# Software Defined Networks

Prof. Ankit Gangwal

Centre for Security, Theory, and Algorithmic Research (CSTAR),
IIIT Hyderabad, India.

# Agenda

## Part I : Traditional Networks

- Introduction
- Network Devices
- Shortcomings & Challenges

## Part II : Software Defined Networks

- Concept
- Decoupling of Control & Data Plane
- Advantages
- Architecture
  1. Infrastructure Layer
  2. Control Layer
  3. Application Layer
  4. Communication Interfaces
- Working

# Agenda

## Part III : OpenFlow Protocol

- Introduction
- Components of OpenFlow Network
  1. Controller
  2. Secure Channel
  3. Flow Table
- OpenFlow Protocol Messages
- Instruction & Action Set
- Packet Matching
- Pipeline Processing
- Table Miss
- Flow Removal
- A Flow Table Entry

# Agenda

## Part IV : SDN Resources, Issues & Use Cases

- Software Switch Implementation Compliant With OpenFlow
- Controller Implementation Compliant With OpenFlow
- SDN Programming Languages
- Issues in SDN
  1. Performance Issues
  2. Management Issues
  3. Security Issues
  4. Reliability Issues
- Use Cases of SDN
  - FlowVisor

## Part V : References

# Traditional Networks

# Traditional Networks

Network consists of :

- Data Plane,
- Control Plane,
- Management Plane.

**1. Data Plane:**

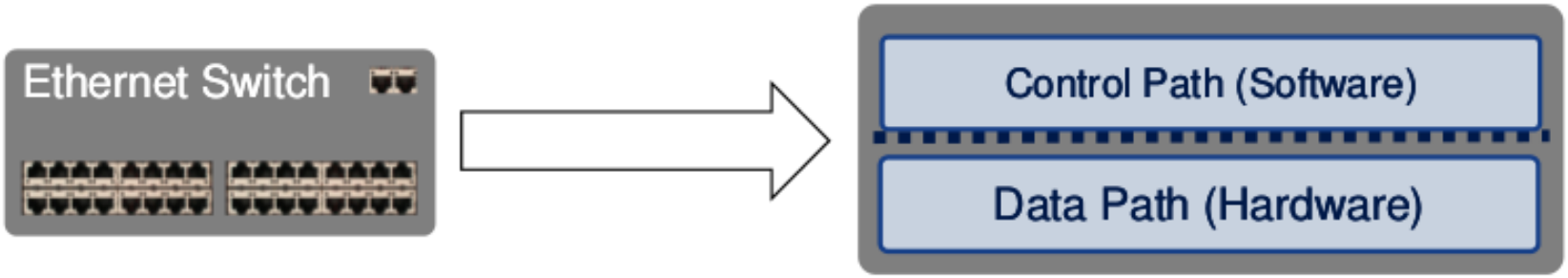
- **Definition**: Handles the actual forwarding of data packets through the network, based on decisions made by the control plane.

**2. Control Plane:**

- **Definition**: Makes decisions about the best paths for data packets to travel across the network, using routing and switching protocols.

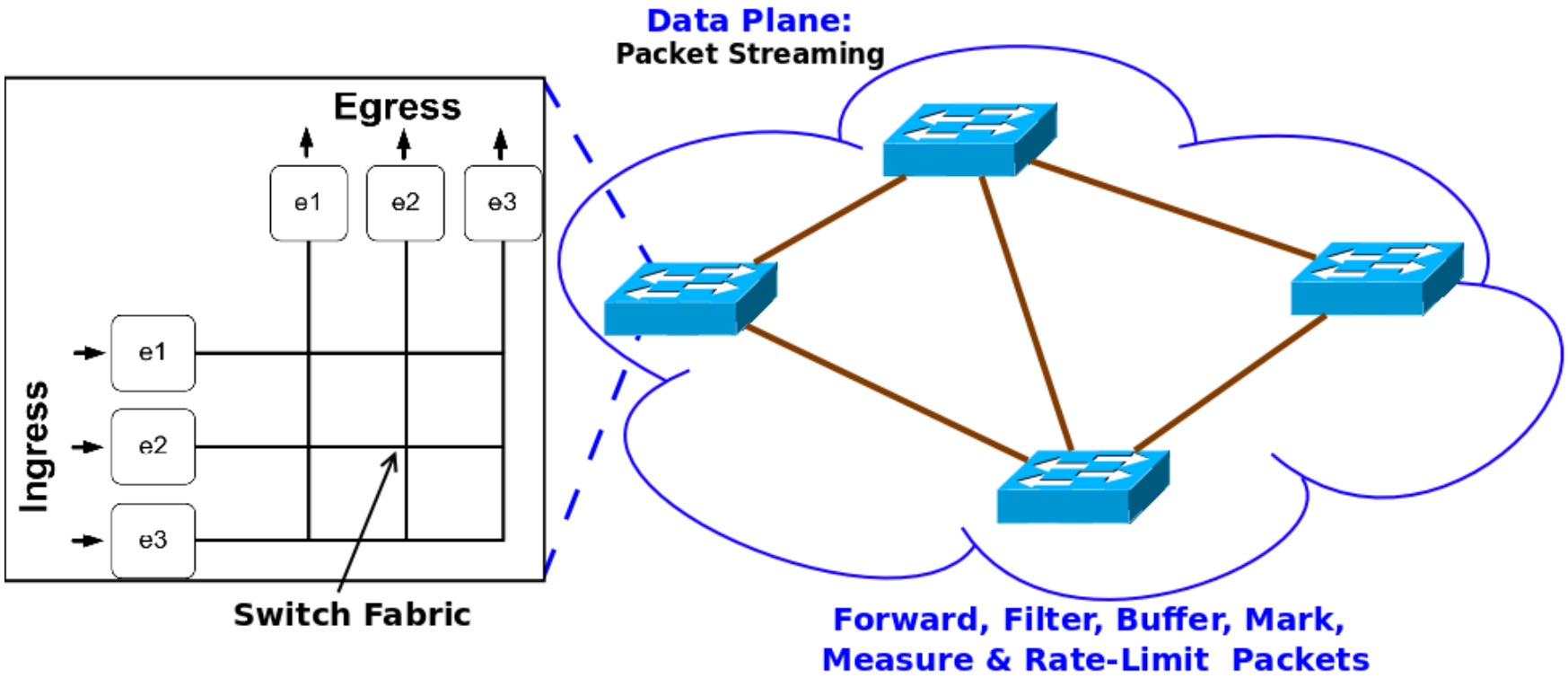
**3. Management Plane:**

- **Definition**: Provides interfaces for configuring, monitoring, and managing network devices, but with limited programmability.



