

Advanced Computer Networks

263-3501-00

RDMA, Network Virtualization

Patrick Stuedi

Spring Semester 2013

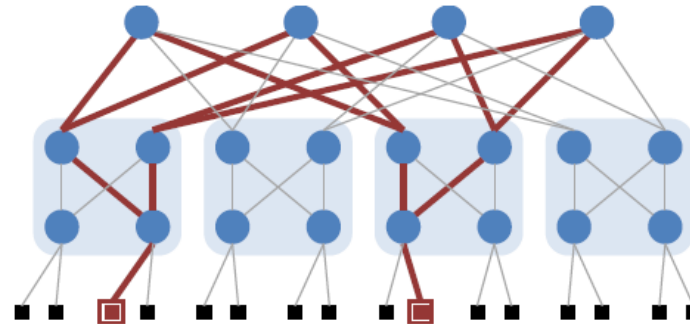
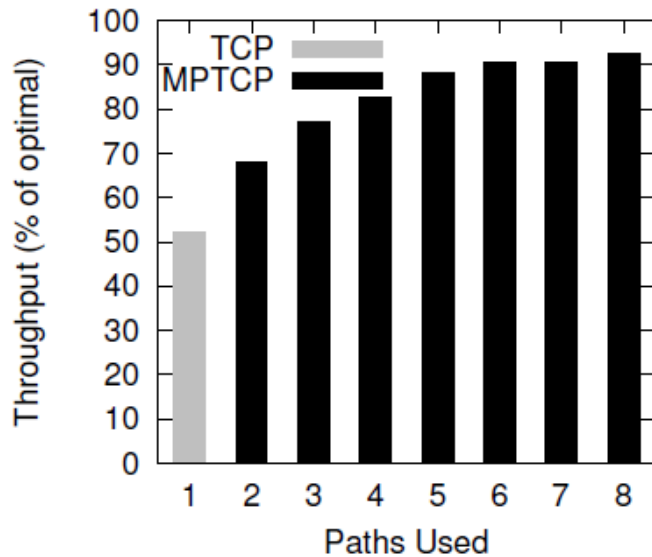
Last Week

- Scaling Layer 2
 - Portland
 - VL2
- TCP
 - Incast
 - Data Center TCP (DCTCP)
 - Multipath TCP

Today

- TCP Offloading
- Remote Direct Memory Access
- Network Virtualization
 - Software Defined Networking
 - OpenFlow

Multipath TCP



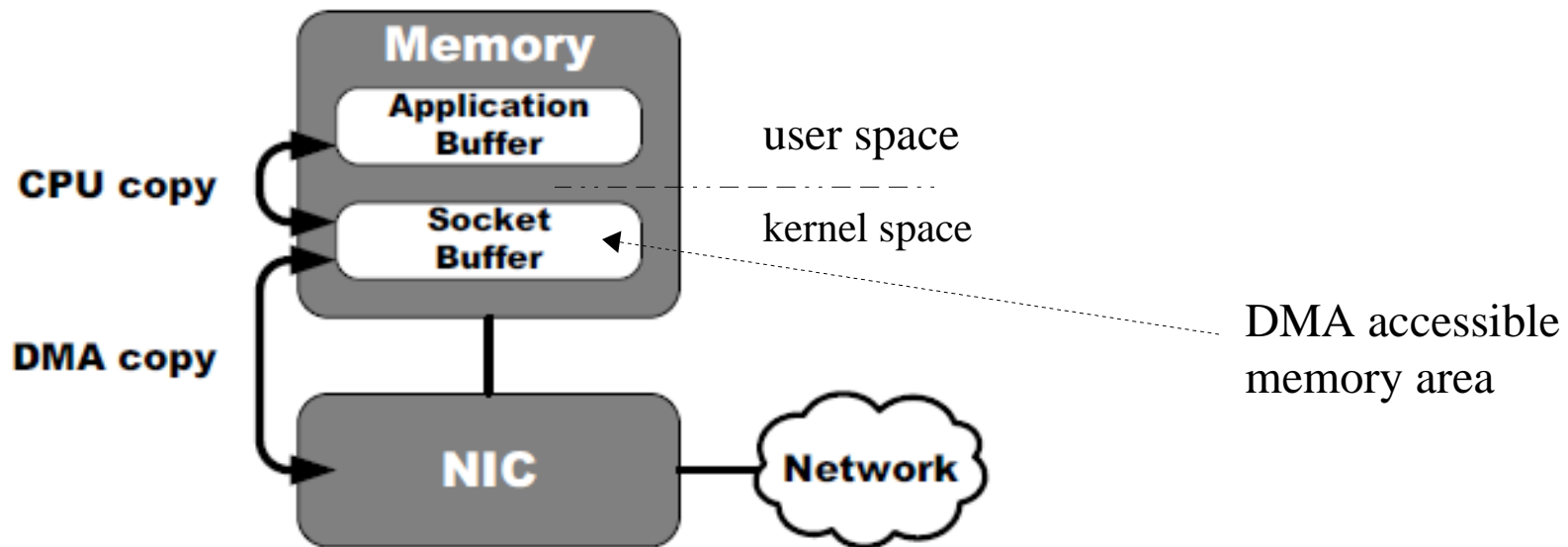
- Benchmark:
 - Partition the network into two parts, src and dst
 - For each host in src connect to host in dst, avoid duplicates
- Regular TCP with ECMP flow hashing
 - Static hashing, creates hotspots (certain switches overly loaded)
- Multipath TCP with ECMP subflow hashing
 - Packets always travel on the least congested subflow

Network latencies in Data centers

Component	Delay	Round-Trip
Network Switch	10-30 μ s	100-300 μ s
Network Interface Card	2.5-32 μ s	10-128 μ s
OS Network Stack	15 μ s	60 μ s
Speed of Light (in Fiber)	5ns/m	0.6-1.2 μ s

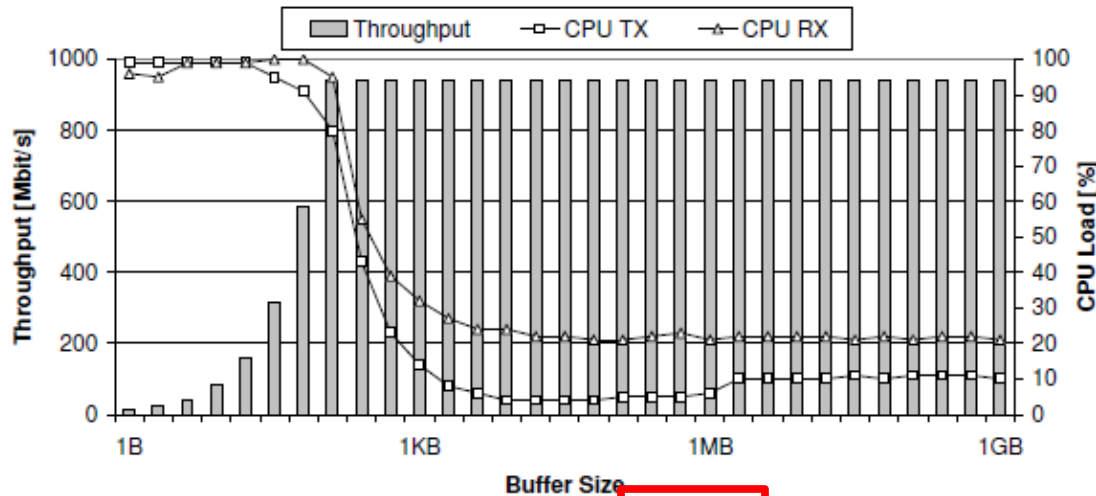
- Factors that contribute to latency in TCP datacenters
 - Delay: cost of a single traversal of the component
 - RTT: total cost in a round-trip traversing 5 switches in each direction
- OS overhead per packet exchanged between two hosts attached to the same switch: $(2 \cdot 15) / (2 \cdot 2.5 + 2 \cdot 15 + 10) = 66\%$ (!!)

Packet Processing Overhead

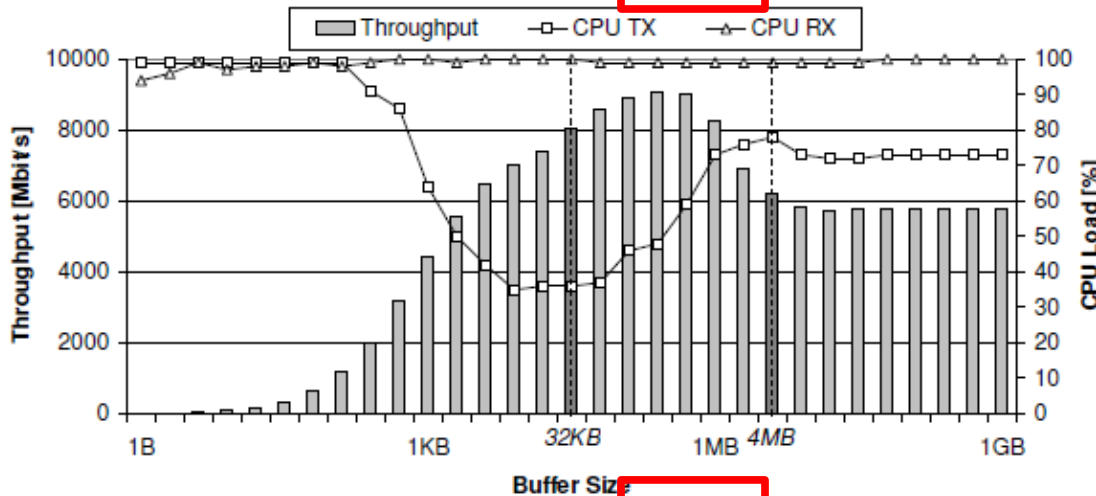


- Sending-side:
 - Data is copied from the application buffer into a socket buffer
 - Data is DMA copied into NIC buffer
- Receiver side:
 - Data is DMA copied from NIC buffer into socket buffer
 - Data is copied into application buffer
 - Application is scheduled (context switching)

Throughput and CPU load at 1Gbit/s and 10Gbit/s



(a) TCP bulk data transfer on 1 Gbps Ethernet.



(b) TCP bulk data transfer on 10 Gbps Ethernet.

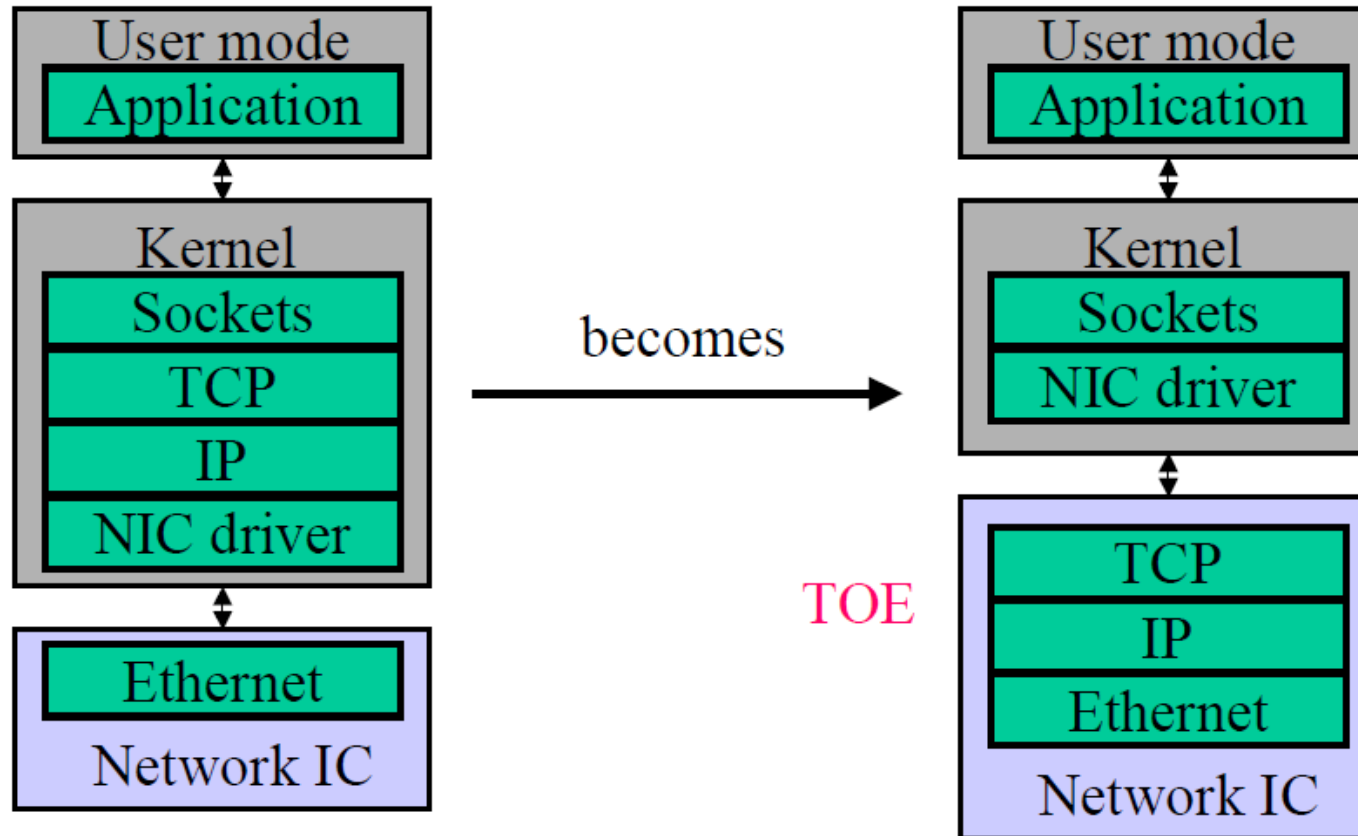
- Throughput limited because of high CPU load

- RX side typically more CPU intensive because highly asynchronous

TCP Offloading

- What is TCP offloading
 - Moving IP and TCP processing to the Network Interface (NIC)
- Main justification for TCP offloading
 - Reduction of host CPU cycles for protocol header processing, checksumming
 - Fewer CPU interrupts
 - Fewer bytes copied over the memory bus
 - Potential to offload expensive features such as encryption

TCP Offload Engines (TOEs)



Problems of TCP offloading

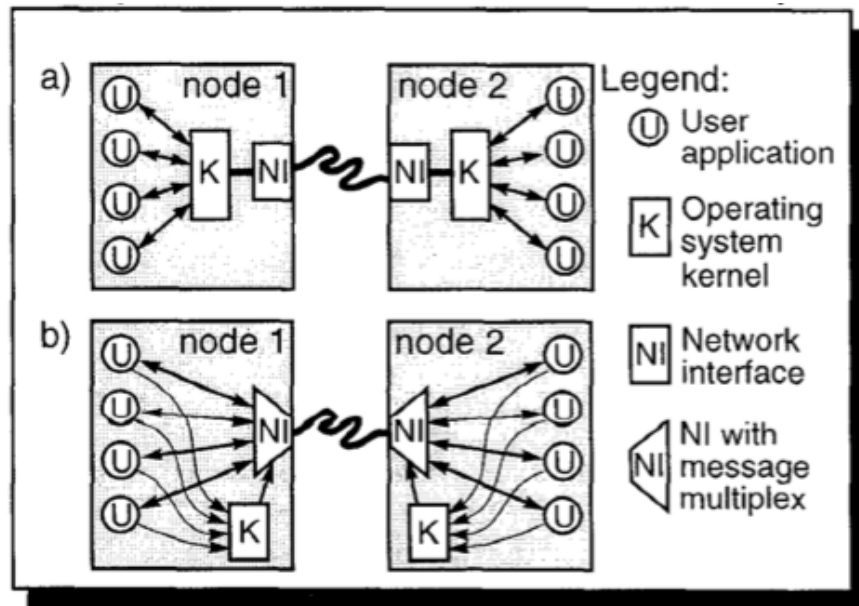
- Moore's Law worked against "smart" NICs
 - CPU's used to get faster
- Now many cores, cores don't get faster
 - Network processing is hard to parallelize
- TCP/IP headers don't take many CPU cycles
- TOEs impose complex interfaces
 - Protocol between TOE & CPU can be worse than TCP
- Connection management overhead
 - For short connections, overwhelms any savings

Where TCP offload helps

- Sweet spot for TCP offload might be apps with:
 - Very high bandwidth
 - Relatively low end-to-end latency network paths
 - Long connection durations
 - Relatively few connections
- Typical examples of these might be:
 - Storage-server access
 - Cluster interconnects

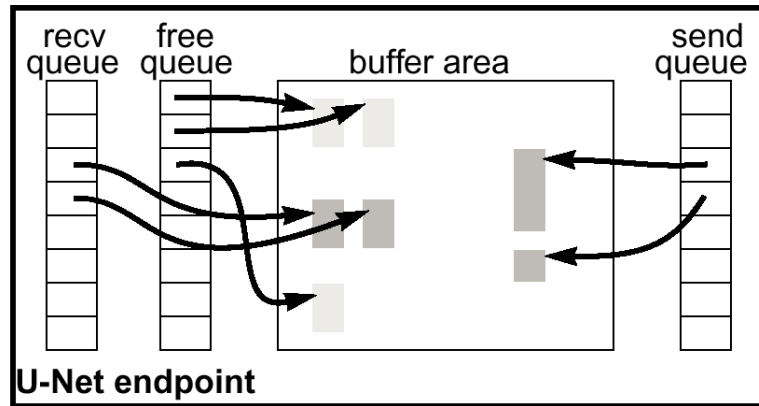
User-level networking: Remove OS from the data path

