

50.012 Networks

Lecture 16: SDN

2021 Term 6

Assoc. Prof. CHEN Binbin



Outline

The SDN control plane

Generalized Forwarding

- match
- action
- OpenFlow examples of match-plus-action in action

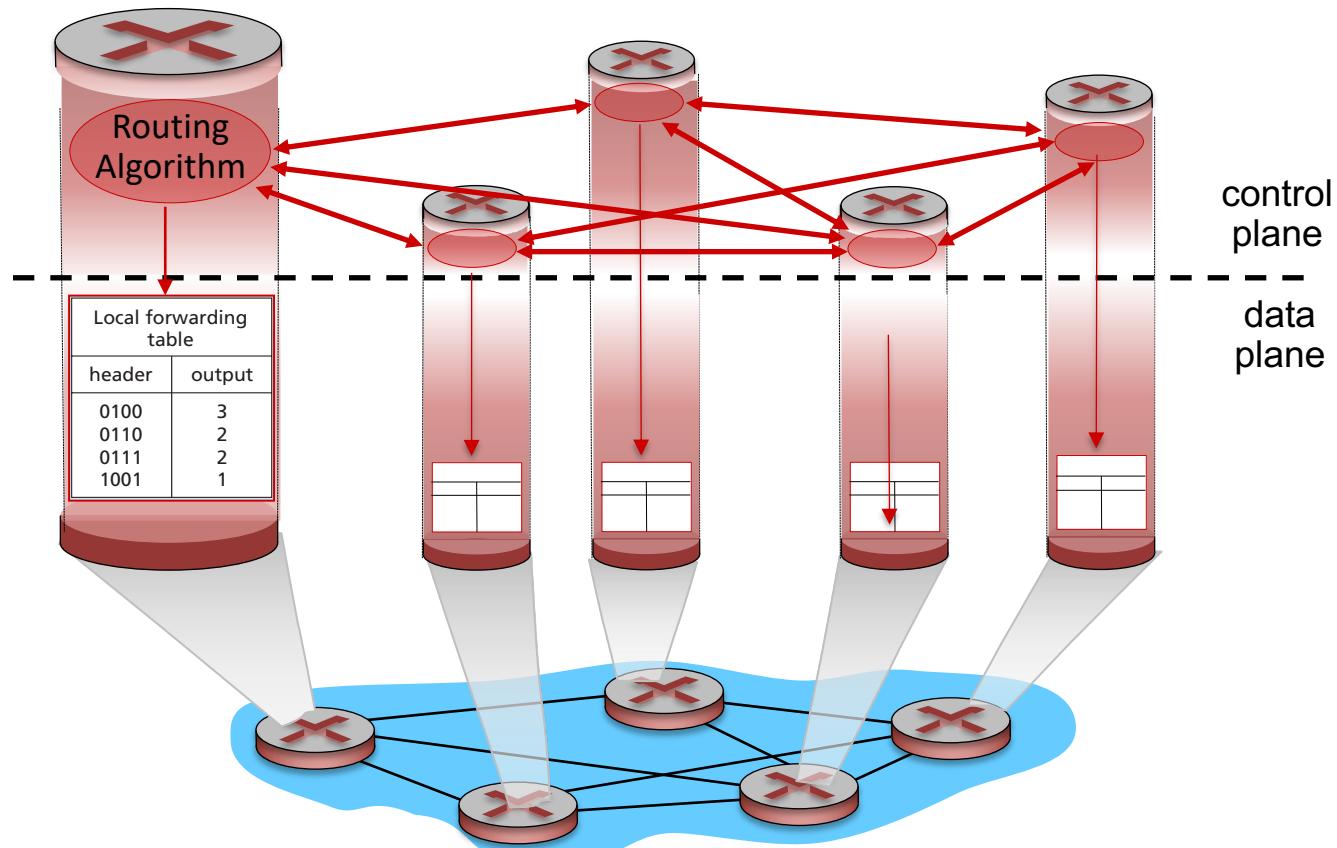
Read textbook Section 4.4, 5.5

Software defined networking (SDN)

- Internet network layer: historically has been implemented via distributed, per-router approach
 - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
 - different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

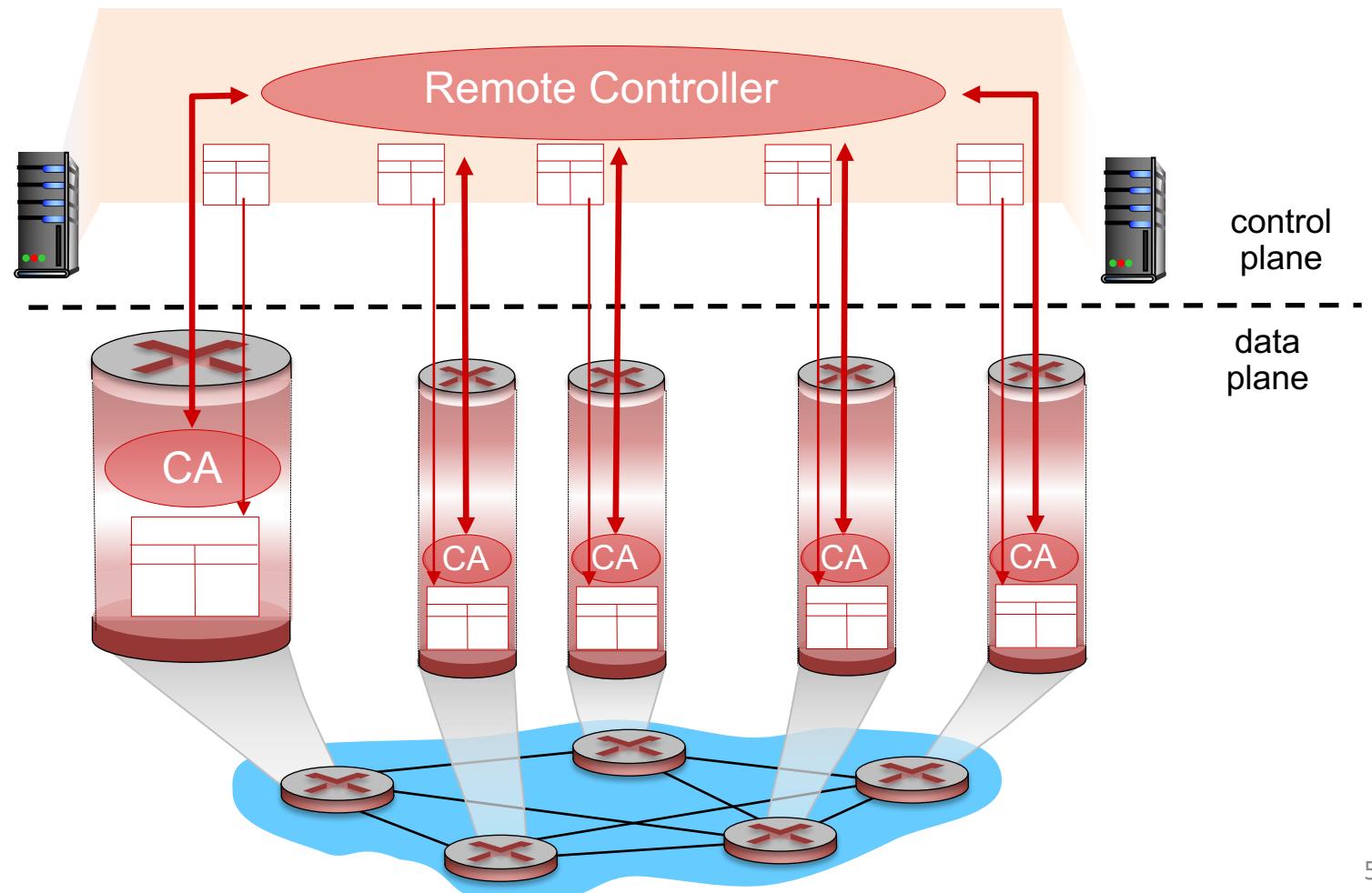
Recall: per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables