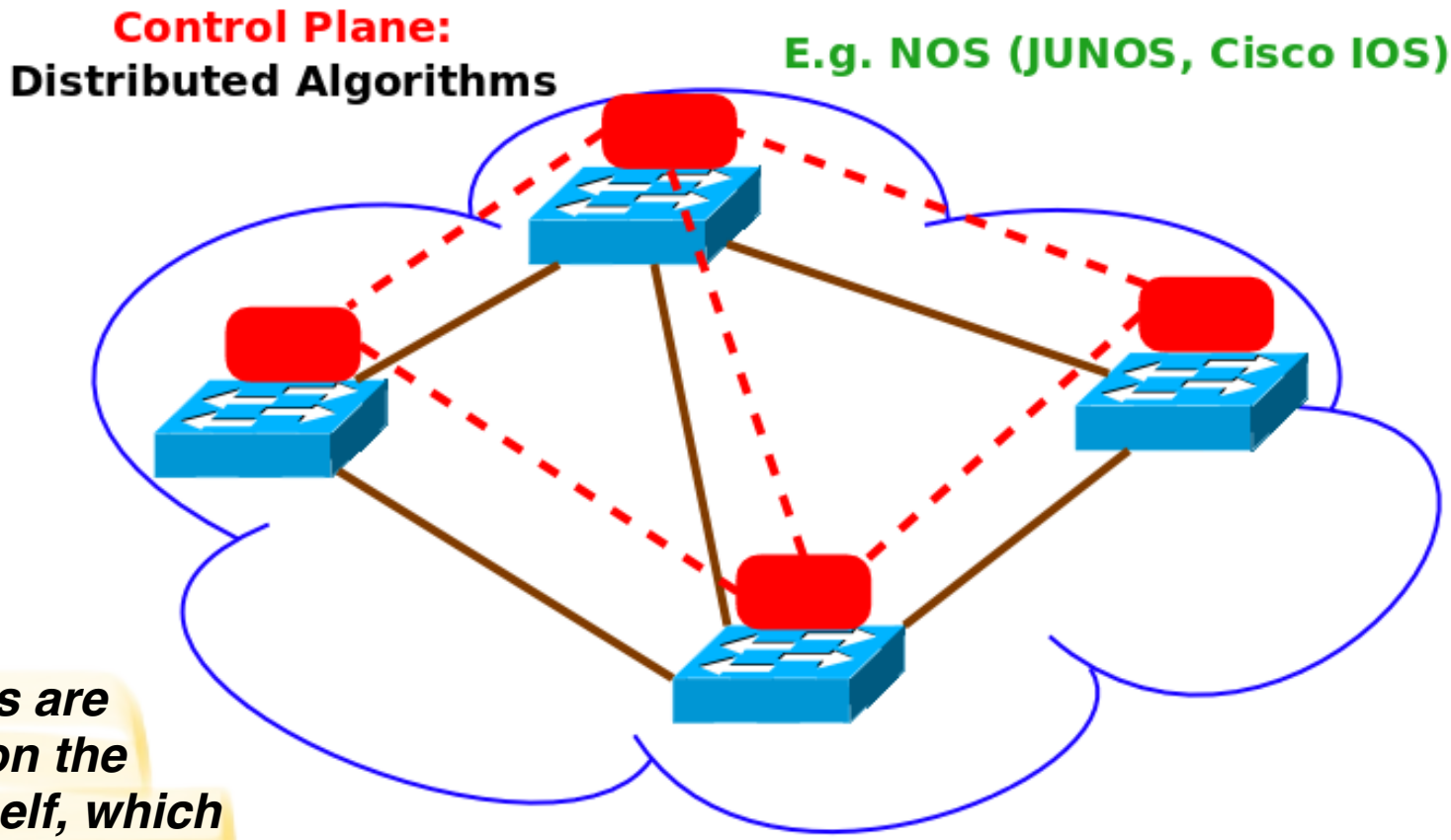
# Traditional Networks

## Data Plane



# Traditional Networks

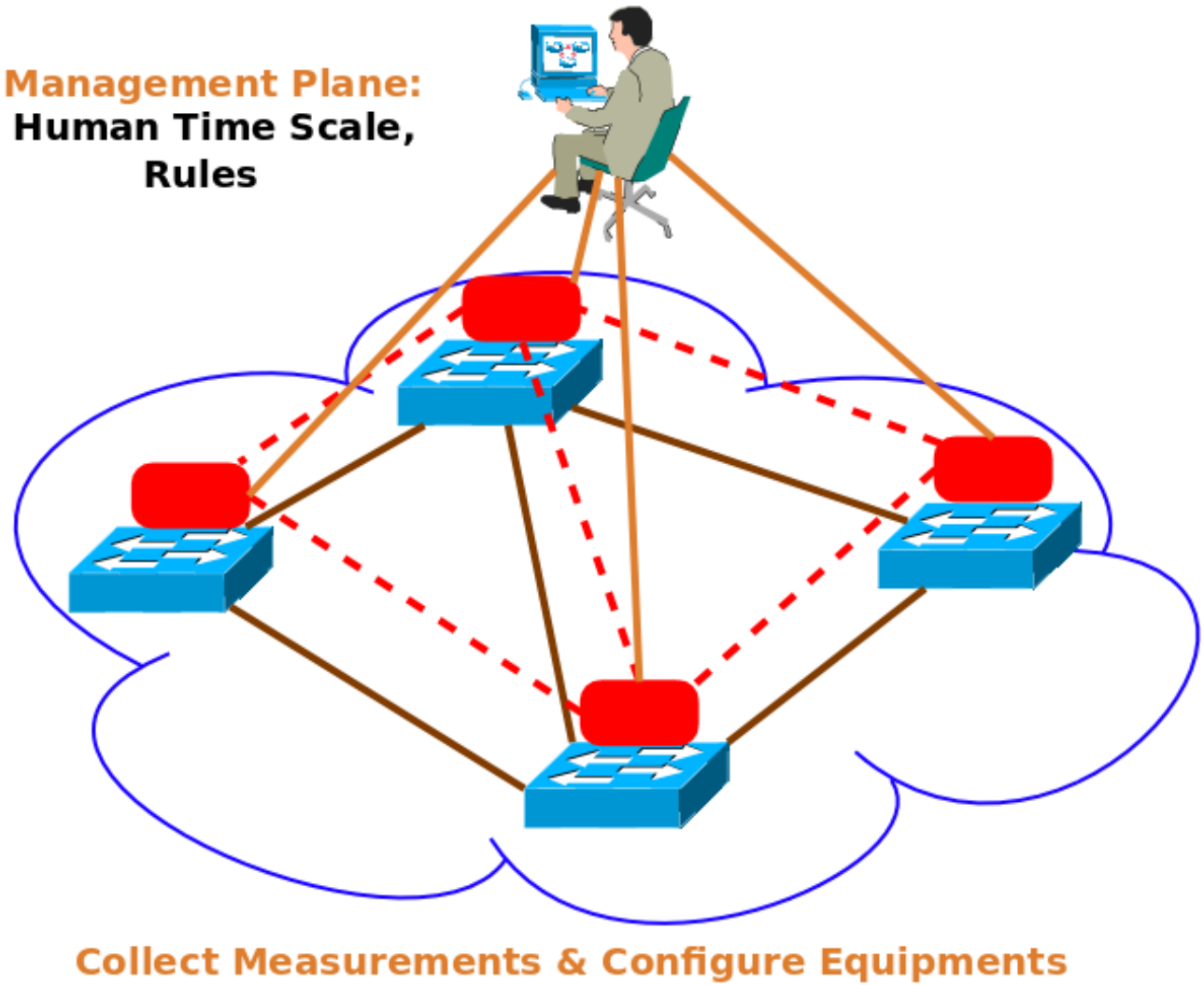
## Control Plane




***Controllers are mounted on the routers itself, which creates the chaos by being less flexible.***

# Traditional Networks

## Management Plane



# Traditional Networks

**Network Devices:**

- ✓ Hardware + Operating Systems + Applications.
- ✓ Built into an **OPAQUE** box.
- ✓ Mix & match not allowed.

# Traditional Networks

**Results:**

- x Very high **effective** cost of box.
  - Deprivations due to replacements, to support newer functionality.
- x Networks have remained the same.
  - **Un-Programmable** by owner.
- x Complete vendor dependence.
  - **Innovations** are limited to vendors or their partners.

# Traditional Networks

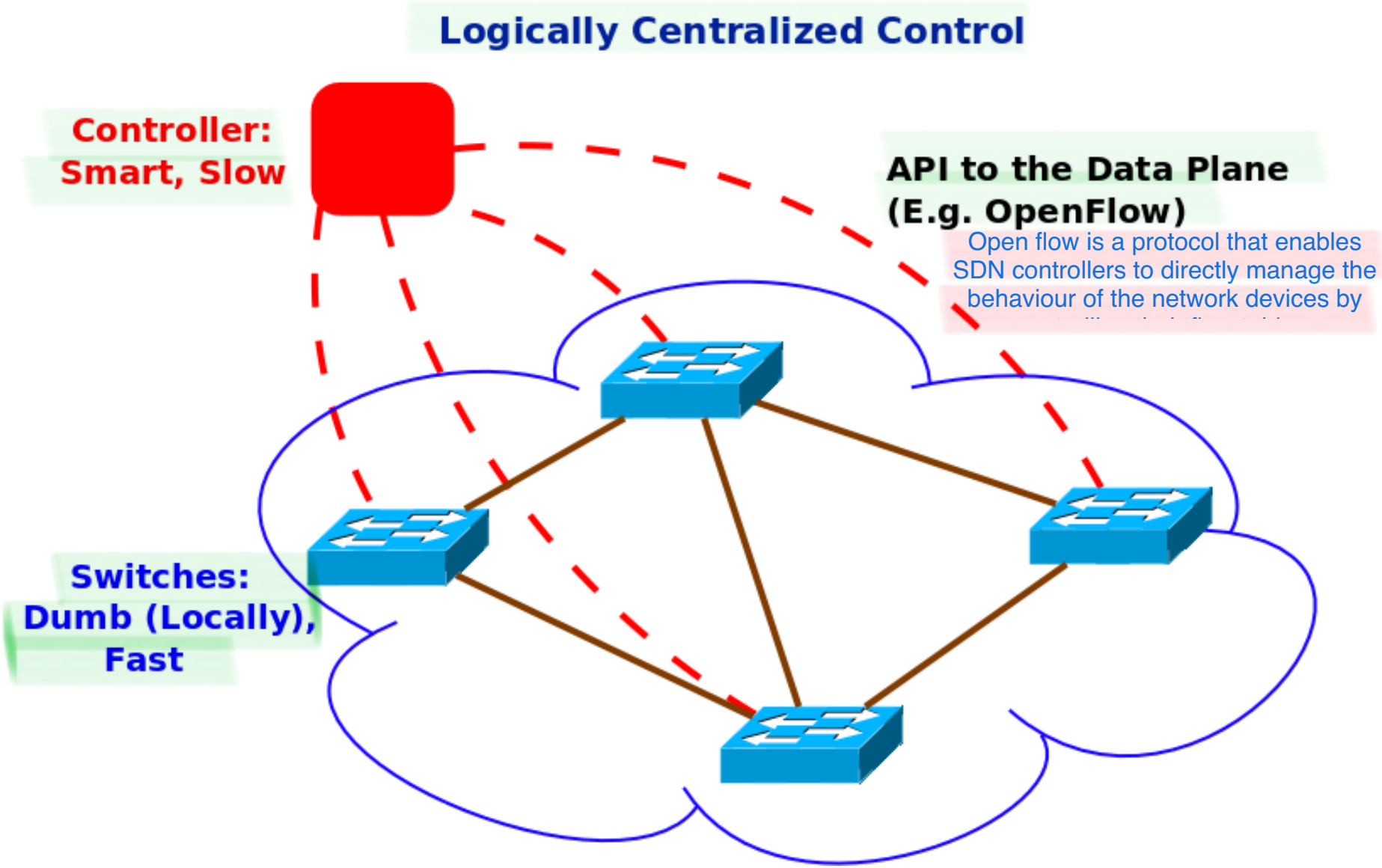
**Challenges:**

- Long hardware fabrication cycles.
- Network management is still complex.
- Testing new protocols in real network.
  - Touching the box is not allowed by IT !

# Software Defined Networks

# Software Defined Networks

## SDN = Dumb Switches + Smart Controller



# Software Defined Networks

## SDN decouples **CONTROL** and **DATA** Plane.

### Data Plane

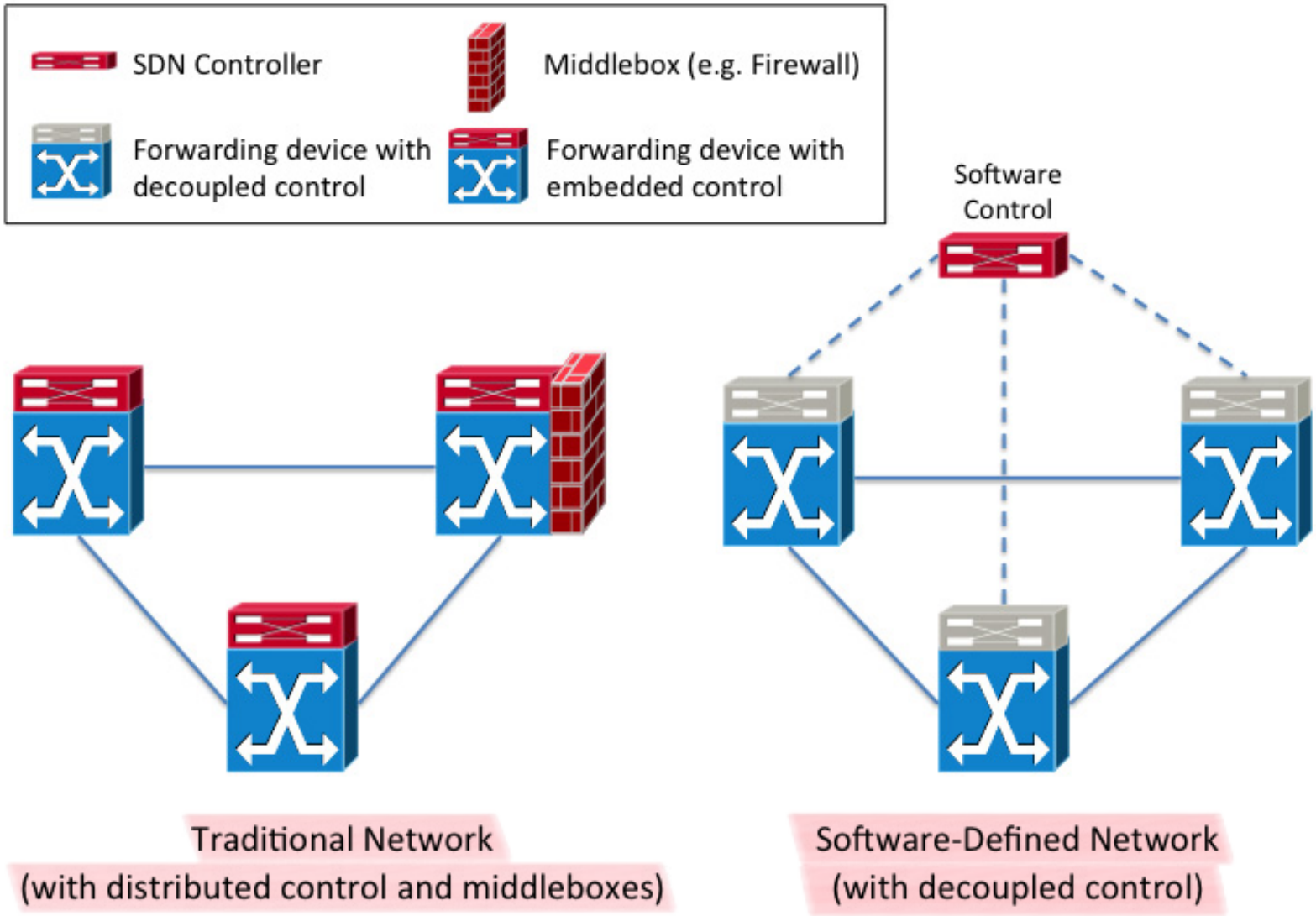
- Hardware functionality.
- Forwards packet.

### Control Plane

- Creates routing (forwarding) table.
- Can sit **out** of the box.
- Protocol used: **OpenFlow** [1].

This means that the controller is centraliser out of the box, and not mounted to the switch already.