- Transport offloading is not enough!
 - Still have system call overhead, context switch, memory copying
- U-Net:
 - Eicken, Basu, Buch, Vogels, Cornell University, 1995
 - Virtual network interface that allows applications to send and receive messages without operating system intervention
 - Move all buffer management and packet processing to user-space (zero-copy)



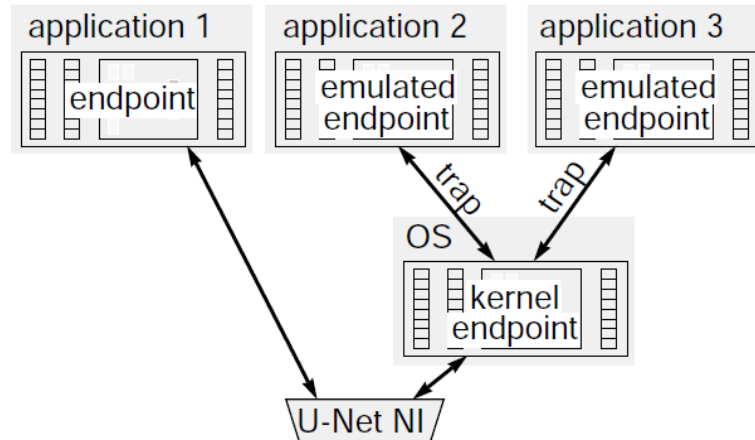
- traditional networking architecture
 - Kernel controls the network
 - All communication via kernel
- U-Net architecture:
 - Application access network directly via MUX
 - Kernel involved only in connection setup

U-Net Building Blocks



- **End points**
 - application's handle into the network
- **Buffer area**
 - hold message data for sending or buffer space for receiving
- **Message queues**
 - hold **descriptors** pointing to buffer area

U-Net communication

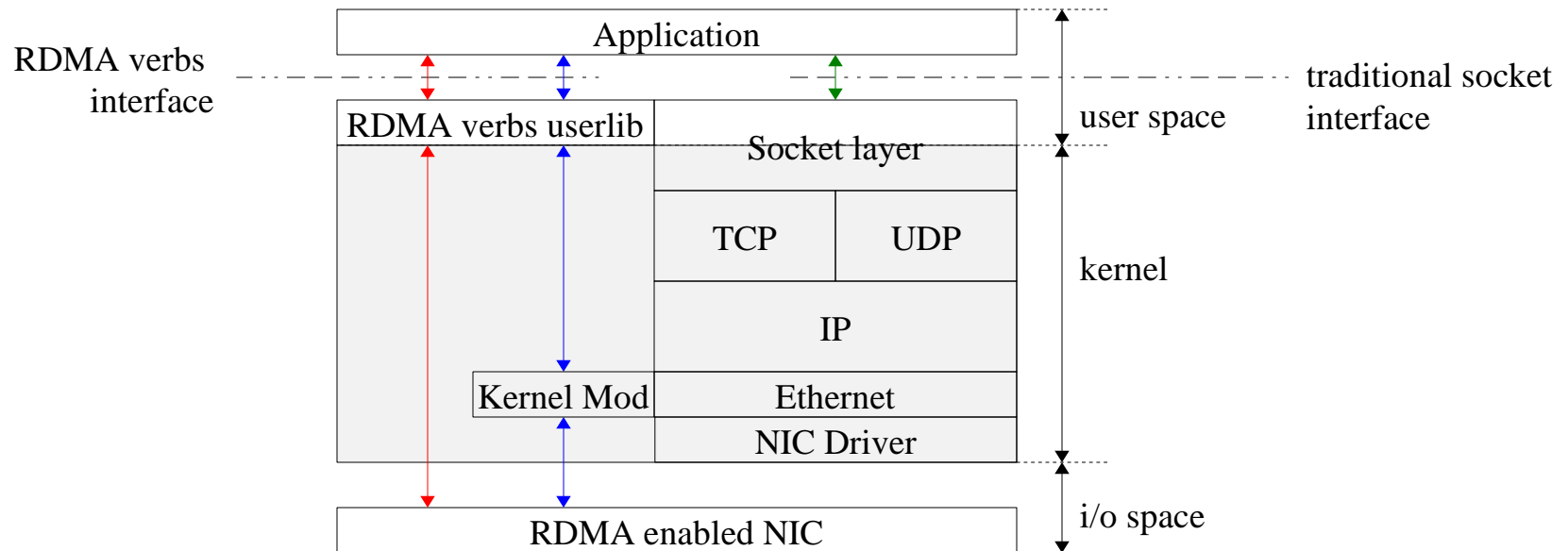


- Initialization:
 - Create one or more endpoints
 - Register user buffers with endpoints and associated them with a **tag**
- Sending
 - Composes the data in the endpoint buffer area
 - Push a descriptor for the message onto the send queue
 - NIC transmits the message after marking it with the appropriate message tag.
- Receiving:
 - Push a message descriptor with pointers to the buffers onto the receive queue.
 - Incoming messages get de-multiplexed based on the message **tag**
 - Data is placed within the target buffer of the application by the NIC

History of User-Level Networking

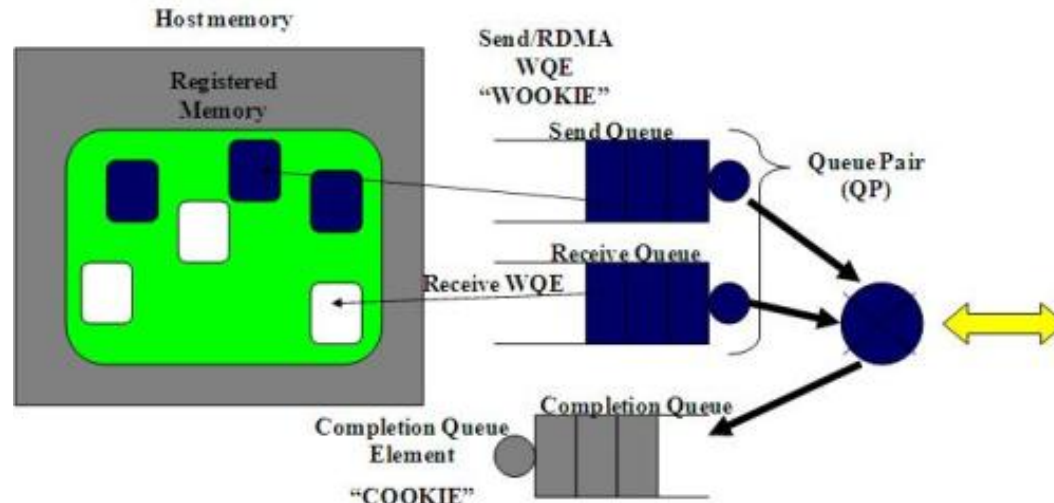
- U-Net one of the first (if not the first) system to propose OS-bypassing
- Other early works
 - SHRIMP: Virtual Memory Mapped Interfaces, IEEE Micro, 1995
 - “Separating Data and Control Transfer in Distributed Operating Systems”, Thekkath et. al., ASPLOS'94
- Efforts of U-Net eventually resulted in the *Virtual Interface Architecture* (VIA)
 - Specification jointly proposed by Compaq, Intel and Microsoft, 1997
- VIA architecture has led to the implementation of various high performance networking stacks: Infiniband, iWARP, Roce:
 - Commonly referred to as RDMA network stacks
 - RDMA = Remote Direct Memory Access

RDMA Architecture



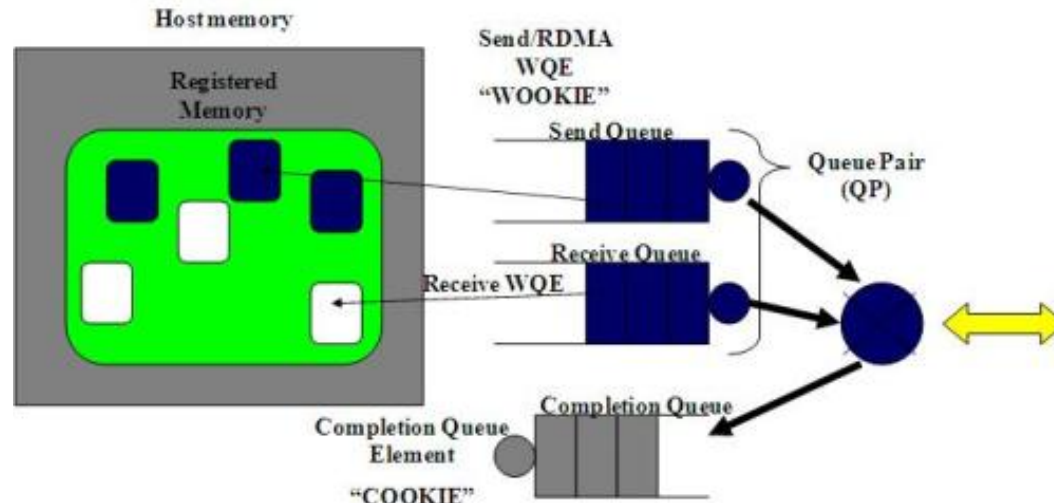
- **Traditional socket interface** involves kernel
- RDMA interface involves kernel only on **control path**, but access the RDMA capable NIC (rNIC) directly from user space on the **data path**
- Dedicated **verbs interface** used for RDMA, instead of traditional socket interface

RDMA Queue Pairs (QPs)



- Applications use 'verbs' interface to
 - Register memory:
 - Operating system will make sure the memory is pinned and accessible by DMA
 - Create a queue pair (QP)
 - send/recv queue
 - Create a completion queue (CQ)
 - RNIC puts a new completion-queue element into the CQ after an operation has completed
 - Send/Receive data
 - Place a work-request element (WQE) into the send or recv queue
 - WQE points to user buffer and defines the type of the operation (e.g., send, recv, ..)

RDMA Queue Pairs (QPs)



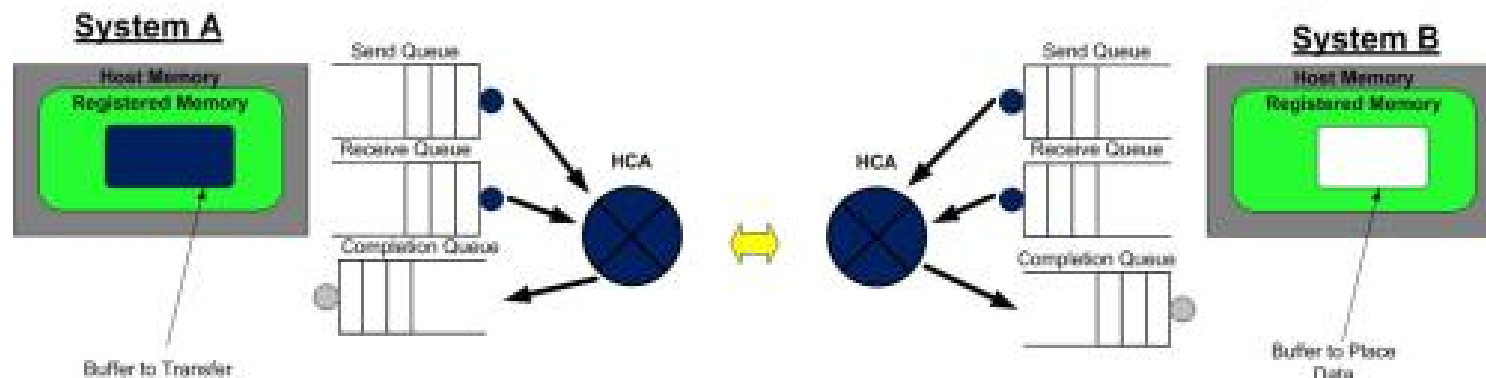
- Applications use 'verbs' interface to
 - Register memory:
 - Operating system will make sure the memory is pinned and a
 - Create a queue pair (QP)
 - send/rcv queue
 - Create a completion queue (CQ)
 - RNIC puts a new completion-queue element into the CQ after an operation has completed
 - Send/Receive data
 - Place a work-request element (WQE) into the send or rcv queue
 - WQE points to user buffer and defines the type of the operation (e.g., send, rcv, ..)

This is much like
U-NET

RDMA operations

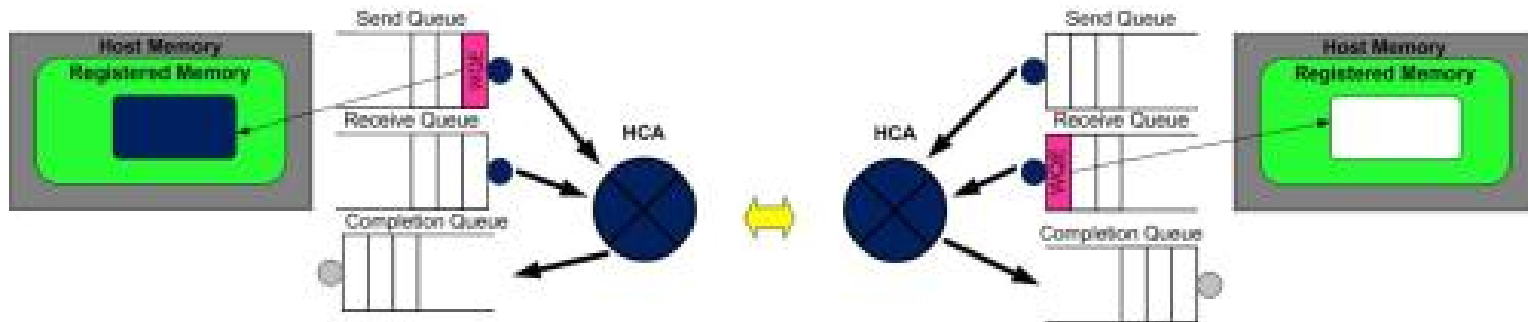
- Send/Receive:
 - Two-sided operation: data exchange naturally involves both ends of the communication channel
 - Each send operation must have a matching receive operation
 - Send WR specifies where the data should be taken from
 - Receive WR on the remote machine specifies where the inbound data is to be placed
- RDMA (Remote Direct Memory Access)
 - Two independent operations: RDMA Read and RDMA Write
 - Only the application issuing the operation is actively involved in the data transfer
 - An RDMA Write not only specifies where the data should be taken from, but also where it is to be placed (remotely)
 - An RDMA Read requires a buffer advertisement prior to data exchange

Example: RDMA Send/Recv (1)



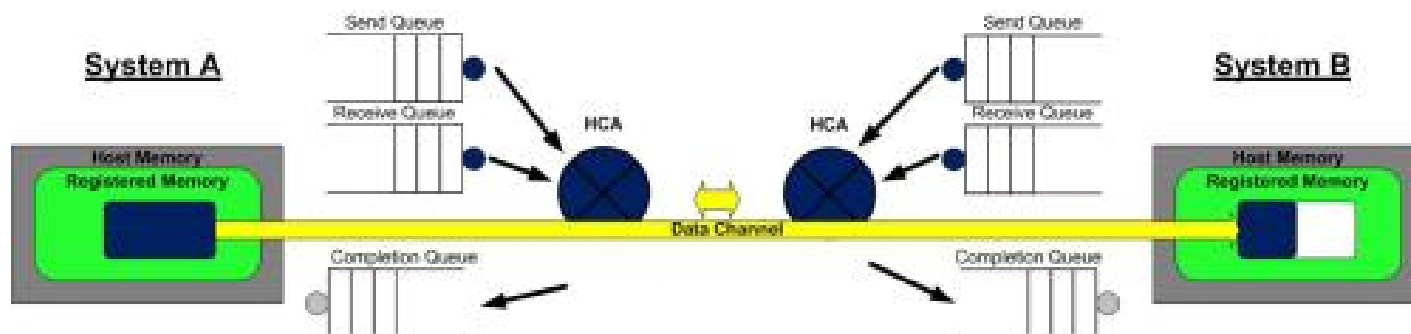
- Sender and receiver have created their Qps and Cqs
- Sender has registered a buffer for sending
- Receiver has registered a buffer for receiving

Example: RDMA Send/Recv (2)



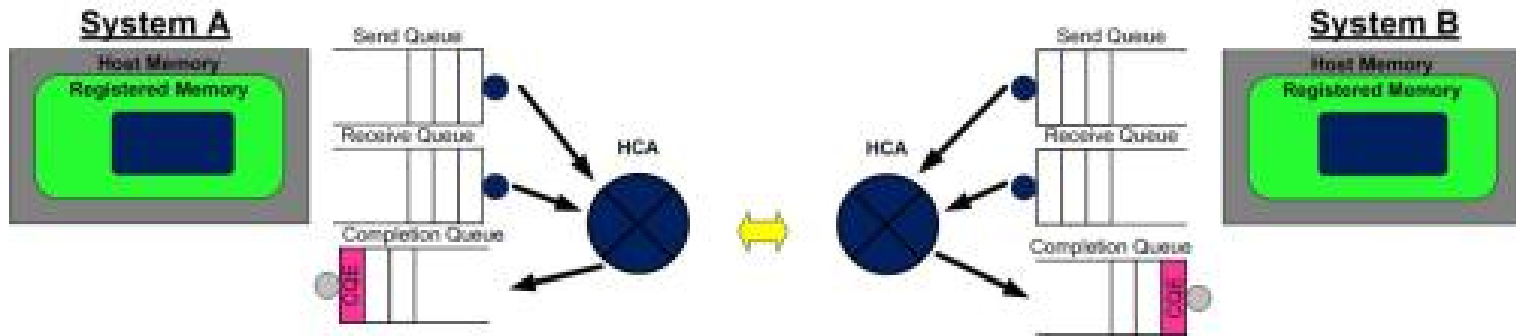
- Receiver places a WQE into its receive queue
- Sender places a WQE into its send queue

