python API
Java API
Modules
Disco & Topo
Device Mgr
Topo
Stats
Flows
Southbound API
OpenFlow

Existing SDN Controllers

1. NOX/POX
2. Trema
3. Ryu
4. Floodlight
5. Pyretic
6. Frenetic
7. Open Daylight
8. And many more

- Some artificial differences: language
- More important differences:
  - API
  - Functionality

NOX/POX:

- **NOX CONTROLLER:**

➢ It is the original OpenFlow controller.

➢ It serves as a network control platform, that provides a high level programmatic interface for management and the development of network control applications.

➢ Its system-wide abstractions turn networking into a software problem.

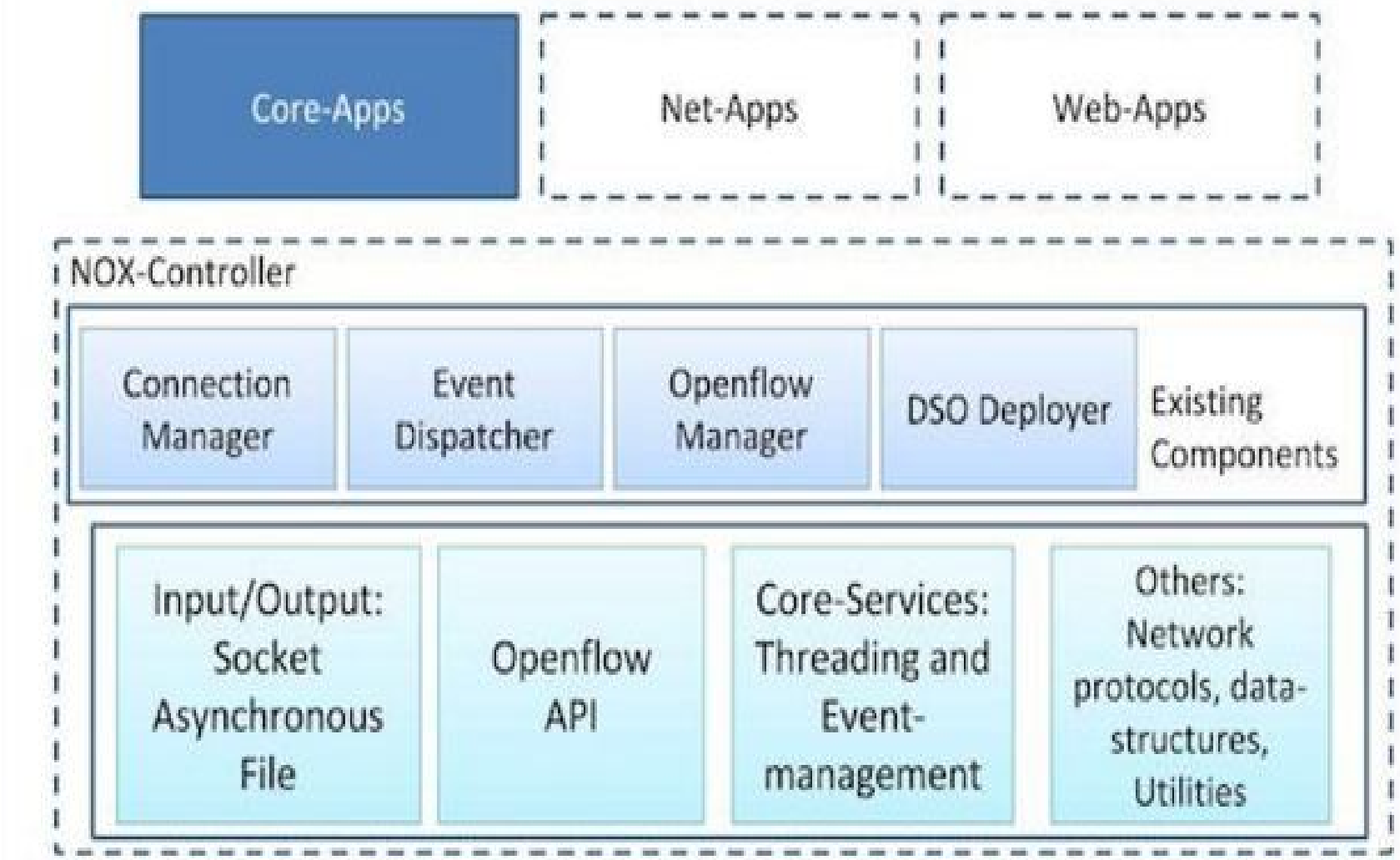
➢ Applications on NOX typically determine how each flow is routed or not routed in the network.

- **NOX Versions:**

1. NOX classic: This is the version that has been available under the GPL since 2009. Supports C/C++ and/or Python
2. NOX: The "new NOX." Only contains support for C++ and has lesser applications than the classic; however, this version is faster and has better codebase.
3. POX: Typically termed as NOX's sibling. Provides Python support.