# Decoupling of CONTROL and DATA Plane

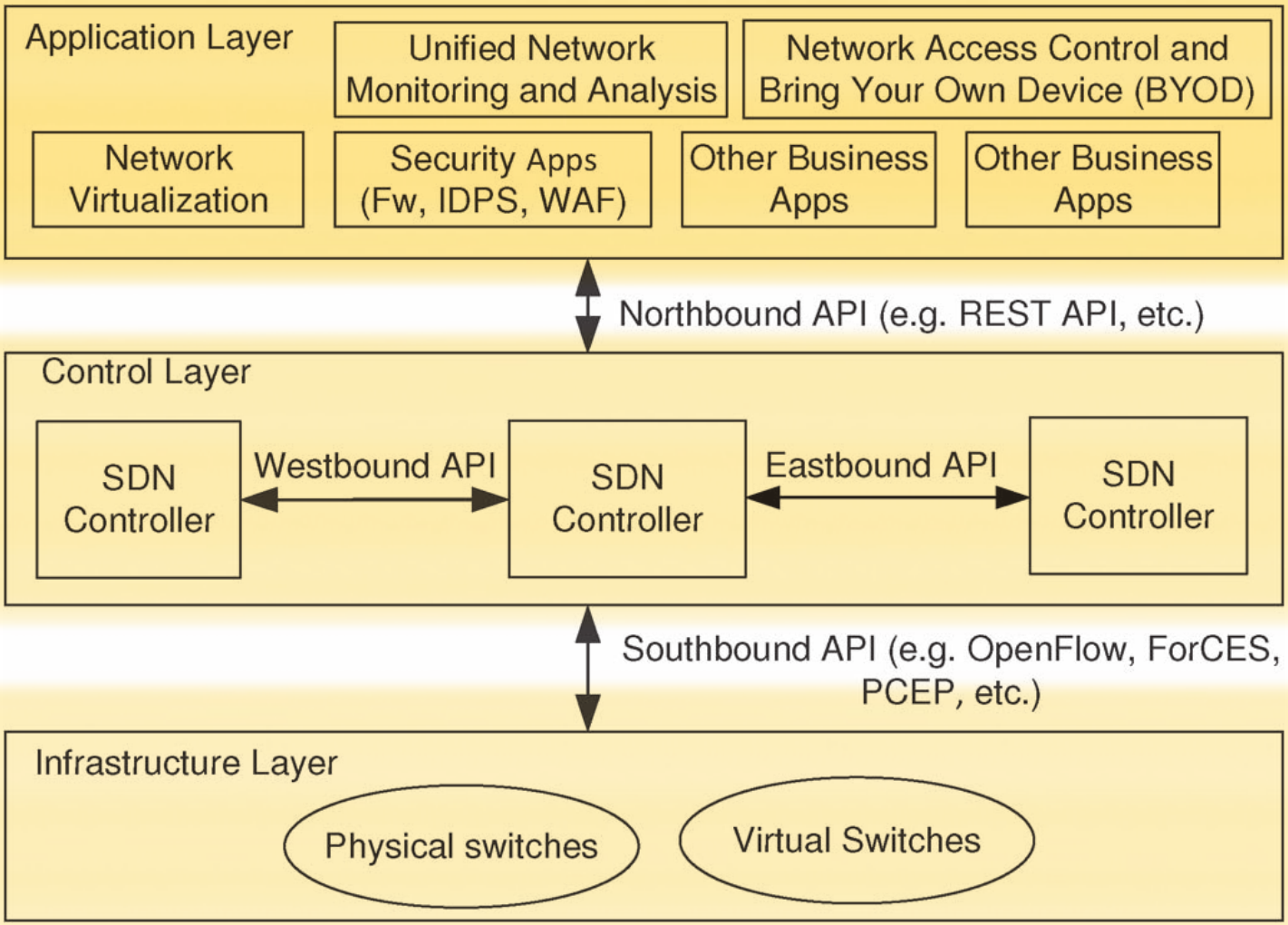


# SDN Benefits

- ~~Long hardware fabrication cycles.~~
  - ✓ Speed-to-market: No hardware fabrication cycles.
  - ✓ Fast upgrades.
- ~~Network management is still complex.~~
  - ✓ More flexibility with programmability.
  - ✓ Ease of customization and integration with other software applications.
  - ✓ Program a network V/s Configure a network.
- ~~Testing new protocols in real network.~~
  - ✓ Facilitate innovation in Network.
  - ✓ Independent innovations at each layer.

# SDN Architecture

*Southbound API: Connects the SDN controller to network devices, enabling direct control and management of the network's data plane (e.g., OpenFlow).*
*Northbound API: Connects the SDN controller to applications and services, allowing them to programmatically interact with the network for management and automation purposes (e.g., RESTful APIs).*



- Also known as the **Data Plane**.
- It consists mainly of Forwarding Elements (FEs) including physical and virtual Switches.
- FEs are accessible via an open interface.
- Allows packet switching and forwarding.
- **Every FE must support Southbound APIs (The reason why traditional FE can't be used as per SDN standards).**

# SDN Architecture

## Control Layer

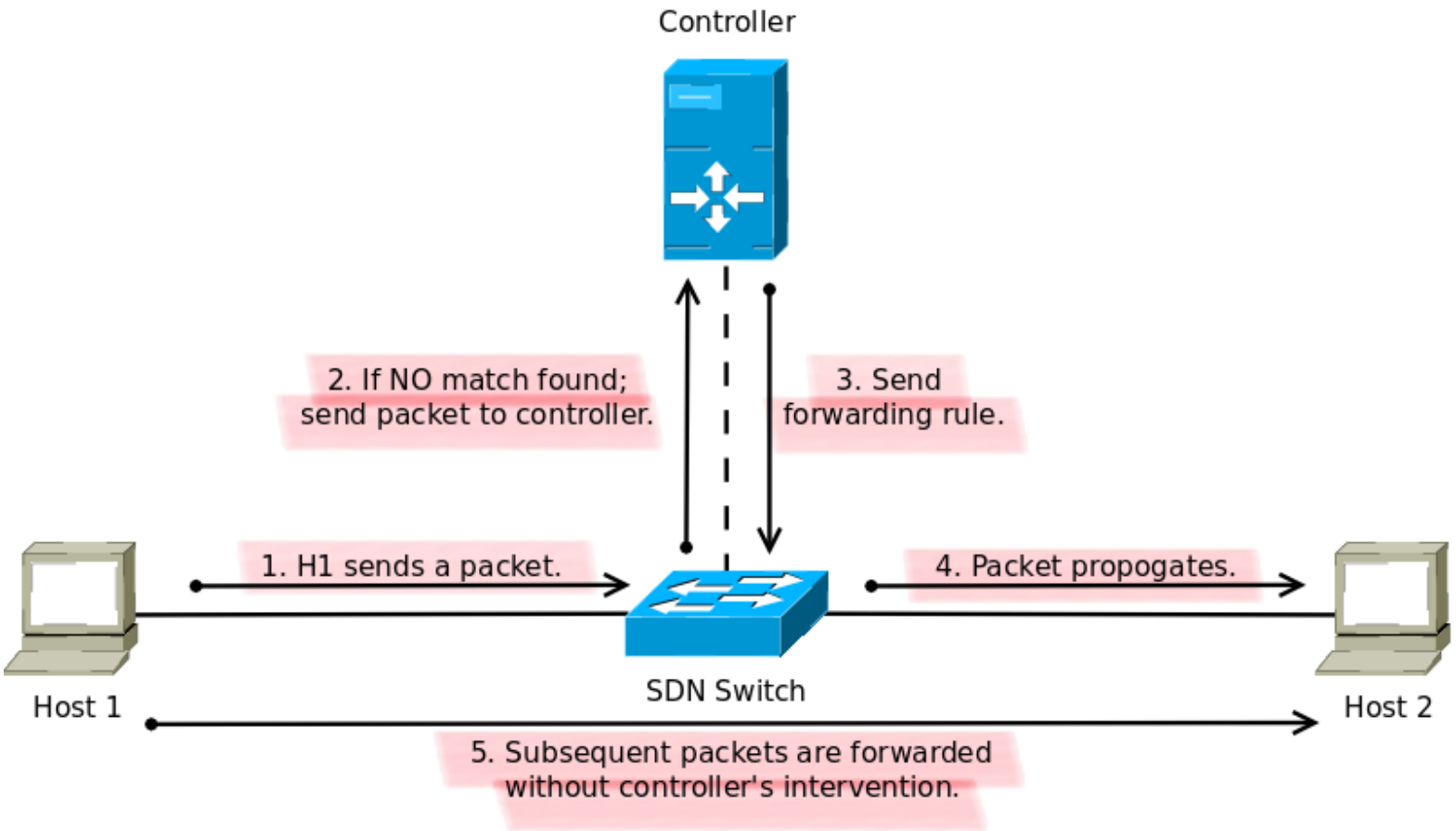
- Also known as the **Control Plane**.
- It consists of a set of software-based SDN Controllers.
- Provides a consolidated control functionality.
- Supervises the network forwarding behaviour through an open interface.
- Three communication interfaces allow the Controllers to interact: **Southbound, Northbound, East/Westbound**.

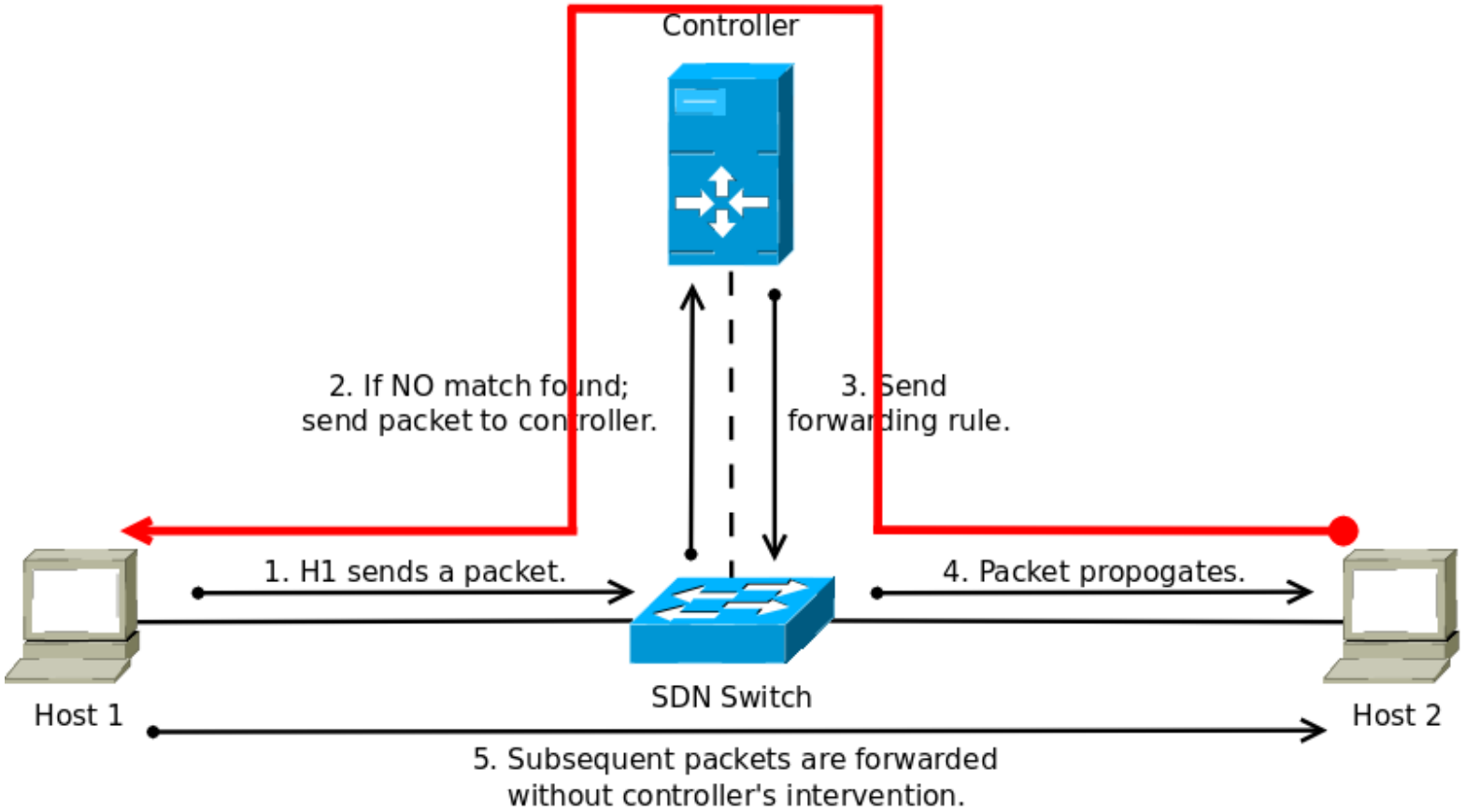
- **Southbound Interface**
  - Allows the Controller to interact with the forwarding elements in the infrastructure layer.
  - E.g. OpenFlow, a protocol maintained by ONF.