Example: RDMA Send/Recv (3)



- Data is transferred between the hosts
 - Involves two DMA transfers, one at the sender and one at the receiver

Example: RDMA Send/Recv (4)

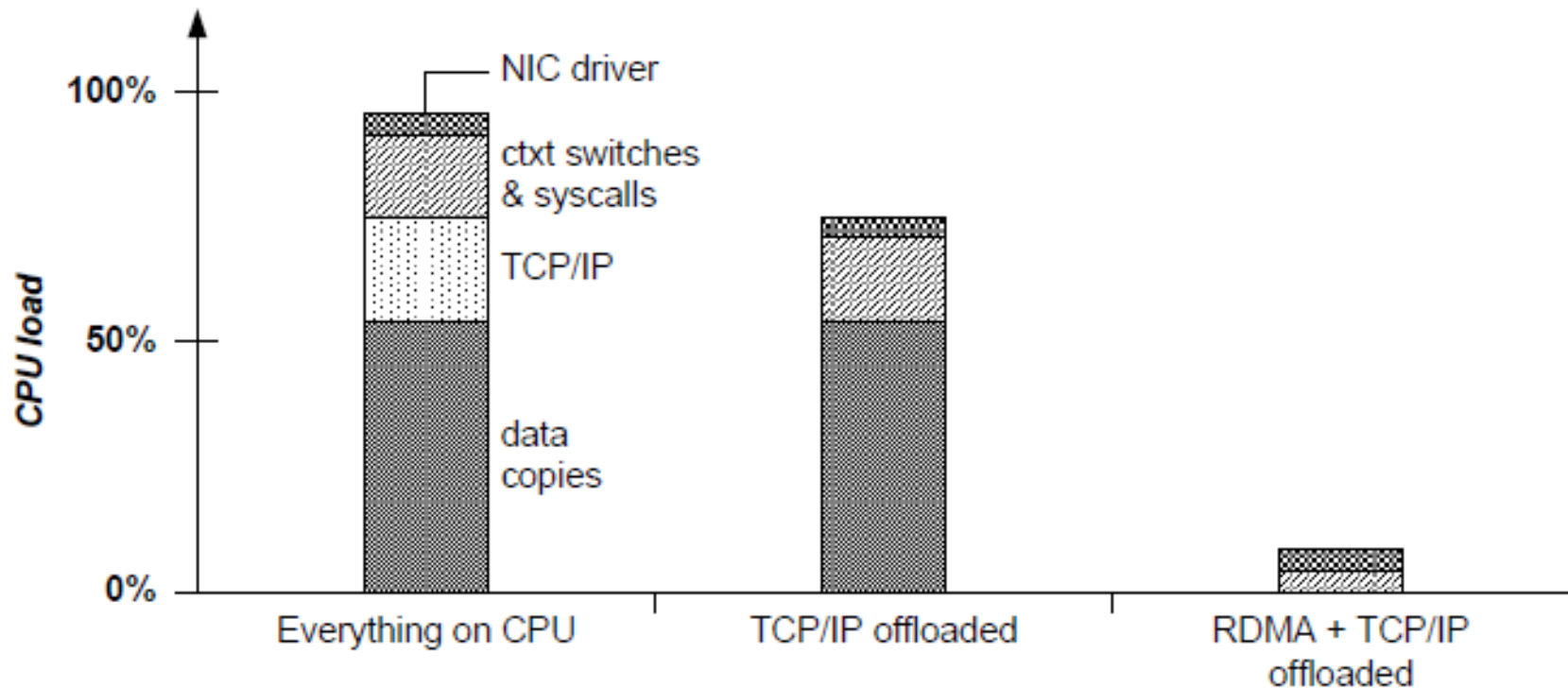


- After operation has finished, a CQE is placed into the completion queue of the sender

RDMA implementations

- Infiniband
 - Compaq, HP, IBM, Intel Microsoft and Sun Microsystems
 - Provides RDMA semantics
 - First spec released 2000
 - Based on point-to-point switched fabric
 - Designed from ground up (has its own physical layer, switches, NICs, etc)
- IWARP (Internet Wide Area RDMA Protocol)
 - RDMA semantics implemented over offloaded TCP/IP
 - Requires custom NICs, but uses Ethernet
- RoCE
 - RDMA semantics implemented directly over Ethernet
- All of those implementation can be programmed through the **verbs interface**

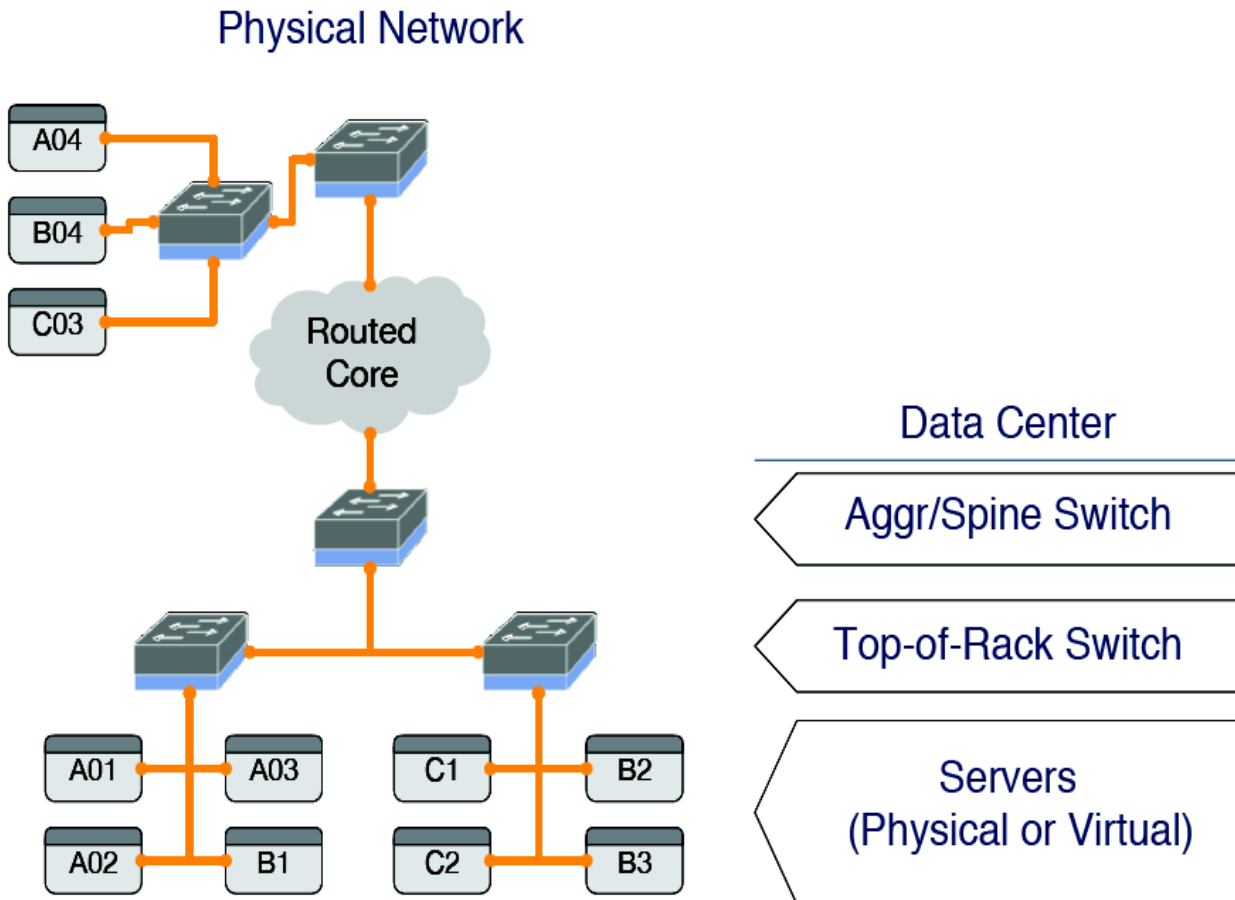
Typical CPU loads for three network stack implementations



Network Virtualization

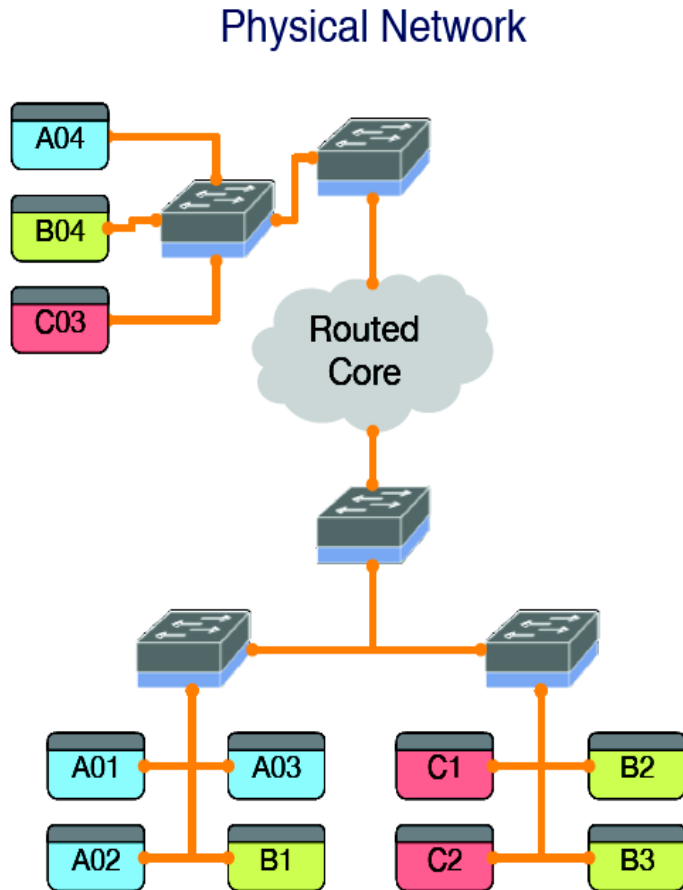
Network Virtualization

Example: Enterprise Data Center



Network Virtualization

Challenge: Multiple Applications/Tenants



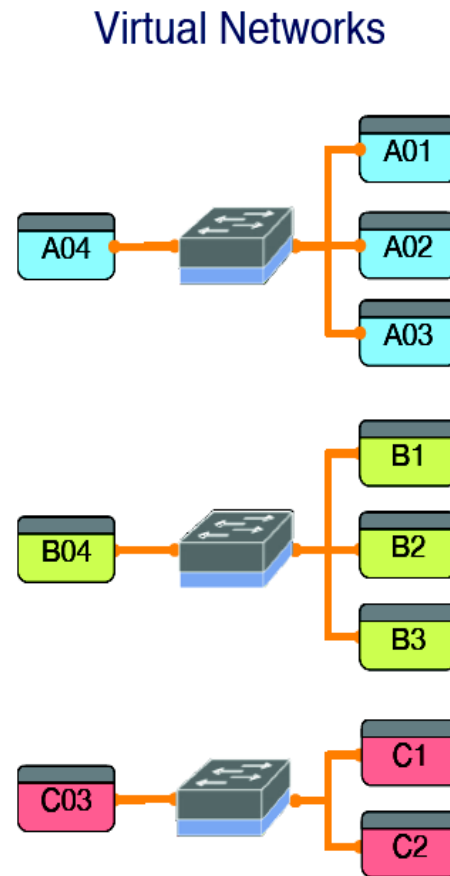
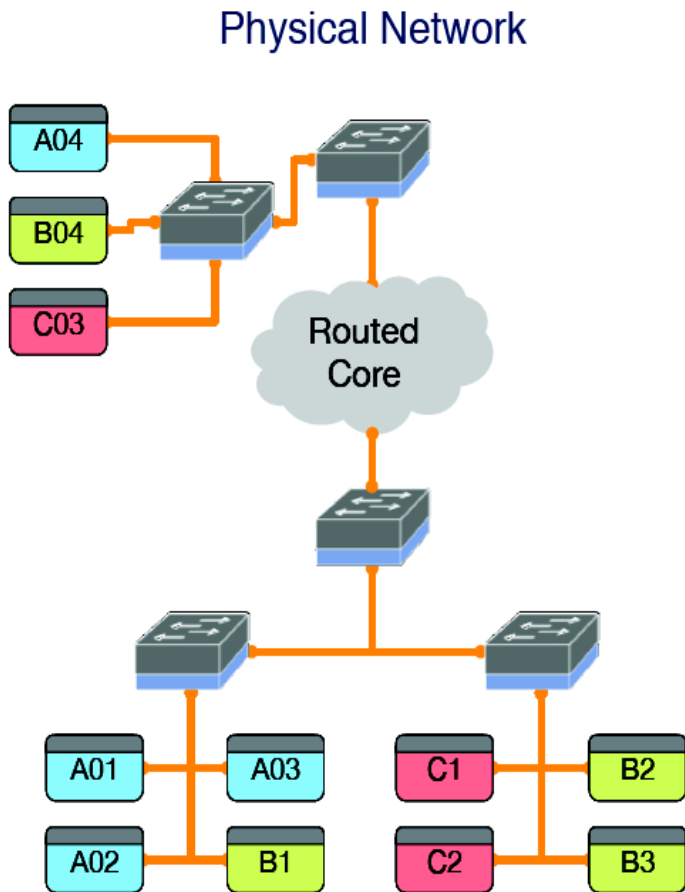
Application 1
Payment Services

Application 2
Production Intranet

Application 3
Test and Development

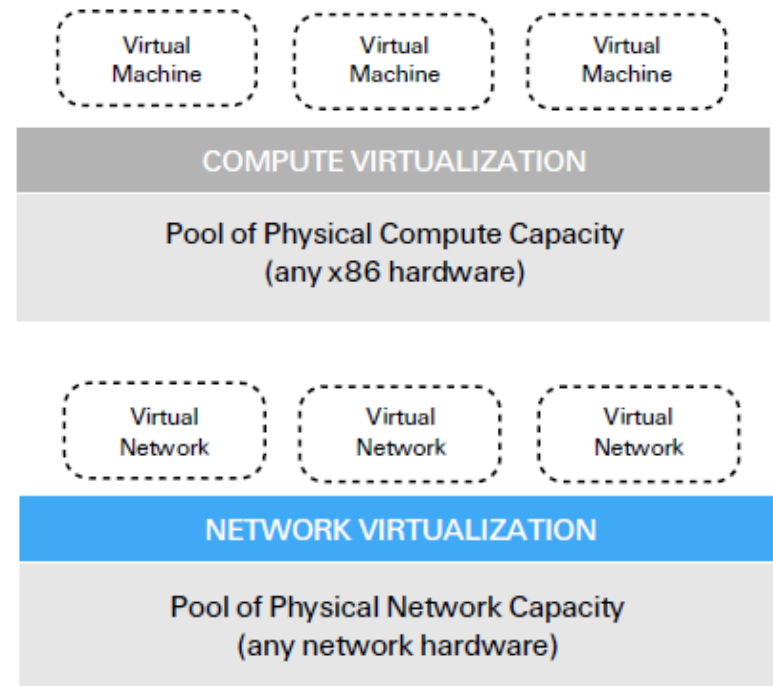
Network Virtualization

Challenge: Multiple Applications/Tenants



Network Virtualization: Goals

- Virtualize the network like we virtualize servers today:
 - Server virtualization:
decouple workload from server hardware
(e.g., CPU, Memory, I/O)
 - Network virtualization:
decouple network services from physical network hardware




Network Virtualization: Goals (2)

- Give each application/tenant its own virtual network with its own
 - Topology
 - Bandwidth,
 - Broadcast domain
 - ...

- Delegate administration for virtual networks

Network Virtualization: Goals (2)

- Give each application/tenant its own virtual network with its own
 - Topology
 - Bandwidth,
 - Broadcast domain
 - ...
- Delegate administration for

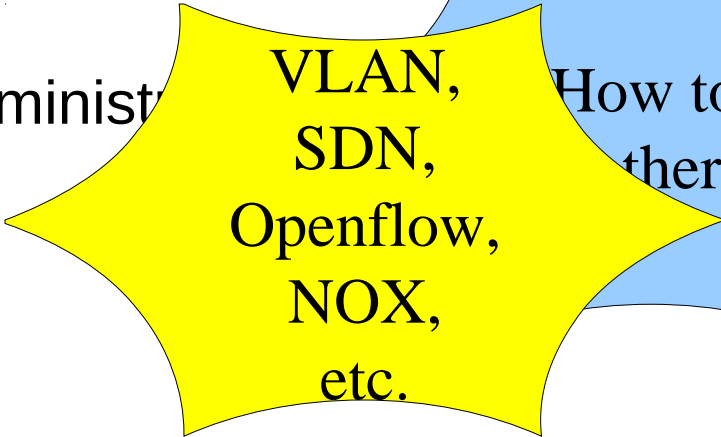


How to
get there?

Network Virtualization: Goals (2)

- Give each application/tenant its own virtual network with its own
 - Topology
 - Bandwidth,
 - Broadcast domain
 - ...

