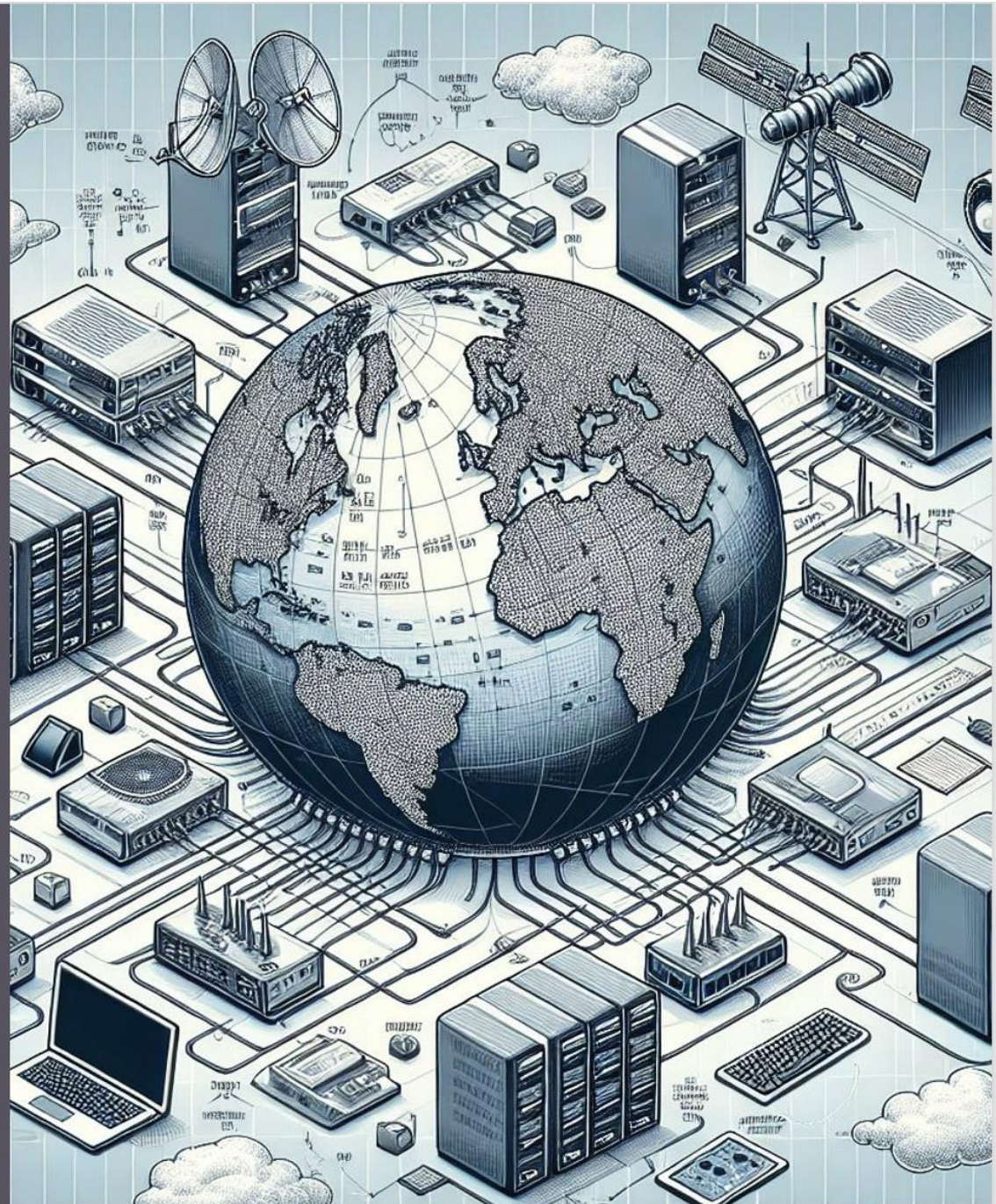
UAB COLLEGE OF ARTS AND SCIENCES
The University of Alabama at Birmingham
Department of Computer Science