NOX ARCHITECTURE:

- At the top, we have applications: Core, Net and Web.
- With the current NOX version, there are only two core applications: OpenFlow and switch, and both network and web applications are missing.
- The middle layer shows the in-built components of NOX.
- The connection manager, event dispatcher and OpenFlow manager are self-explanatory, whereas the dynamic shared object (DSO) deployer basically scans the directory structure for any components being implemented as DSOs.
- NOX events can be broadly classified as core events and application events.
- The core events map directly to OpenFlow messages received by controlled switches, such as:


Core-Apps
Net-Apps
Web-Apps
NOX-Controller
Connection Manager
Event Dispatcher
Openflow Manager
DSO Deployer
Existing Components
Input/Output: Socket Asynchronous File
Openflow API
Core-Services: Threading and Event-management
Others: Network protocols, data-structures, Utilities

	NOX	NOX classic
Core apps	OpenFlow, Switch	Messenger, SNMP, switch
Network Apps	—	Discovery, Topology, Authenticator, Routing, Monitoring
Web Apps	—	Webservice,Webserver,WebServiceClient
Language Support	C++ Only	C++ and Python
GUI	NO	YES

OpenFlow-Events	Description
Datapath_join_event	When a new switch is detected.
Datapath_leave_event	When a switch leaves the network.
Packet_in_event	Called for each new packet received.
Flow_mod_event	When a flow has been added or modified.
Flow_removed_event	When a flow in the network expires/removed.
Port_status_event	Indicates a change in port status.
Port_stats_in	When a port statistics message is received.

- In addition to core events, components themselves may define and throw higher level events which may be handled by any other events.
- NOX classic has various events such as **host_event** and **flow_in_event** by authenticator application, and **link_event** by the discovery application.
- An important functionality of NOX architecture is the interactions between components and the core, and among the components.
- NOX includes a concept called **container**, also termed **kernel**.
- The kernel does not directly operate on components, but on component contexts, which contain all per component information, including the component instance itself.
- Applications provide a component factory for the container.
- While loading the component in, the container asks for a factory instance by calling **get_factory()** and then constructs (and destroys) all the component instances using the factory.

- On the other hand, for accessing the container and to discover other components, the container passes a **context instance** for them.
- NOX often used in academic network research to develop SDN application: –
  - SANE: An approach to representing the network as a filesystem
  - Ethane: Application for centralized, network-wide security.
- Different from standard network development environments (such as building routers within or on top of Linux), NOX allows a centralized programming model for an entire network.
- NOX is designed to support both large enterprise networks of hundreds of switches (supporting many thousands of hosts) and smaller networks of a few hosts.

- **NOX Characteristics:**
  - Users implement control in C++
  - Supports Open Flow v.1.0
    - A fork (CPqD) supports 1.1, 1.2, and 1.3
  - Programming model
    - Controller registers for events
    - Programmer writes event handler
- **When to use NOX:**
  - You know C++
  - You are willing to use low-level facilities and semantics of OpenFlow
  - You need good performance

POX:

- IT is an open source development platform for Python-based software-defined networking (SDN) control applications, such as OpenFlow SDN controllers.
- IT enables rapid development and prototyping, is becoming more commonly used than **NOX**.
- **POX features:**

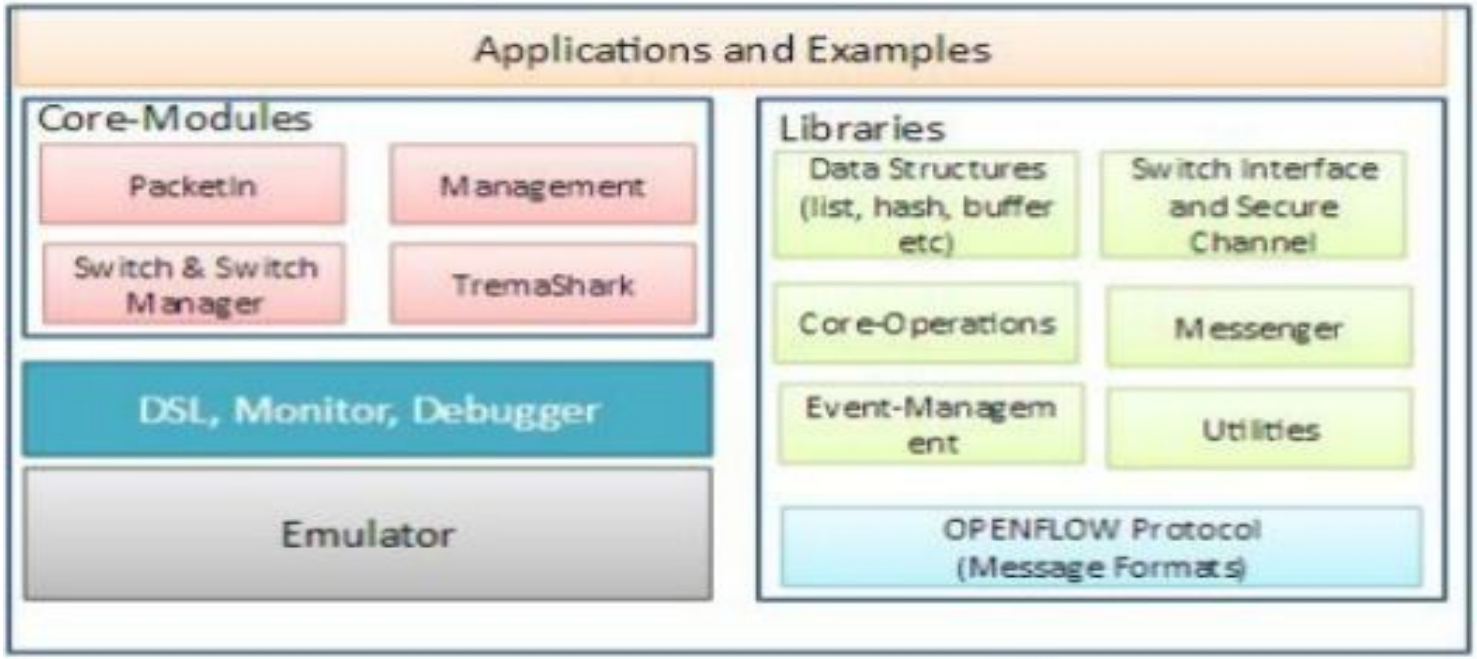
1. “Pythonic” OpenFlow interface.
2. Reusable sample components for path selection, topology discovery, etc.
3. “Runs anywhere” – Can bundle with install-free PyPy runtime for easy deployment.
4. Specifically targets Linux, Mac OS, and Windows.
5. Topology discovery.
6. Supports the same GUI and visualization tools as NOX.
7. Performs well compared to NOX applications written in Python.