- Delegate administration



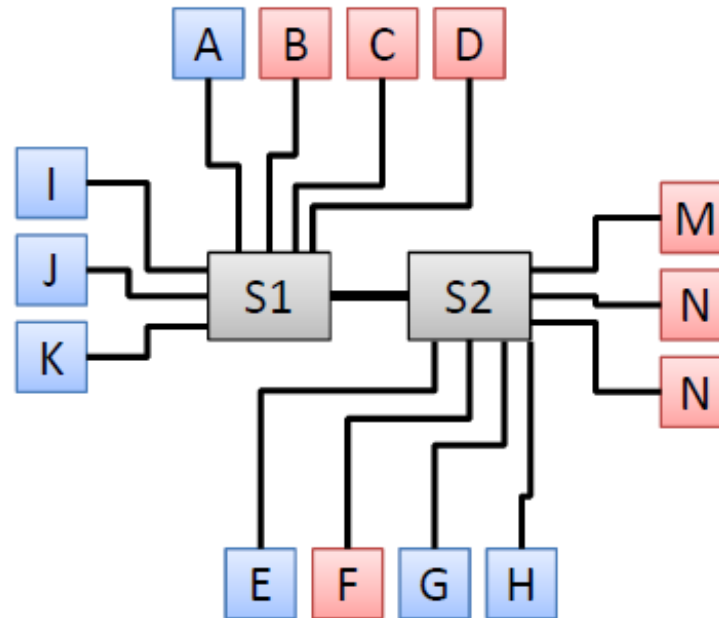
VLAN,
SDN,
Openflow,
NOX,
etc.



How to
there?

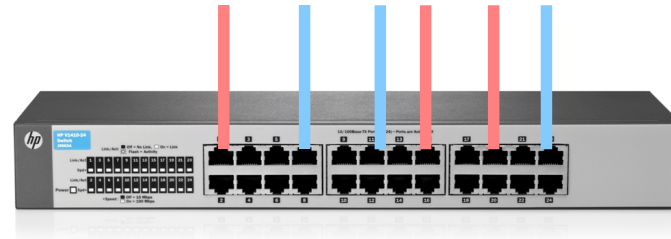
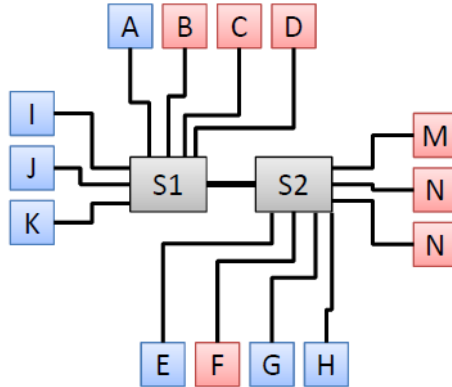
Virtual Local Area Network (VLAN)

- Remember from Lecture 2 (Principles):



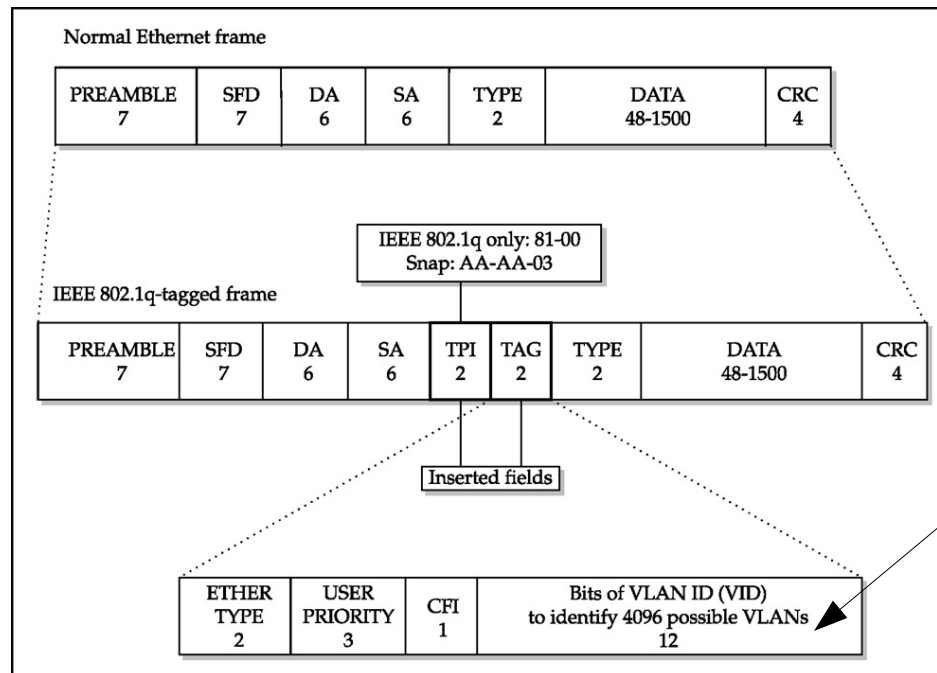
- Create multiple virtual LANs from a single physical network
 - Nodes in different LANs can't communicate
 - Broadcast isolation (e.g. ARP)

Defining VLANs



- Static: Port-based
 - Switch port statically defines the VLAN a host is part of
- Dynamic: MAC-based
 - MAC address defines which VLAN a host is part of
 - Typically configured by network administrator

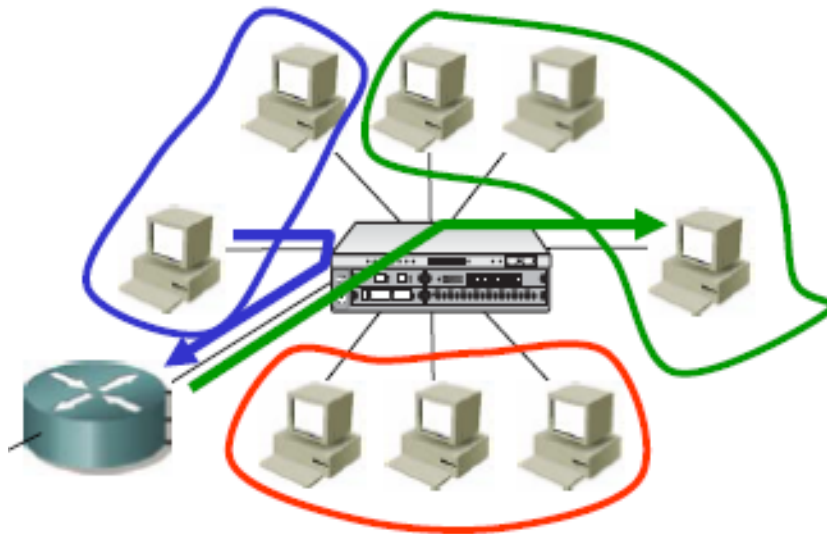
802.1Q tagging



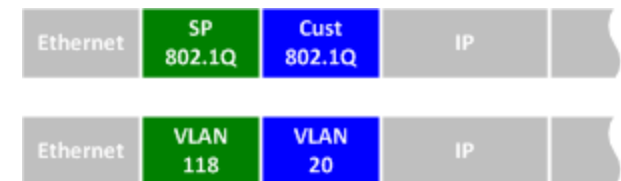
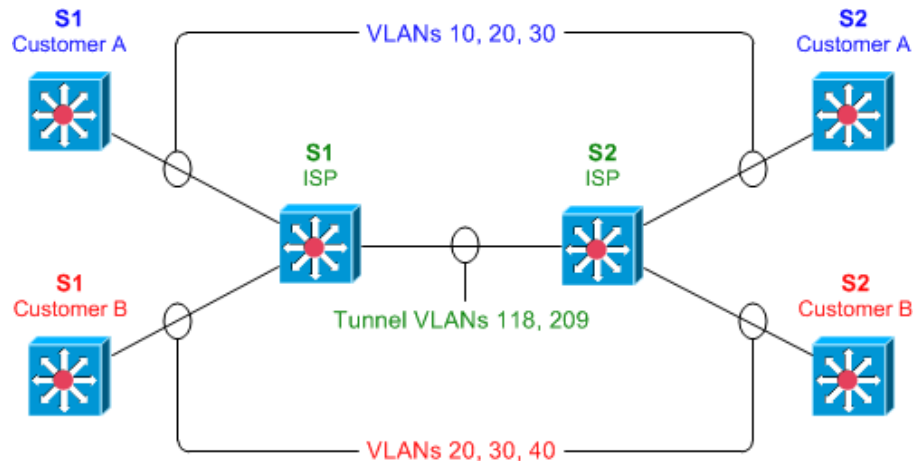
- VLAN tagging
 - Tag added to layer-2 frame at ingress switch
 - Tag stripped at egress switch
 - Tag defines on which VLAN the packet is routed
- Allows a single interconnect to transport data for various VLAN
- Trunk port: port that sends and receives tagged frames on multiple VLANs

Communication between VLANs

- Forwarding between VLANs requires going through a router
 - VLAN tag typically gets lost at router boundaries



802.1q Double Tagging




- Useful for Internet service providers
 - Allowing them to use VLANs internally while mixing traffic from clients that are already VLAN-tagged
- How it works
 - Tunnel one VLAN through another VLAN
 - VLAN tunnels add a second VLAN id to the frame (called the outer tag)
 - Switching of packets entering the tunnel is done based on the outer tag
 - Outer tag is removed at tunnel egress switch

Problems with VLANs

- Requires separate configuration of every network node
- Static configuration
 - Requires administrator
- Number of VLANs limited (4096)

Problems with VLANs

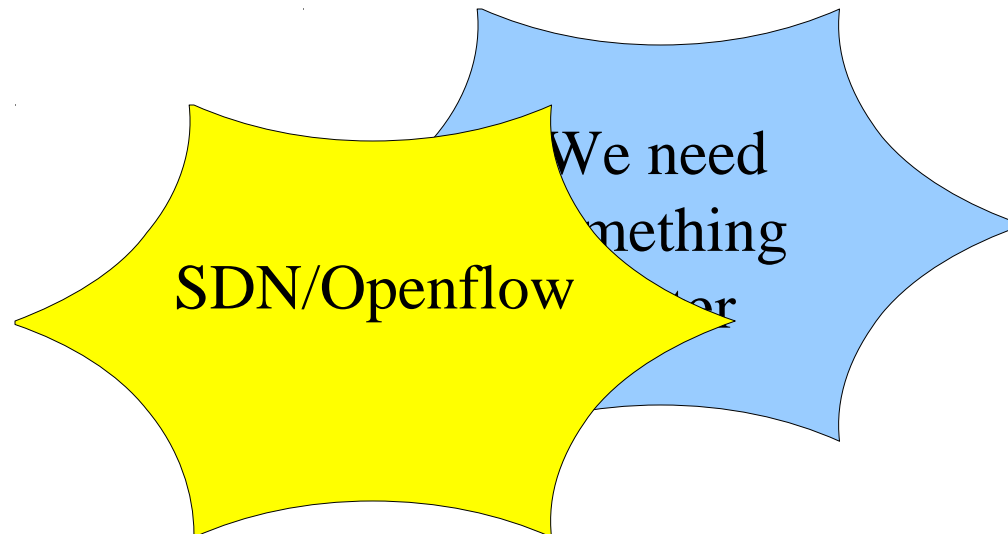
- Requires separate configuration of every network node
- Static configuration
 - Requires administrator
- Number of VLANs limited (4096)



We need
something
better

Problems with VLANs

- Requires separate configuration of every network node
- Static configuration
 - Requires administrator
- Number of VLANs limited (4096)

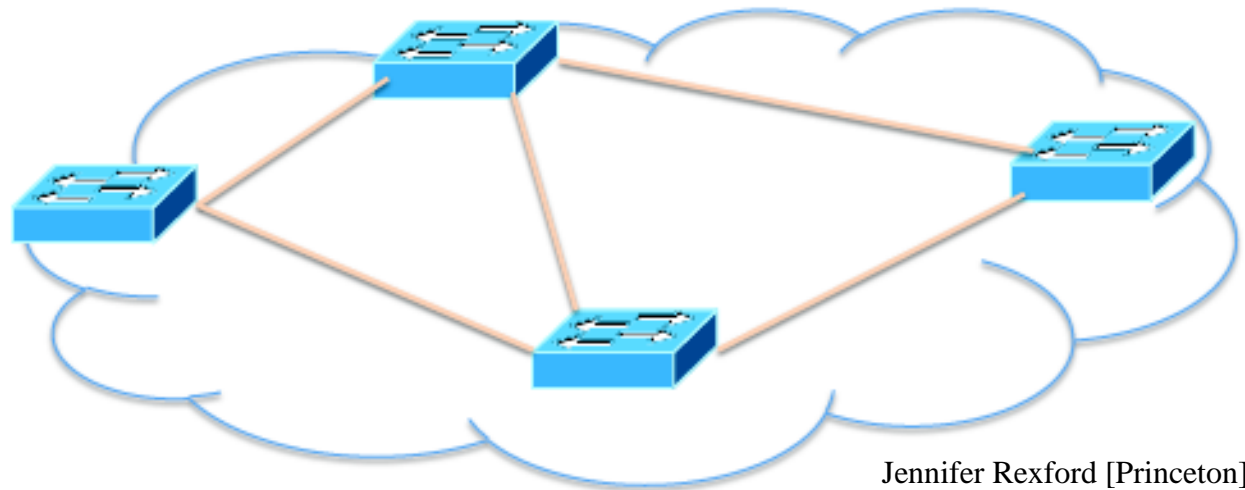


Software Defined Networks (SDN)

Traditional Networks: Data Plane

- Task: Forward, filter, buffer, mark, rate-limit and measure packets

Data plane:
Packet
streaming

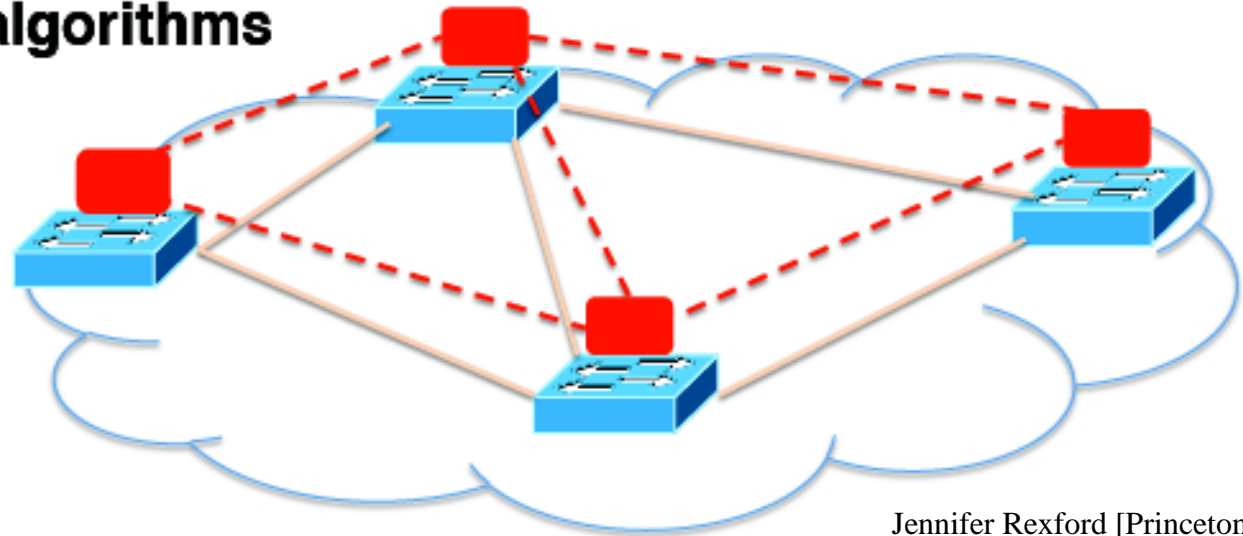


Jennifer Rexford [Princeton]

Traditional Networks: Control Plane

- Task: Track topology changes, compute routes, install forwarding tables
 - Example: Ethernet Spanning Tree Protocol