CS 334/534 NETWORKING

Dr. Ragib Hasan

Lecture 3.2:
VLAN and SDN



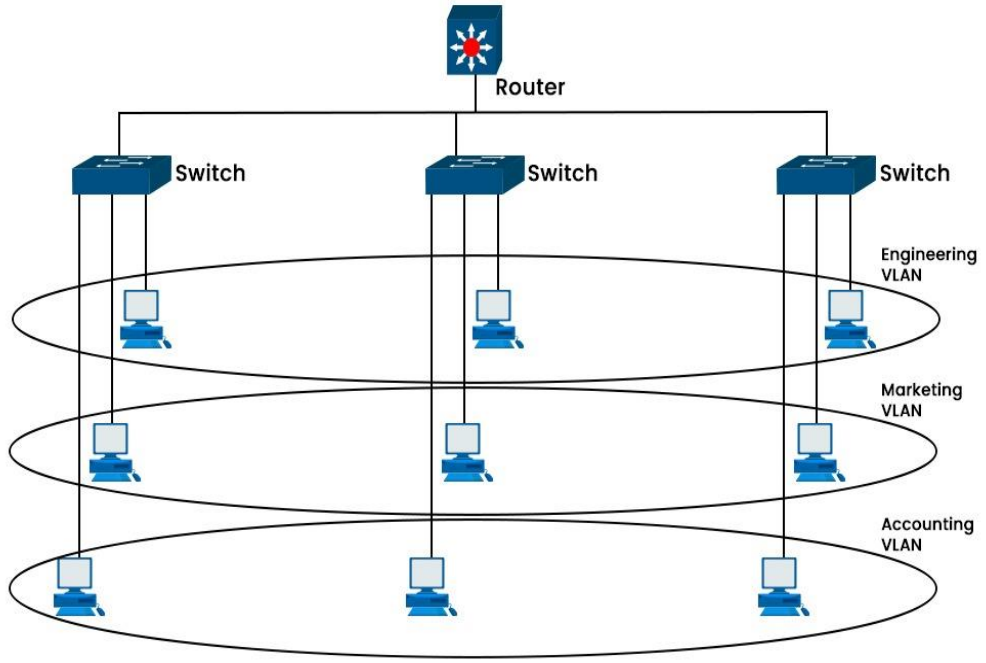
Lecture goals

- VLAN (Virtual LAN)
- SDN (Software Defined Network)
- OpenFlow Protocol
- Load Balancers

- Book reference: Chapter 3, section 3.2 to 3.4

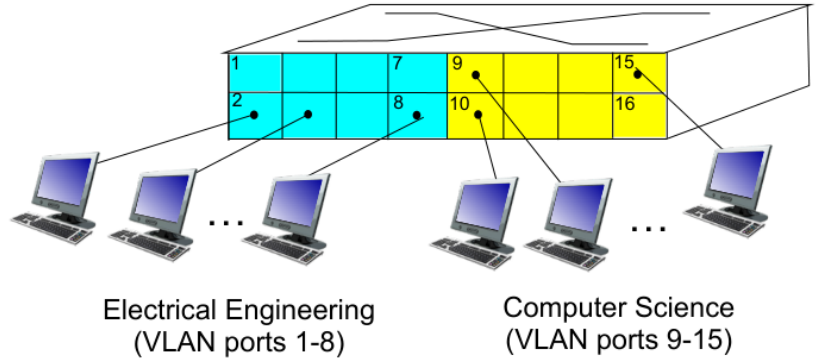
VLAN (Virtual LAN)

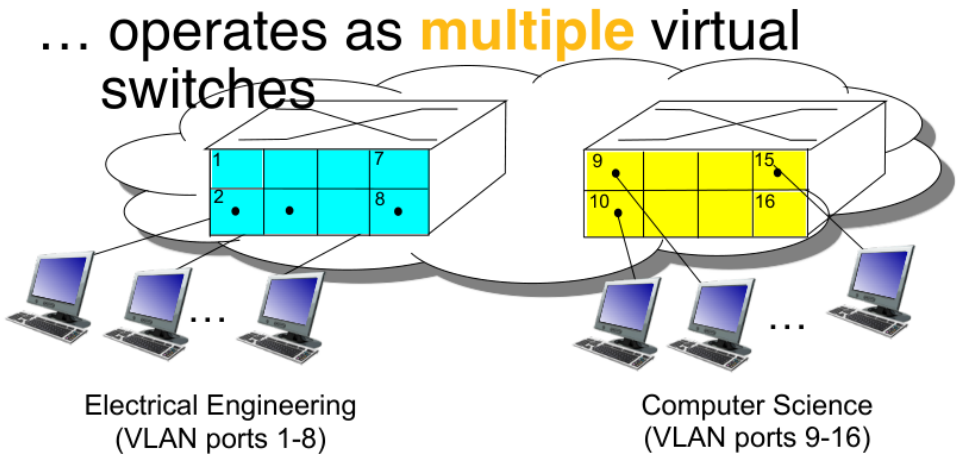
- A Virtual LAN (VLAN) is a **logical network segment** created within a physical network.
- It allows devices to be **grouped together** even if they are not connected to the same physical switch.
- Each VLAN acts as a **subgroup** of switch ports in an Ethernet LAN.
- Configured on **switches** by assigning interfaces to specific broadcast domains.



