ILLINOIS

CS 438: Computer Networks (Spring 2026)

Software-Defined Networking (SDN)

Slides adapted from *Computer Networking: A Top-Down Approach.*

Learning Objectives

- A different view of the network layer:
  - data plane and control plane
- Software-defined networking (SDN)

Two key network-layer functions

network-layer functions:

- *forwarding:* move packets from a router's input link to appropriate router output link
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

analogy: taking a trip

- *forwarding:* process of getting through single interchange
- *routing:* process of planning trip from source to destination

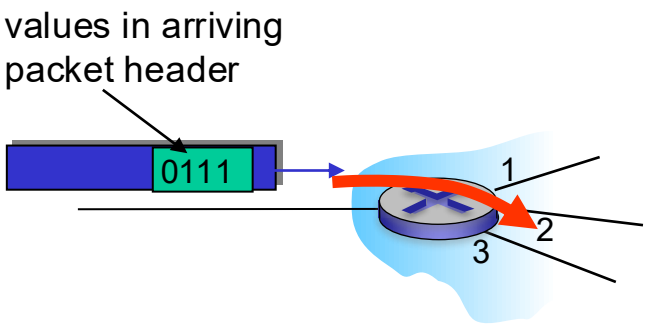
forwarding

routing

Network layer: data plane, control plane

Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port



Control plane

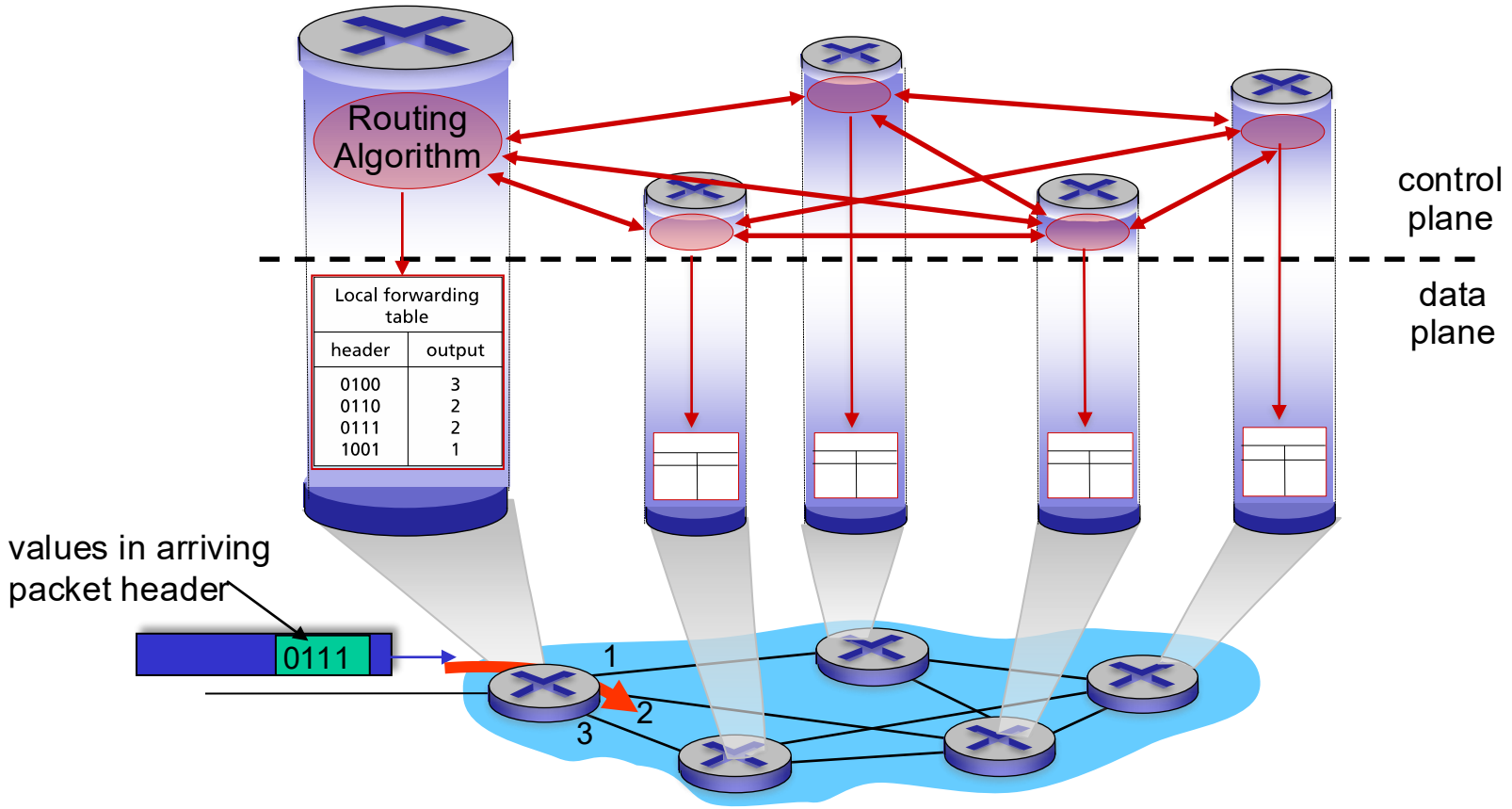
- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

Software defined networking (SDN)

- Internet network layer: historically implemented via distributed, per-router control 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

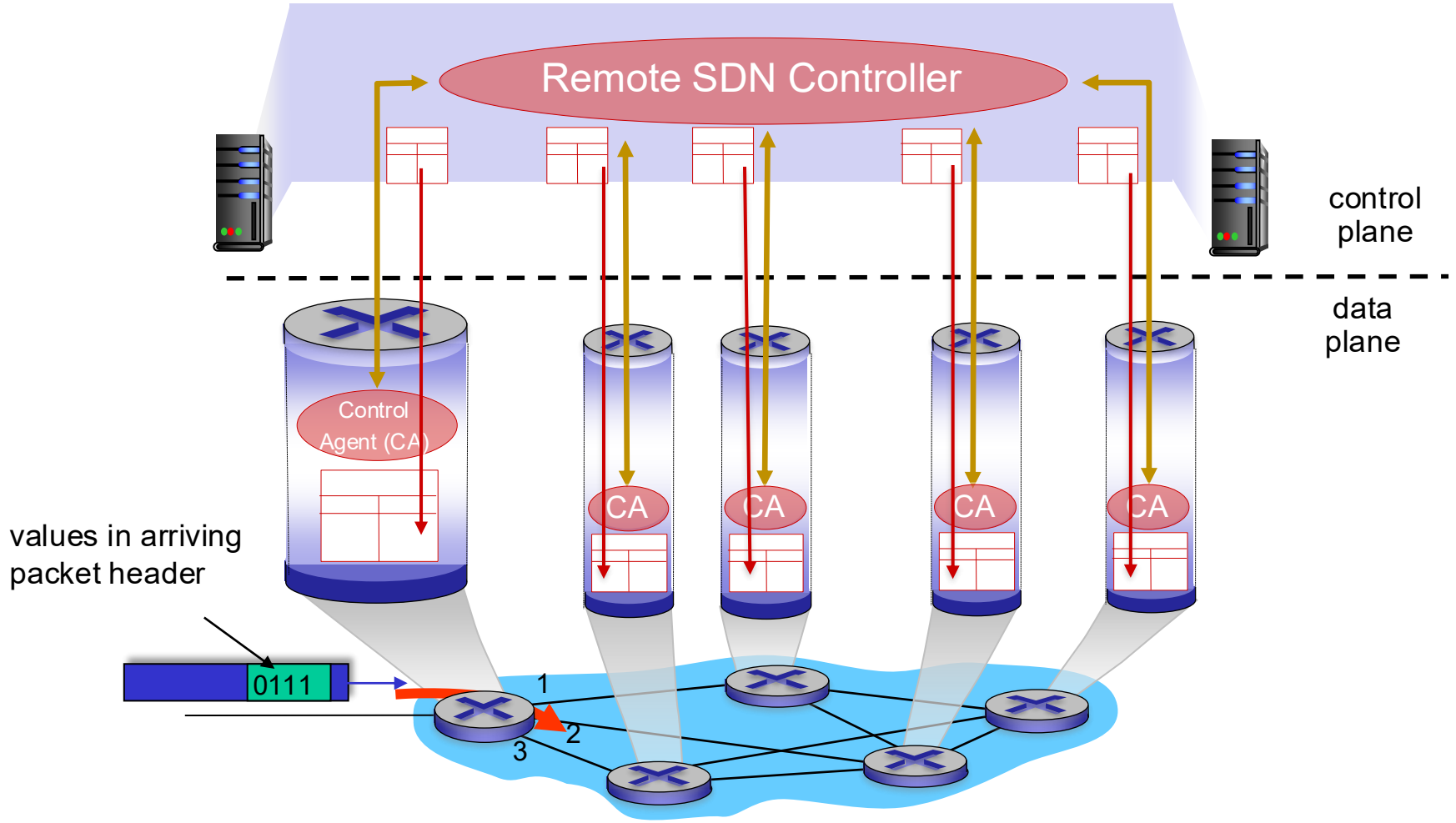
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