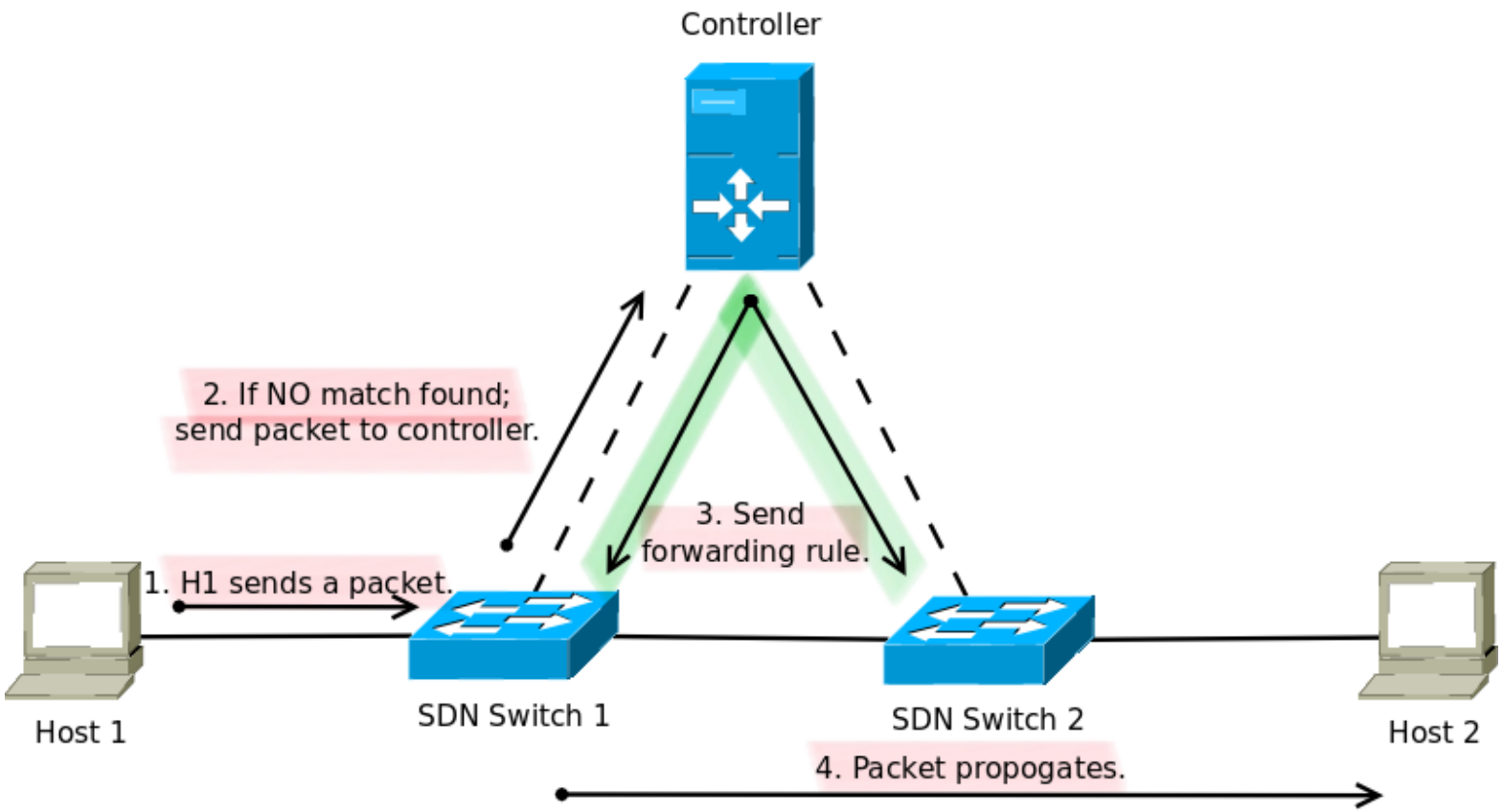
- **Southbound Interface**
  - Allows the Controller to interact with the forwarding elements in the infrastructure layer.
  - E.g. OpenFlow, a protocol maintained by ONF.
- **Northbound Interface**
  - Enables the programmability of the Controllers.
  - E.g. REpresentational State Transfer **(REST)**-based APIs.

- **Southbound Interface**
  - Allows the Controller to interact with the forwarding elements in the infrastructure layer.
  - E.g. OpenFlow, a protocol maintained by ONF.
- **Northbound Interface**
  - Enables the programmability of the Controllers.
  - E.g. REpresentational State Transfer **(REST)**-based APIs.
- **East/Westbound Interface**
  - It is an envisioned communication interface.
  - Which is not currently supported by an accepted standard.
  - It is mainly meant for enabling communication between federations of Controllers to synchronize state for high availability.

- It mainly consists of the end-user business applications.
- It consume the SDN communications and network services.
- E.g. network visualization and security business applications.


Controller
2. If NO match found;
send packet to controller.
3. Send
forwarding rule.
1. H1 sends a packet.
4. Packet propogates.
Host 1
SDN Switch
Host 2
5. Subsequent packets are forwarded
without controller's intervention.

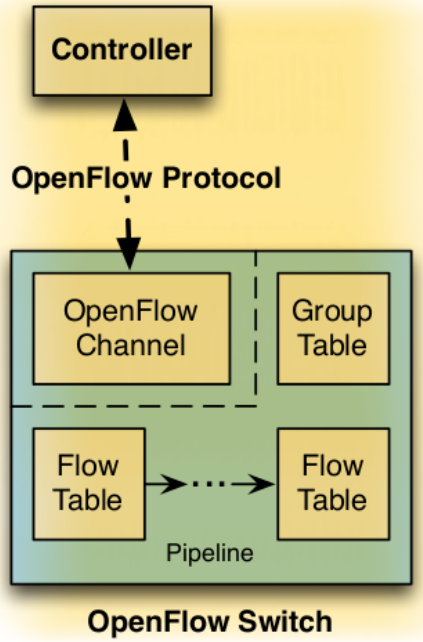

Controller
2. If NO match found;
send packet to controller.
3. Send
forwarding rule.
1. H1 sends a packet.
4. Packet propogates.
Host 1
SDN Switch
Host 2
5. Subsequent packets are forwarded
without controller's intervention.


Controller
2. If NO match found;
send packet to controller.
3. Send
forwarding rule.
1. H1 sends a packet.
Host 1
SDN Switch 1
SDN Switch 2
Host 2
4. Packet propogates.

# OpenFlow Protocol

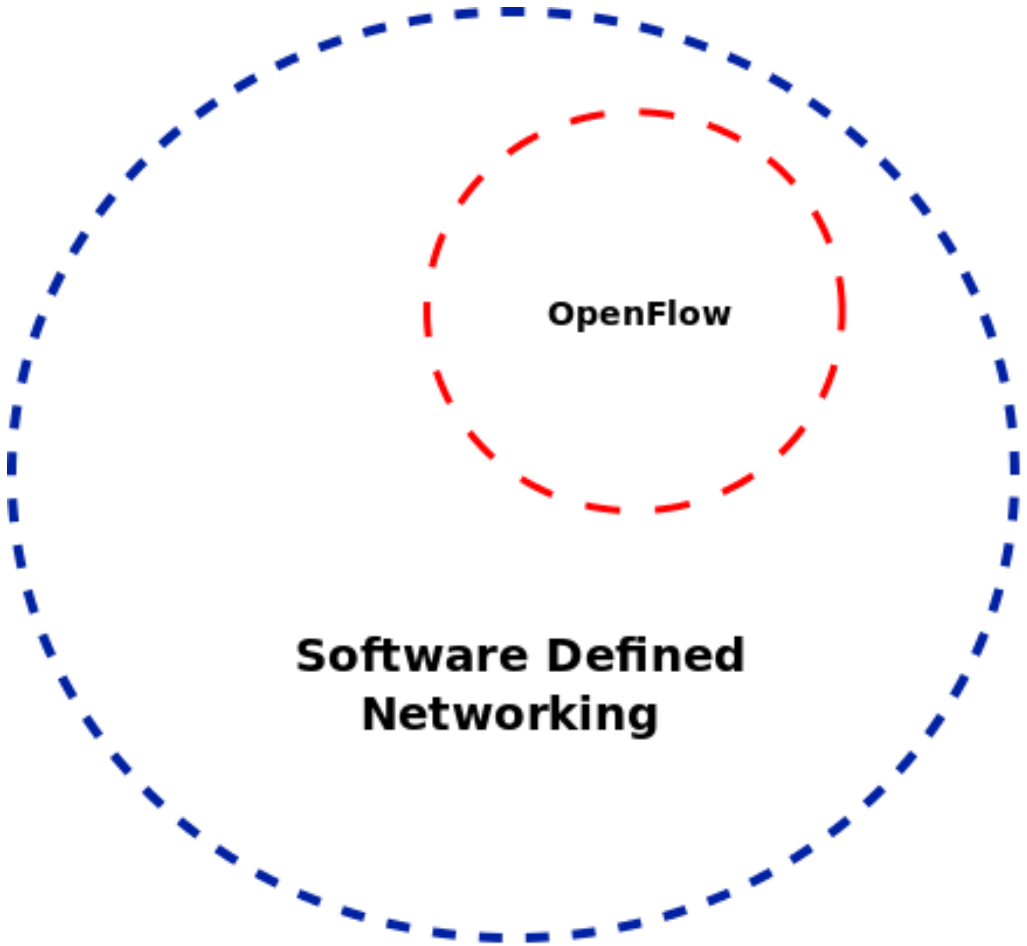
# OpenFlow Protocol

- OpenFlow is a **Layer 2** communication protocol.
- OpenFlow is the first standard communications interface defined between the **Control** and **Forwarding** layers of SDN architecture.
- OpenFlow allows **direct access** & **manipulation** of the Forwarding Plane of network devices such as Switches and Routers.