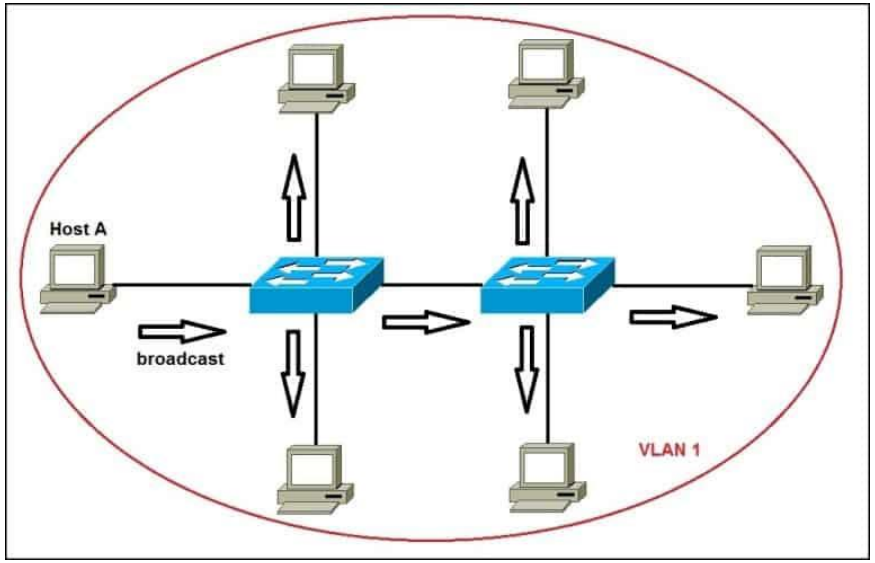
VLAN (Virtual LAN)

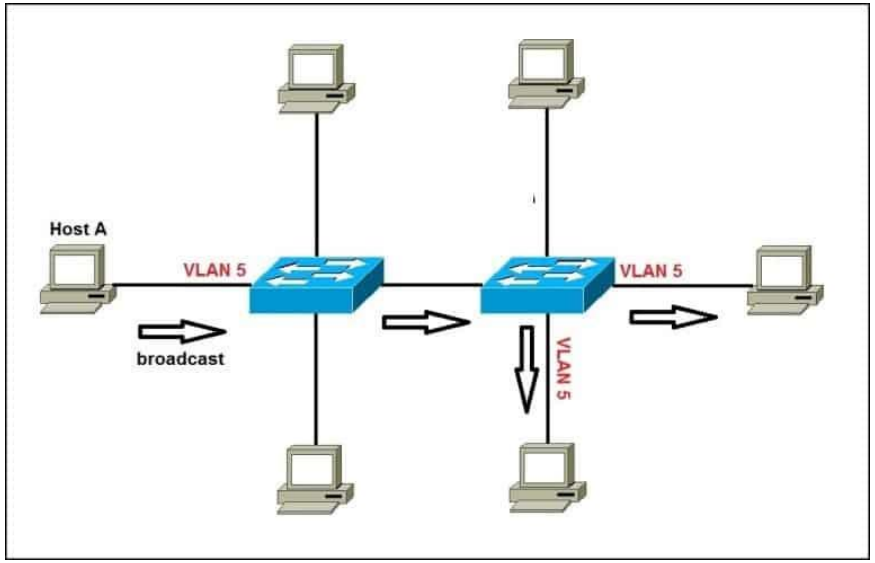
- It Increase **broadcast domains** while reducing their size.
- Enhance security:
  - Reduce the number of hosts receiving flooded frames.
  - Isolate sensitive data by placing hosts on a separate VLAN.
- Improve flexibility:
  - Group users by department instead of physical location.
- Simplify network changes:
  - Easily reconfigure ports to the appropriate VLAN.





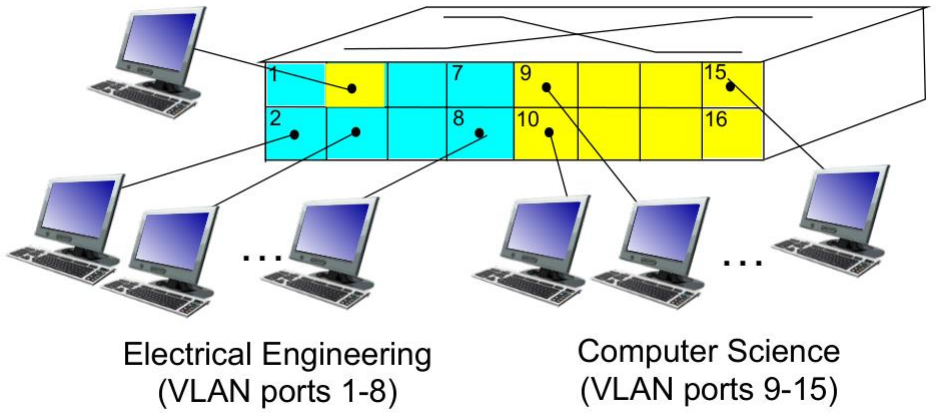
VLAN (Virtual LAN)