SDN overview

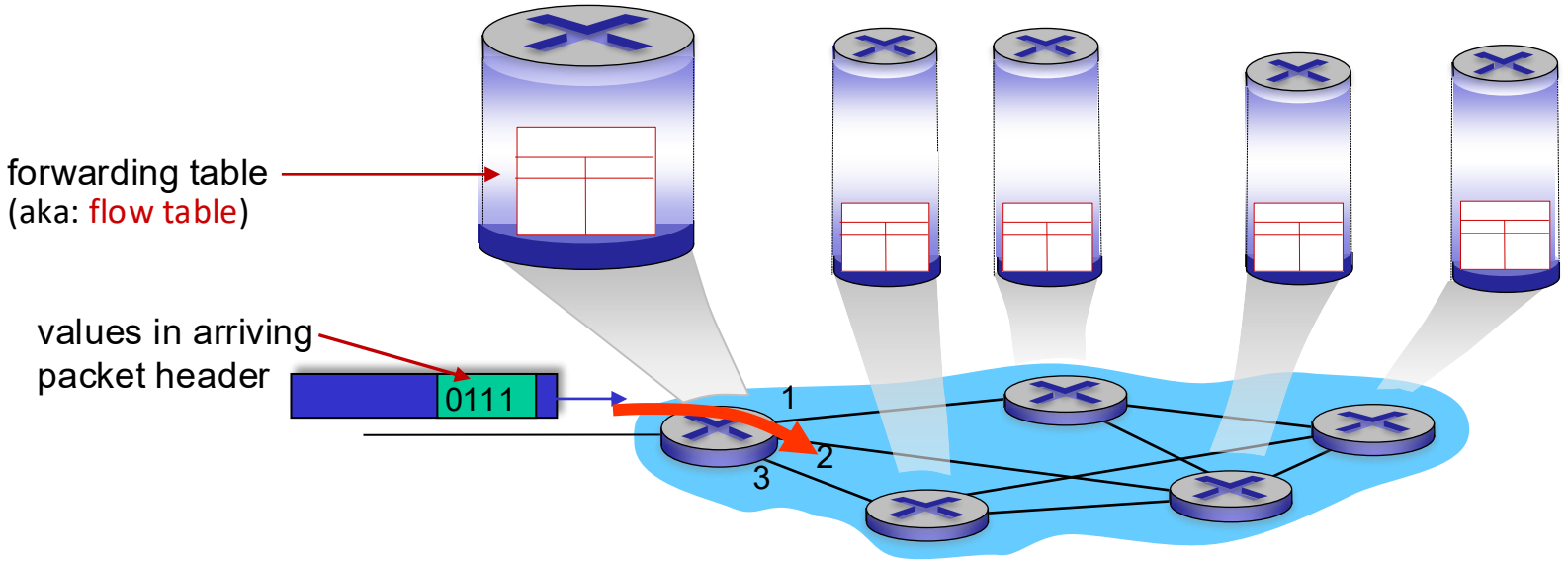
Remote controller computes, installs forwarding tables in routers



Generalized forwarding: SDN data plane

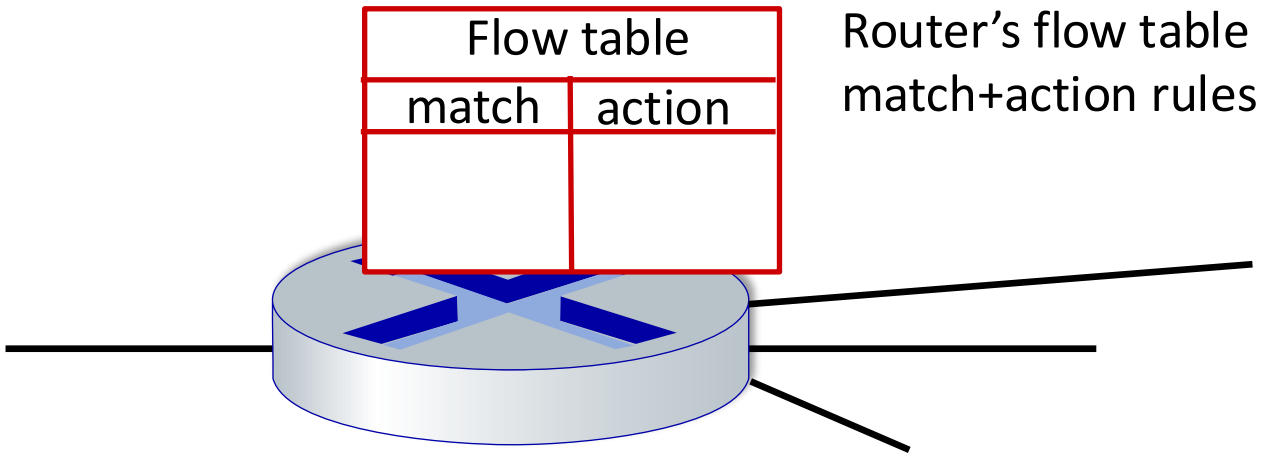
Each router/switch contains a forwarding table (aka: flow table)

- "match plus action" abstraction: match bits in arriving packet, take action
  - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding*
    - many header fields can determine action
    - many actions possible: drop/copy/modify/log packet



Flow table abstraction

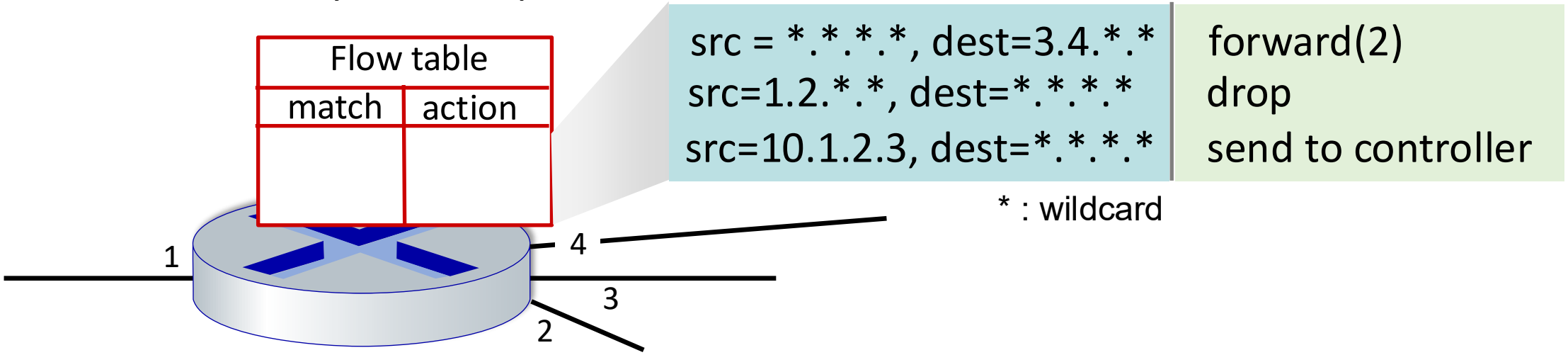
- flow: defined by header field values (in link-, network-, transport-layer fields)
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - priority: disambiguate overlapping patterns
  - counters: #bytes and #packets