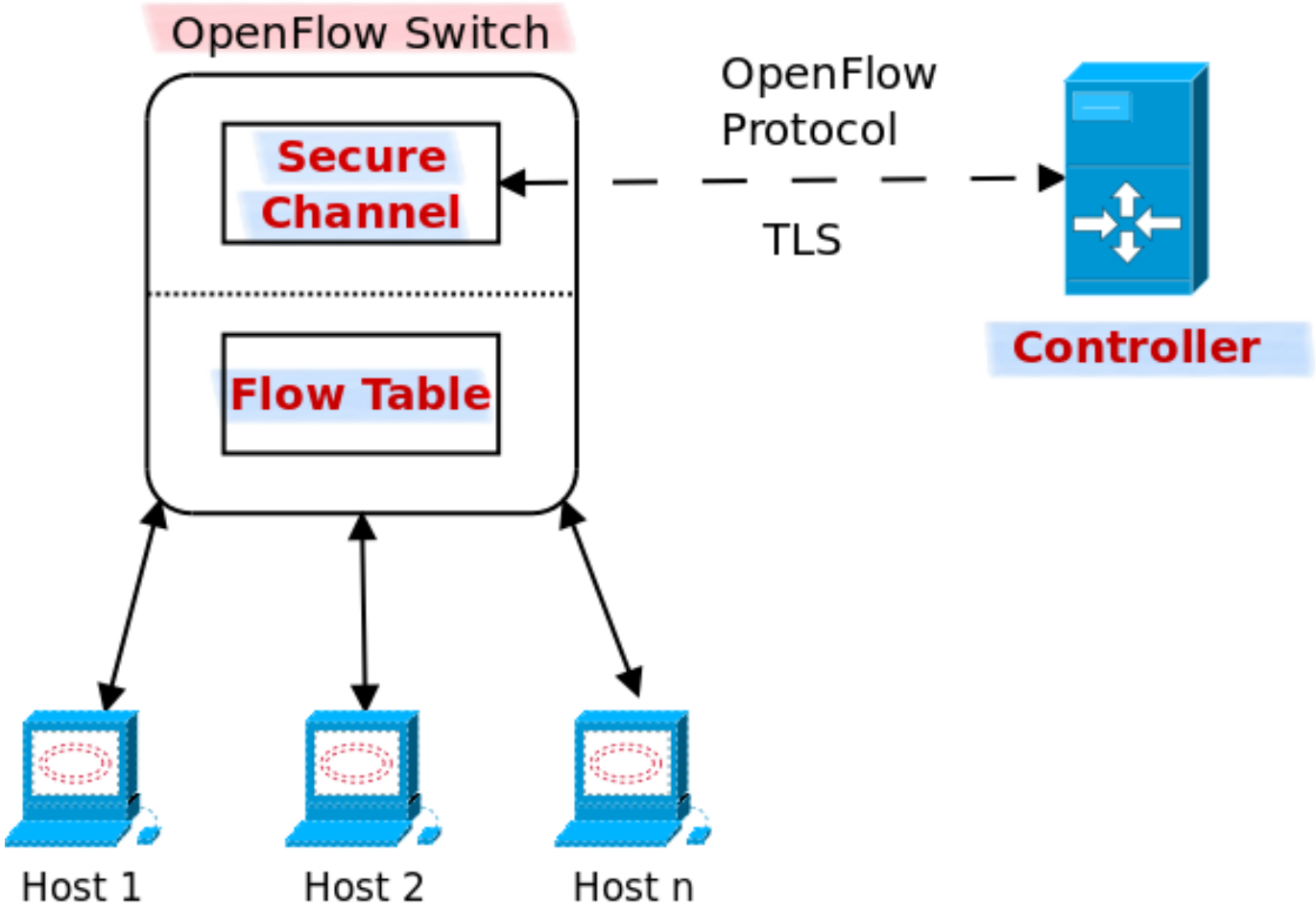


# OpenFlow **does not** equal SDN

- General Myth : OpenFlow is SDN
- Reality : OpenFlow is one flavour, or a subset, of SDN



# Components of OpenFlow Network



# Components of OpenFlow Network

1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.

# Components of OpenFlow Network

1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.
2. **Secure Channel**
   - Connects a Switch to the Controller, allowing commands and packets to be sent between a Controller & the Switch through OpenFlow protocol.

# Components of OpenFlow Network

1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.
2. **Secure Channel**
   - Connects a Switch to the Controller, allowing commands and packets to be sent between a Controller & the Switch through OpenFlow protocol.
3. **Flow Table**
   - Consists of a **Flow Entry** & an **Action** associated with each flow entry to tell the Switch how to process the flow.

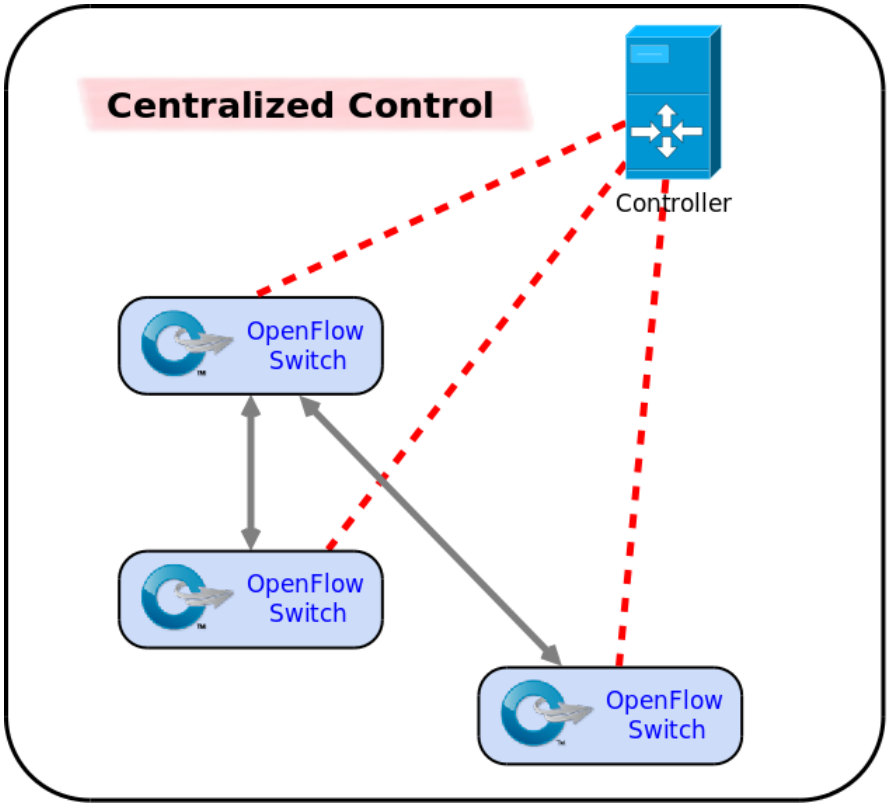
# Components of OpenFlow Network

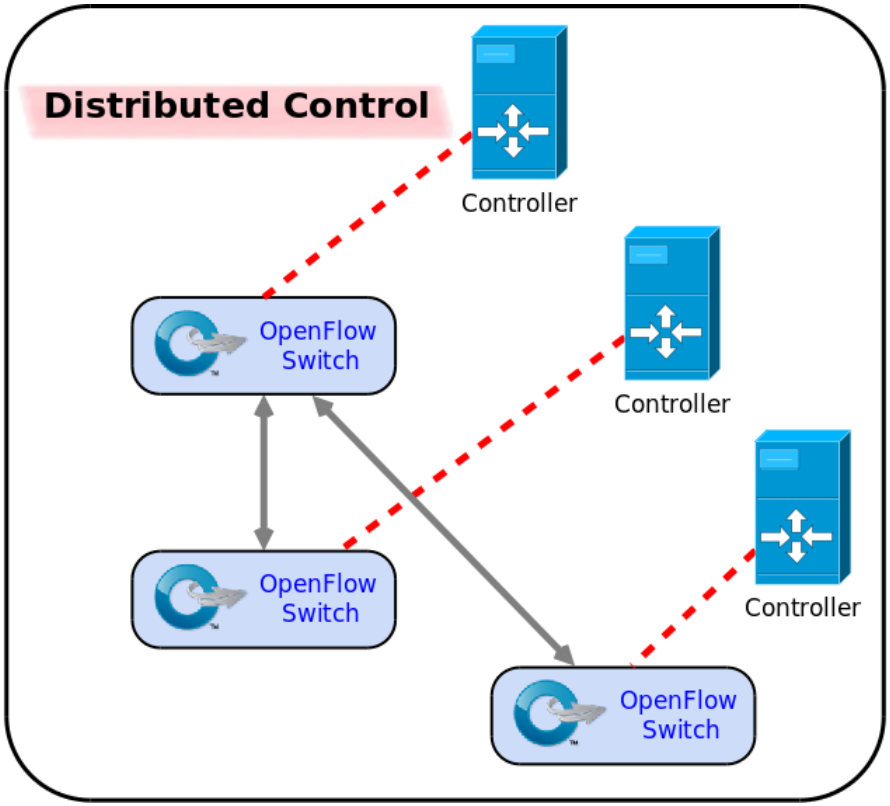
1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.
2. **Secure Channel**
   - Connects a Switch to the Controller, allowing commands and packets to be sent between a Controller & the Switch through OpenFlow protocol.
3. **Flow Table**
   - Consists of a **Flow Entry** & an **Action** associated with each flow entry to tell the Switch how to process the flow.

*** Network Operating System (NOS)** is the set of Controllers that forms an execution environment.

# Controller

## Centralized V/s Distributed



# Controller

## Measuring Performance & Handling Failure

***Throughput : Number of flow rule setups per second in the switch by the controller.***
***Latency : Time in milliseconds between the packet arriving at the switch and sending flow rule to the switch via controller under low load conditions.***

### Measuring Controller Performance

- **Throughput,** measuring the maximum flow set-up rate that a Controller can maintain.
- **Latency,** measuring the Controller's request processing time under low-load conditions.

### Handling Controller Failure

- Use a backup Controller.
- Use hybrid Switches. They start working as legacy Switch upon failure of the Controller.

# Secure Channel

- SC is the **Interface** that connects each OpenFlow Switch to Controller.
- A Controller configures and manages the Switch, receives events from the Switch & send packets out the Switch via this interface.
- SC establishes and terminates the connection between OpenFlow Switch & the Controller using Connection Setup & Connection Interruption procedures.
- The SC connection is a TLS (or TCP) connection. Switch and Controller mutually authenticate by exchanging certificates signed by a site-specific private key.

# Flow Table

## Open Flow Protocol Messages

**Features**: Requests or provides switch capabilities and supported features.
**Config**: Manages and retrieves switch configuration settings.
**Modify State**: Changes the switch's flow tables or group entries.
**Read State**: Retrieves statistics and status information from the switch.
**Packet-Out**: Sends a packet from the controller to a specific switch port.
**Barrier**: Ensures all previous commands are completed before processing new ones.

1. **Controller-to-Switch** - Initiated by the Controller & used to directly manage or inspect the state of the Switch.
   - *Features, Config, Modify State, Read-State, Packet-Out, Barrier.*

2. **Asynchronous** - Asynchronous messages are sent without the Controller soliciting them, from a Switch.
   - *Packet-in, Flow Removed / Expiration, Port-status, Error.*

3. **Symmetric** - Symmetric messages are sent without solicitation, in either direction.
   - *Hello, Echo.*

**Packet-In**: Sends a packet from the switch to the controller for processing.
**Flow Removed/Expiration**: Notifies the controller when a flow entry is removed or expires.
**Port Status**: Reports changes in the status of a switch port (e.g., up/down).
**Error**: Indicates an issue or failure in processing a command.
**Hello**: Establishes the connection and negotiates the OpenFlow version between the switch and controller.
**Echo**: Tests the liveness and measures the latency of the connection between the switch and controller.

# Flow Table

## Instructions & Action Set

- Each flow entry contains a set of instructions that are executed when a packet matches the entry.
- **Instructions** contain either a set of actions to be added to the action set or modify pipeline processing.
- **Action Set** contains a list of actions to be applied immediately to the packet.
- An Action set is always associated with every entry. By default it is empty.
- A flow entry modifies action set using **Write-Action** or **Clear-Action** instruction.