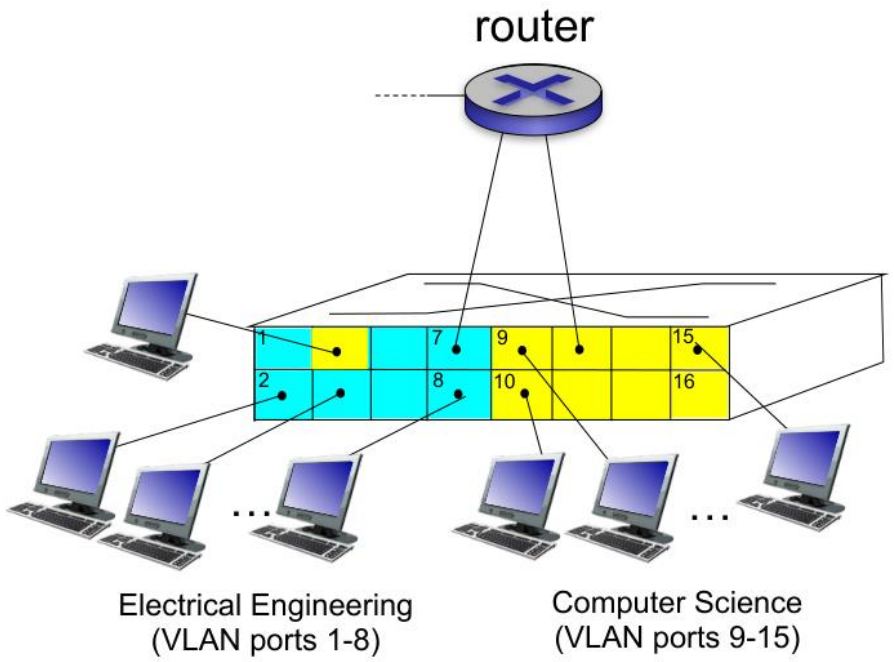
VLAN (Virtual LAN)

- **Dynamic membership:**
  - Ports can be dynamically assigned among VLANs

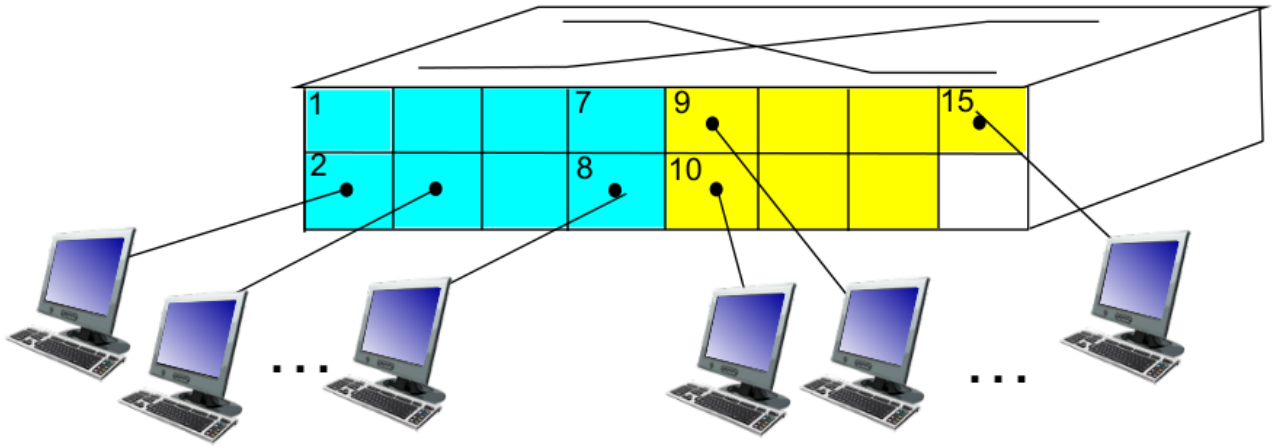


VLAN (Virtual LAN)

- **Dynamic membership:**
  - Ports can be dynamically assigned among VLANs
- **Forwarding between VLANS:**
  - Done via routing (just as with separate switches)
  - In practice vendors sell combined switches plus routers