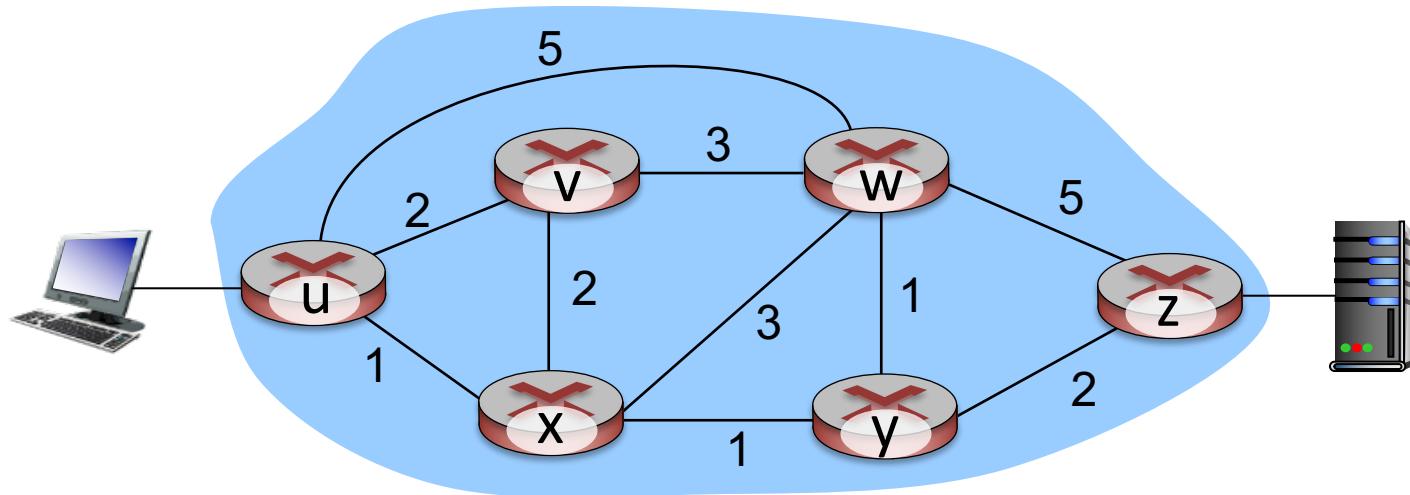


Recall: logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



Traffic engineering: difficult traditional routing

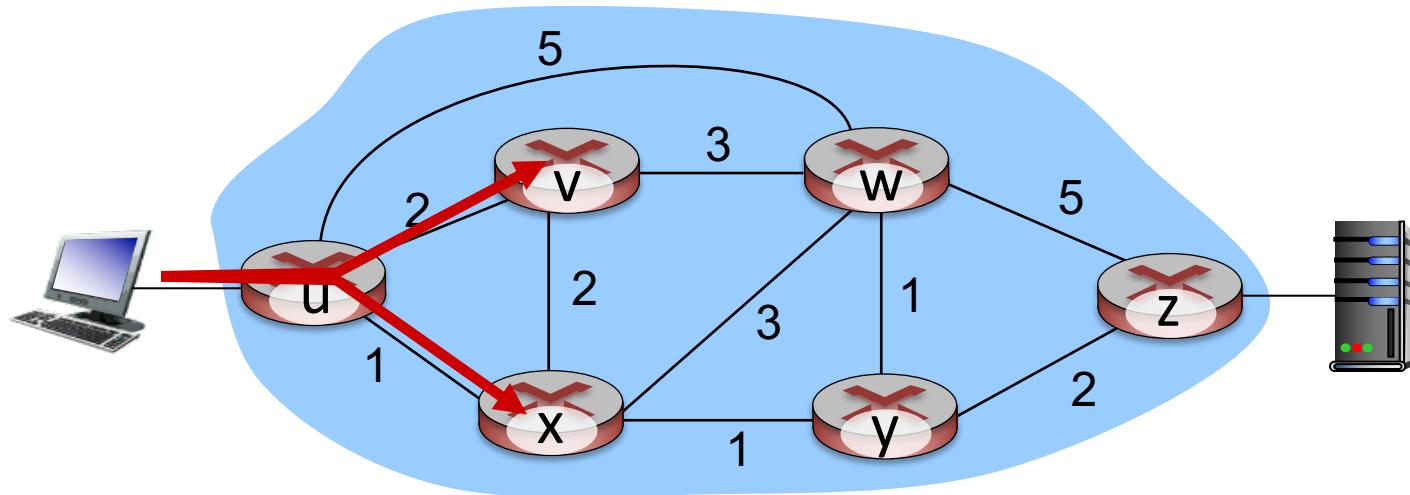


Q: what if network operator wants u-to-z traffic to flow along $uvwz$, x-to-z traffic to flow $xwyz$?

A: need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

Link weights are only control “knobs”: wrong!

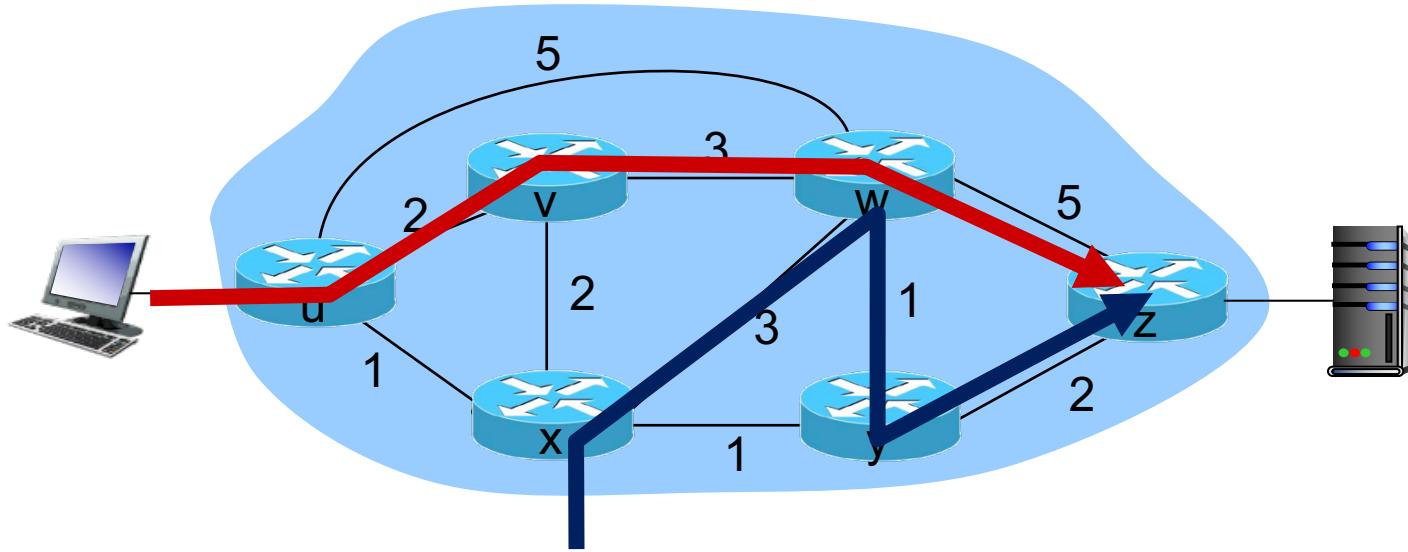
Traffic engineering: difficult



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

A: can't do it (or need a new routing algorithm)

Traffic engineering: difficult



Q: what if w wants to route blue and red traffic differently?