- **When to use POX:**
  - If you know (or can learn) Python and are not concerned about controller performance
  - Rapid prototyping and experimentation
    - Research, experimentation, demonstrations
    - Learning concepts

TREMA:

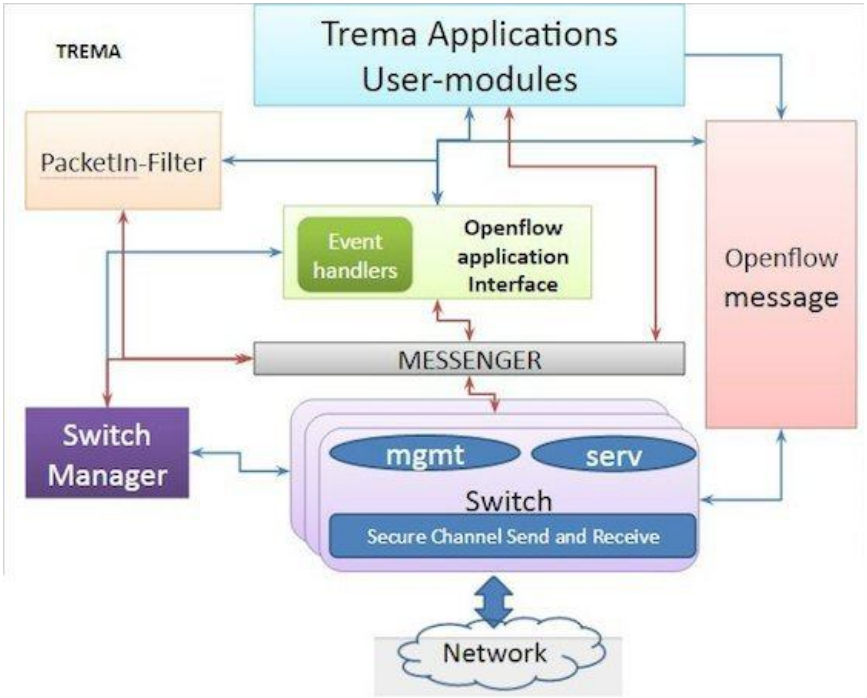
- A framework for developing OpenFlow controllers in Ruby and C
- An OpenFlow programming framework for developing an OpenFlow controller that was originally developed by NEC.
- Provides basic infrastructure services as part of its core modules that support the development of user modules in Ruby or C.
- Developers can individualize or enhance the base controller functionality by defining their own controller subclass object.
- The core modules provide a message bus that allows the communication.
- The infrastructure provides a command-line interface and configuration file system for configuring and controlling applications, managing messaging and filters, and configuring virtual networks.

- Trema is more a software platform for OpenFlow developers/researchers/enthusiasts than a production controller.
- For example, if the user wants an OpenFlow controller to manage the network topology, she just has to build and run Trema with an existing topology-manager application.
- Via Network Domain Specific Language (DSL) – Trema-based OpenFlow controller can interoperate with any element agent that supports OpenFlow without require a specific agent.



- Trema architecture includes core modules such as packet-in filter, switch and switch manager, OpenFlow application interface, necessary libraries, network DSL for configurations, an emulator for integrated development, and Tremashark to support debugging.

- Trema employs a multi-process model, in which many functional modules are loosely coupled via a messenger.
- The functional modules could be any of the user modules (applications) or the core modules.
- The modules interact via messenger using the following six important APIs — the first three are for receiving and last three are for sending messages:




```
1 add_message_received_callback [RECEIVE NOTIFICATION MESSAGE]
2 add_message_requested_callback [RECEIVE REQUEST MESSAGE]
3 add_message_replied_callback [RECEIVE REPLY MESSAGE]
4 send_message [SEND NOTIFICATION MESSAGE]
5 send_request_message [SEND REQUEST MESSAGE]
6 send_reply_message [SEND REPLY MESSAGE]
```

- The fig. above represents the functional diagram of Trema
- The messenger acts as a glue for linking user modules (applications), core modules and monitoring systems.