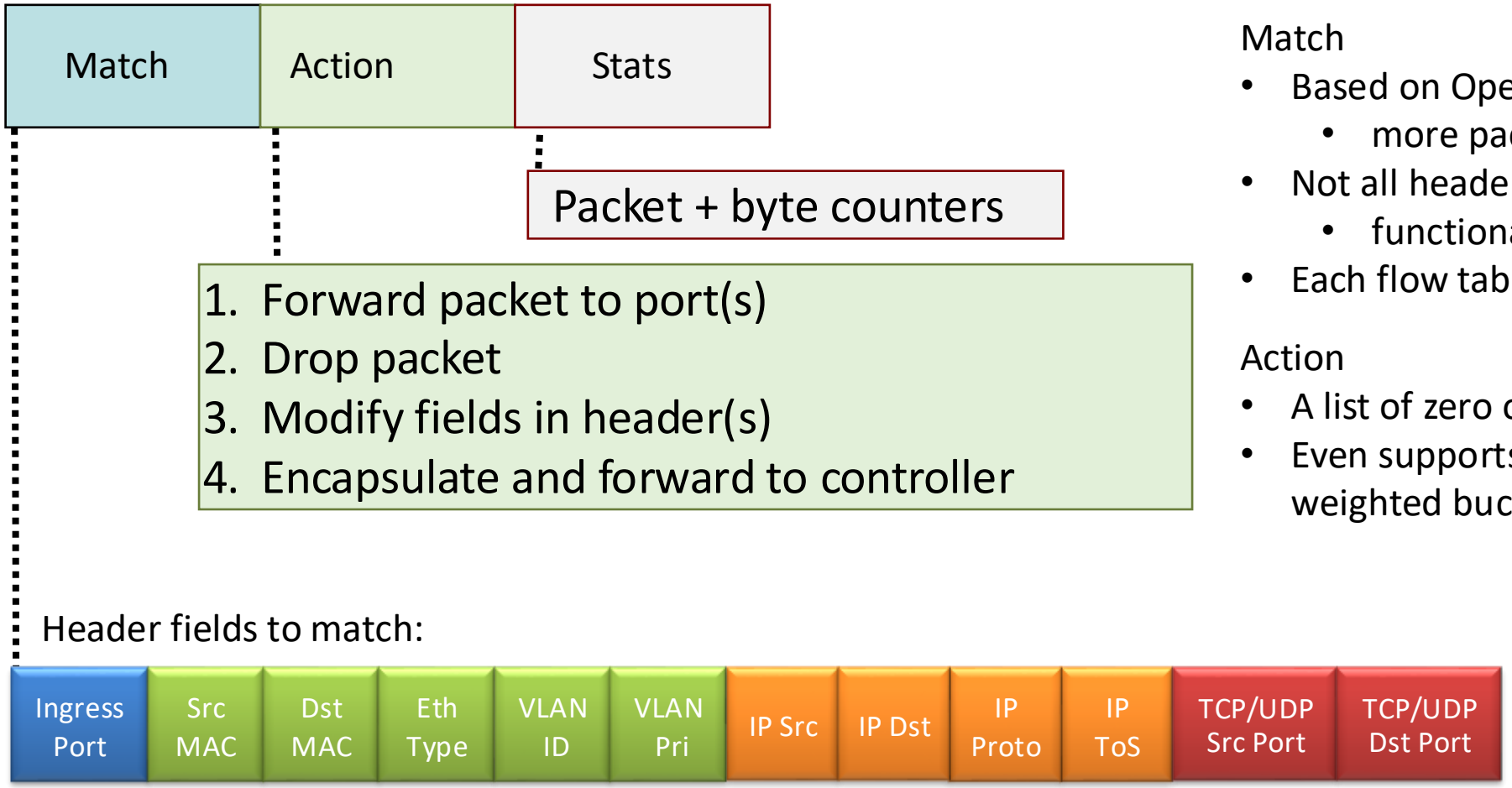
Router's flow table defines match+action rules

Flow table abstraction

- flow: defined by header fields
- generalized forwarding: simple packet-handling rules
  - match: pattern 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



OpenFlow: flow table entries



Match

- Based on OpenFlow 1.0
  - more packet matching fields later
- Not all header fields can be matched
  - functionality vs. complexity
- Each flow table entry has a priority

Action

- A list of zero or more actions
- Even supports action groups with weighted buckets

OpenFlow: examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	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	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

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

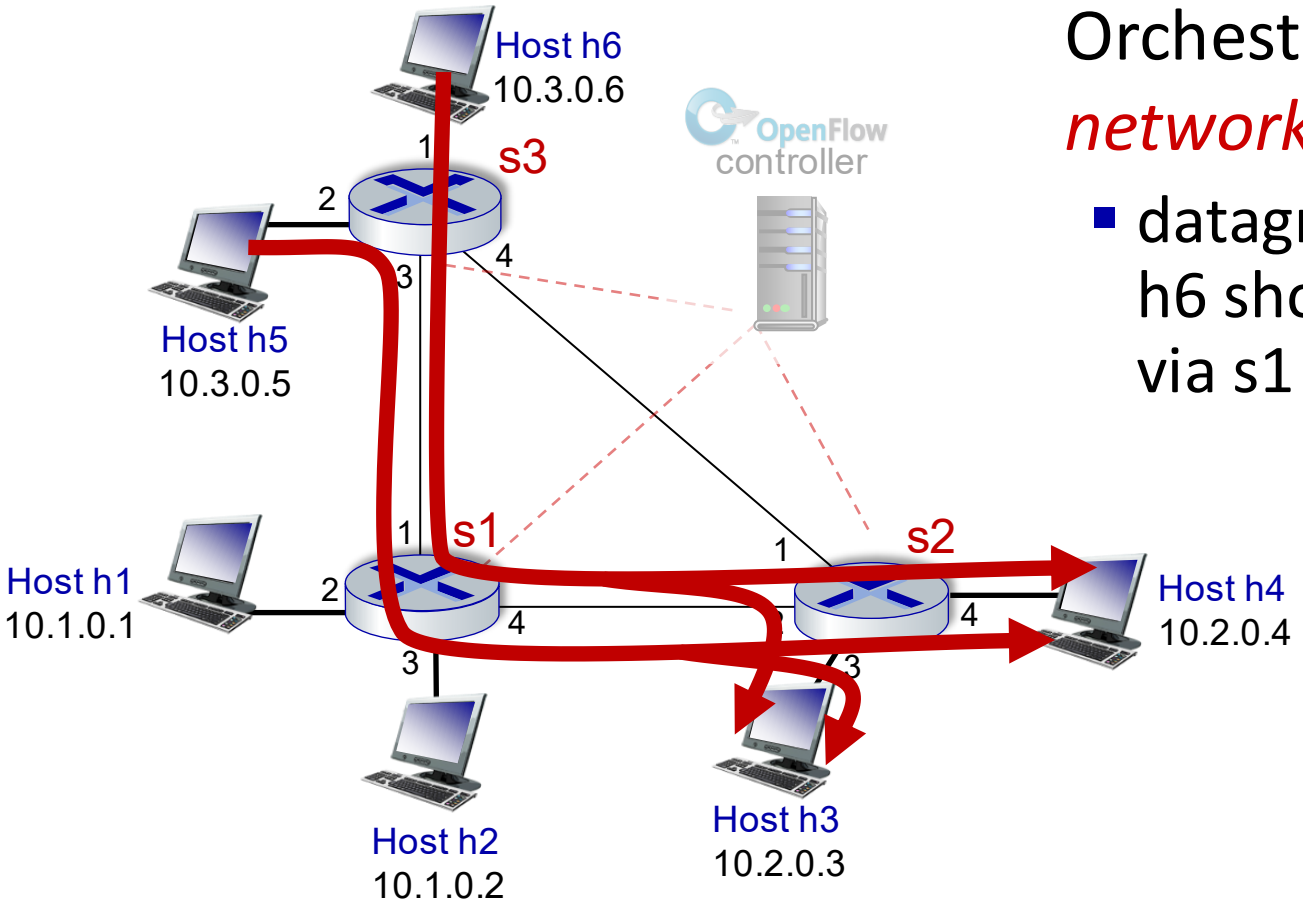
OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23:11:E1:02	*	*	*	*	*	*	*	*	*	port3