**Apply-Actions**: This instruction immediately applies a set of actions to the packet (e.g., modify header fields, send to a specific port).
**Write-Actions**: This instruction adds actions to the "Action Set," which will be applied later when the packet exits the pipeline.
**Clear-Actions**: Clears the action set for the packet, essentially removing any previously set actions.

# Flow Table

## Instructions

List of Instructions to modify action set:

1. **Apply Actions**
   - Apply the specified actions immediately.
2. **Clear Actions**
   - Clear all the actions in the set immediately.
3. **Write Actions**
   - Merge the specified actions to the current set.
4. **Write Metadata**
   - Write the meta data field with the specified value.
5. **Goto -Table**
   - Indicated the next table in the processing pipeline.

# Flow Table

## Actions

Actions are of two type:

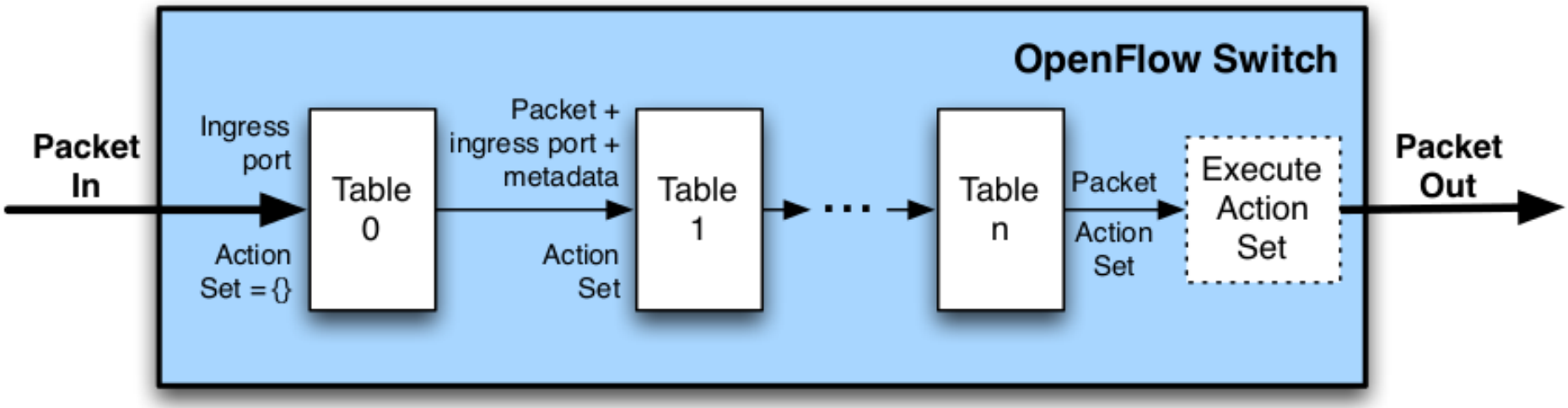
1. **Required Actions**
   - Output - Forward a packet to the specified port.
   - Drop
2. **Optional Actions**
   - Set-Queue
   - Push/Pop Tag
   - Set-Field

# Flow Table

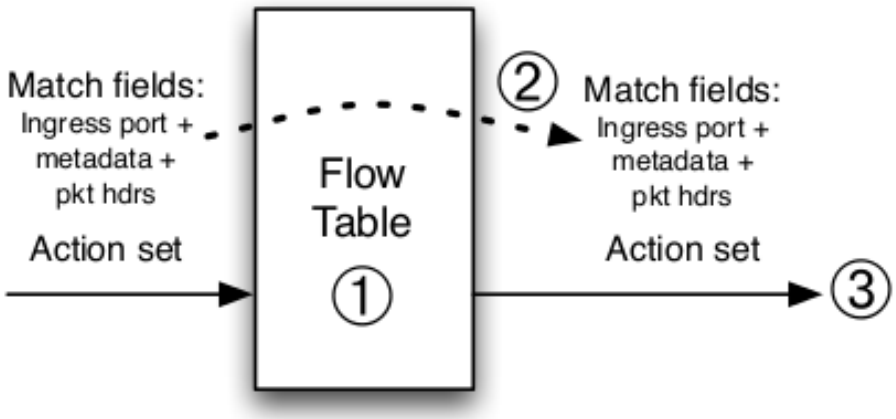
## Packet Matching

- OpenFlow pipeline contains multiple flow tables starting with **Table 0**.
- Each flow table contains one or more flow entries. Matching starts with the first flow entry.
- If a Match is found :
  - Instructions associated with flow entry are executed.
  - Instruction may direct the packet to the next flow table in pipeline.
  - When processing stops, the associated action set is applied.
- Instructions describe packet forwarding, packet modification and pipeline processing.

**Packets are matched against multiple tables in the pipeline:**



**Per-table packet processing:**



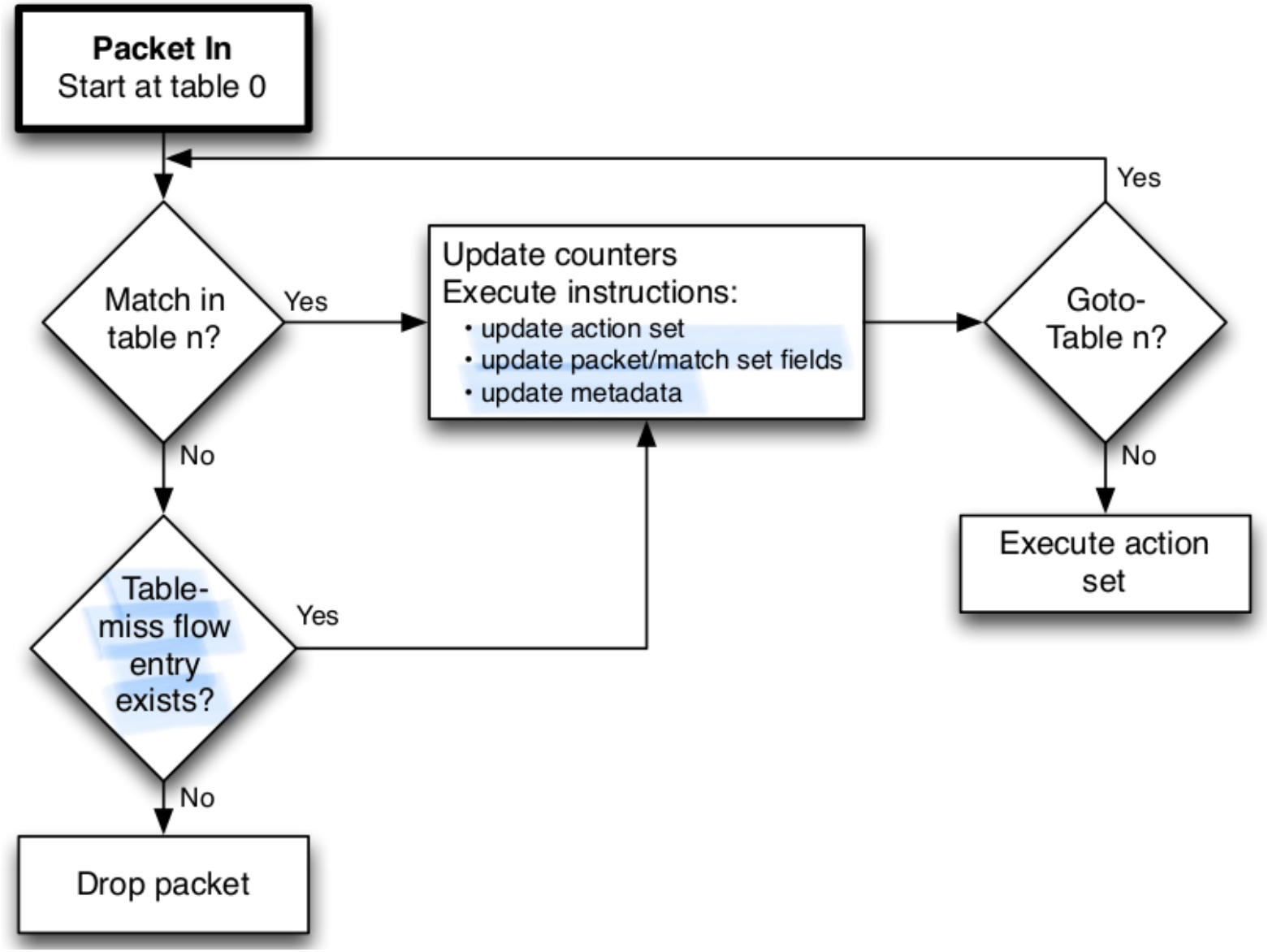

① Find highest-priority matching flow entry

② Apply instructions:
- i. Modify packet & update match fields (apply actions instruction)
- ii. Update action set (clear actions and/or write actions instructions)
- iii. Update metadata

③ Send match data and action set to next table

# Flow Table

## Flowchart detailing packet flow through an OpenFlow Switch



# Flow Table

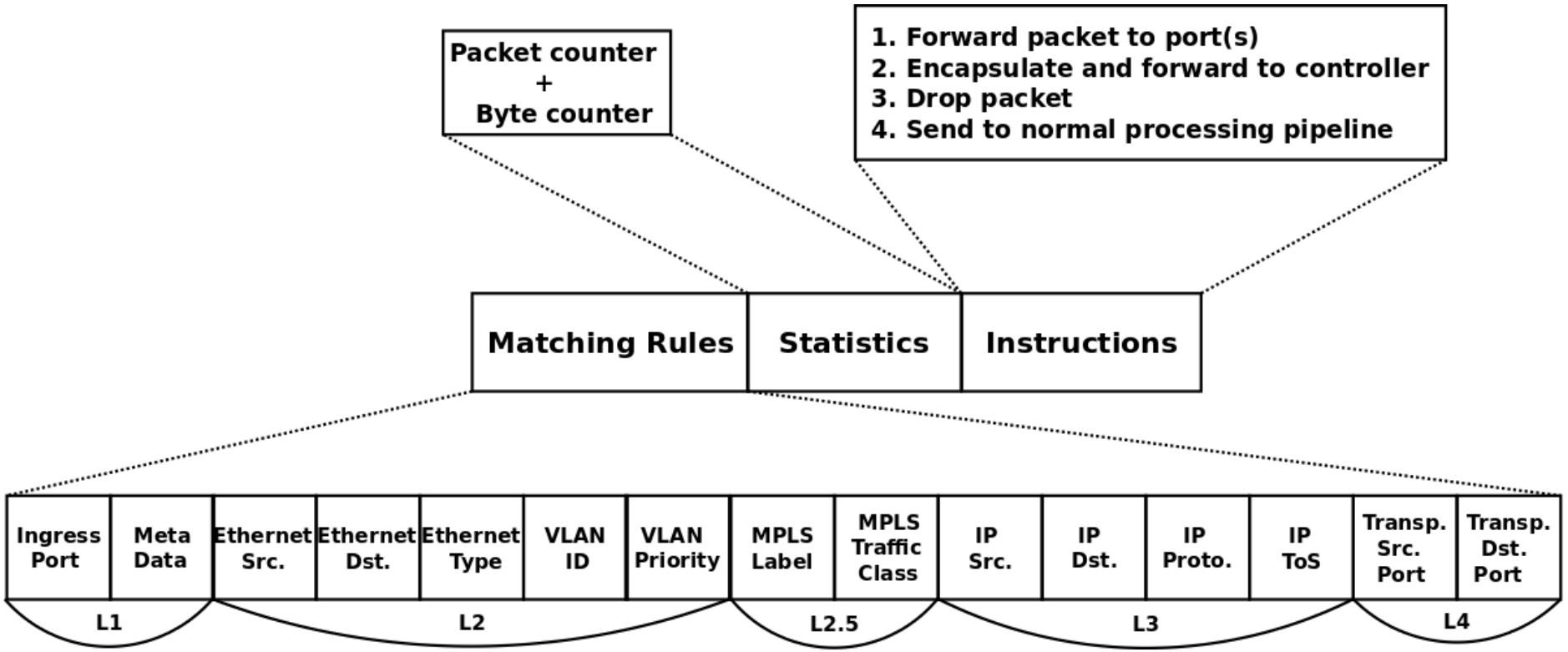
## Table Miss

- Every flow table must support a **table-miss** flow entry to process table misses.
- The table-miss flow entry specifies how to process packets unmatched by other flow entries in the flow table.
  - E.g. send packets to the Controller, direct packets to a subsequent table or drop packets.
- If a table is missed & a table-miss entry exists then it causes **penalty** in form of **delay**.