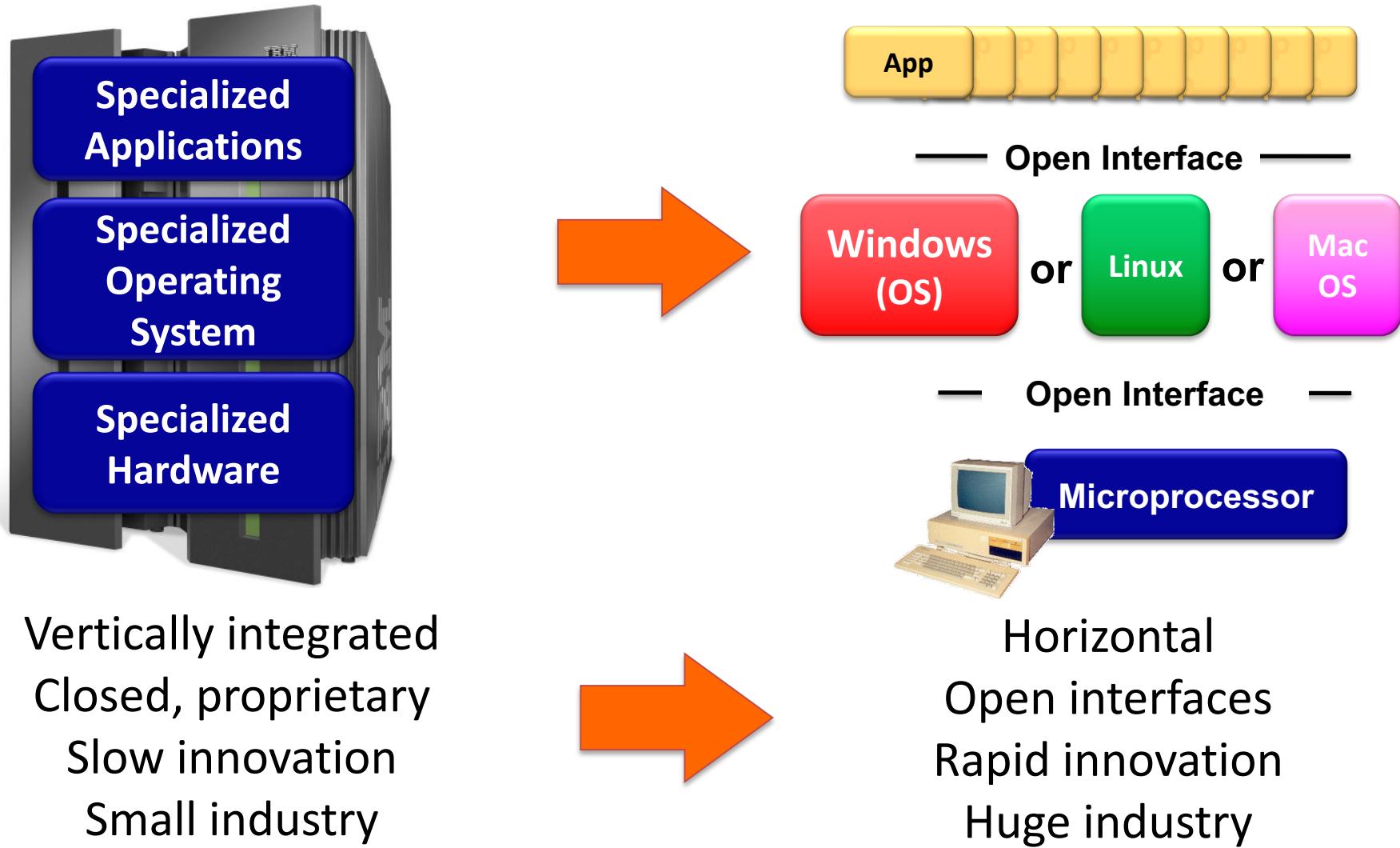
A: can't do it (with destination based forwarding, and LS, DV routing)

Software defined networking (SDN)

Why a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (see generalized forwarding & OpenFlow) allows “programming” routers
 - centralized “programming” easier: compute tables centrally and distribute
 - distributed “programming” more difficult: compute tables as result of distributed algorithm (protocol) implemented in each and every router
- open (non-proprietary) implementation of control plane

Analogy: mainframe to PC evolution*



Software defined networking (SDN)

4. programmable control applications

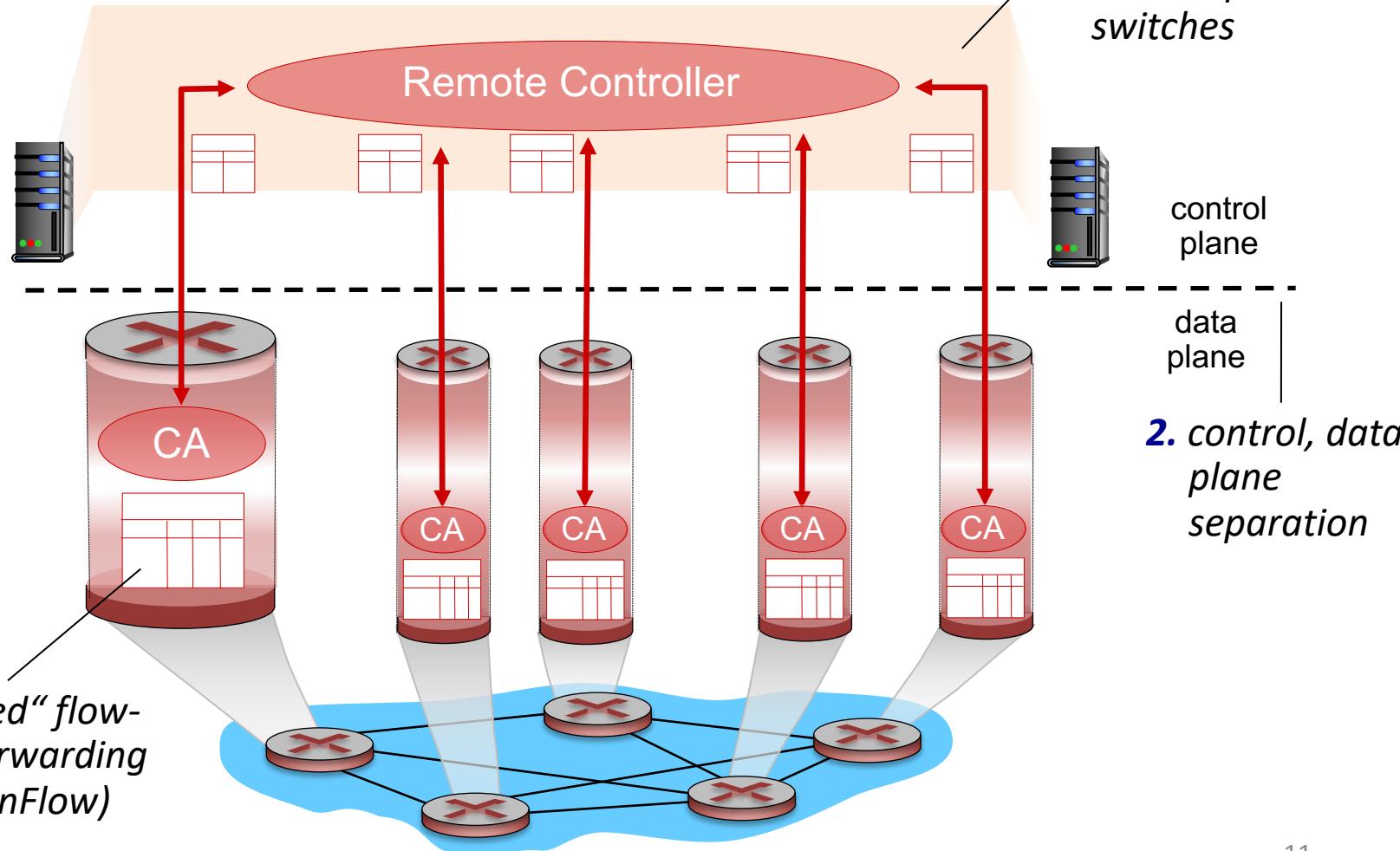
routing

access control

...