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

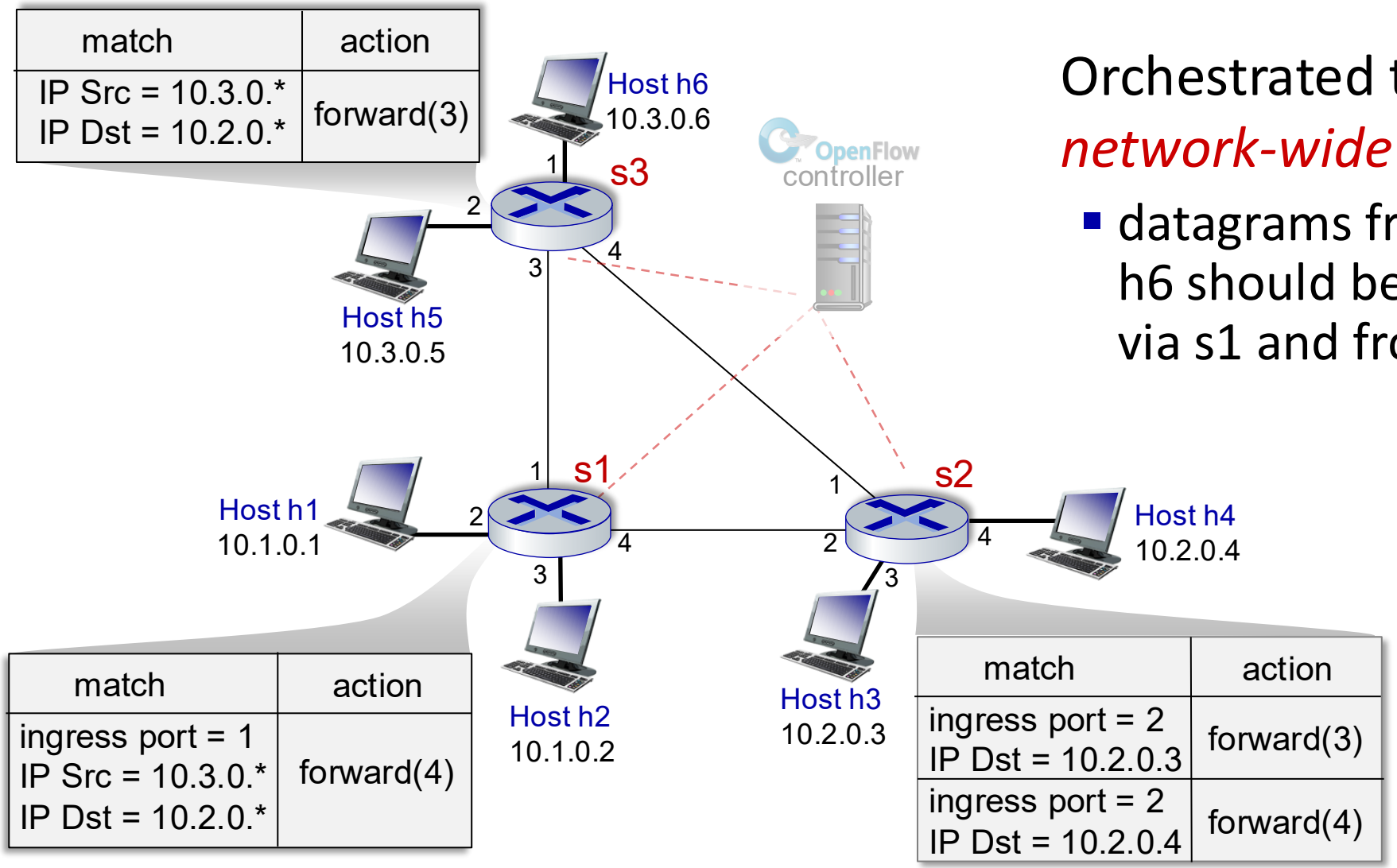
OpenFlow example



Orchestrated tables can create *network-wide* behavior, e.g.,:

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

OpenFlow example



Orchestrated tables can create *network-wide* behavior, e.g.,:

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

OpenFlow abstraction

- match+action: abstraction 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

SDN data plane: summary

- "match plus action" abstraction: match bits in arriving packet header(s) in any layers, take action
  - matching over many fields (link-, network-, transport-layer)
  - local actions: drop, forward, modify, or send matched packet to controller
  - "program" n*etwork-wide* behaviors
- simple form of "network programmability"
  - programmable, per-packet "processing"
  - more generalized programming for datagram processing at line rate: P4 (Programming Protocol-independent Packet Processors)

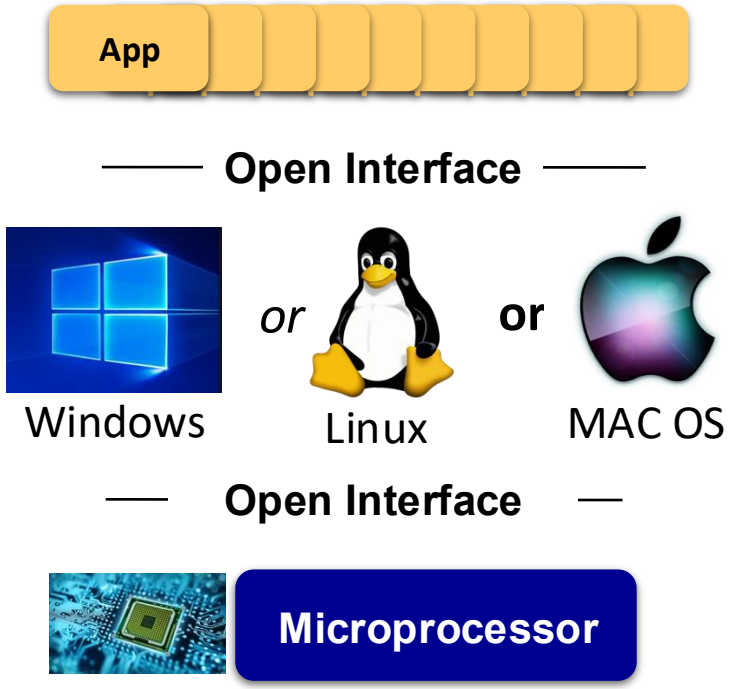
SDN control plane: logically centralized