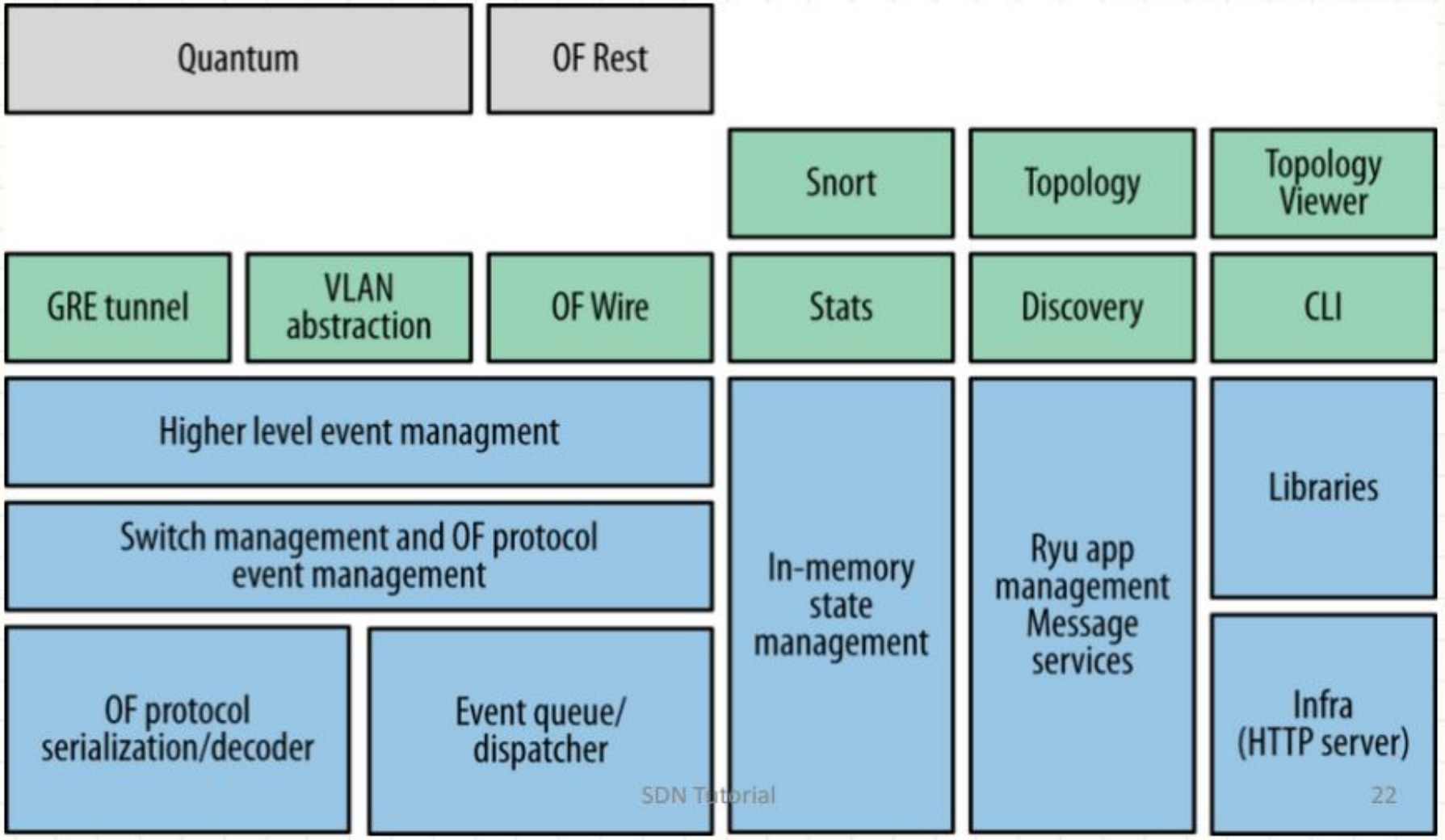
- **Core Modules:**
  - The core modules of the Trema framework mostly include OpenFlow foundational modules and typically those that are useful for multiple applications.
- **Switch and Switch-Manager:**
  - These two core modules implement the necessary functionalities for interactions with the OpenFlow switches.
  - They also maintain all the necessary information about the switches.
  - The switch manager is responsible for creating the instance (switch daemons) of a switch.
  - There will be one switch_daemon process for each OFS instance and only one instance of switch_manager.
  - The service_name of a switch daemon starts initially with **switch.(OFS IPaddr:port)**

- **Packet-In Filter:**
  - The PacketIn message is a way for the OpenFlow switch to send a captured packet to the controller.
  - Switch sends the packets to the controller only when it is asked to do so by the controller or when there is no appropriate entry in the switch's flow table.
  - It is responsible for handing the packets that arrive at the controller from the OpenFlow switches.
- **TremaShark:**
  - Tremashark is the Wireshark plugin for keeping track of any interprocess communication events among the functional modules.
  - The events could vary from messages to secure-channel status to the queue statuses.
- **Libraries:**
  - Protocol: OpenFlow.
  - Interfaces: OpenFlow application, switch, management.
  - Commonly used data structures: Linked list, doubly-linked list, hash table, timers.
  - Utilities: Log, stats, wrapper.
  - Network Protocols: TCP, IP, UDP, ether and etherIP, ICMP and IGMP.