load balance

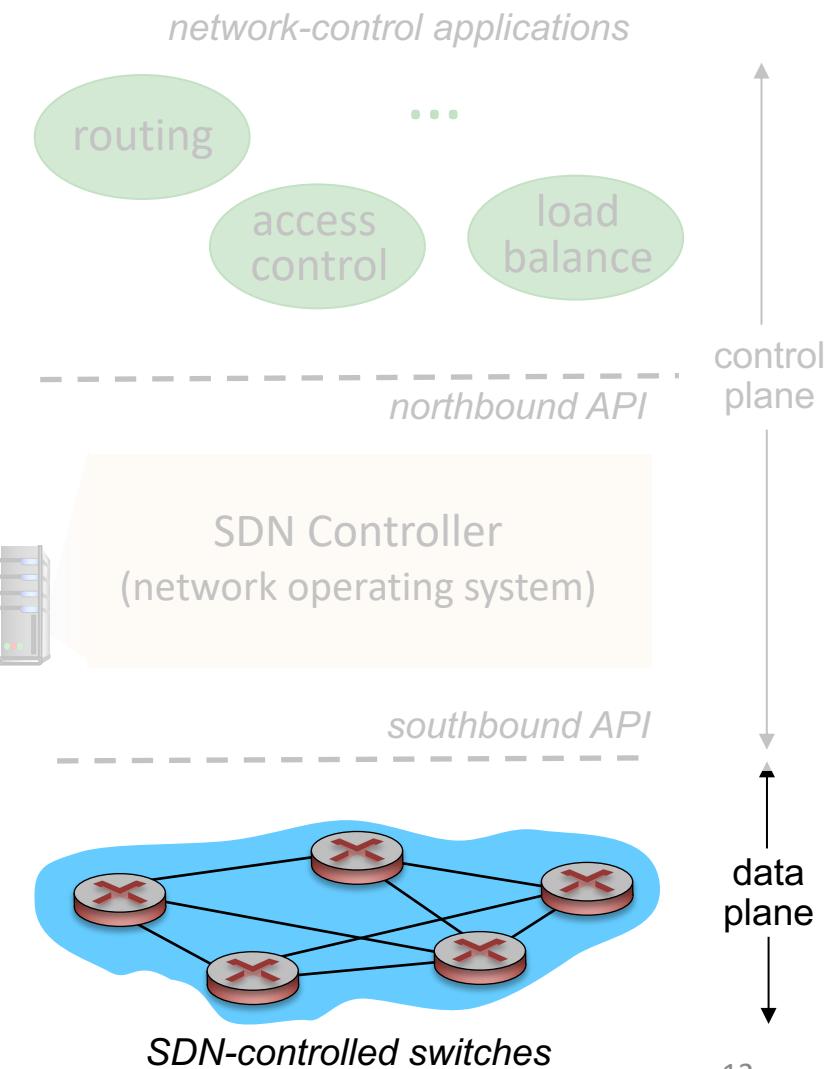
3. control plane functions external to data-plane switches



SDN perspective: data plane switches

Data plane switches

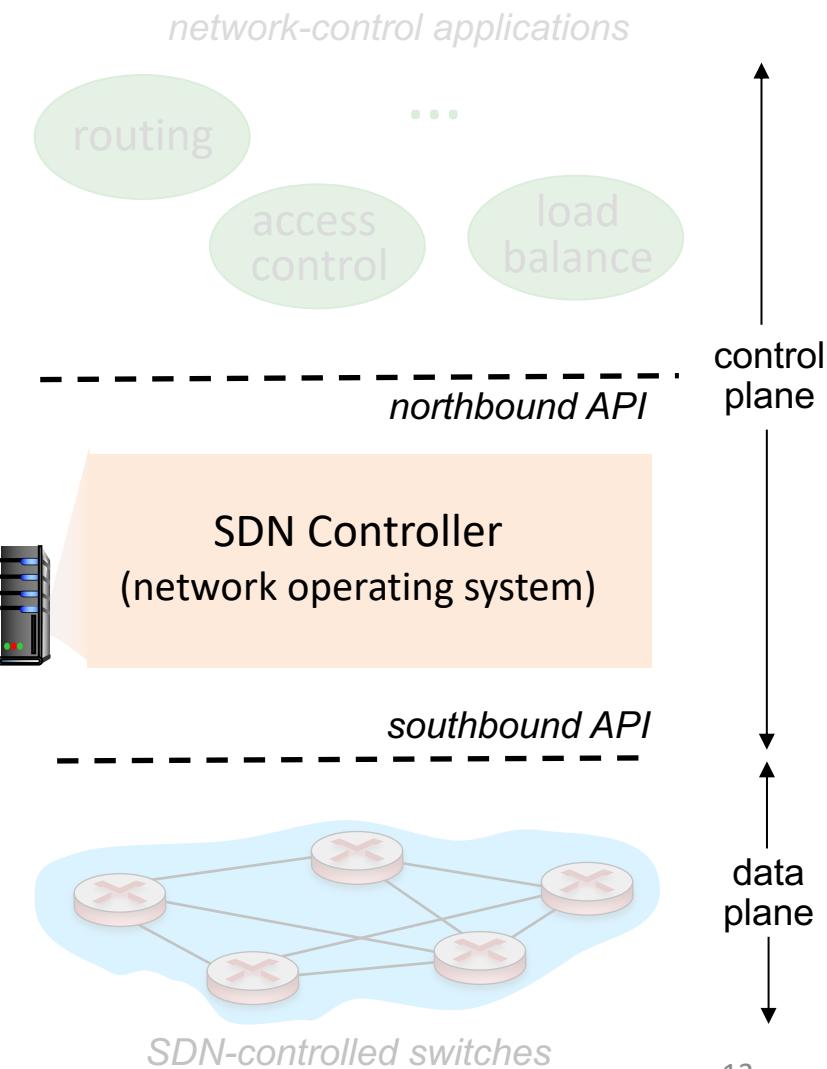
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)



SDN perspective: SDN controller

SDN controller (network OS):

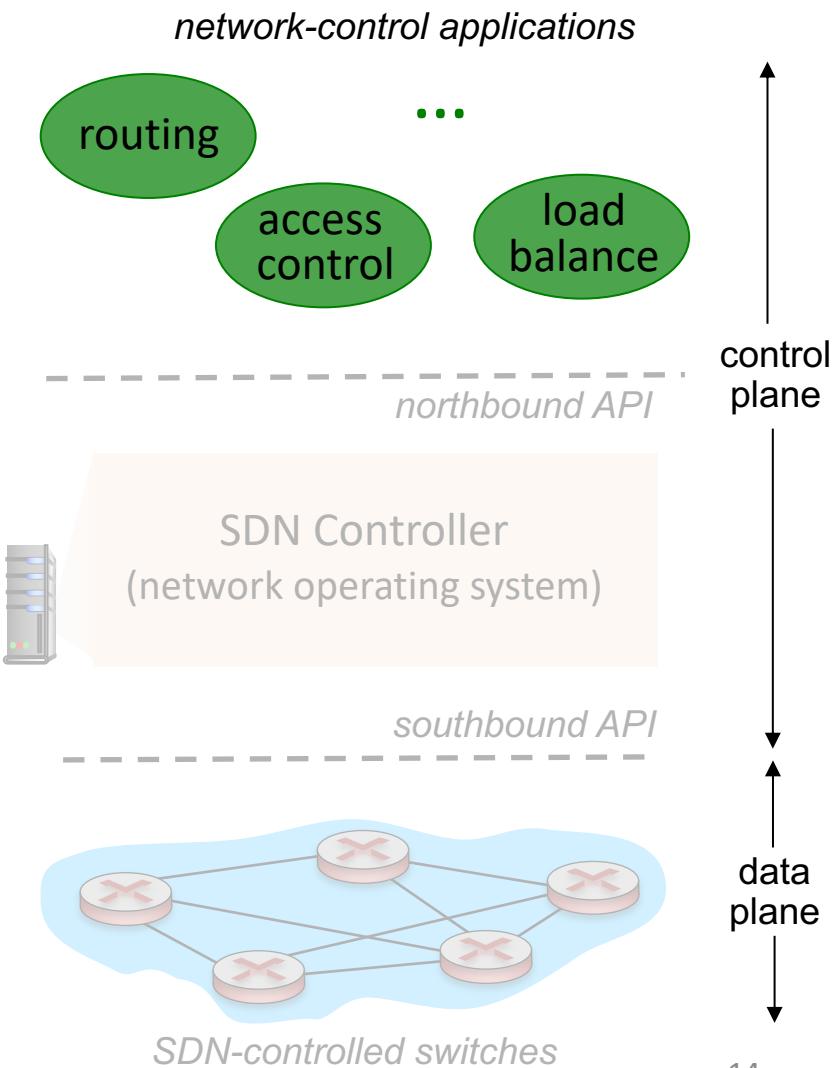
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