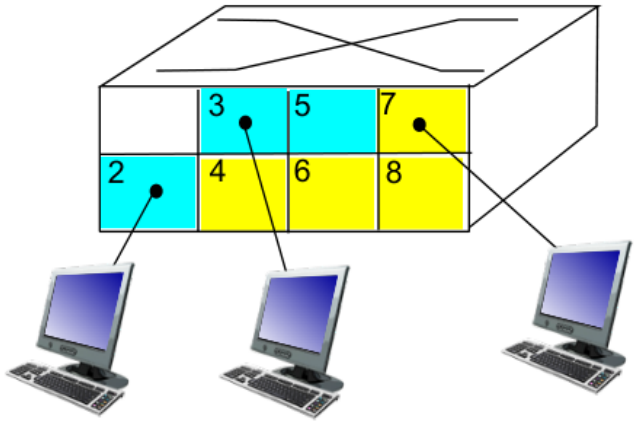


Spanning Multiple Switches



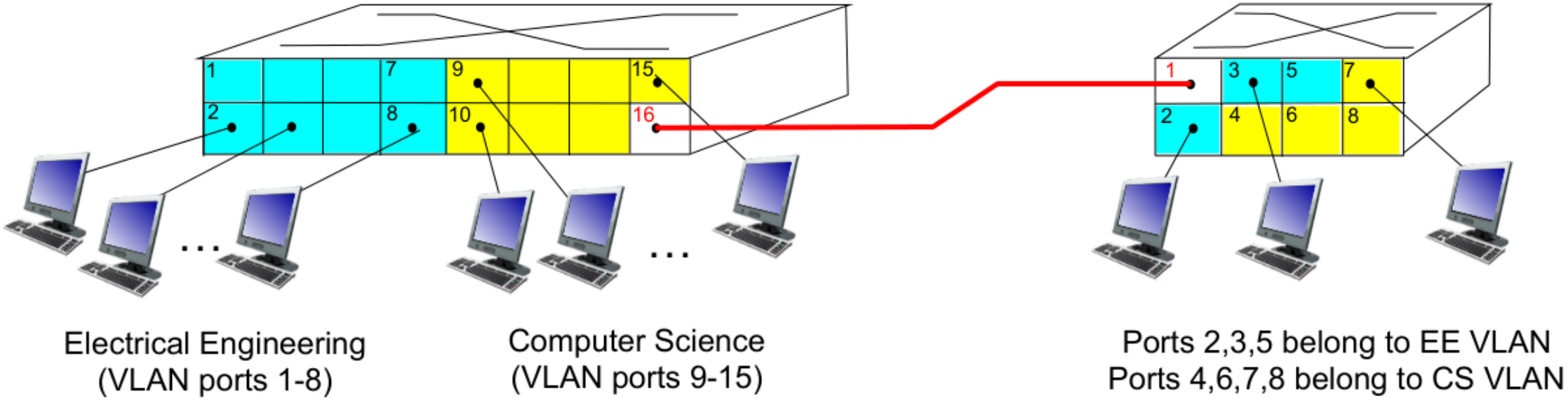
Electrical Engineering
(VLAN ports 1-8)

Computer Science
(VLAN ports 9-15)

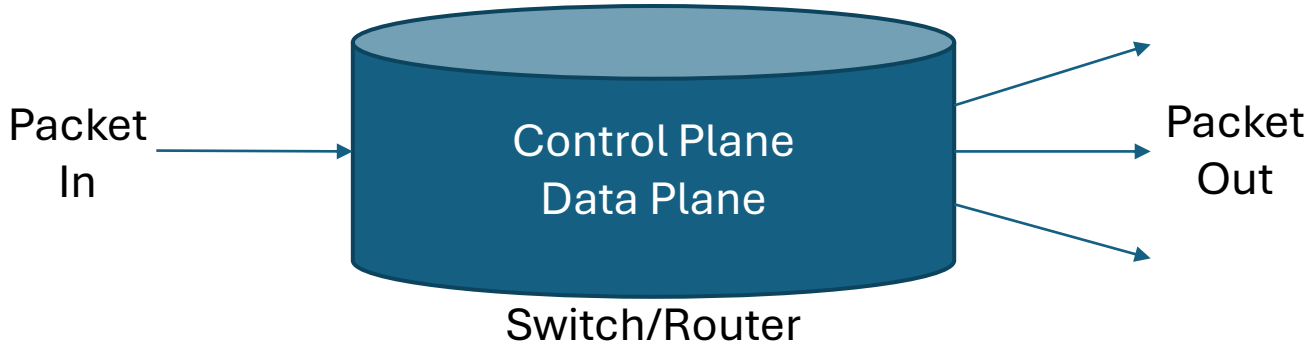


Ports 2,3,5 belong to EE VLAN
Ports 4,6,7,8 belong to CS VLAN

Spanning Multiple Switches



Functions of a switch/router



- Receive a packet and send to appropriate destination
- Prevent a packet from reaching a certain destination

Control Plane vs Data Plane

Control Plane

- Update flow table to specify where packets should go
- Update flow table to specify where packets should not go

Data Plane

- Receive a packet
- Forward packet based on flow table

SDN

- SDN is a network architecture that separates the control plane from the data plane, allowing for centralized management and programmability of network resources.