Ryu:

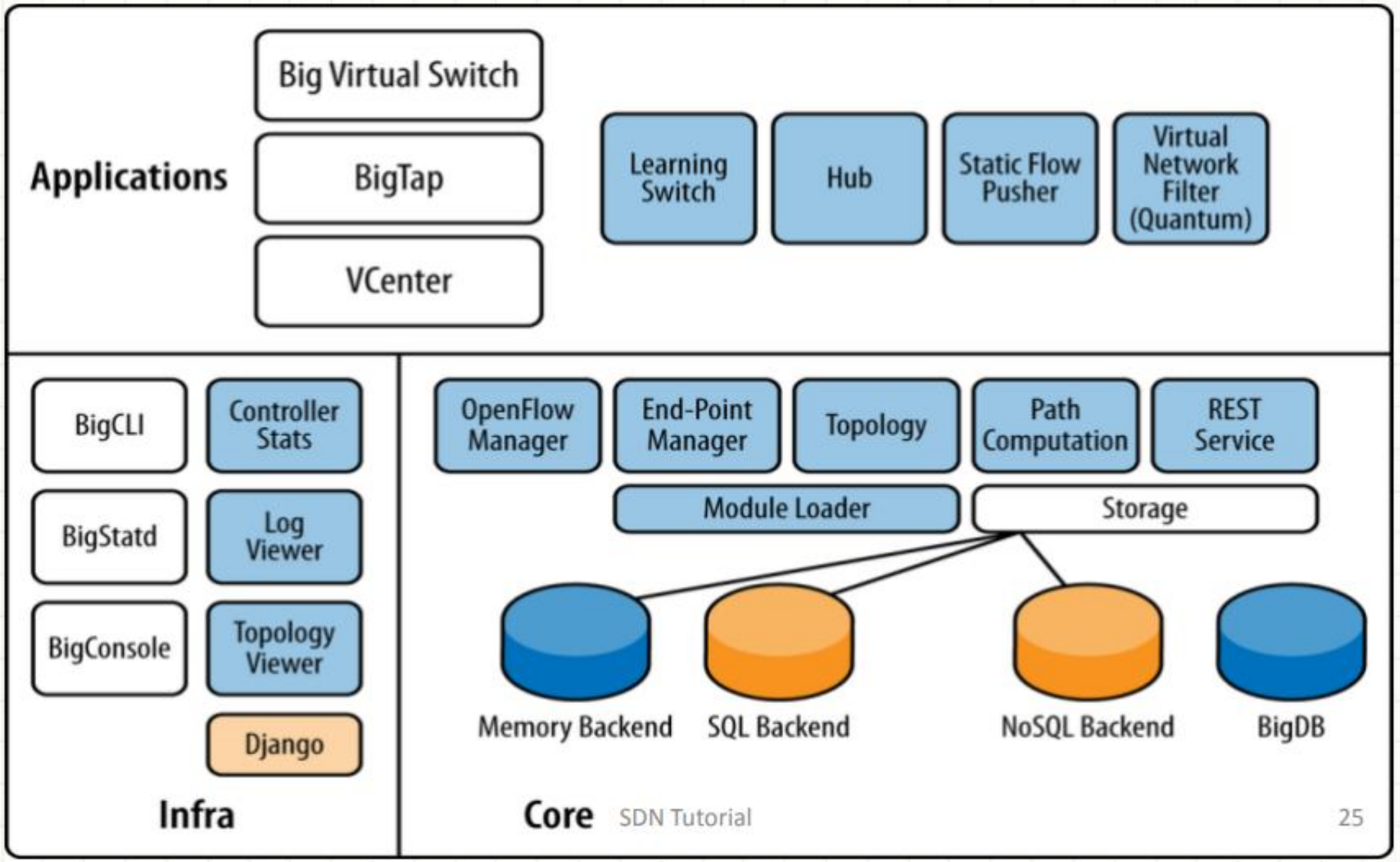
- Component-based, open source framework implemented entirely in Python.
- Components include:
  - OpenFlow wire protocol support
  - Event management
  - Messaging
  - In memory state management
  - Application management
  - Reusable libraries
- Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications.
- Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions.
- Has an Open-stack Quantum plug-in supports both GRE(Generic Routing Encapsulation) based overlay and VLAN
- Aims to be an “Operating System” for SDN

Ryu Architecture:



Big Switch Networks/FLOODLIGHT

- Core module handles I/O from switches and translates OpenFlow messages into Floodlight events, creating an event-driven, asynchronous application framework.
- Floodlight incorporates a threading model that allows modules to share threads with other modules.
- Floodlight is Java/Jython centric
    - Jython: Python for the Java Platform