Control plane:
Distributed algorithms

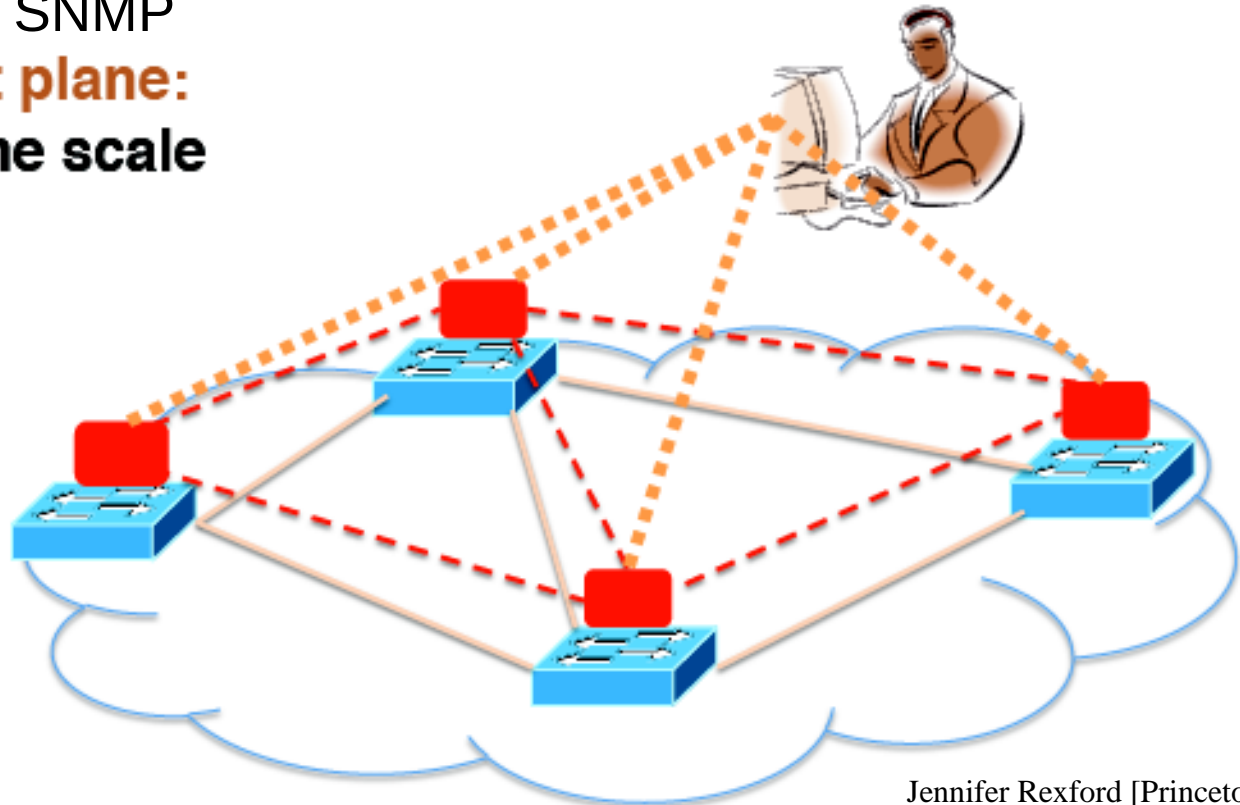


Traditional Networks: Management Plane

- Task: Collect measurements and configure equipment

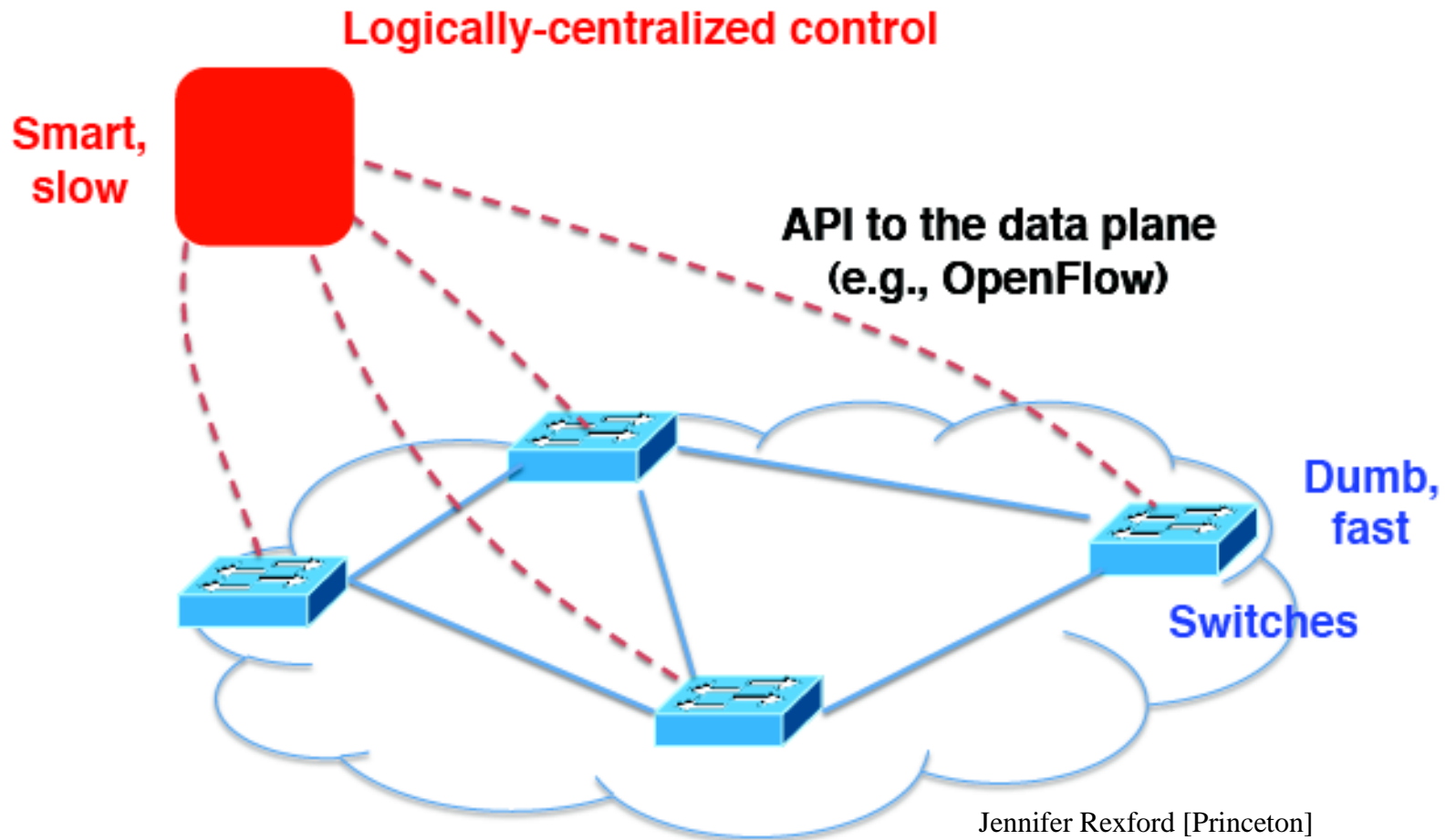
- Example: SNMP

Management plane:
Human time scale



Jennifer Rexford [Princeton]

Software Defined Networking (SDN): From a Bird's Eye View

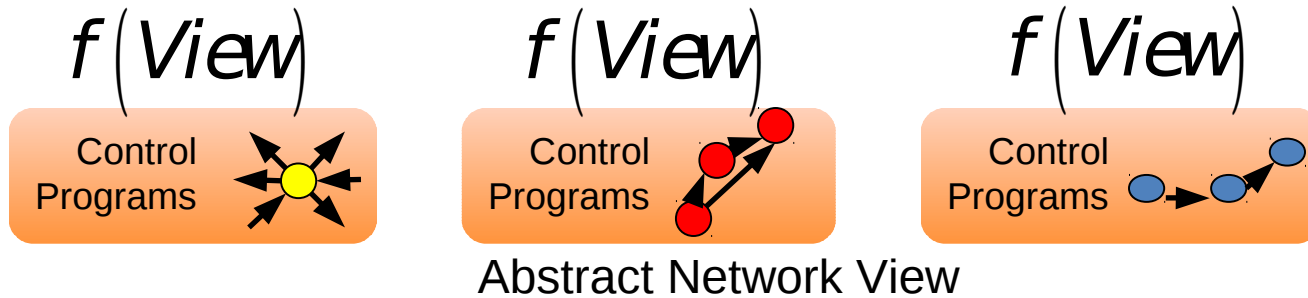


Software Defined Networking:



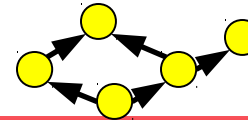
- Architecture for future enterprise and data center networks
- Originally developed for University campus networks
 - To allow researchers testing out new features
- Ideas
 - Separate network intelligence from data path
 - Extract control functionality into a logically centralized controller