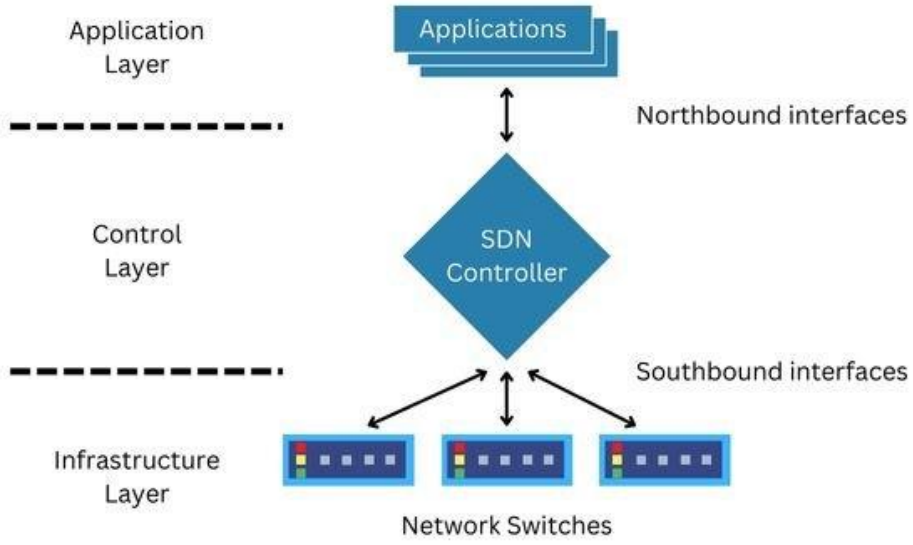
Traditional Networks	SDN-Based Networks
Control plane and data plane are integrated into network devices (routers, switches).	Control plane is centralized and separate from network devices.
Configuring devices is **manual and complex**.	Centralized control allows **automatic and dynamic** configurations.
Vendor-dependent hardware.	Open and flexible network control.

Real-Life Example:

- **Traditional Networking**: Imagine a traffic system where each traffic light works independently without knowing the status of others.
- **SDN-Based Networking**: A centralized traffic control system monitors and optimizes traffic lights dynamically, reducing congestion and improving flow.

Key Components of SDN

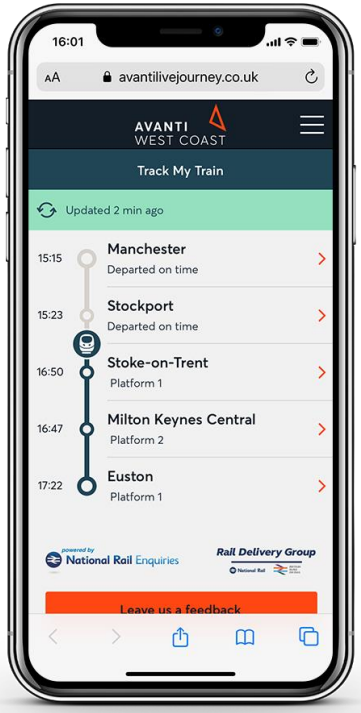
- **Application Layer**:
  - Applications that request network resources.
- **Control Layer:**
  - The "brain" of SDN, controlling network traffic centrally using software controllers.
- **Infrastructure Layer (Data Plane):**
  - Physical and virtual switches/routers that forward data based on control plane instructions.



SDN Example

Smart railway system:

- **Application Layer:** Passengers (apps) request train services.
- **Control Layer:** The train control center directs train movements.
- **Infrastructure Layer**: The trains (data plane) follow the control center's instructions.



Benefits of SDN

- **Centralized Control & Automation –** No need to configure every switch manually.
- **Scalability & Flexibility –** Easy to add new devices without complex configurations.
- **Cost-Effective –** Uses open-source controllers instead of expensive vendor hardware.
- **Security & Traffic Optimization –** Network policies can be enforced centrally.
- Cloud service providers like **Google, AWS, and Microsoft Azure** use SDN to dynamically **allocate network resources**.

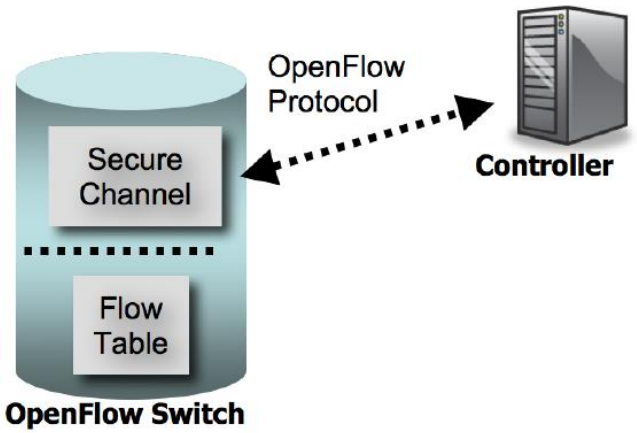
OpenFlow - The Foundation of SDN

OpenFlow is the first SDN standard protocol that **allows an SDN controller to communicate with network devices** (switches/routers).