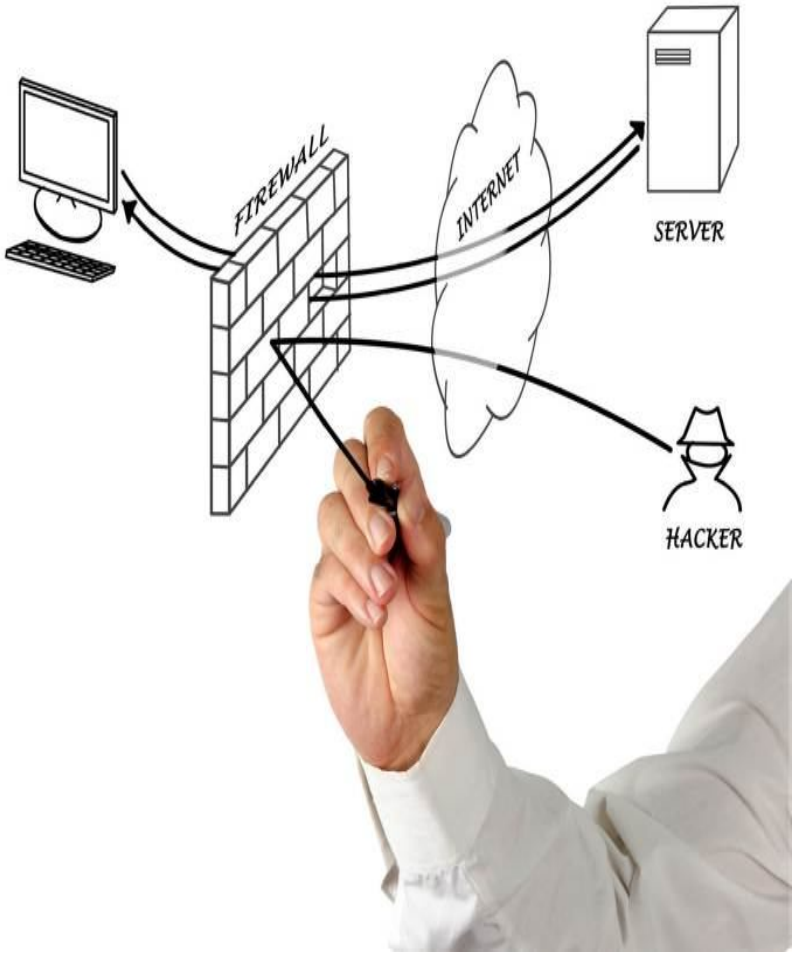

Applications
Big Virtual Switch
BigTap
VCenter
Learning Switch
Hub
Static Flow Pusher
Virtual Network Filter (Quantum)
BigCLI
Controller Stats
BigStatd
Log Viewer
BigConsole
Topology Viewer
Django
Infra
OpenFlow Manager
End-Point Manager
Topology
Path Computation
REST Service
Module Loader
Storage
Memory Backend
SQL Backend
NoSQL Backend
BigDB
Core

- **Features:**
- Offers a module loading system that make it simple to extend and enhance.
- Easy to set up with minimal dependencies
- Supports a broad range of virtual/physical OpenFlow switches (e.g.NOX3, OpenFlow, sFlow).
- Can handle mixed OpenFlow and non-OpenFlow networks.
- Designed to be high-performance – core from a commercial product from Big Switch Networks.
- Support for OpenStack cloud orchestration platform.