Flow entries are removed either

1. At the request of the Controller,
2. Or via the Switch flow expiry mechanism.
   - **idle_timeout** causes the flow entry to be removed when it has matched no packets in the given number of seconds.
   - **hard_timeout** causes the flow entry to be removed after the given number of seconds, regardless of how many packets it has matched.

# A Flow Table Entry



# SDN Resources, Issues & Use Cases

# Software Switch Implementation Compliant With OpenFlow

| Software Switch | Implementation | Overview | Version |
|---|---|---|---|
| Open vSwitch [2] | C/Python | Open source software Switch that aims to implement a Switch platform in virtualized server environments. Supports standard management interfaces and enables programmatic extension and control of the forwarding functions. Can be ported into ASIC Switches. | v1.0 |
| Pantou/OpenWRT [3] | C | Turns a commercial wireless router or Access Point into an OpenFlow-enabled Switch. | v1.0 |
| ofsoftswitch13 [4] | C/C++ | OpenFlow 1.3 compatible user-space software Switch implementation. | v1.3 |
| Indigo [5] | C | Open source OpenFlow implementation that runs on physical Switches and uses the hardware features of Ethernet Switch ASICs to run OpenFlow | v1.0 |

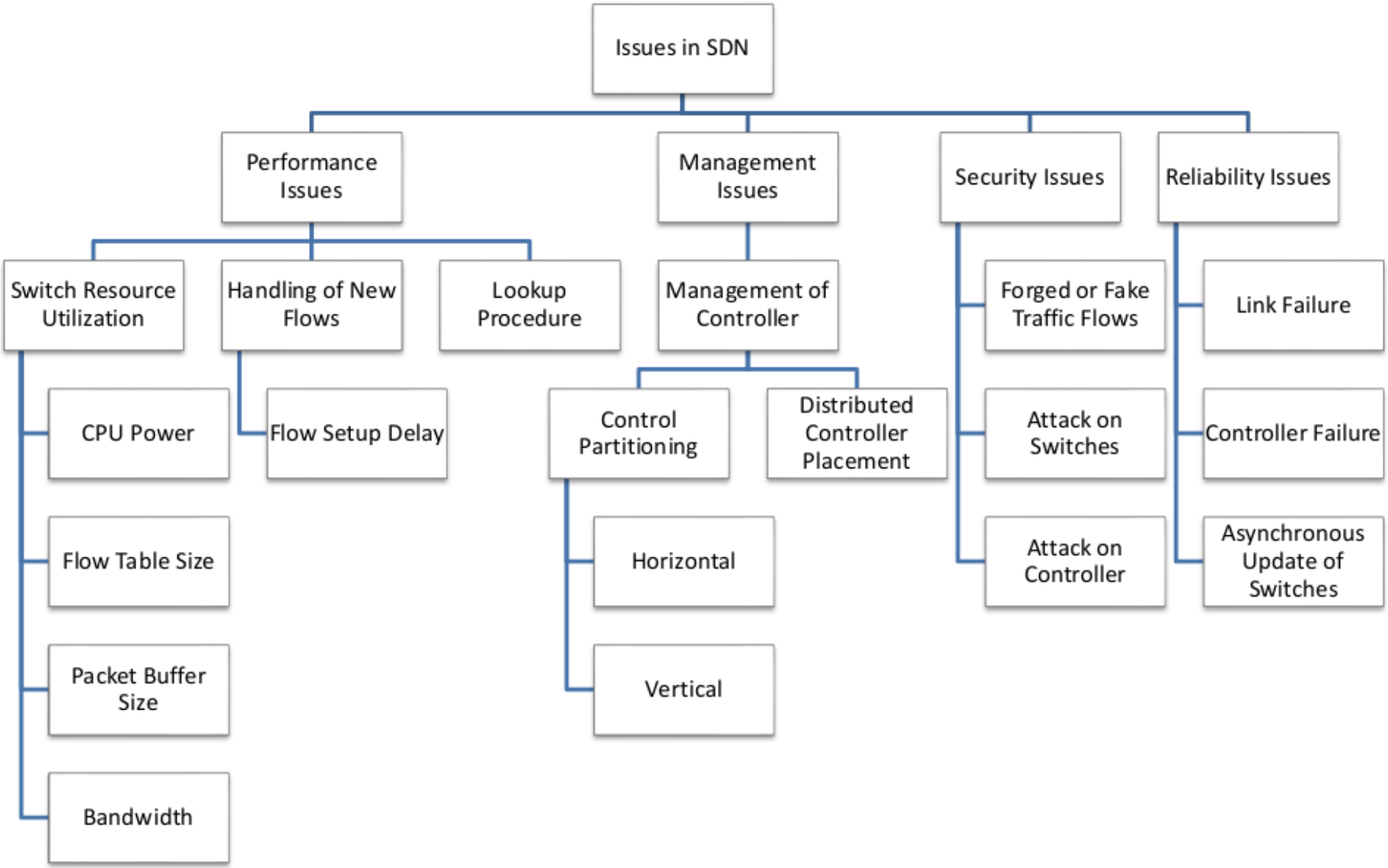
# Controller Implementation Compliant With OpenFlow

| Controller | Implementation | Open Source | Developer |
|---|---|---|---|
| POX [6] | Python | Yes | Nicira |
| NOX [7] | Python/C++ | Yes | Nicira |
| Floodlight [8] | Java | Yes | BigSwitch |
| Ryu [9] | Python | Yes | NTT, OSRG group |
| Beacon [10] | Java | Yes | Stanford |
| Trema [11] | Ruby/C | Yes | NEC |
| Maestro [12] | Java | Yes | Rice University |
| MUL [13] | C | Yes | Kulcloud |
| Jaxon [14] | Java | Yes | Independent Developers |
| Helios [15] | C | No | NEC |
| SNAC [16] | C++ | No | Nicira |
| NodeFlow [17] | JavaScript | Yes | Independent Developers |
| ovs-controller [2] | C | Yes | Independent Developers |
| Flowvisor [18] | C | Yes | Stanford/Nicira |
| RouteFlow [19] | C++ | Yes | CPqD |

# SDN Programming Languages

| **Framework** | **Level of Abstraction** | **Query Language** | **Implementation Language** | **Policies Type** |
|---|---|---|---|---|
| Frenetic [20],[21] | High | Yes | Python | Active |
| NetCore [22] | High | Yes | Python | Active |
| Nettle [23] | Low | No | Haskell | Active |
| FML [24] | High | No | Python/C++ | Passive |
| Procera [25] | High | No | Haskell | Active |

# Issues in SDN



1. CPU Power
   - Every flow is handled by the system CPU.
   - CPU is needed to encapsulate the packet to be transmitted to the Controller for flow setup through the Secure Channel.
   - Limited power of a Switch CPU can restrict the bandwidth between the Switch and the Controller.
2. Flow Tables Size

   OpenFlow Switches maintain complete visibility in a large OpenFlow network which results in increase in flow table size.

3. Packet Buffer Size
   Limited packet buffer size may lead to drop in packets and decrease in throughput.

4. Bandwidth Between Switch and Controller
   Limited bandwidth between Switch and Controller may lead to decrease in performance.

# Performance Issues

## Handling of New Flows

- Performance of Control Plane is determined by the number of new flows per second that the Controller can handle and the delay of a flow setup.
- Goals of OpenFlow was to keep the Data Plane simple and to delegate the control task to a logically centralized Controller.
- As a result, Switches consult the Controller frequently for instructions on how to handle incoming packets of new flows.
- This tends to **congest** Switch-Controller connections, which in turn adds latency to the processing of the first packets of a flow in the Switch buffer.

# Performance Issues

## Lookup Procedure

Two types of flow tables exist:

1. Hash table
   - Stored in Static RAM (SRAM) on the Switch.
   - This type of memory is **off-chip**, leading to increased lookup latencies.
2. Linear table
   - Stored on TCAM (Ternary Content Addressable Memory).
   - Located on Switch-chip, leading to decreased lookup latencies.

- In ordinary Switches, lookup mechanism is the main operation that is performed.
- In OpenFlow-enabled Switches, other operations especially the **"insert"** operation is considered that can lead to a higher power dissipation & a longer access latency.

# Management Issues

1. Distributed Controllers Placement

   Determining the number of the needed Controllers and their placement within the controlled domain is an issue.

2. Control Partitioning

   Refers to partitioning of the network into multiple controlled domains. It can be done in two ways:

   - **Horizontal**

     Multiple Controllers are organized in a flat Control Plane where each one governs a subset of the network Switches.

   - **Vertical**

     Controllers functionalities are organized vertically. Control tasks are distributed to different Controllers depending on criteria such as network view and locality requirements.

# Security Issues

1. Forged or faked traffic flows
   - To attack Switch/Controllers.
   - Can be triggered by faulty device or malicious user.
   - Aims to launch DoS/D-DoS attack against network devices.
2. Attack on Switches
   - To slow down.
   - Or to drop packets.
3. Attack on Controller
   - Inject traffic or forged requests to overload the Controller.
   - Once the Controller is attacked, all lower level Switches are misled and can't correctly deliver packets.

# Reliability Issues

1. Link Failure
   - Failure of link between Controller and Switch.
   - Failure of Switch-to-Switch path link.
2. Controller Failure
3. Asynchronous Update of Switches
   - Controller may not be able to synchronously update all the Switches.