- How OpenFlow Works:
  - Packets arrive at a switch.
  - The switch asks the SDN controller for instructions.
  - The controller decides how to handle the packet.
  - The switch follows the decision and forwards the packet accordingly.

OpenFlow Architecture

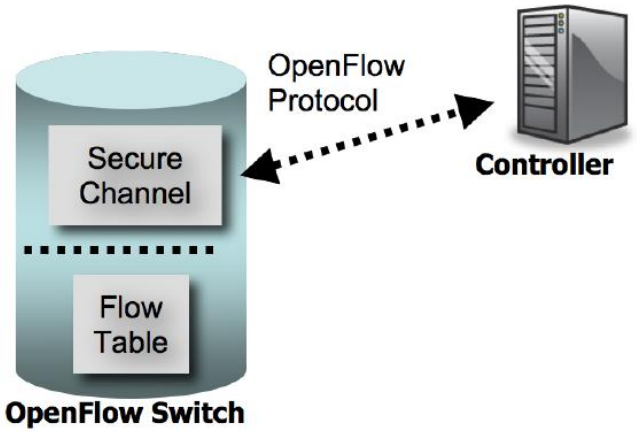
- **SDN Controller**: Acts as the central control unit, instructing switches on how to handle traffic.
- **OpenFlow Switch**: Forwards traffic based on instructions from the controller.
- **Secure Channel:** Ensures secure communication between the controller and switches.



OpenFlow Example

University Campus: block Netflix during exams

- The SDN Controller decides a new rule (e.g., "Block Netflix during exams").
- It sends this rule over a secure channel to the OpenFlow switches.
- Switches update their flow tables with the new rule.
- When a Netflix packet arrives, the switch checks its flow table, sees the "drop" action, and blocks it.



Learning Switches in OpenFlow

Basic OpenFlow Learning Process

- Stores **<destination, source>** pair
- Strictly match **<destination, source>** pair to follow actions

Learning Switches in OpenFlow

Basic OpenFlow Learning Process

- Stores **<destination, source>** pair
- Strictly match **<destination, source>** pair to follow actions
- Initially flow table is empty

Match	Action

Learning Switches in OpenFlow

Basic OpenFlow Learning Process

- Stores **<destination, source>** pair
- Strictly match **<destination, source>** pair to follow actions
- Initially flow table is empty
- **A** sends a packet to **B**

Match	Action

Learning Switches in OpenFlow

Match	Action

Basic OpenFlow Learning Process

- Initially flow table is empty
- **A** sends a packet to **B**
- **S reports the packet to the controller**, which floods it and records that **A is reachable via port 1**.

Learning Switches in OpenFlow

Basic OpenFlow Learning Process

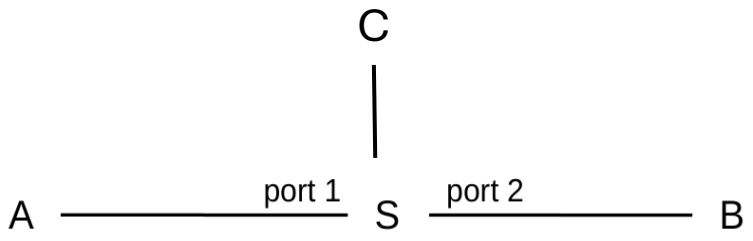
- Initially flow table is empty
- **A** sends a packet to **B**
- **S reports the packet to the controller**, which floods it and records that **A is reachable via port 1**.
- When **B responds**, the controller learns that **B is reachable via port 2** and installs two flow-table entries:
  - dst=B, src=A → forward to port 2
  - dst=A, src=B → forward to port 1.

Match	Action
dst=B, src=A	forward to port 2
dst=A, src=B	forward to port 1

Learning Switches in OpenFlow

Basic OpenFlow Learning Process

- Initially flow table is empty
- **A** sends a packet to **B**
- **S reports the packet to the controller**, which floods it and records that **A is reachable via port 1**.
- When **B responds**, the controller learns that **B is reachable via port 2** and installs two flow-table entries:
  - dst=B, src=A → forward to port 2
  - dst=A, src=B → forward to port 1
- **Problem:** If a third host C sends a packet to B, it does not match the existing rules and is sent to the controller for processing.



Match	Action
dst=B, src=A	forward to port 2
dst=A, src=B	forward to port 1

Learning Switches in OpenFlow

Optimized Approach Using Multiple Flow Tables

- Instead of storing pair-based rules, a multi-table approach is used:
  - **Table 0 (T0):** Matches destination address.
  - **Table 1 (T1):** Matches source address.
- Initial Rules Installed:
  - **T0**: No match → Flood packet, send to T1.
  - **T1**: No match → Send to controller.

T0: Match DST	Action
No Match	Flood packet, send to T1.