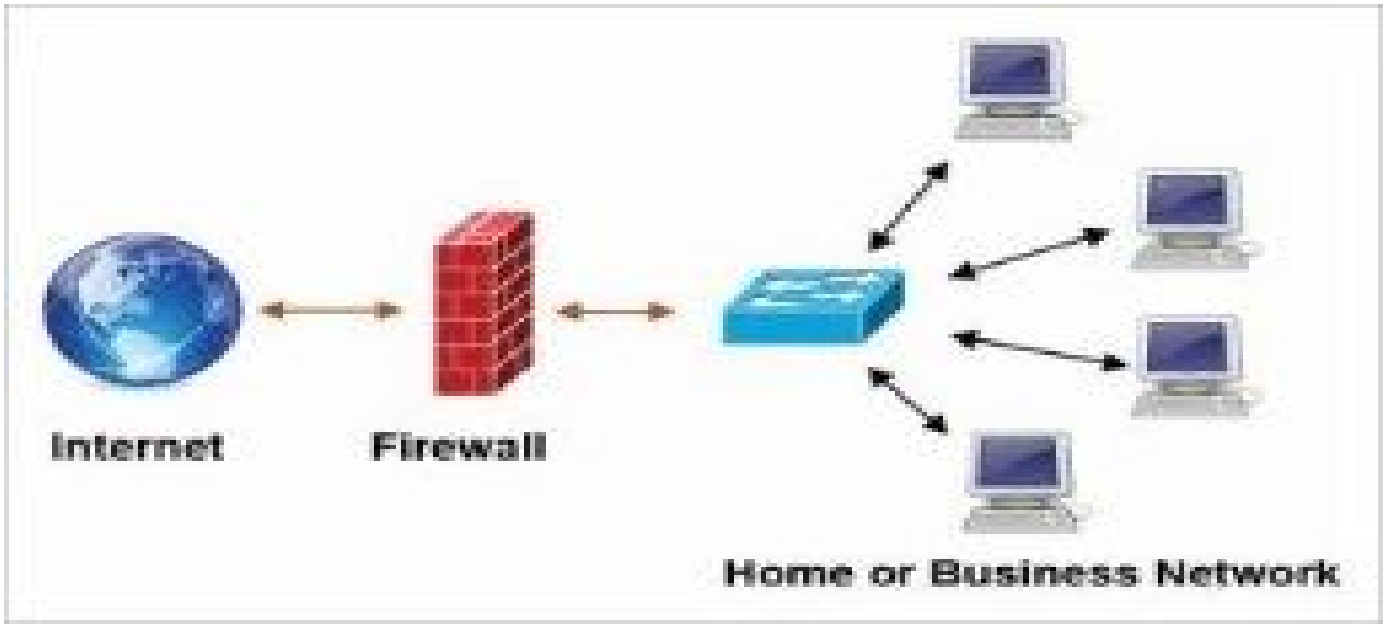
	NOX	POX	Ryu	Floodlight	ODL
Language	C++	Python	Python	Java	Java
Performance	Fast	Slow	Slow	Fast	Fast
Distributed	No	No	Yes	Yes	Yes
OpenFlow	1.0 (CPqD: 1.1, 1.2, 1.3)	1.0	1.0, 1.1, 1.3, 1.4	1.0	1.0, 1.3
Multi-tenant Clouds	No	No	Yes	Yes	Yes
Learning Curve	Moderate	Easy	Moderate	Steep	Steep

Customization of Control Plane : Firewall

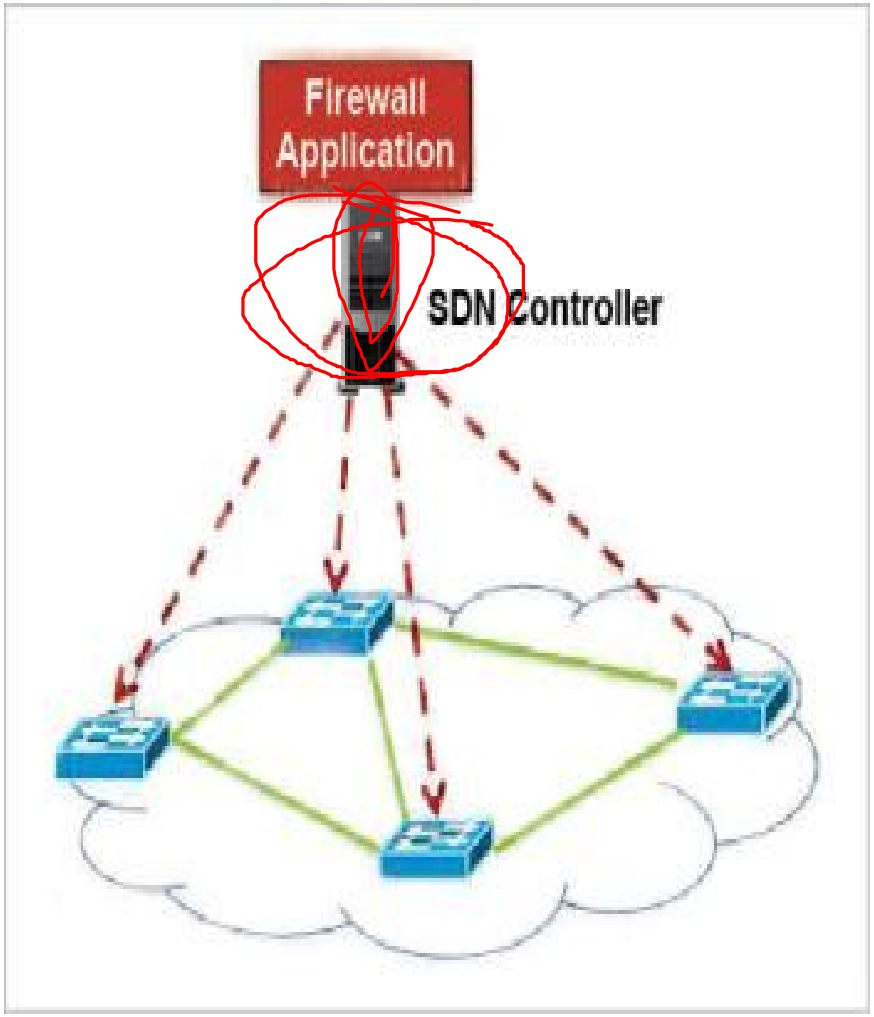
- A Firewall is a system, based on set of rules, that secures incoming network packets coming from various sources, as well as outgoing network packets.
- It can monitor and control the flow of data which comes into the network from different sources, and works on the basis of predefined rules.
- Firewalls typically maintain a barricade between a confidential, protected internal network and another outside network, such as the Internet, which is assumed not to be secure or trusted.