SDN perspective: control applications

network-control apps:

- “brains” of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled*: can be provided by 3rd party: distinct from routing vendor, or SDN controller

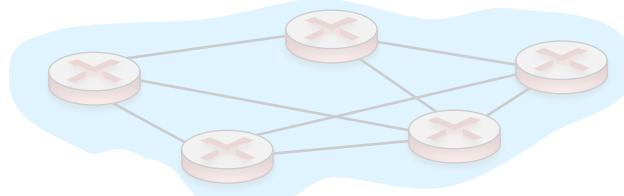
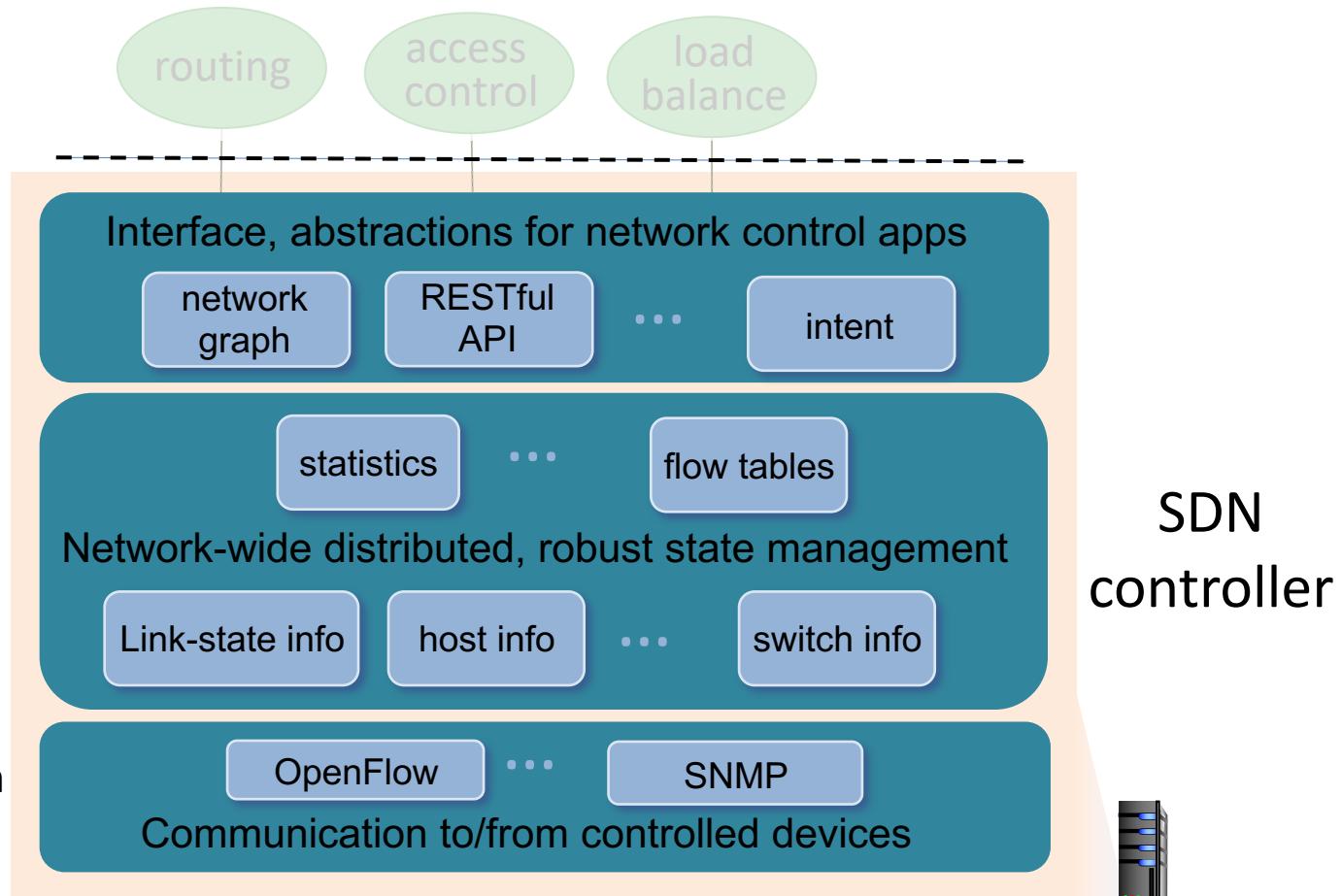


Components of SDN controller

Interface layer to network control apps: abstractions API

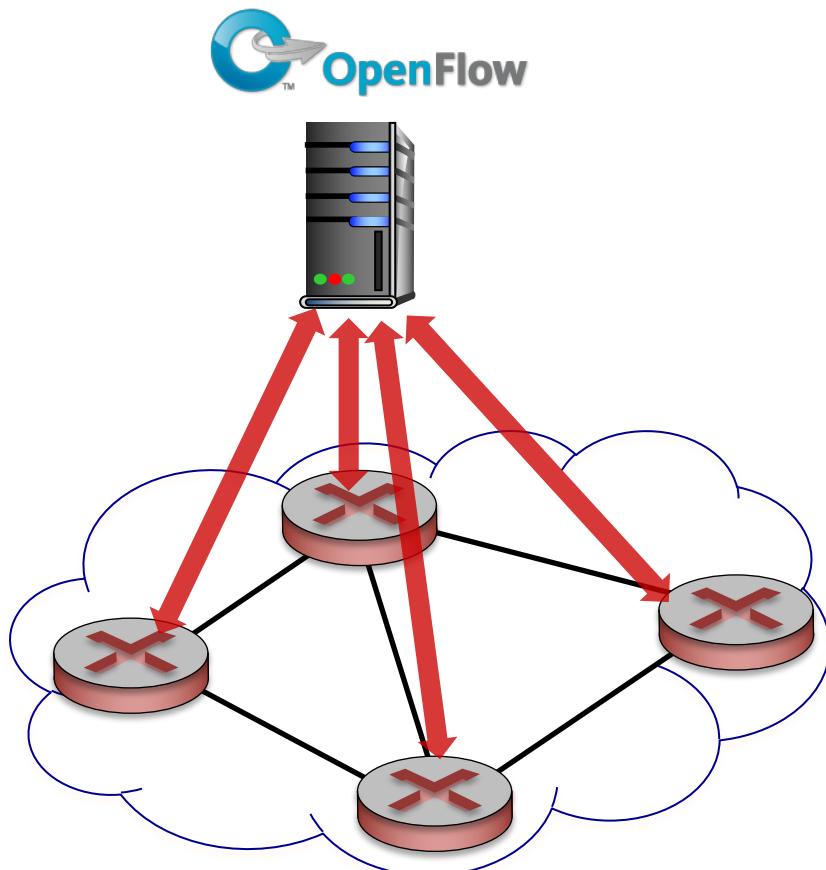
Network-wide state management layer: state of networks links, switches, services: a *distributed database*

communication layer: communicate between SDN controller and controlled switches



OpenFlow protocol

OpenFlow Controller

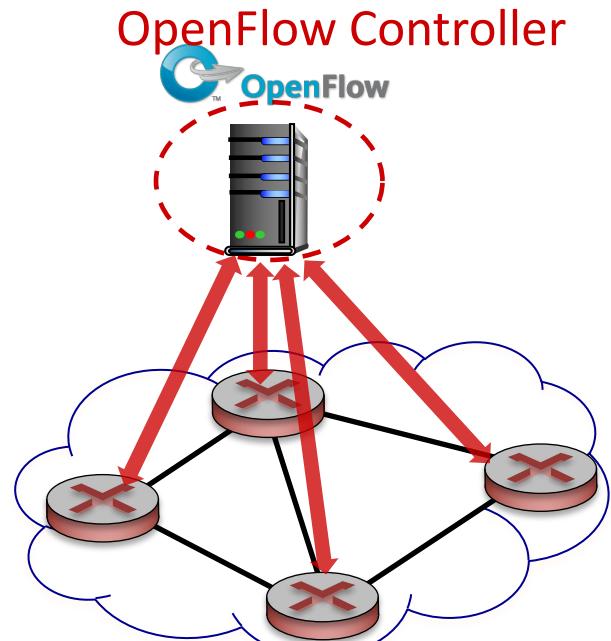


- operates between controller and switch
- TCP used to exchange messages
 - Default port: 6653
- three classes of OpenFlow messages:
 - controller-to-switch
 - asynchronous (switch to controller)
 - symmetric (misc)