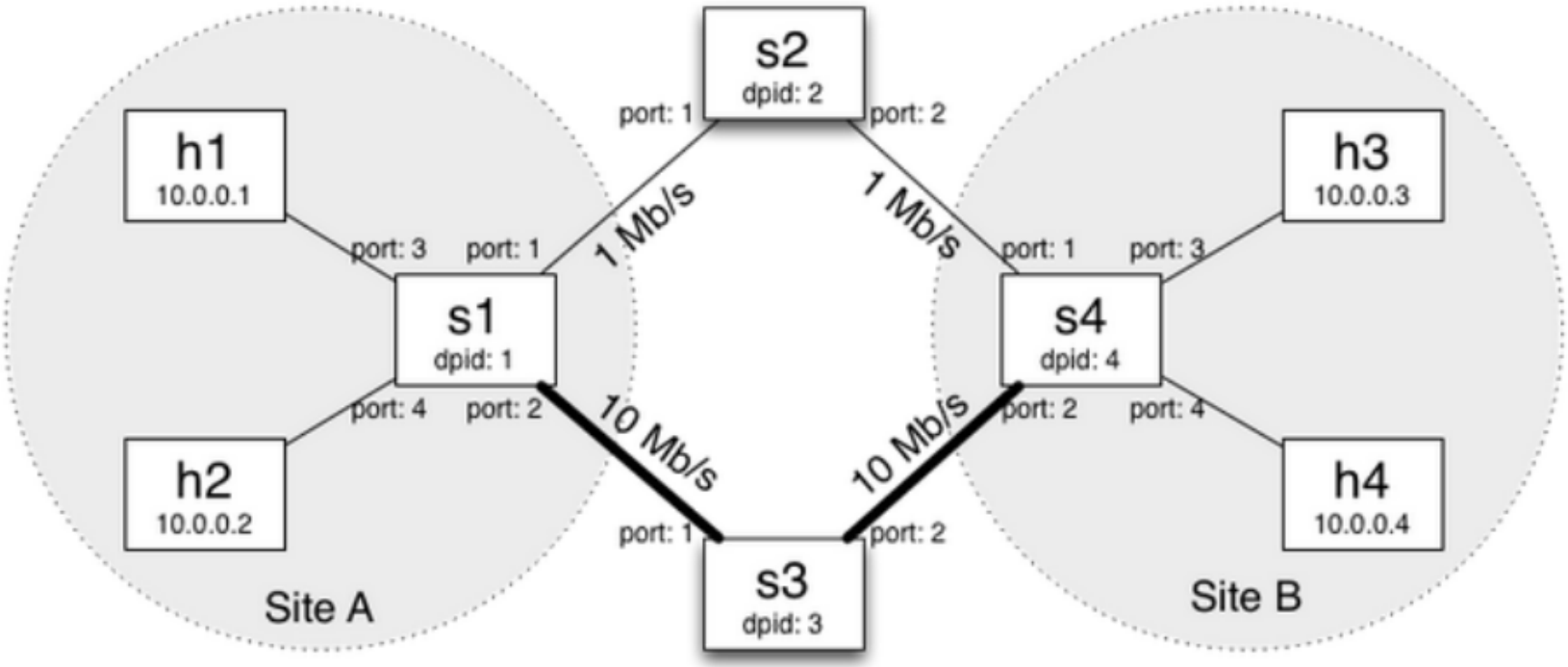
# Use Cases of SDN

## FlowVisor

- **FlowVisor** is an application Controller that sits between the physical devices and the Controllers, slices the flows (called flowspace).
- **Slicing** is to separate the flowspace in distinct subspaces.
- FlowVisor partitions the flow-table in each Switch by keeping track of which flow-entries belong to which Controller.
- Given a packet header, it can decide which flowspace contains it, and hence which slice (or slices) it belongs to.

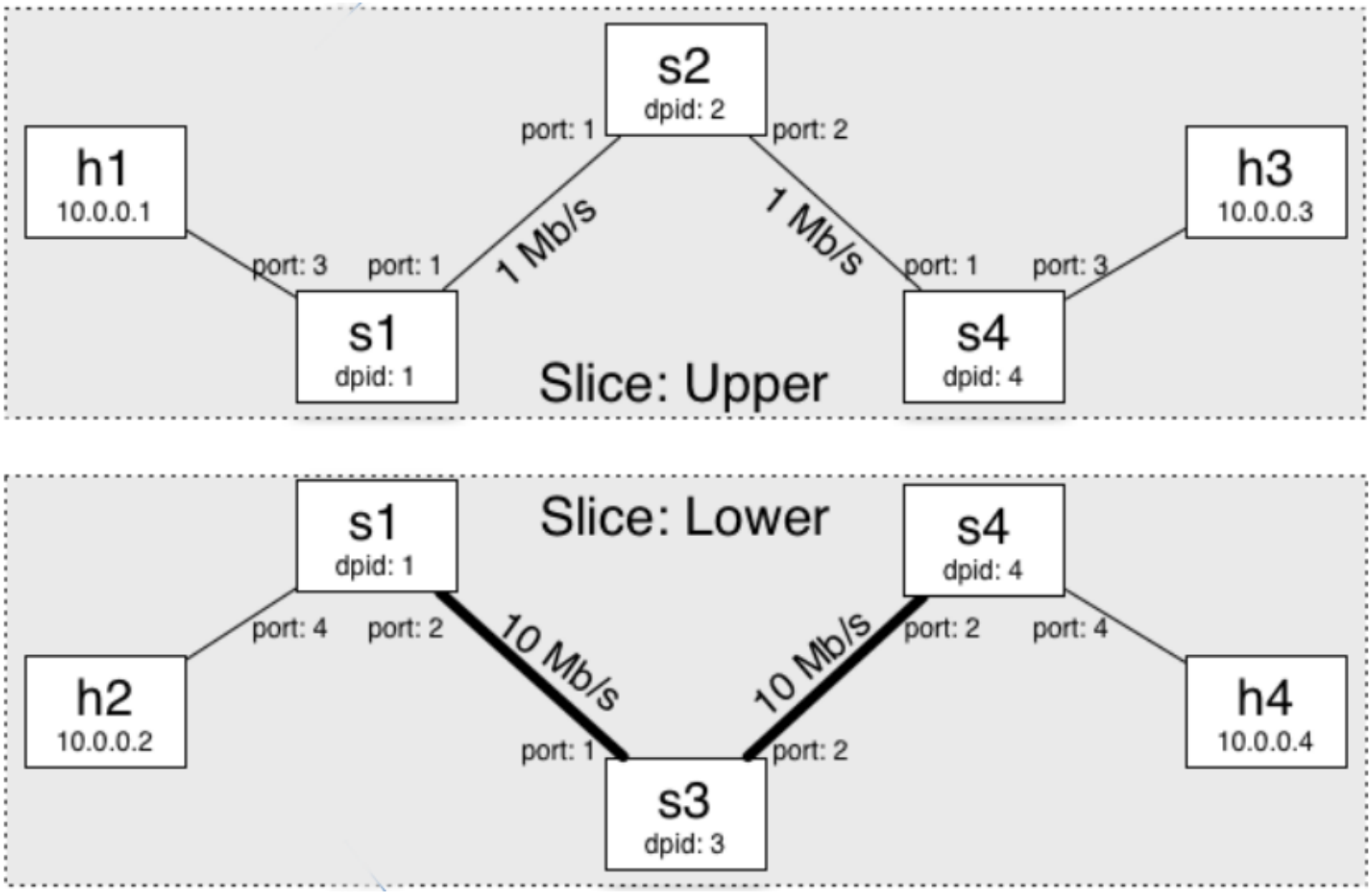
# FlowVisor

## Sample Topology



# FlowVisor

## Execution



- **Imagine a multi tenant datacenter which has multiple customers each having their applications deployed in the data center servers. Say the customers wants to run their own proprietary switching logic (Control Plane Protocols) for their respective traffic.**

- **Imagine a multi tenant datacenter which has multiple customers each having their applications deployed in the data center servers. Say the customers wants to run their own proprietary switching logic (Control Plane Protocols) for their respective traffic.**
  - With the existing network architecture there is no way to address this requirement.

- **Imagine a multi tenant datacenter which has multiple customers each having their applications deployed in the data center servers. Say the customers wants to run their own proprietary switching logic (Control Plane Protocols) for their respective traffic.**
  - With the existing network architecture there is no way to address this requirement.
  - FlowVisor solves this problem by slicing the networks based on some of the attributes either in the packet or based on the interface configurations in the OpenFlow Switches.

# References

# References


1. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. **OpenFlow: Enabling innovation in campus networks**, ACM SIGCOMM Computer Communications Review, Apr. 2008.

2. Open vSwitch & OVS-Controller.
   http://openvswitch.org/

3. Pantou: Openflow 1.0 for OpenWRT.
   http://www.openflow.org/wk/index.php/OpenFlow_1.0_for_OpenWRT

4. ofsoftswitch13 - cpqd.
   https://github.com/CPqD/ofsoftswitch13

5. Indigo: Open source openflow switches.
   http://www.projectfloodlight.org/indigotext

# References


6. POX.
   http://www.noxrepo.org/pox/about-pox/

7. N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. **NOX: towards an operating system for networks**, ACM SIGCOMM Computer Commun. Review, 38(3), pp. 105-110, 2008.

8. Floodlight, an open sdn controller.
   http://floodlight.openflowhub.org/

9. Ryu.
   http://osrg.github.com/ryu/

10. Beacon.
    https://openflow.stanford.edu/display/Beacon/Home

# References


11. Trema openflow controller framework.
https://github.com/trema/trema

12. Z. Cai, AL Cox, and TSE Ng. **Maestro: A system for scalable openflow control**, Technical Report TR10-08, Rice University, December 2010.

13. Mul.
http://sourceforge.net/p/mul/wiki/Home/

14. Jaxon:java-based openflow controller.
http://jaxon.onuos.org/

15. Helios by nec.
http://www.nec.com/

# References


16. Simple Network Access Control (SNAC). http://www.openflow.org/wp/snac/

17. The nodeflow openflow controller. http://garyberger.net/?p=537

18. R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, et al. **Carving research slices out of your production networks with openflow**, ACM SIGCOMM Computer Commun. Review, 40(1), pp. 129-130, 2010.

# References


19. Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Corr^ea, Sidney C. de Lucena, and Maur ıcio F. Magalh aes. **Virtual routers as a service: the routeflow approach leveraging software defined networks**, In Proc. 6th Int. Conf. on Future Internet Technology, CFI '11, pp. 34-37, New York, NY, USA, 2011. ACM.

20. N. Foster et al. **Frenetic: A network programming language**, SIGPLAN Not., vol. 46, no. 9, pp. 279-291, Sep. 2011.

21. N. Foster et al. **Languages for software-defined networks**, IEEE Commun. Mag., vol. 51, no. 2, pp. 128-134, Feb. 2013.

# References


22. C. Monsanto, N. Foster, R. Harrison, and D. Walker. **A compiler and run-time system for network programming languages**, in Proc. 39th Annu. ACM SIGPLAN-SIGACT Symp. POPL, pp. 217-230, 2012.

23. A. Voellmy and P. Hudak. **Nettle: Taking the sting out of programming network routers**, in Proc. 13th Intl. Conf. PADL, pp. 235-249, 2011.

24. T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. **Practical declarative network management**, in Proc. 1st ACM WREN, pp. 1-10, 2009.

25. A. Voellmy, H. Kim, and N. Feamster. **Procera: A language for high level reactive network control**, in Proc. 1st Workshop HotSDN, pp. 43-48, 2012.

# References


26. Fei Hu, Qi Hao, and Ke Bao. **A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation**, IEEE Communication Surveys & Tutorials, Vol. 16, No. 4, pp. 2181-2206, 2014.

27. Yosr Jarraya, Taous Madi, and Mourad Debbabi. **A Survey and a Layered Taxonomy of Software-Defined Networking**, IEEE Communication Surveys & Tutorials, Vol. 16, No. 4, pp. 1955-1980, 2014.

28. Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. **Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks**, IEEE Communications Surveys & Tutorials, Vol. 16 , No. 3, pp. 1617-1634, 2014.

# References


29. Nick Feamster, Jennifer Rexford, Ellen Zegura. **The Road to SDN: An Intellectual History of Programmable Networks**, ACM SIGCOMM Computer Communication Review, Vol. 44, No. 2, pp. 87-98, 2014.

30. B. Lantz, B. Heller, and N. McKeown. **A Network in a Laptop: Rapid Prototyping for Software-Defined Networks**, Proc. of ACM Hotnets'10, 2010.

31. Diego Kreutz,Fernando M. V. Ramos, Paulo Verissimo. **Towards Secure and Dependable Software-Defined Networks**, HotSDN'13, ACM, New York, USA, pp. 55-60, 2013.

32. https://www.opennetworking.org/

33. http://www.opennetsummit.org/

# Thank You!!! Enjoy Networking!!!