- Firewalls can be categorized as either hardware or software firewalls.
- Network (Hardware) firewalls are software programs running on different hardware appliances in the network.
- Software based firewalls provide a layer of software on a host, which controls network traffic in and out of that particular machine.



- **SDN based Firewalls:**
- Internal traffic is not seen and cannot be filtered by a traditional firewall.
- An SDN based firewall works both as a packet filter and a policy checker.
- The first packet goes through the controller and is filtered by the SDN firewall (SDN application bundled with SDN Controller).
- The subsequent packets of the flow directly match the flow policy defined in the controller.
- The firewall policy is centrally defined and enforced at the controller.



Firewall implementation

- Two Approaches:
  1. Pre-installing the rules onto the switch’s flow table
  2. Handling the packets directly as they come in
- Mostly 2nd approach is preferred because it provides flexibility in management.
- But one drawback is that too many packets can be delivered to controller and take up large portion of its resources.
- So it is necessary to block unnecessary packets at the switch level.

Implementing an SDN based firewall

- It requires the installation of **Mininet** and the **POX controller**.
- **Mininet** is a network emulator that enables the creation of a network of virtual hosts, switches, controllers, and links.
- It constructs a virtual network that appears to be a real physical network.
- We can create a network topology, simulate it and implement the various network performance parameters such as bandwidth, latency, packet loss, etc, with Mininet, using simple code.
- The firewall application is provided with a list of MAC address pairs, i.e., an access control list (ACL).
- When a connection gets established between the controller and the switch, the application installs static flow rule entries in the OpenFlow table to disable all communication between each MAC pair.

- At the specified switch, block all traffic coming from host 10.1.1.1:
  - **00-00-02-00-00-00-00-00 BLOCK scrip = 10.1.1.1**
- At the specified switch, block all traffic coming from host 10.1.2.2, if the packet TOS(type of service) is marked with 32 and it destined for 10.1.3.1:
  - **00-00-03-00-00-00-00-00 BLOCK scrip = 10.1.2.2 dstip = 10.1.3.1 tos = 32**
- At the specified switch, redirect traffic destined for 10.1.2.1 and, instead, send it to 10.1.2.2:
  - **00-00-01-00-00-00-00-00 REDIRECT dstip = 10.1.2.1 TO 10.1.2.2**
- At the specified switch, mark the TOS field of all packets sent by 10.1.1.1 with value 40:
  - **00-00-04-00-00-00-00-00 MARK scrip = 10.1.1.1 tos = 40**

- **Firewall execution in SDN :**
  - Check the MAC address against the firewall rules available in the POX controller.
  - Is transparent=False, and either Ether type is LLDP or the packets destination address is a Bridge Filtered Address? If yes, DROP.
  - Is the destination multi-cast? If yes, FLOOD.
  - Install the flow table entry in the switch so that this entry will be used when a similar packet reappears.

Firewall Customization In POX Controller

- To start this, you will find a skeleton class file at ***pox/pox/misc/12_firewall.py***.
- This is currently not blocking any traffic and you will need to modify this skeleton code to add your own logic later.
- Here, some rules are added to block the packet from a specific address.
- To test the firewall first, put the *l2_firewall.py* in the *pox/pox/misc* directory and run the POX controller as shown below:
  - **./pox.py log.level -- DEBUG forwarding. 12_firewall**
- This code runs the firewall on the POX controller and filters the traffic by the rules specified in it.
- It is shown in the video

Georgia Institute of Technology

Dr. Nick Feamster
Associate Professor

Software Defined Networking

In this course, you will learn about software defined networking and how it is changing the way communications networks are managed, maintained, and secured.

School of Computer Science

Implementation using SDN Concepts.

- Wide area networks
- Service provider and carrier networks
- Campus networks
- Hospitality networks
- Mobile networks
- In-line network functions
- Optical networks

1. Campus Network:

- Campus networks are a collection of LANs in a concentrated geographical area.
- Usually the networking equipment and communications links belong to the owner of the campus.
- This may be a university, a private enterprise, or a government office, among other entities.
- Campus end users can connect through wireless access points (APs) or through wired links.
- They can connect using desktop computers, laptop computers, shared computers, or mobile devices such as tablets and smartphones.
- The devices with which they connect to the network may be owned by their organization or by individuals.

- Furthermore, those individually owned devices may be running some form of access software from the IT department, or the devices may be completely independent.
- There are a number of networking requirements that pertain specifically to campus networks. These include
    1. Differentiated levels of access,
    2. Bring your own device (BYOD),
    3. Access control and security,
    4. Service discovery, and
    5. End-user firewalls.

1. Campus Network:

- Campus networks are a collection of LANs in a concentrated geographical area.
- Usually the networking equipment and communications links belong to the owner of the campus.
- This may be a university, a private enterprise, or a government office, among other entities.
- Campus end users can connect through wireless access points (APs) or through wired links.
- They can connect using desktop computers, laptop computers, shared computers, or mobile devices such as tablets and smartphones.
- The devices with which they connect to the network may be owned by their organization or by individuals.

OpenFlow Evolution Summary