OpenFlow: controller-to-switch messages

Key controller-to-switch messages

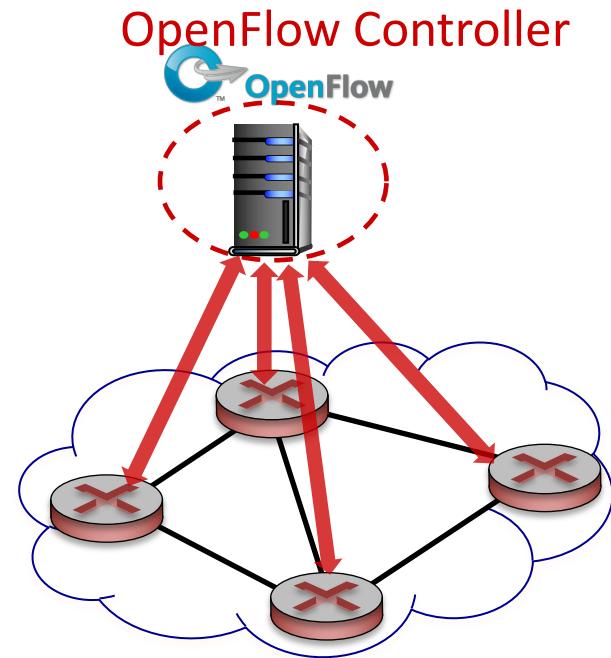
- ***configuration***: controller queries/sets switch configuration parameters
- ***modify-state***: add, delete, modify flow entries in the OpenFlow tables
- ***read-state***: collect statistics and counter values from the flow table and ports
- ***send-packet***: sends a specific packet (in its payload) out of a specific switch port



OpenFlow: switch-to-controller messages

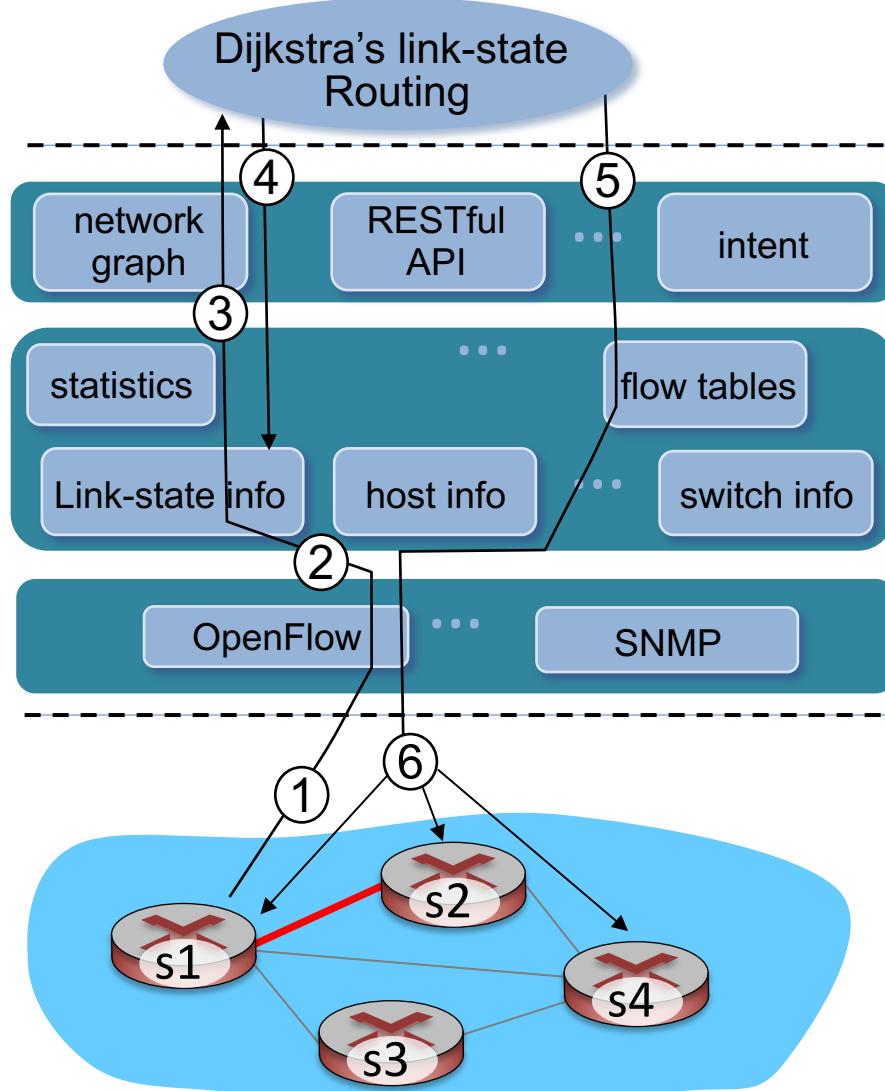
Key switch-to-controller messages

- *packet-in*: transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed*: flow table entry deleted at switch
- *port status*: inform controller of a change on a port.



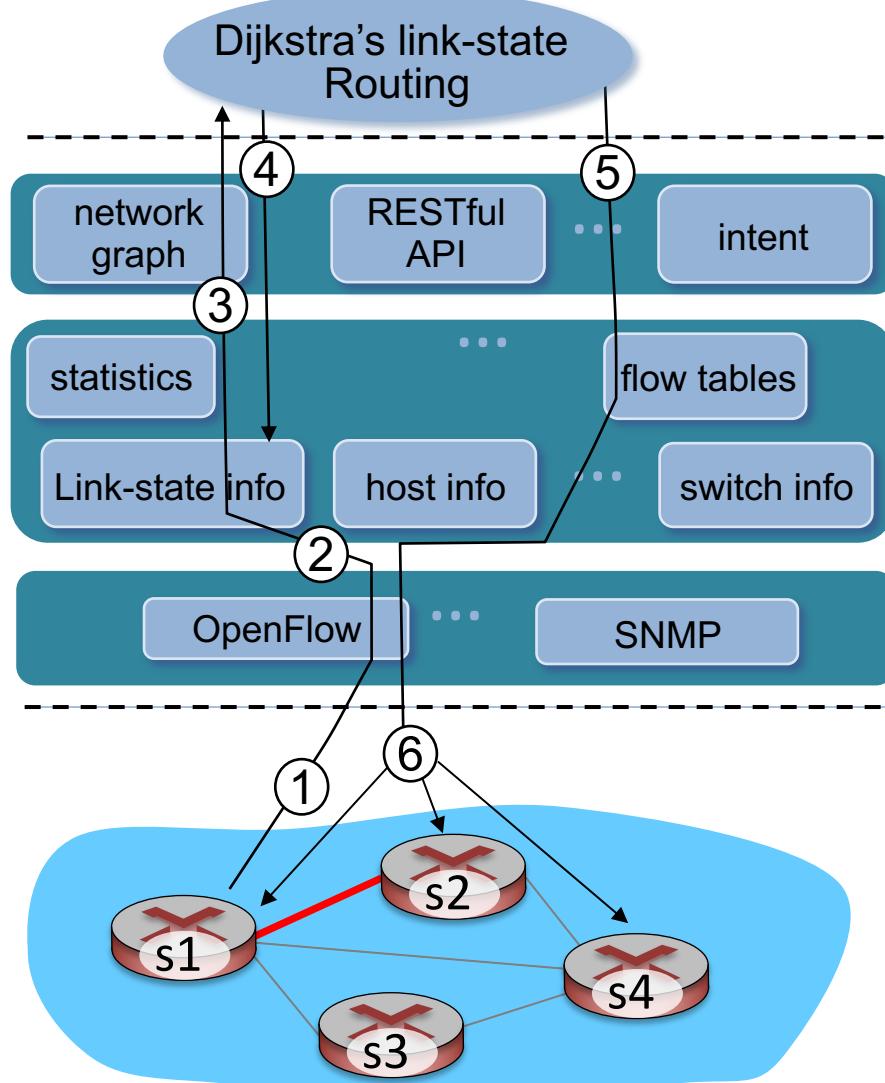
Fortunately, network operators don't “program” switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