Why a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding 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
  - foster innovation: let 1000 flowers bloom

SDN analogy: mainframe to PC revolution

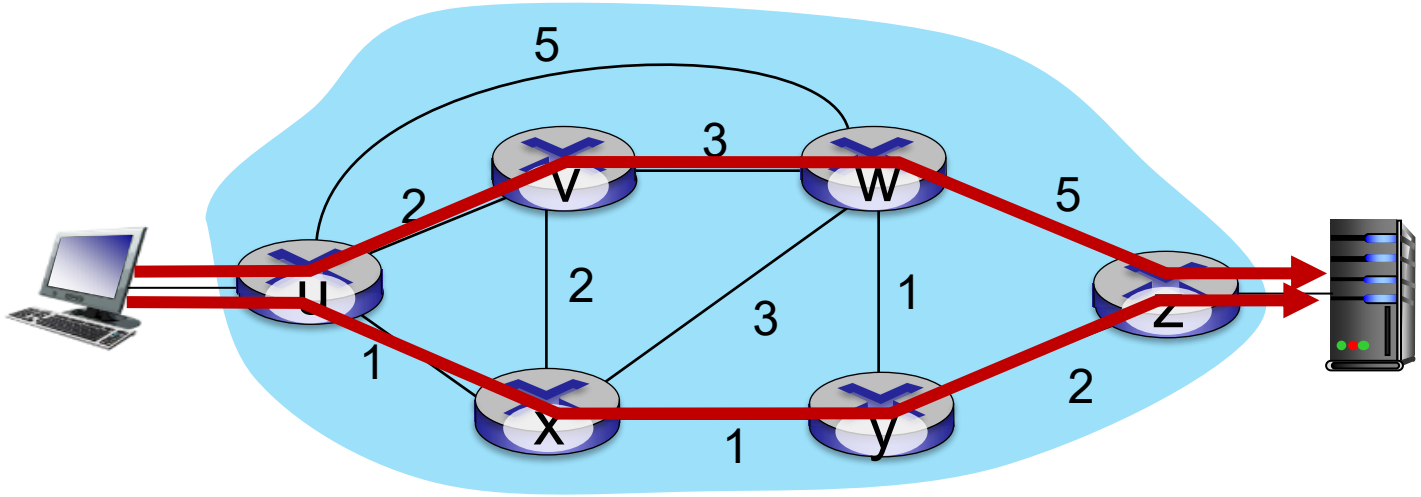
Vertically integrated
Closed, proprietary
Slow innovation
Small industry



Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

Traffic engineering: difficult with traditional routing

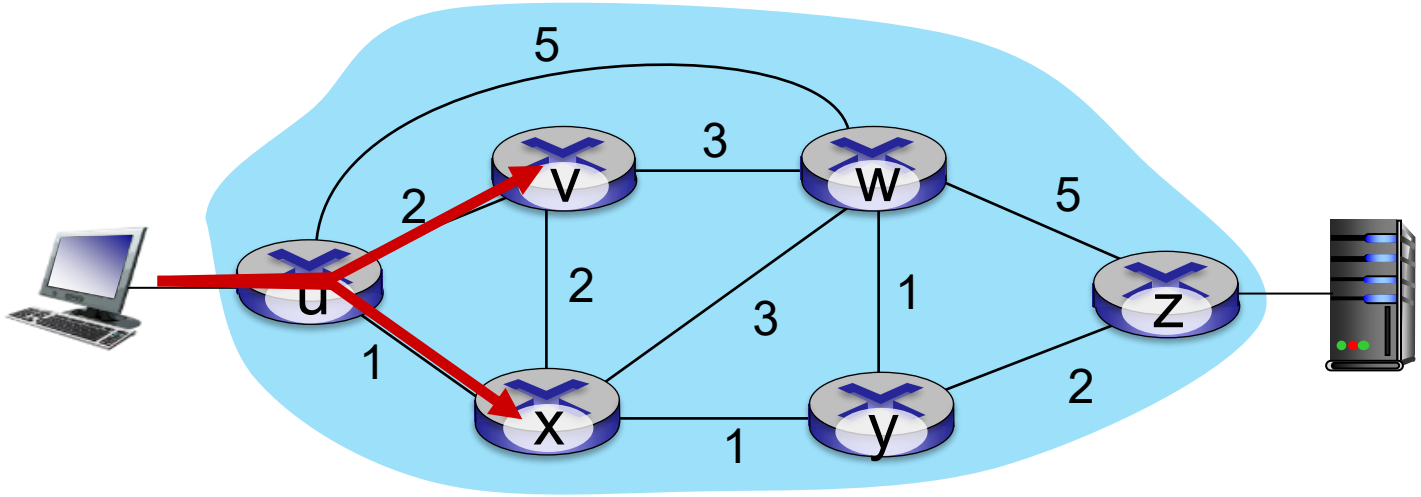


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

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

link weights are only control "knobs": not much control!

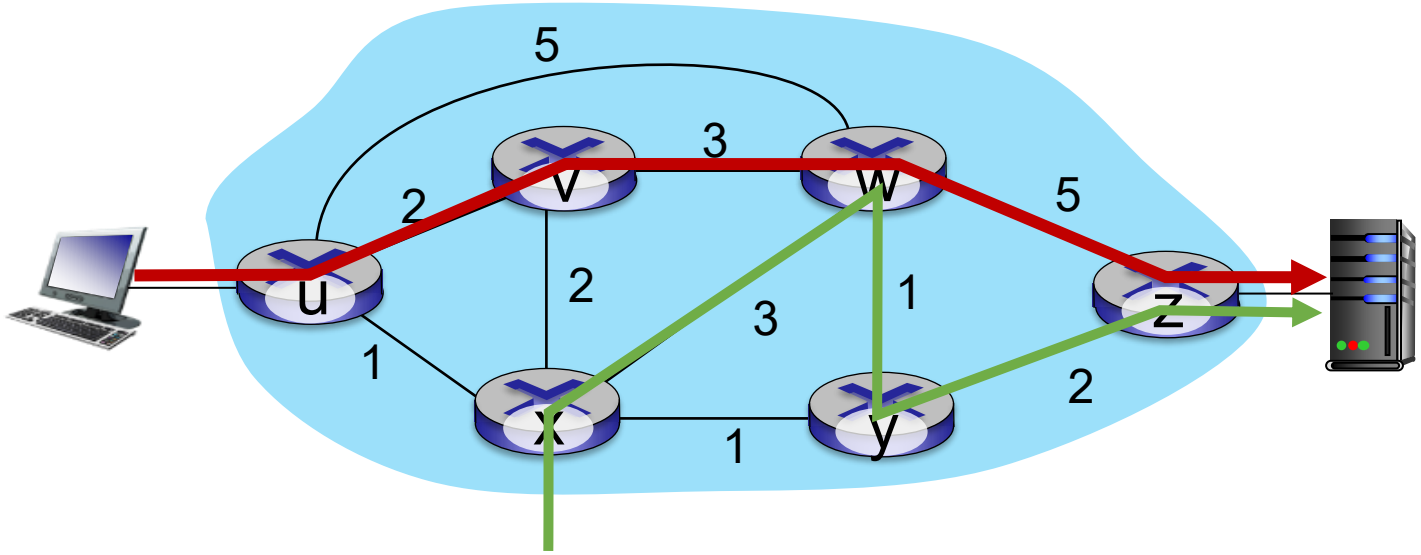
Traffic engineering: difficult with traditional routing