Building Blocks of SDN

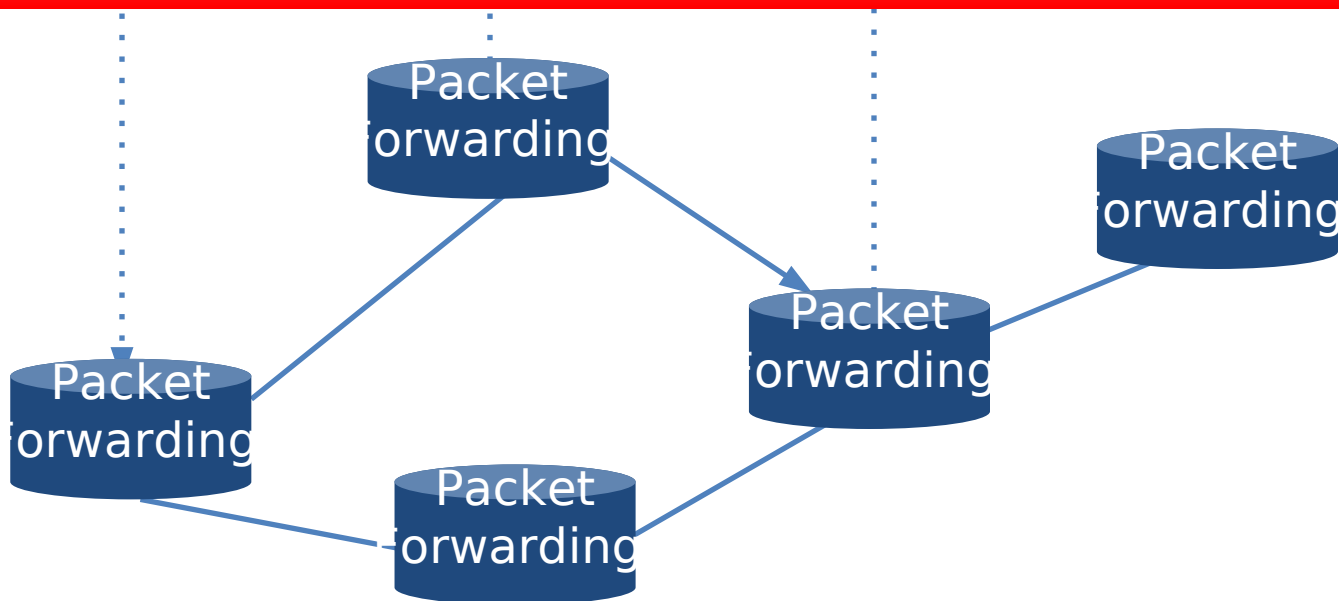


Network Virtualization

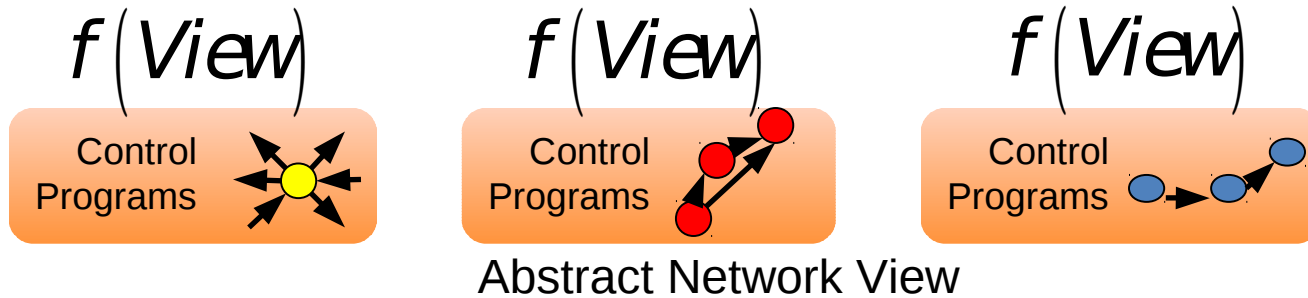
Global Network View



Network OS

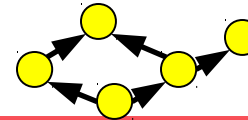


Building Blocks of SDN

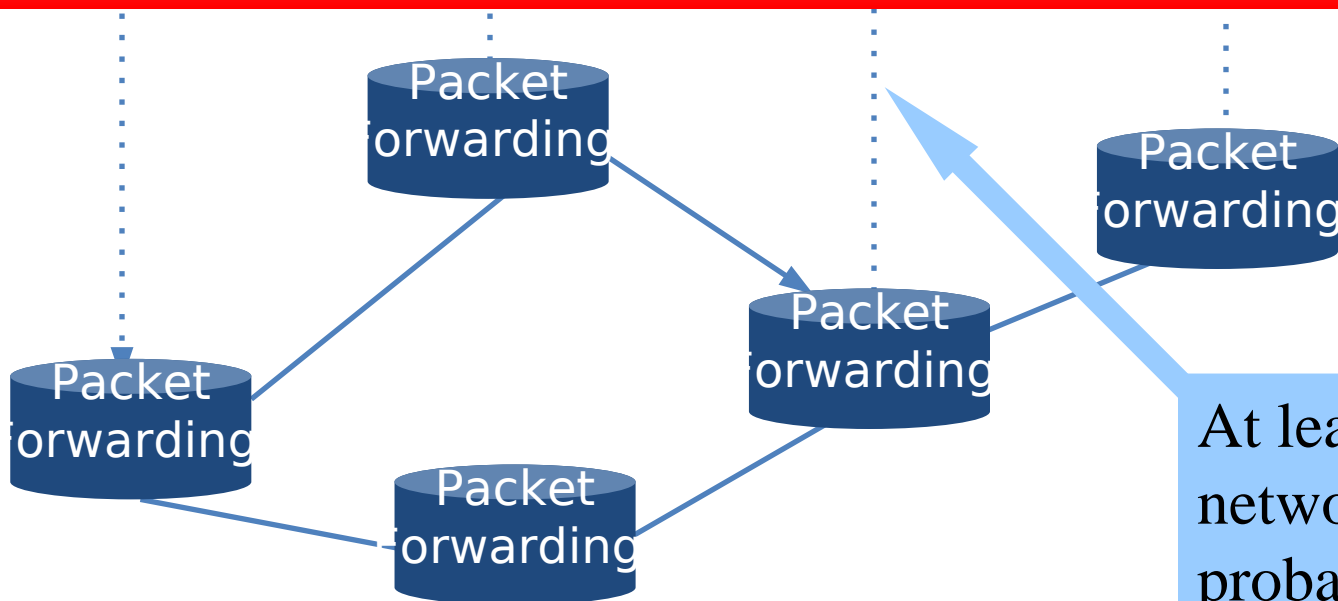


Network Virtualization

Global Network View

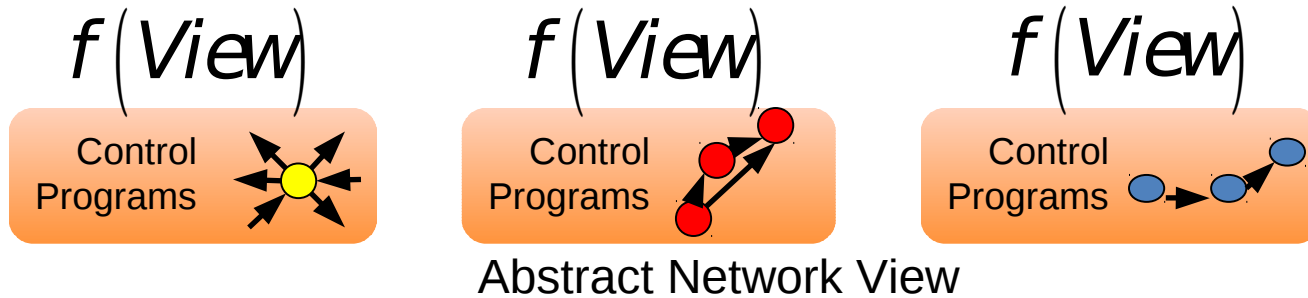


Network OS

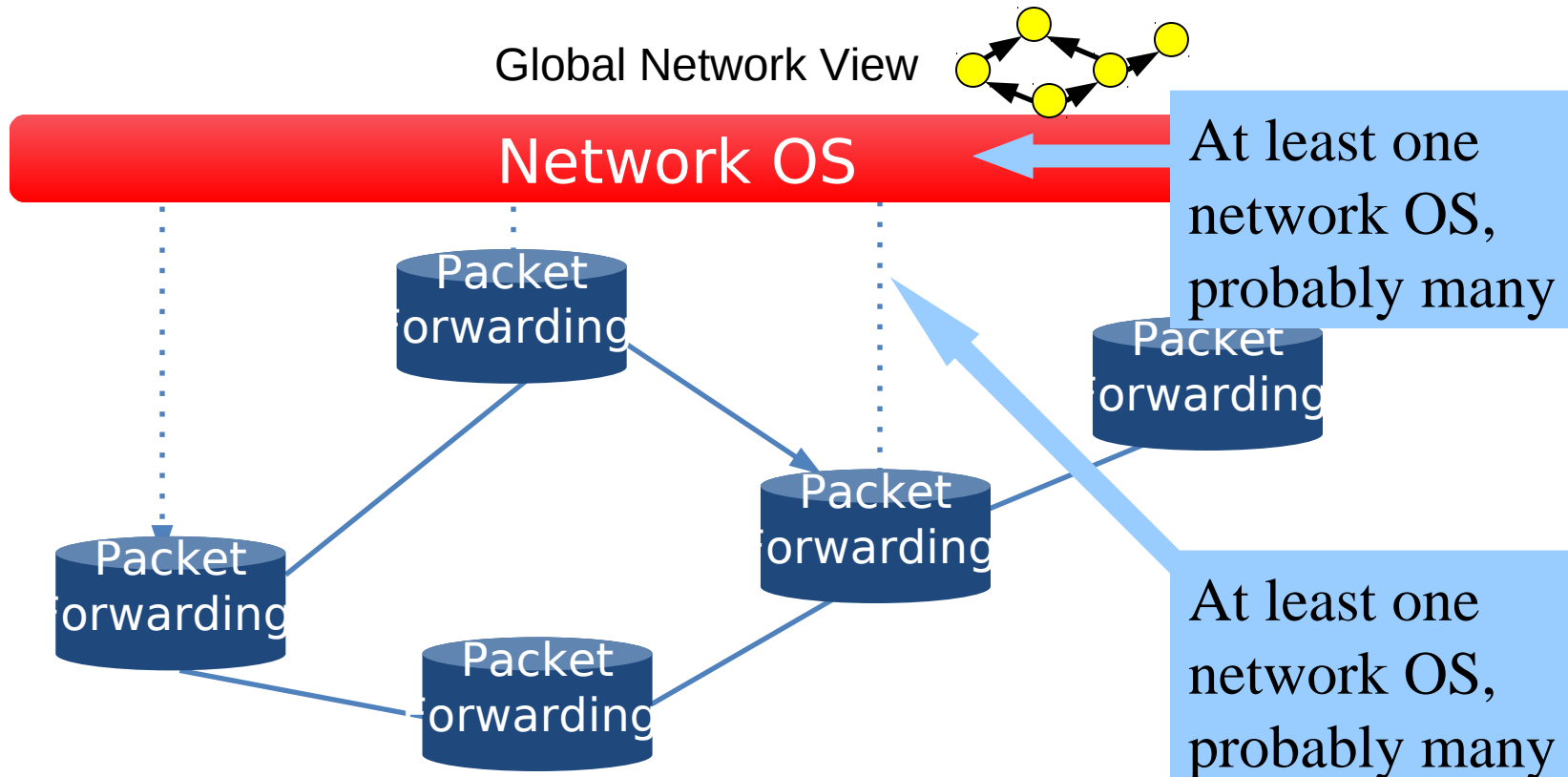


At least one
network OS,
probably many

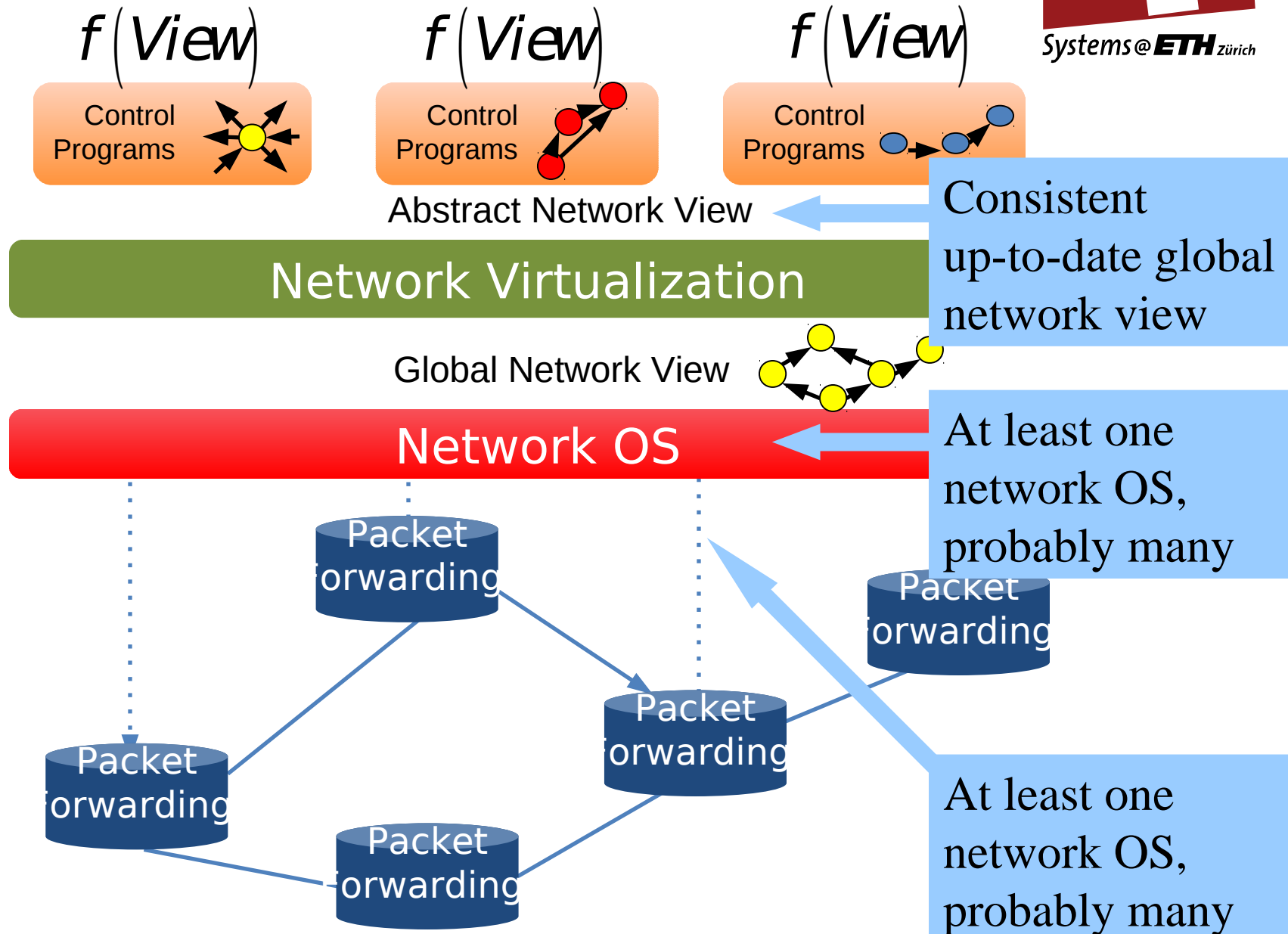
Building Blocks of SDN



Network Virtualization



Building Blocks of SDN



Building Blocks of SDN

$f(\text{View})$

Control
Programs



```
firewall.c
```

```
...
```

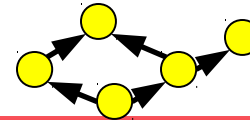
```
if( pkt->tcp->dport == 22)  
    dropPacket(pkt);
```

```
...
```

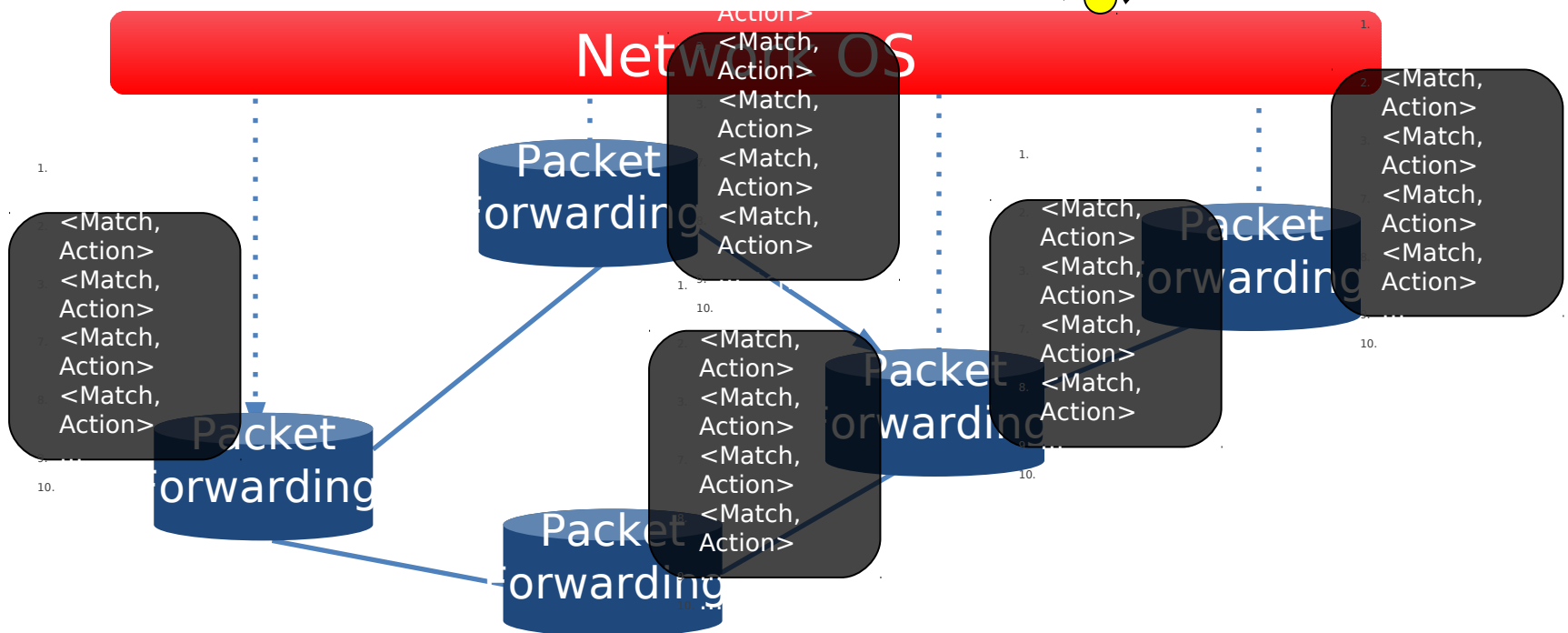
Abstract Network View

Network Virtualization

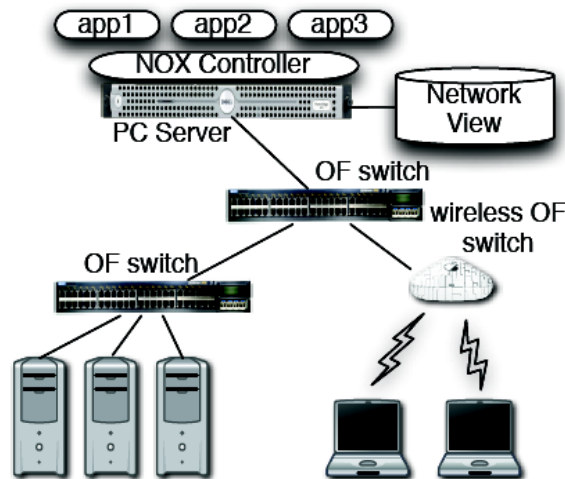
Global Network View



Network OS



Network OS



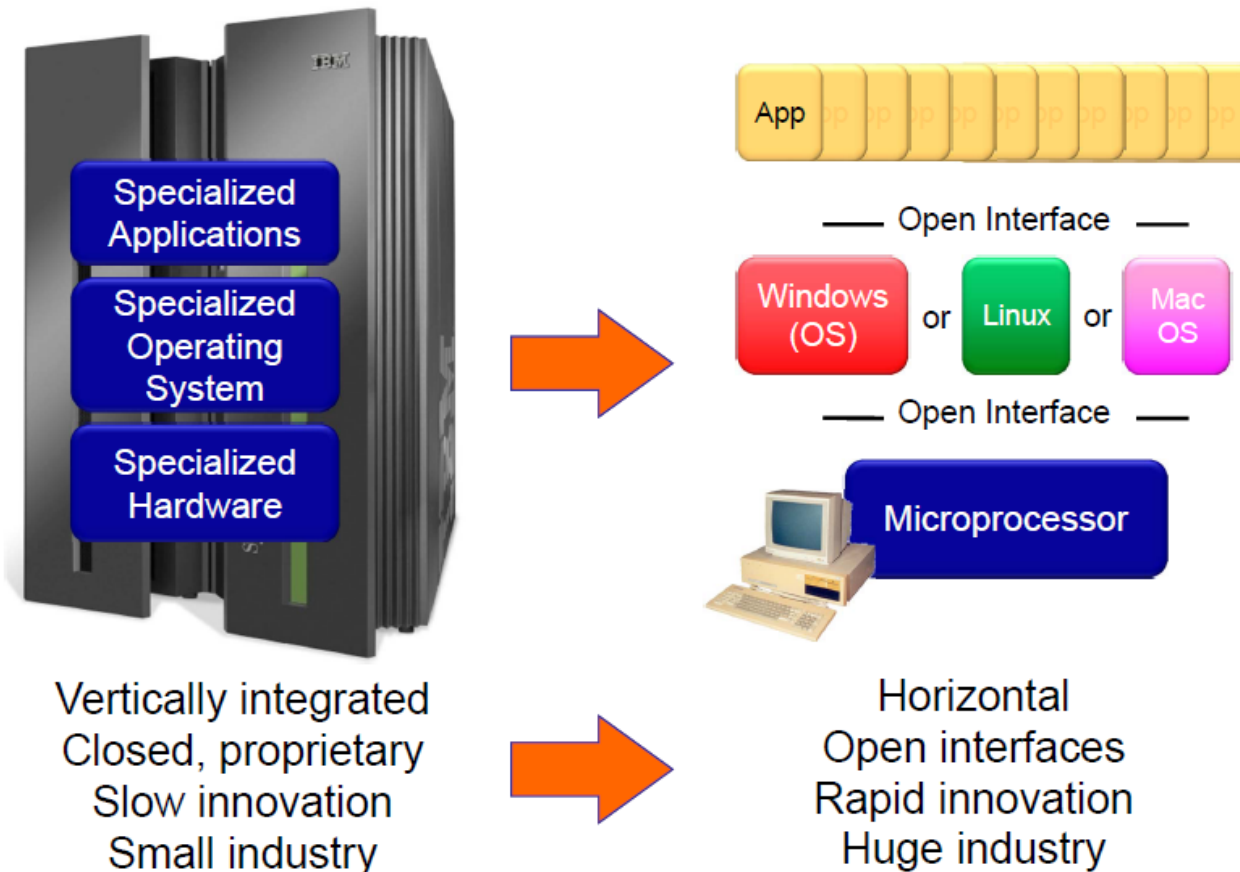
- Distributed system that creates a consistent up-to-date network view
- Runs on servers (controllers) in the network
- Uses forwarding abstraction to get/put state from/to switches
- Platform for running SDN applications (or control programs)
- Example controllers: NOX, Onix, Floodlight, ... + more

Control program

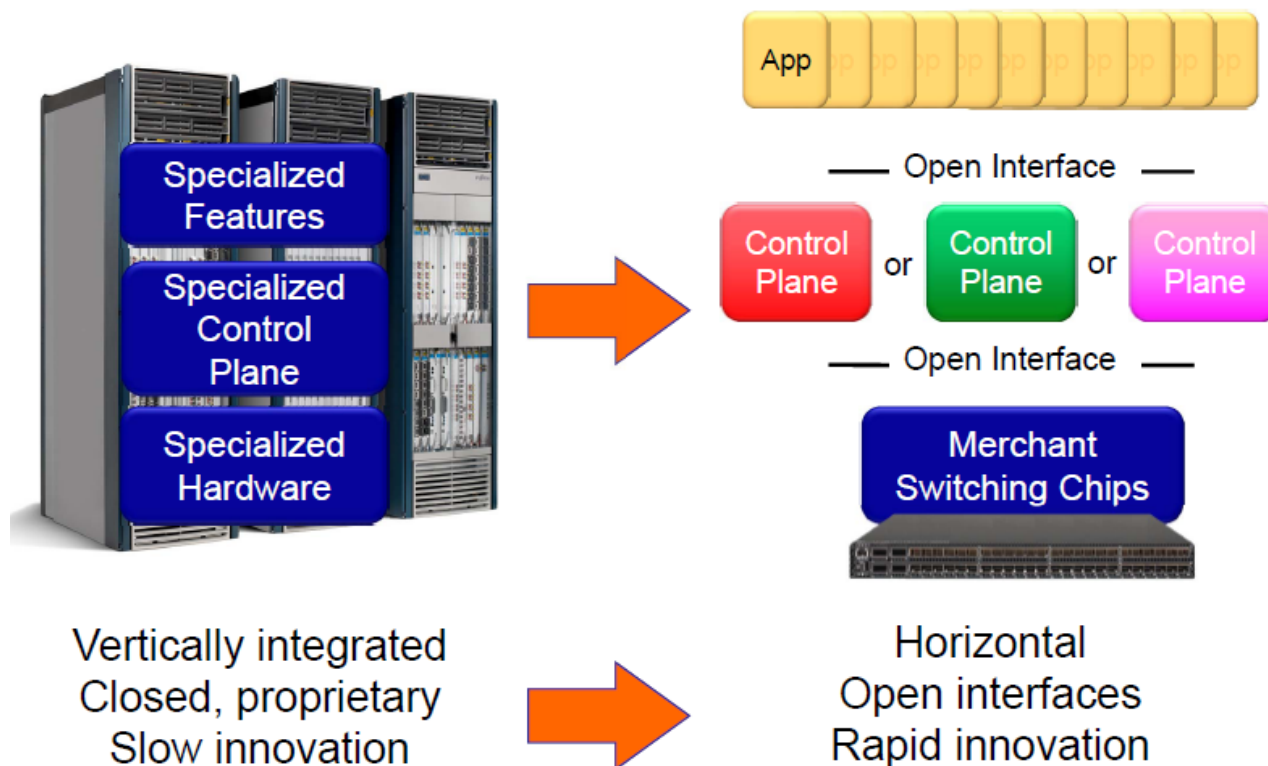
```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
    vlanid = user_to_vlan_function(user)
    # For packets from the user, add a VLAN tag
    attr_out[IN_PORT] = port
    attr_out[DL_SRC] = nox.reverse_resolve(host).mac
    action_out = [(nox.OUTPUT, (0, nox.FLOOD)),
                  (nox.ADD_VLAN, (vlanid))]
    install_datapath_flow(dp, attr_out, action_out)
    # For packets to the user with the VLAN tag, remove it
    attr_in[DL_DST] = nox.reverse_resolve(host).mac
    attr_in[DL_VLAN] = vlanid
    action_in = [(nox.OUTPUT, (0, nox.FLOOD)),
                 (nox.DEL_VLAN)]
    install_datapath_flow(dp, attr_in, action_in)
    nox.register_for_user_authentication(setup_user_vlan)
```