SDN: control/data plane interaction example



- ① S1, experiencing link failure using OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called whenever link status changes. It is called.
- ④ Dijkstra's routing algorithm accesses network graph info, link state info in controller, computes new routes

SDN: control/data plane interaction example



- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ Controller uses OpenFlow to install new tables in switches that need updating

Outline

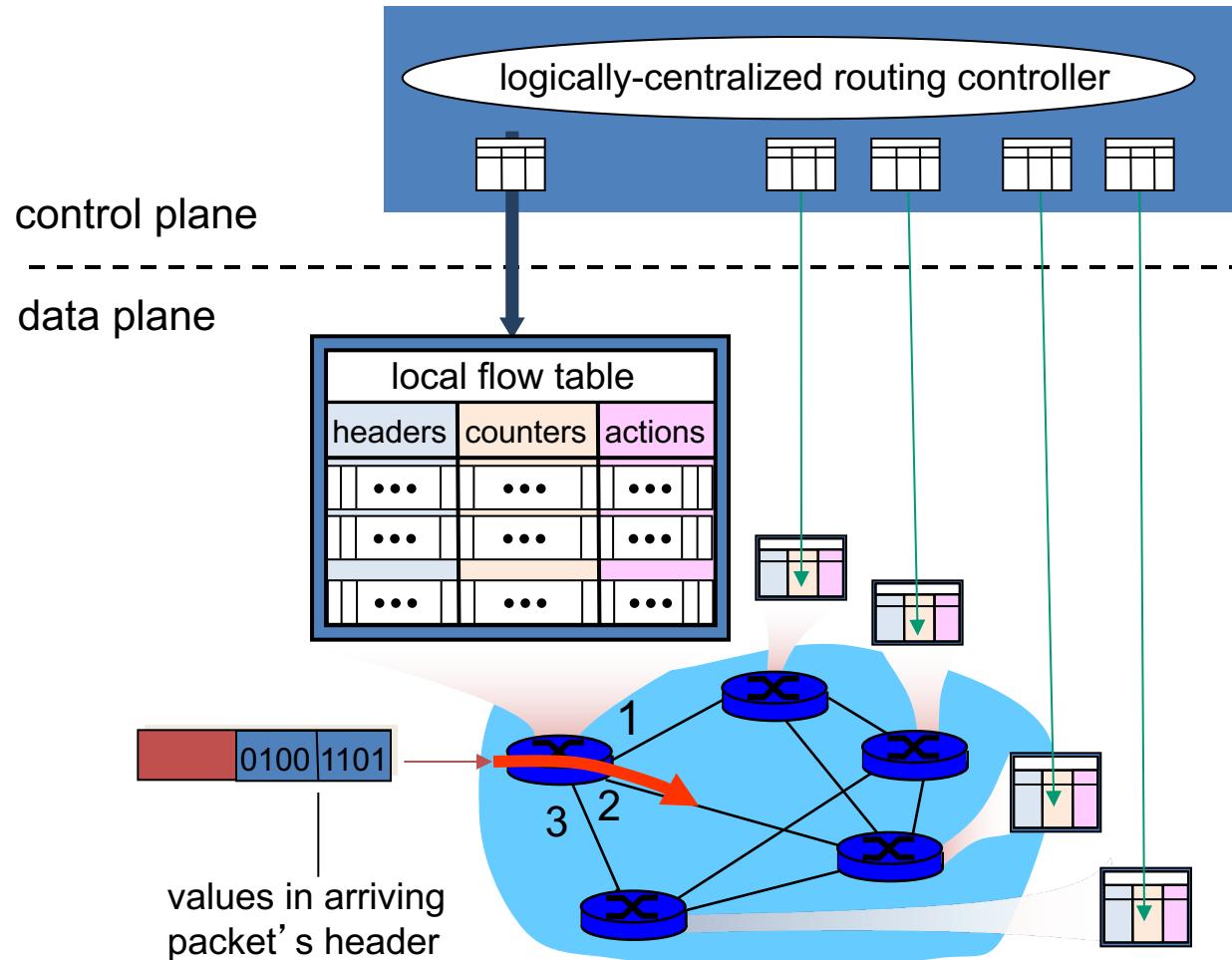
The SDN control plane

Generalized Forwarding

- match
- action
- OpenFlow examples of match-plus-action in action

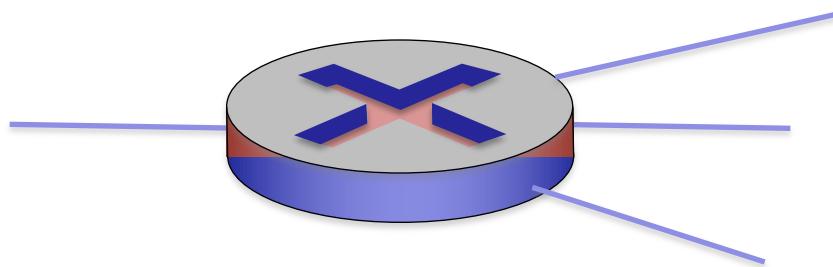
Generalized Forwarding and SDN

Each router contains a *flow table* that is computed and distributed by a *logically centralized routing controller*



OpenFlow data plane abstraction

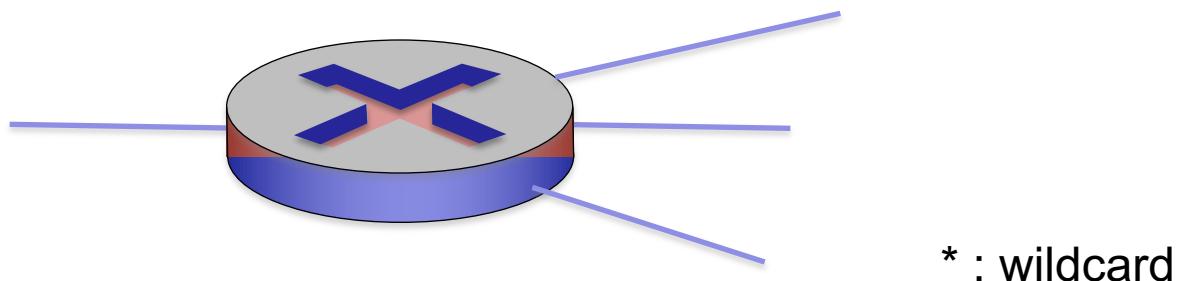
- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
 - *Pattern*: match values in packet header fields
 - *Actions*: *for matched packet*: drop, forward, modify, matched packet or send matched packet to controller
 - *Priority*: disambiguate overlapping patterns
 - *Counters*: #bytes and #packets



Flow table in a router (computed and distributed by controller) define router's match+action rules