T1: Match SRC	Action
No Match	Send to controller.

Learning Switches in OpenFlow

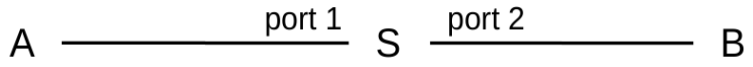
Optimized Approach Using Multiple Flow Tables

- When A sends to B

T0: Match DST	Action
No Match	Flood packet, send to T1.

T1: Match SRC	Action
No Match	Send to controller.

Learning Switches in OpenFlow



Optimized Approach Using Multiple Flow Tables

- When A sends to B
  - **Check B in T0**
    - No Match: Flood packet and check in T1

B →

T0: Match DST	Action
No Match	Flood packet, send to T1.

T1: Match SRC	Action
No Match	Send to controller.

Learning Switches in OpenFlow

Optimized Approach Using Multiple Flow Tables

- When A sends to B
  - **Check B in T0**
    - No Match: Flood packet and check in T1
  - **Check A in T1**
    - No Match: Send to controller

T0: Match DST	Action
No Match	Flood packet, send to T1.

A →

T1: Match SRC	Action
No Match	Send to controller.

Learning Switches in OpenFlow

Optimized Approach Using Multiple Flow Tables

- When A sends to B, the packet is flooded and sent to the controller, which installs:
  - **T0**: match dst=A → forward via port 1, send to T1
  - **T1**: match src=A → do nothing

T0: Match DST	Action
Dst = A	Forward via port 1, send to T1

T1: Match SRC	Action
Src = A	Do nothing

Learning Switches in OpenFlow

Optimized Approach Using Multiple Flow Tables

- When B responds to A:
  - **Check A in T0**
    - Match: Forward via port 1, and check in T1

A →

T0: Match DST	Action
Dst = A	Forward via port 1, send to T1

T1: Match SRC	Action
Src = A	Do nothing

Learning Switches in OpenFlow

Optimized Approach Using Multiple Flow Tables

- When B responds to A:
  - **Check A in T0**
    - Match: Forward via port 1, and check in T1
  - **Check B in T1**
    - No Match: Send to controller

T0: Match DST	Action
Dst = A	Forward via port 1, send to T1

B →

T1: Match SRC	Action
Src = A	Do nothing

Learning Switches in OpenFlow

Optimized Approach Using Multiple Flow Tables

- When B responds to A, S forward to A via port 1, and the controller installs:
  - **T0**: match dst=B → forward via port 2, send to T1
  - **T1**: match src=B → do nothing
- This ensures that once a communication path is learned, the **controller is not involved in further packet forwarding**.

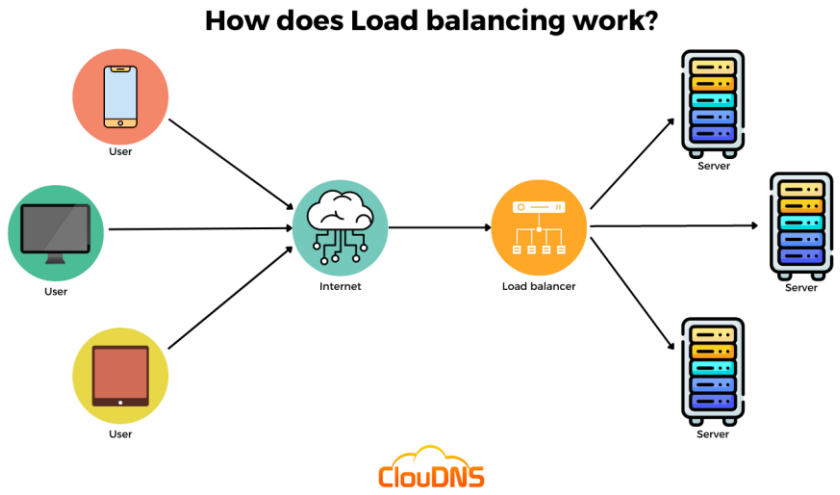
T0: Match DST	Action
Dst = A	forward via port 1, send to T1
Dst = B	forward via port 2, send to T1

T1: Match SRC	Action
Src = A	Do nothing
Src = B	Do nothing

Load Balancer

A load balancer is a network component that distributes incoming traffic across multiple servers to **ensure high availability and prevent overload**.

- How Load Balancers Work
  - **Client Requests:** A user accesses a website.
  - **Load Balancer Checks:** It decides which server should handle the request.
  - **Traffic is Forwarded:** The best server processes the request.



SDN + Load Balancers in Action

- **Problem**: A video conferencing app (e.g., Zoom) faces traffic spikes.
- **Solution**:
  - SDN Controller **detects congestion**.
  - OpenFlow **reroutes traffic** to idle servers.
  - Load Balancer **distributes users** evenly.
- **Result**: No lag during meetings!