- Control program operates on view of network
 - **Input:** global network view (graph/database)
 - **Output:** configuration of each network devices
- Control program is not a distributed system
 - Abstractions hide details of distributed state
- Event-driven programming: register handlers for events
- Example: NOX control program to set VLAN tagging rules on user authentication

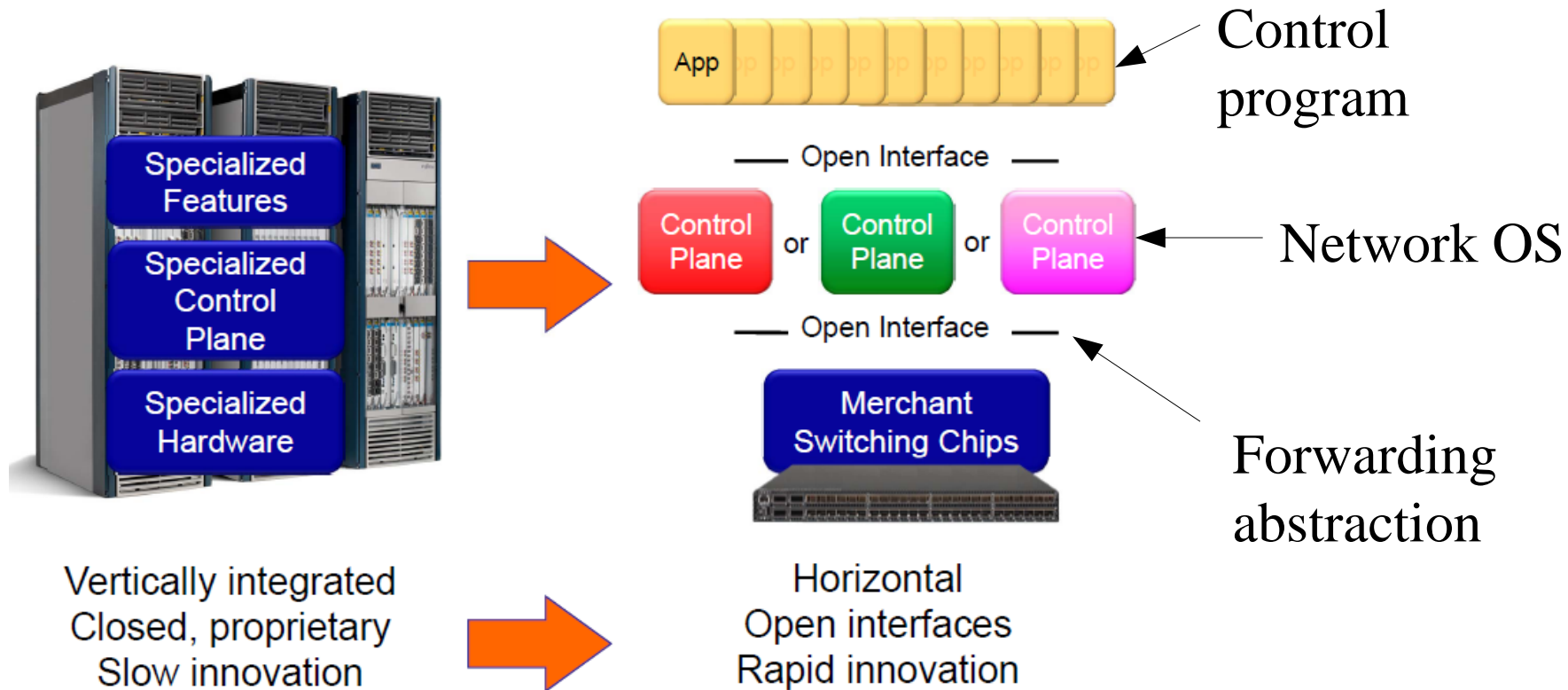
A Nice Analogy (1)



A Nice Analogy (2)



A Nice Analogy (3)



OpenFlow

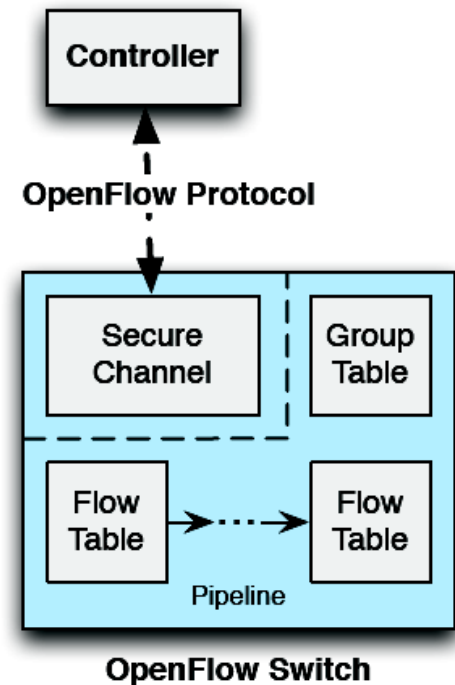
OpenFlow Basics

- Started in 2004 with the PhD thesis of Martin Casado
 - Now CEO at Nicira Networks
- Original motivation: drive innovation for new network architectures/protocols
- Gap in the tool space, current systems either
 - do NOT perform at the scale and speed we want OR
 - are NOT open to be programmed by researchers

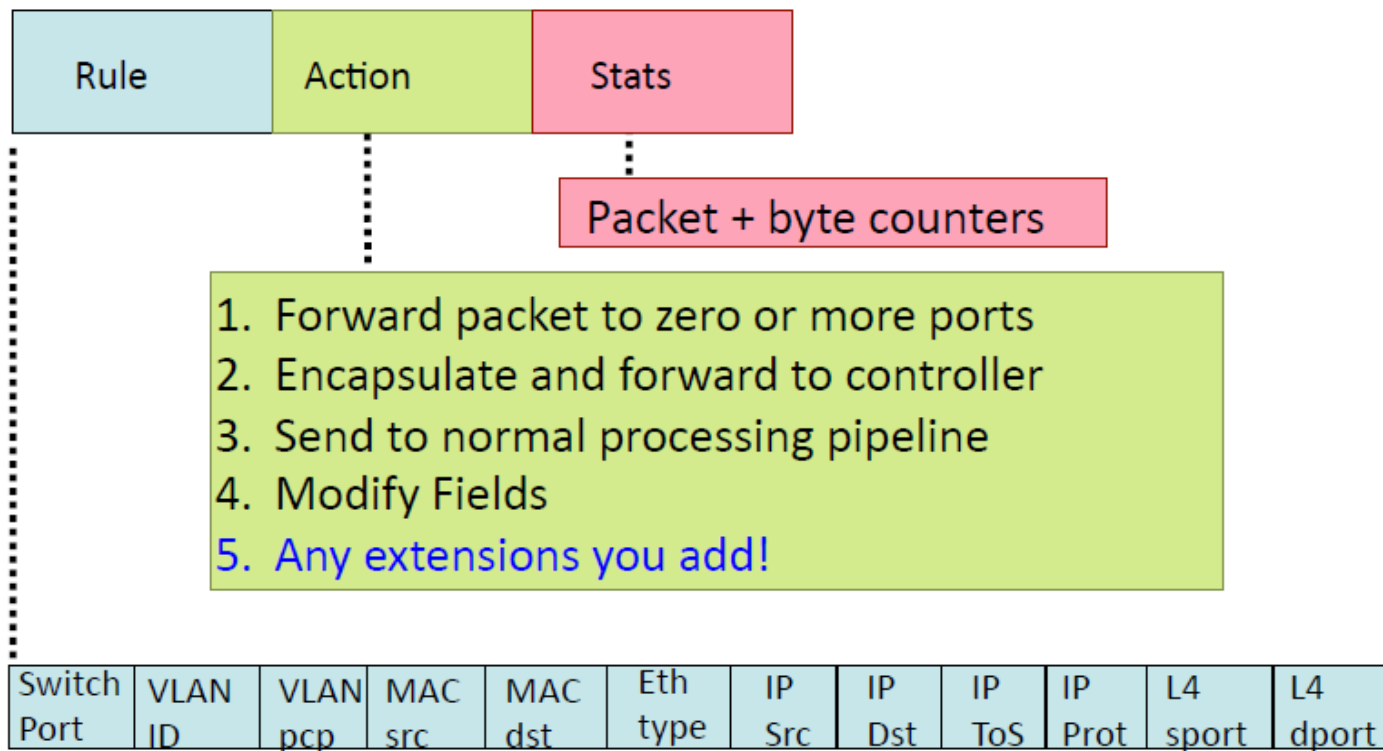
	Performance Fidelity	Scale	Real User Traffic?	Complexity	Open
Simulation	medium	medium	no	medium	yes
Emulation	medium	low	no	medium	yes
Software Switches	poor	low	yes	medium	yes
NetFPGA	high	low	yes	high	yes
Network Processors	high	medium	yes	high	yes
Vendor Switches	high	high	yes	low	no

OpenFlow Building Blocks

- Controller talks to OpenFlow switch through a secure channel
- Switch contains:
 - One or more flow tables
 - A group table
- Flow tables:
 - Contain flow entries
 - Packets matched against flow entries
 - Flow entry determines which packet matches and what action will be taken
- Group table
 - Set of group entries
 - Each group entry has: identifier, type, counters and action bucket
 - Allows for additional action to be set on a packet: actions common for all packets of the same group



Flow Table Entries



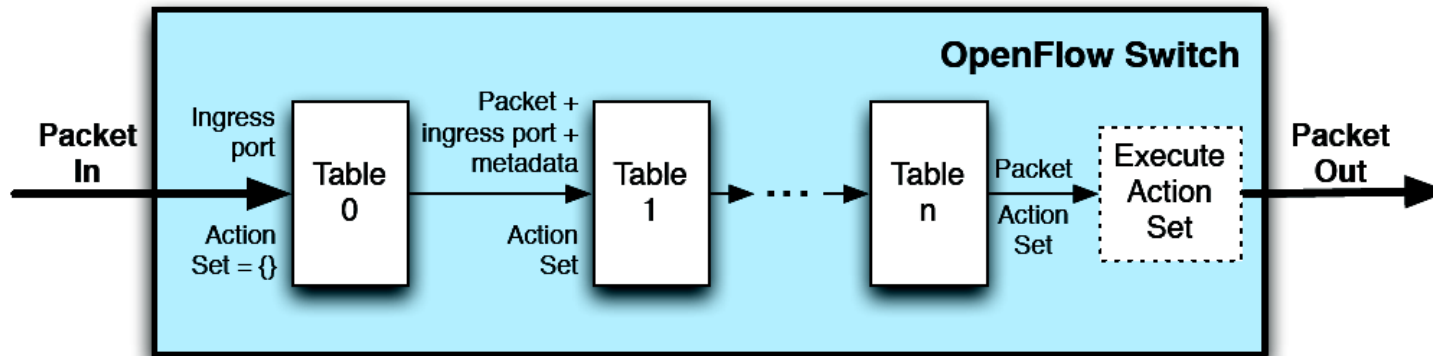
+ mask what fields to match

[illegible]

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

[illegible]

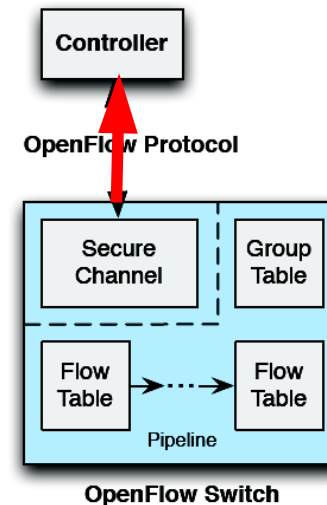
Pipeline Processing



- Flow tables sequentially numbered
- A flow table entry adds actions to an action set of a packet
- A flow table entry may explicitly direct the packet to another flow table
 - Can only direct a packet to a flow table with number which is greater than my own table's number
- If table entry does NOT direct a packet to another table, the pipeline stops and the associated action set is executed

OpenFlow Asynchronous Events

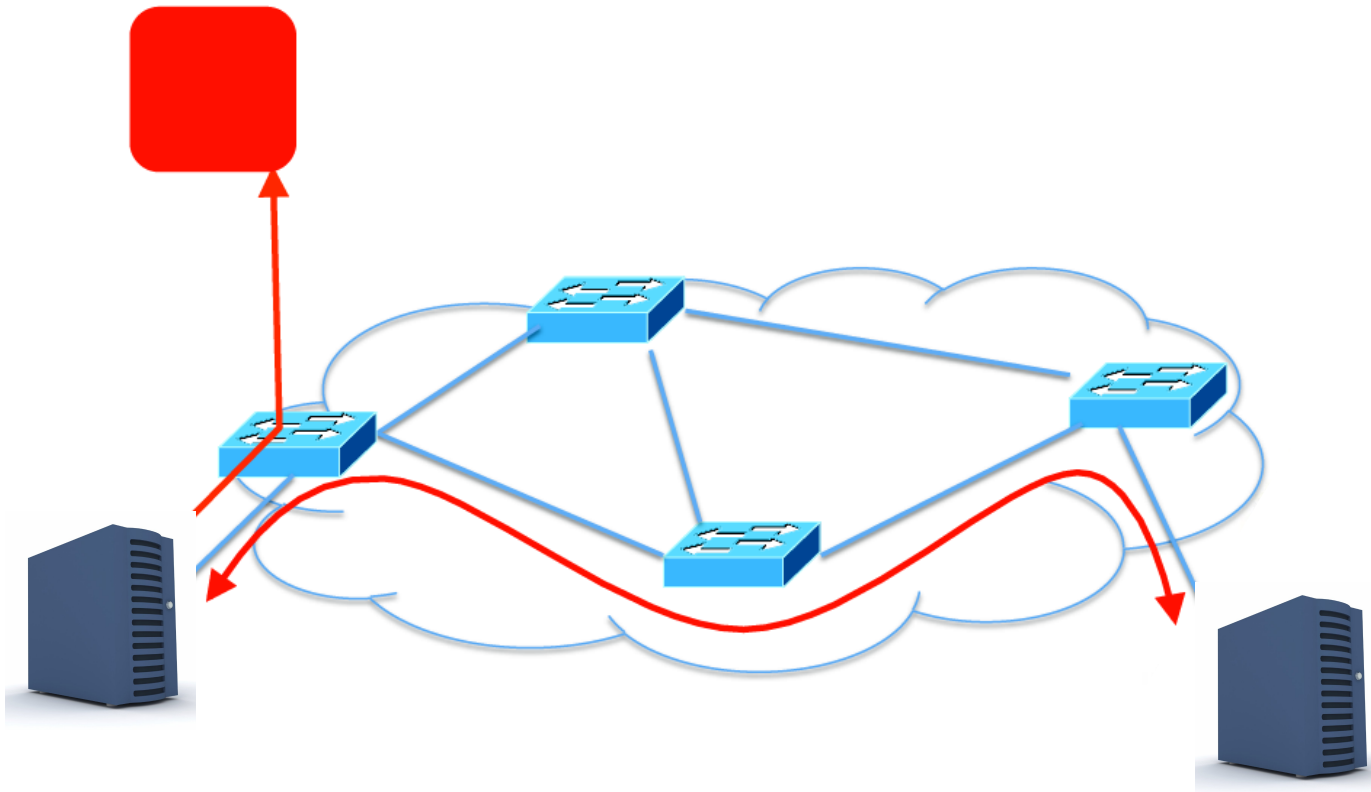
- Sent by the switch without being requested from the controller in case of a
 - Packet arrival for which there is not matching flow entry
 - Switch state change (e.g., new switch added)
 - Error