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 with traditional routing



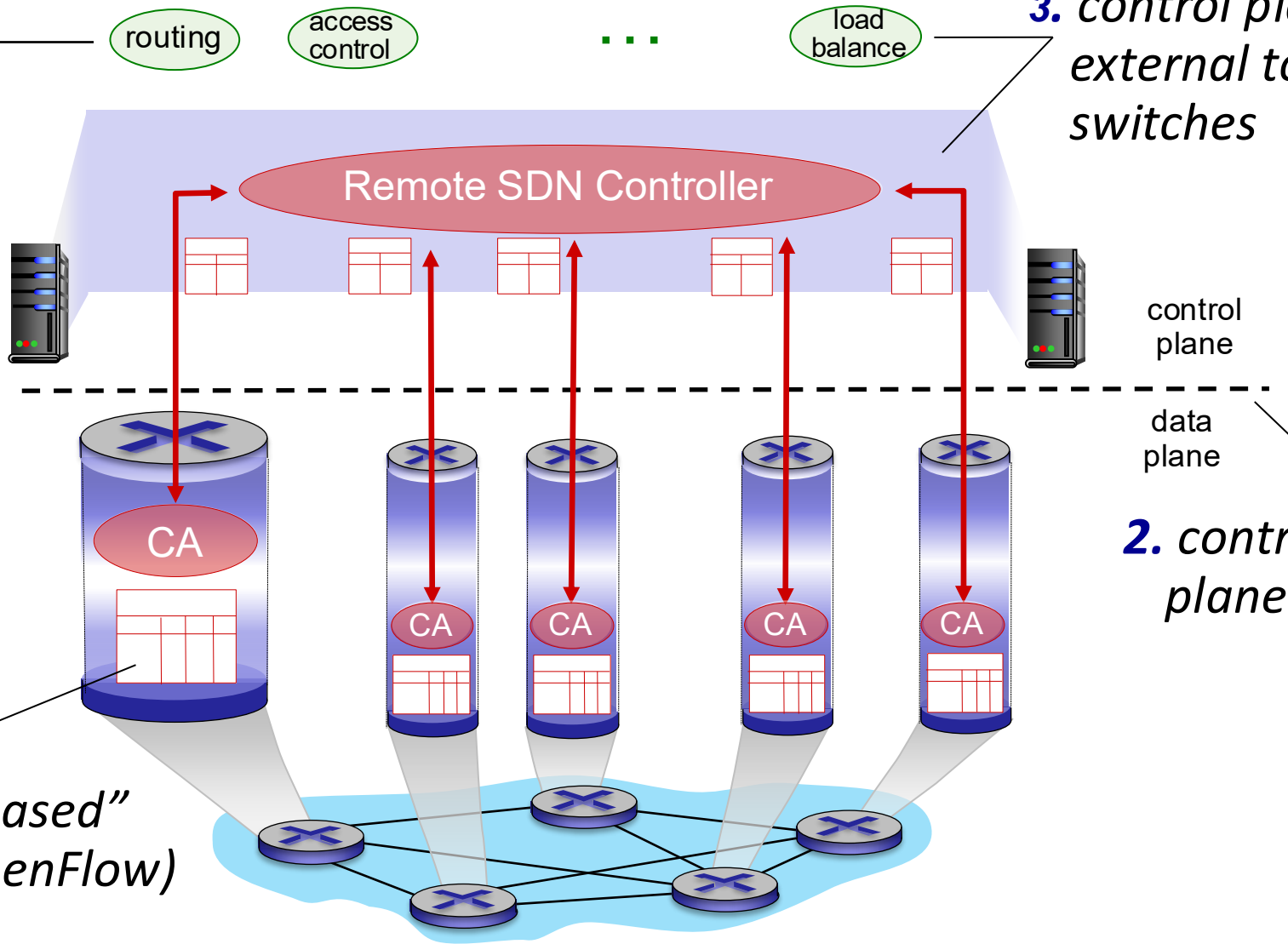
Q: what if w wants to route green and red traffic differently from w to z?

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

We learned that generalized forwarding and SDN
can be used to achieve *any* routing desired

SDN architecture

4. *programmable network control applications*



3. *control plane functions external to data-plane switches*

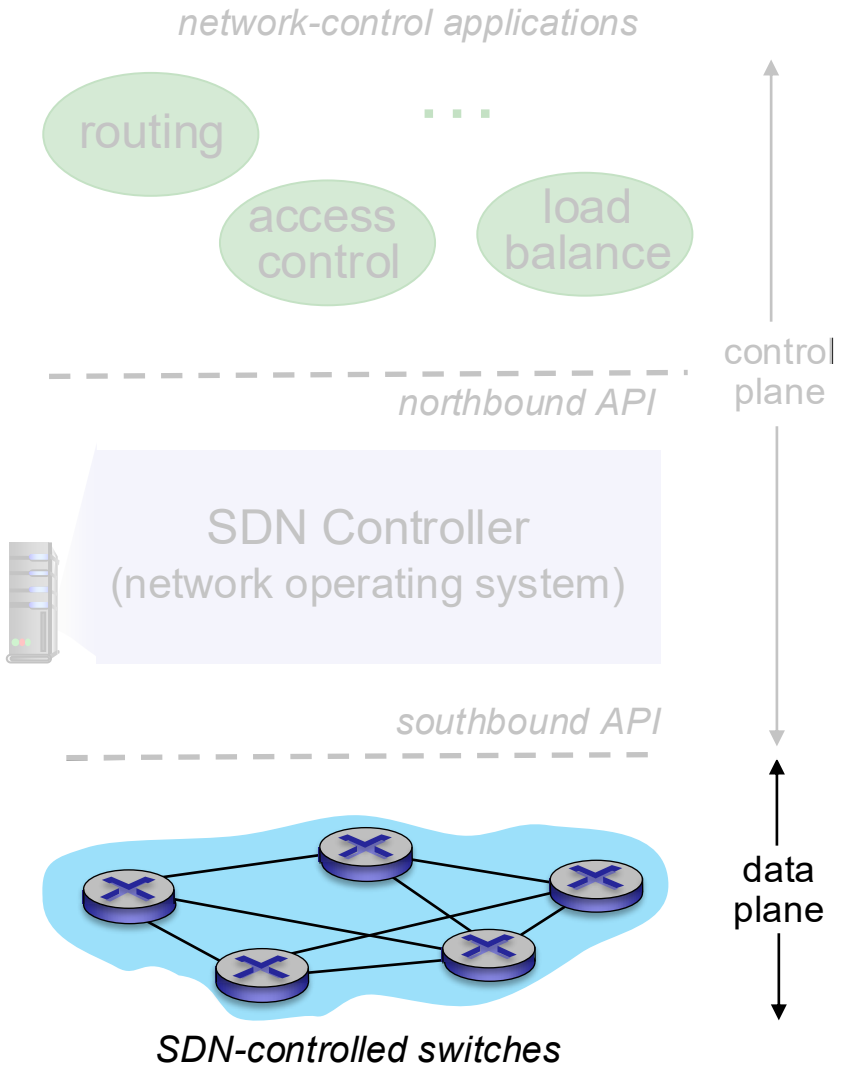
2. *control, data plane separation*

1: *generalized "flow-based" forwarding (e.g., OpenFlow)*

SDN architecture

Data-plane switches:

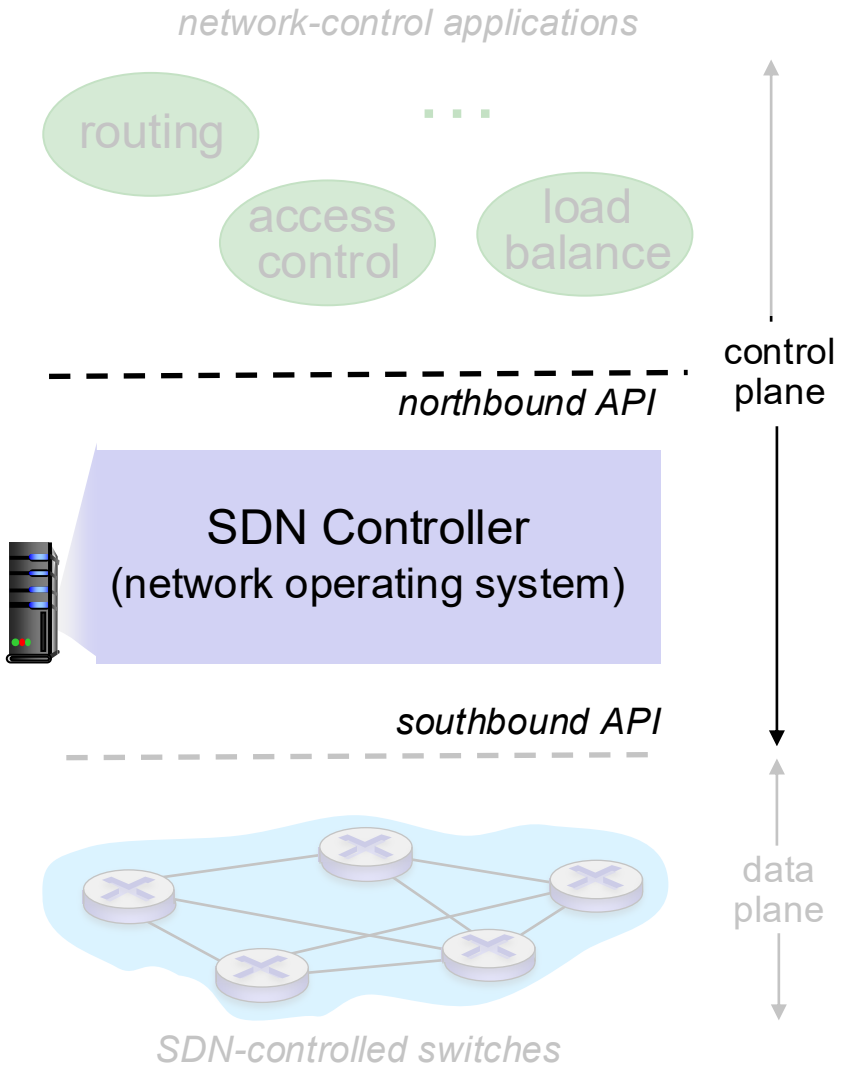
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with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



SDN architecture

SDN controller (e.g., NOX):

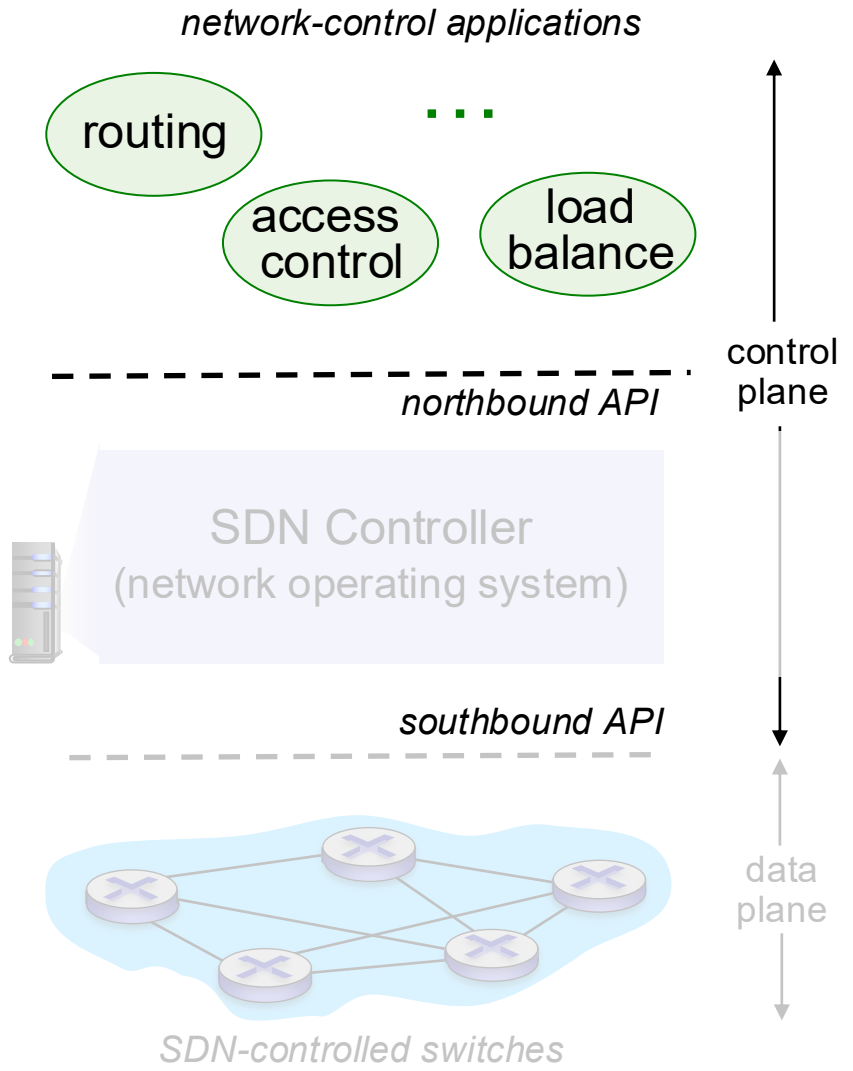
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