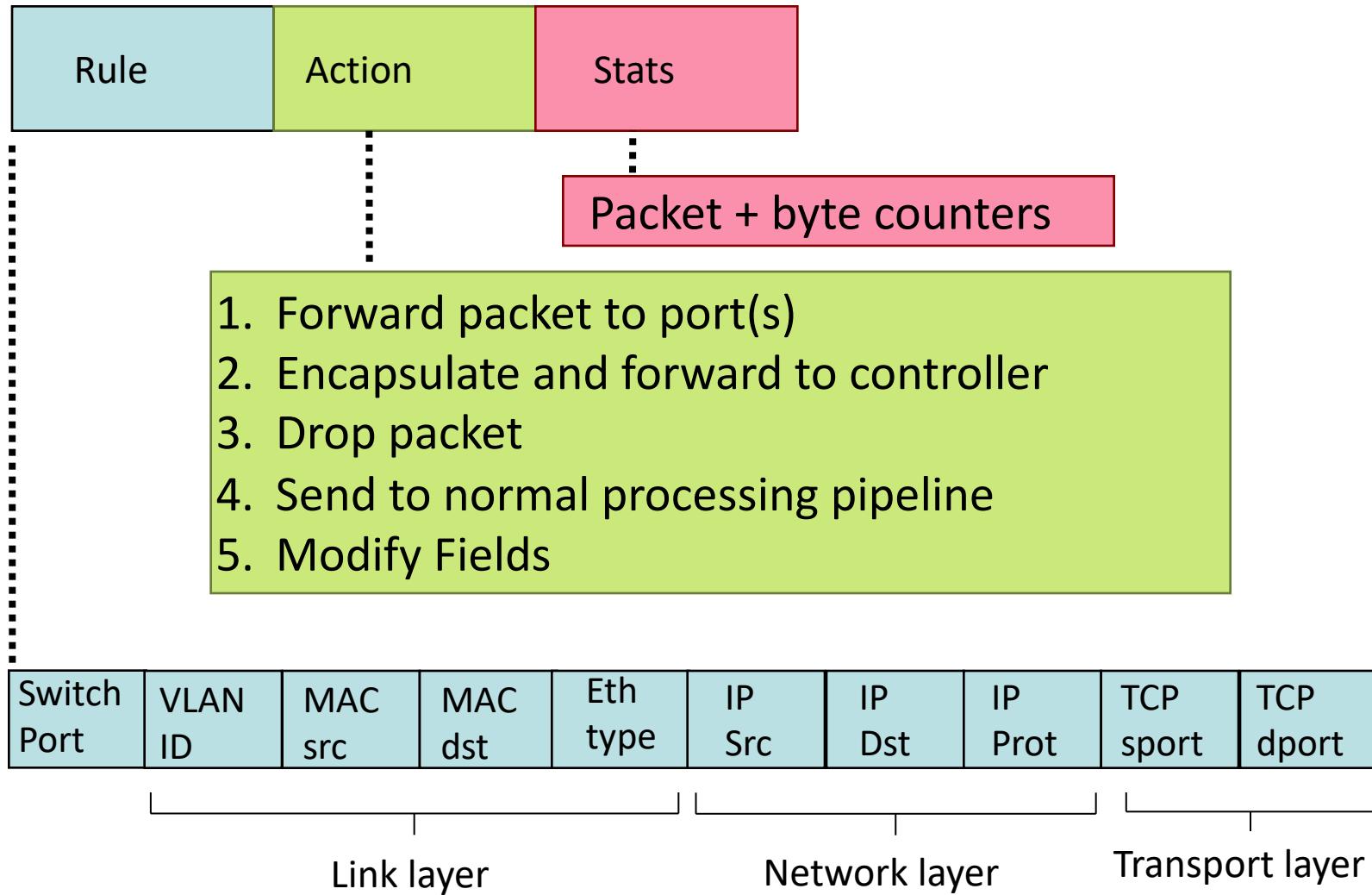
OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
 - *Pattern*: match values in packet header fields
 - *Actions*: *for matched packet*: drop, forward, modify, matched packet or send matched packet to controller
 - *Priority*: disambiguate overlapping patterns
 - *Counters*: #bytes and #packets



1. $\text{src}=1.2.*.*$, $\text{dest}=3.4.5.* \rightarrow \text{drop}$
2. $\text{src} = *.*.*.*$, $\text{dest}=3.4.*.* \rightarrow \text{forward}(2)$
3. $\text{src}=10.1.2.3$, $\text{dest} = *.*.*.* \rightarrow \text{send to controller}$

OpenFlow: Flow Table Entries



Examples

Destination-based forwarding:

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

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

do not forward (block) all datagrams sent by host 128.119.1.1

Examples

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	port3

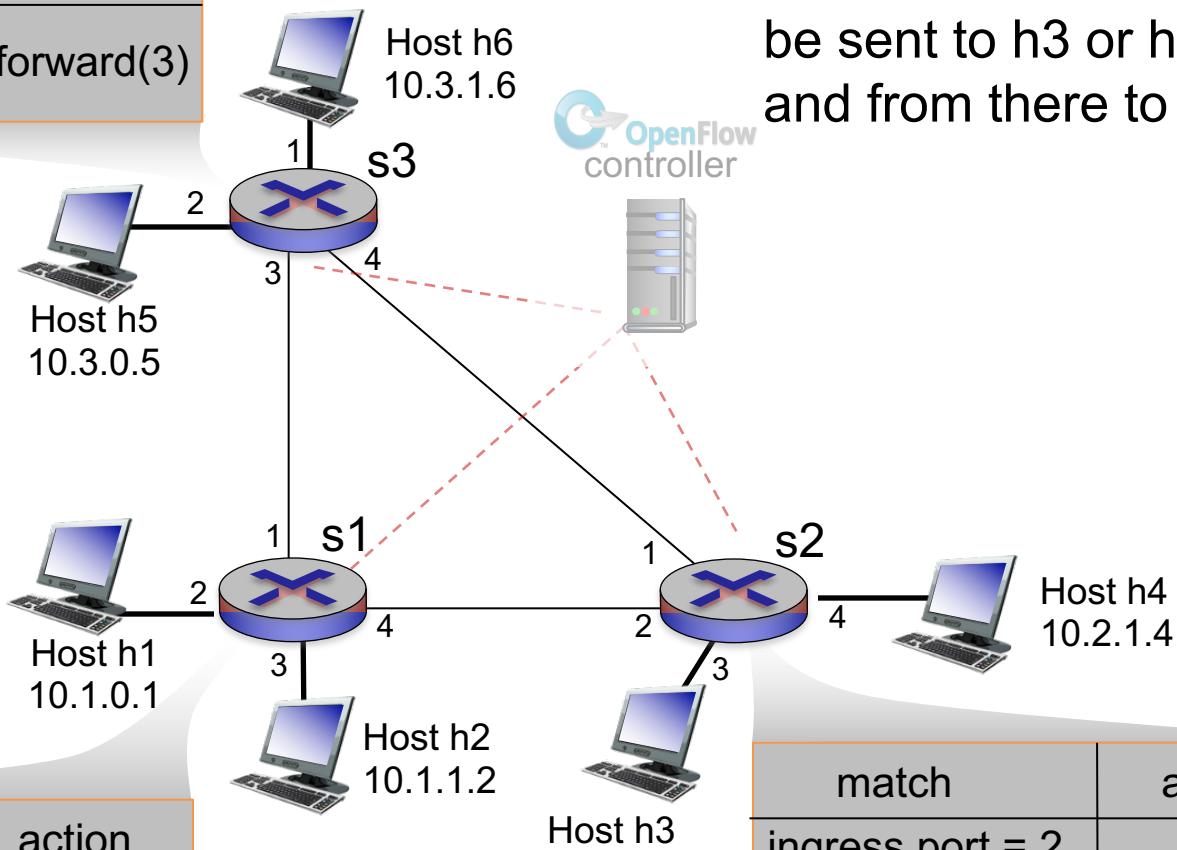
*layer 2 frames from MAC address 22:A7:23:11:E1:02
should be forwarded to output port 3*

OpenFlow abstraction

- *match+action*: unifies different kinds of devices
- Router
 - *match*: longest destination IP prefix
 - *action*: forward out a link
- Switch
 - *match*: destination MAC address
 - *action*: forward or flood
- Firewall
 - *match*: IP addresses and TCP/UDP port numbers
 - *action*: permit or deny
- NAT
 - *match*: IP address and port
 - *action*: rewrite address and port

OpenFlow example

match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)



Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

match	action
ingress port = 1 IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(4)

match	action
ingress port = 2 IP Dst = 10.2.0.3	forward(3)
ingress port = 2 IP Dst = 10.2.1.4	forward(4)