Example OpenFlow Applications

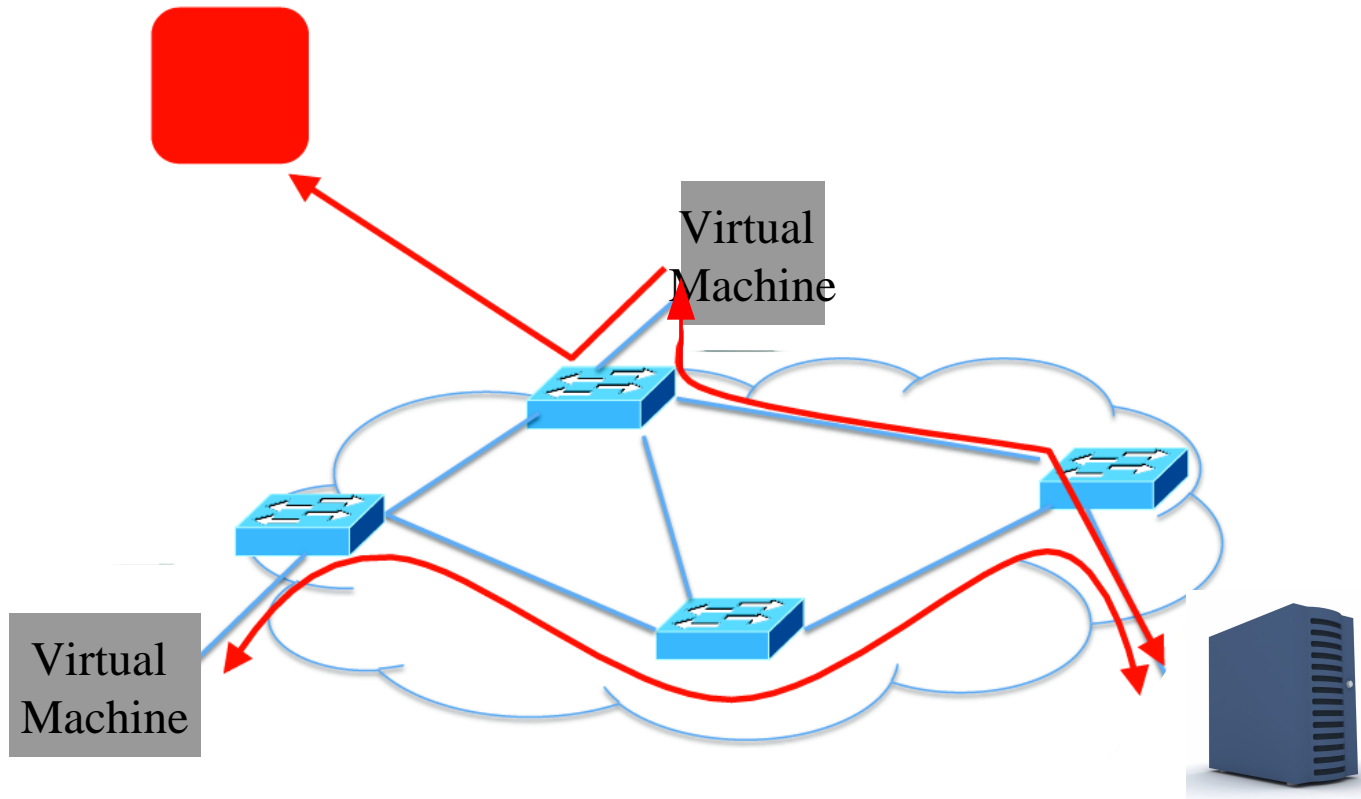
- Dynamic Access Control
- Seamless mobility
- Network virtualization
- Energy efficient networking

Dynamic Access Control



- Inspect first packet of a connection
- Consult the access control policy
- Install rules to block or route traffic

Seamless Mobility/Migration



- Observe hosts sends traffic from new location
- Modify flow tables to re-route the traffic

Saving Energy with OpenFlow (1)

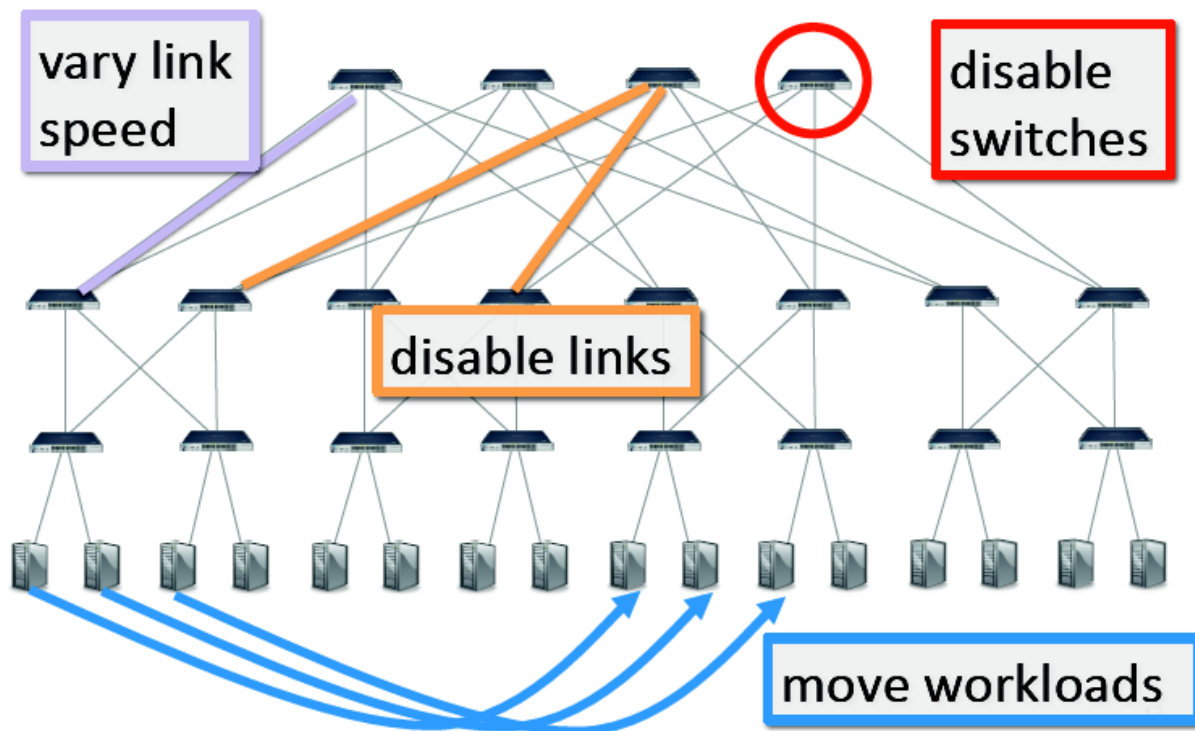


4

- **Problem:** Power consumption of switches is high, even no ports are connects
- Key idea: Use OpenFlow to dynamically turn off switching elements that are not needed

Saving Energy with OpenFlow (2): The Network Power Knobs

- Actually, we have even more options: vary link speed, disable switch, move VMs, disable links



Remember:

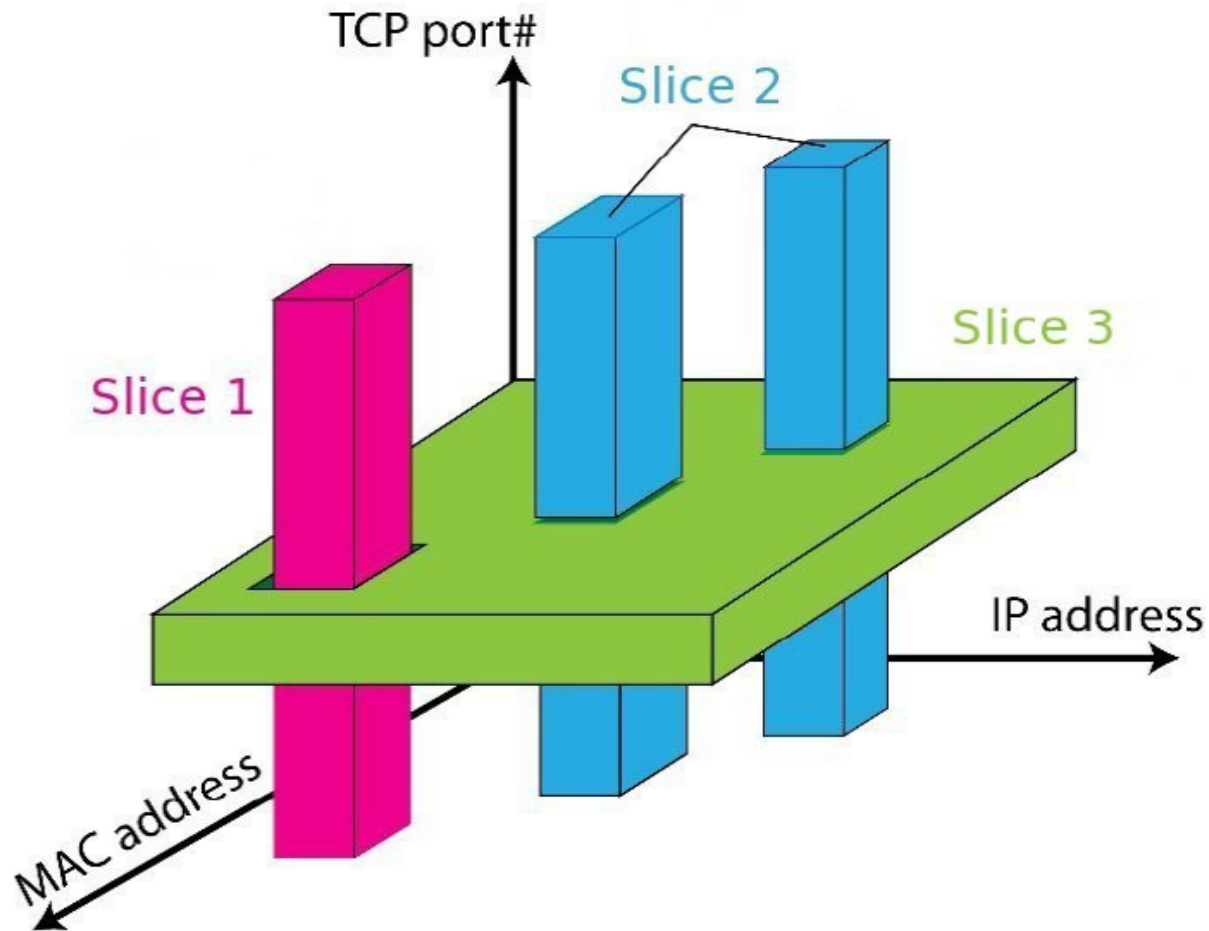
Network Virtualization Goals

- Give each application/tenant its own virtual network with its own
 - Topology
 - Bandwidth,
 - Broadcast domain
 - ...
- Delegate administration for virtual networks

FlowVisor: Slicing the Network

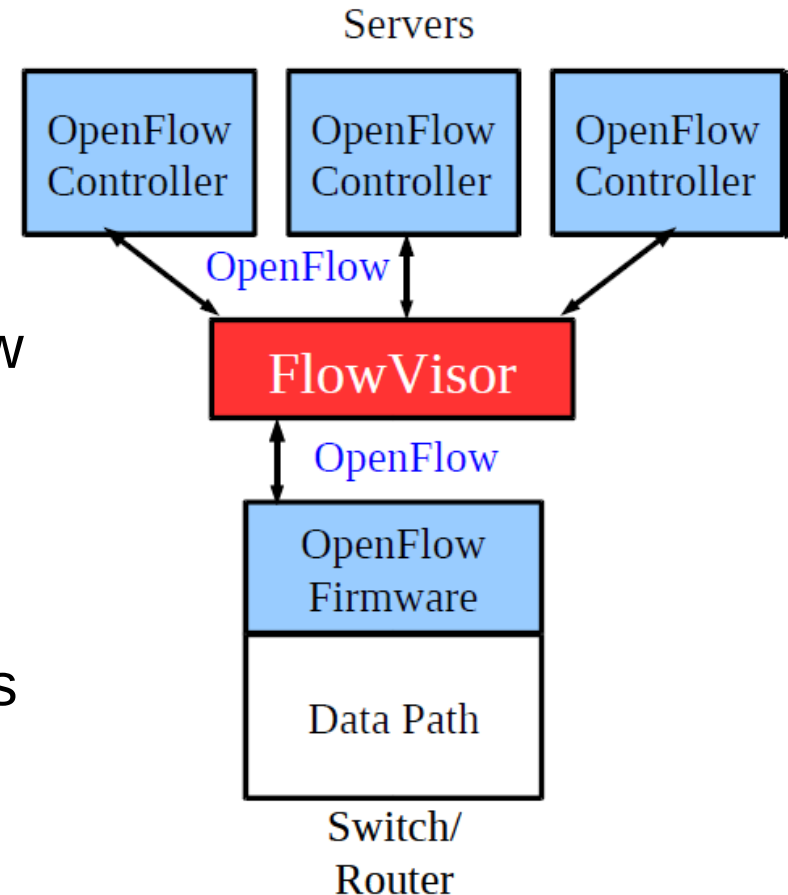
- Divide the physical network into logical **slices**
 - Each slice/service controls its own packet forwarding
 - Give different slices to different application or owners
 - Enforce strong isolation between slices
- A **network slice** is a collection of sliced switches/routers
- Each slice believes it owns the data path
- Slicing Policy: specifies resource limits for each slice
 - Link Bandwidth
 - Topology
 - Maximum number of forwarding rules

Mapping packets to Slices



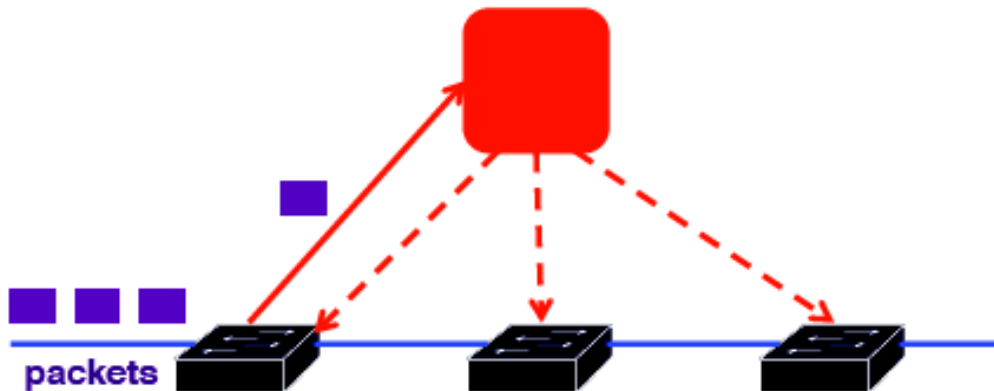
FlowVisor Implementation

- FlowVisor is an SDN/OpenFlow controller
 - Talks OpenFlow to the switches
- FlowVisor runs multiple OpenFlow controller, one for each slice
 - Talks OpenFlow to the 'Slice' controller
- FlowVisor intercepts and re-writes OpenFlow messages from the 'Slice' controllers



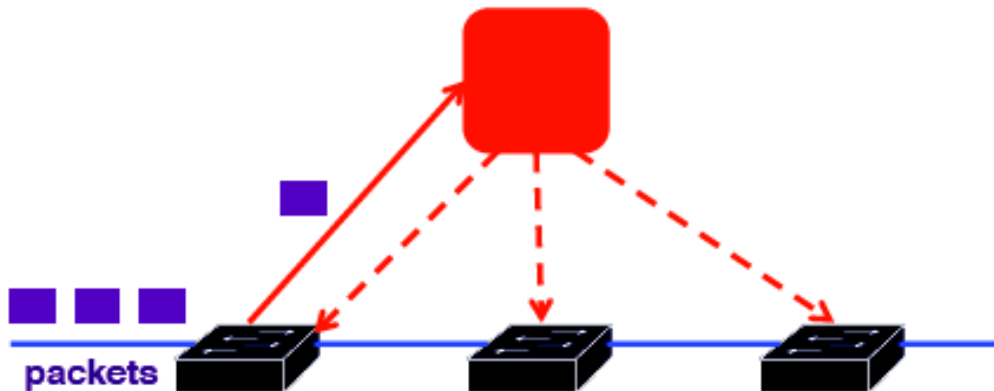
OpenFlow Challenges: Controller Delay and Overhead

- Controller is much slower than the switches
- Processing packets leads to delay and overhead
- Need to keep most packets in “fast path”



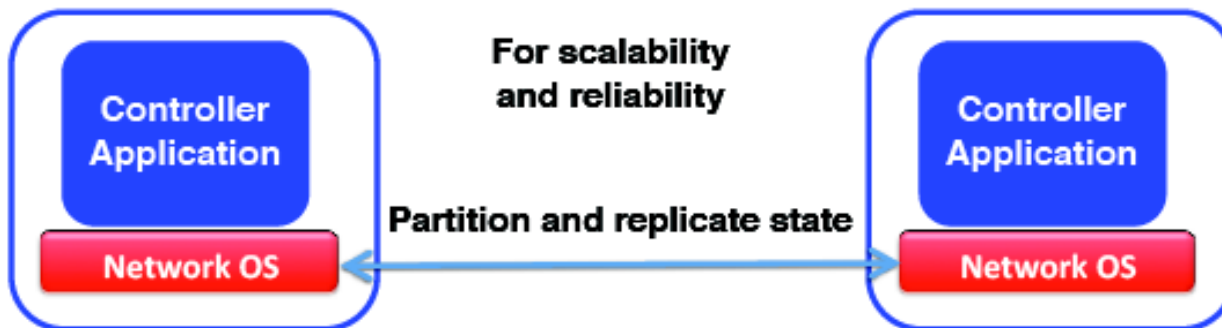
OpenFlow Challenges: Controller Delay and Overhead

- Controller is much slower than the switches
- Processing packets leads to delay and overhead
- Need to keep most packets in “fast path”



OpenFlow Challenges (2): Distributed Controller

- Controller is “single-point of failure” and potential bottleneck
- Partition or replicate controller for scalability and reliability
- Problems: keeping state consistent



References

- “U-Net: User-level network interface for parallel and distributed computing”, SOSP 1995
- “Ethane: Taking Control of the Enterprise”, Sigcomm 2007
- “OpenFlow: Enabling Innovation in Campus Networks”, SIGCOMM Comput. Communication Review, 2008
- “Can the Production Network Be the Testbed?”, OSDI 2010
- “OpenFlow Switch Specification 1.1.0”, <http://www.openflow.org/>
- “DevFlow: Scaling Flow Management for High-Performance Networks”, Sigcomm 2011
- “A Survey of Virtual LAN Usage in Campus Networks”, IEEE Communications, 2011

Flow Table Entries: Example (2)

Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

VLAN Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f..	*	vlan1	*	*	*	*	*	port6, port7, port9

FlowVisor Message Handling