SDN architecture

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

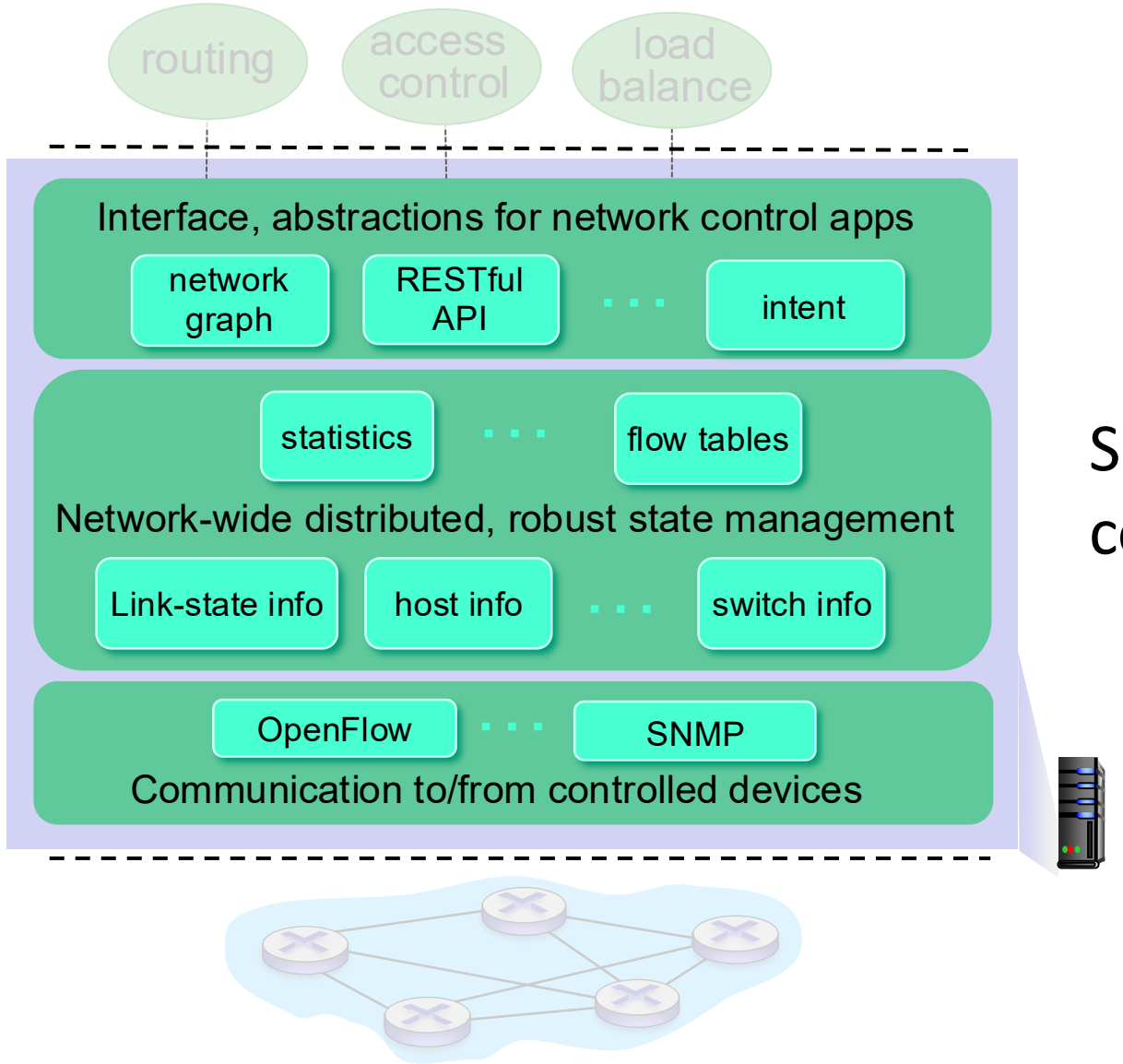


Components of SDN controller

interface layer to network control apps: abstractions API

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

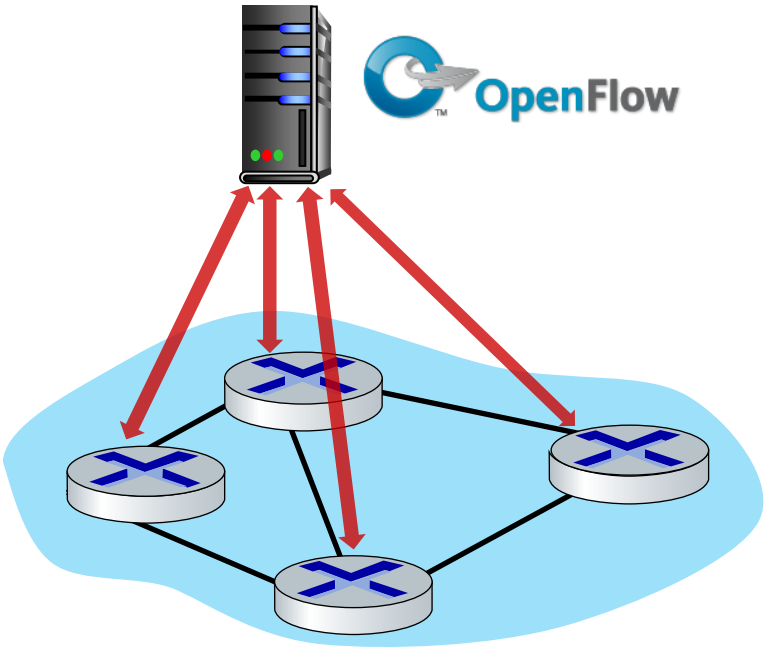
communication: communicate between SDN controller and controlled switches



OpenFlow protocol

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc.)
- distinct from OpenFlow API
  - API used to specify generalized forwarding actions

OpenFlow Controller

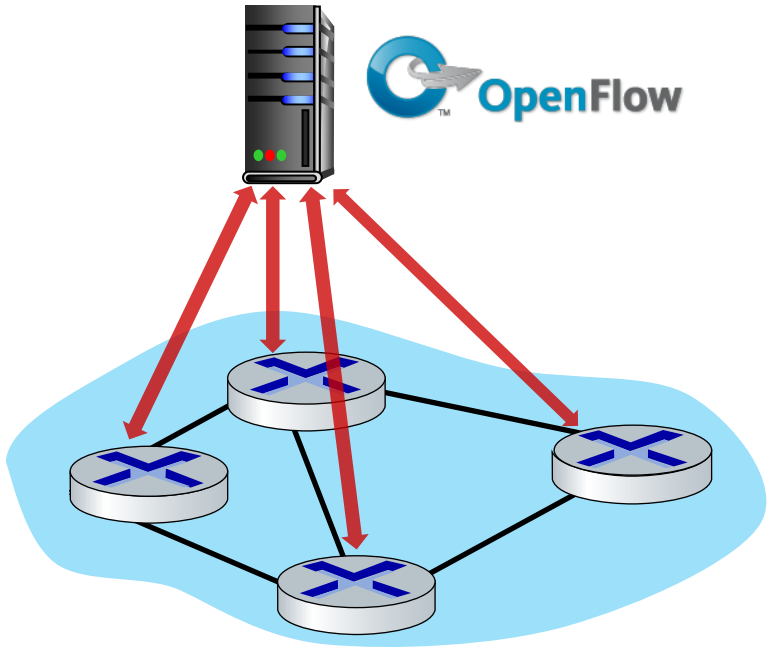


OpenFlow: controller-to-switch messages

Key controller-to-switch messages

- *Features:* controller queries switch features, switch replies
- *Configuration:* controller queries/sets switch configuration parameters
- *Modify-State:* add, delete, modify flow entries in the OpenFlow tables
- *Packet-out:* controller can send this packet out of specific switch port

OpenFlow Controller



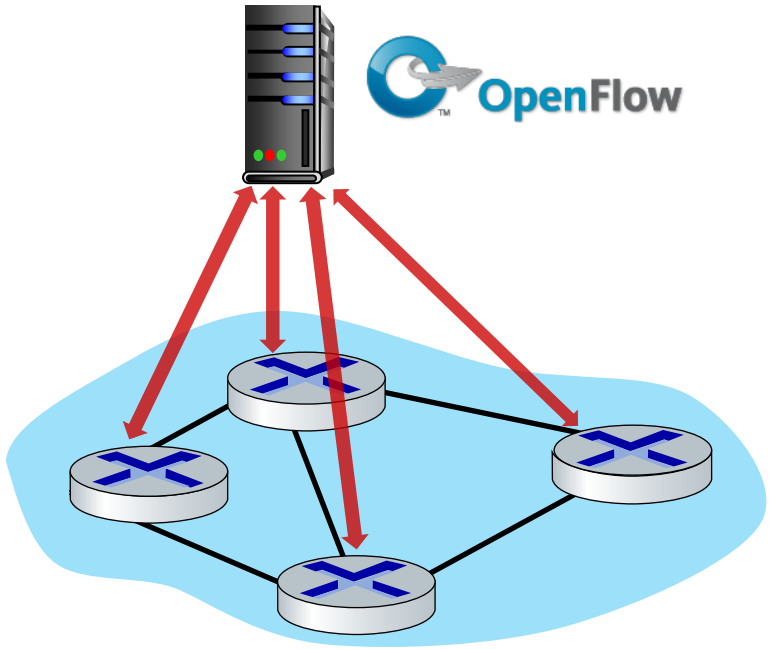
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

OpenFlow: asynchronous messages

Key asynchronous (switch-to-controller) messages

- *Packet-in:* transfer packet (and its control) to controller (vs. packet-out message from controller)
- *Flow-Removed:* flow table entry deleted at switch (especially due to timeout)
- *Port-status:* inform controller of a change on a port

OpenFlow Controller

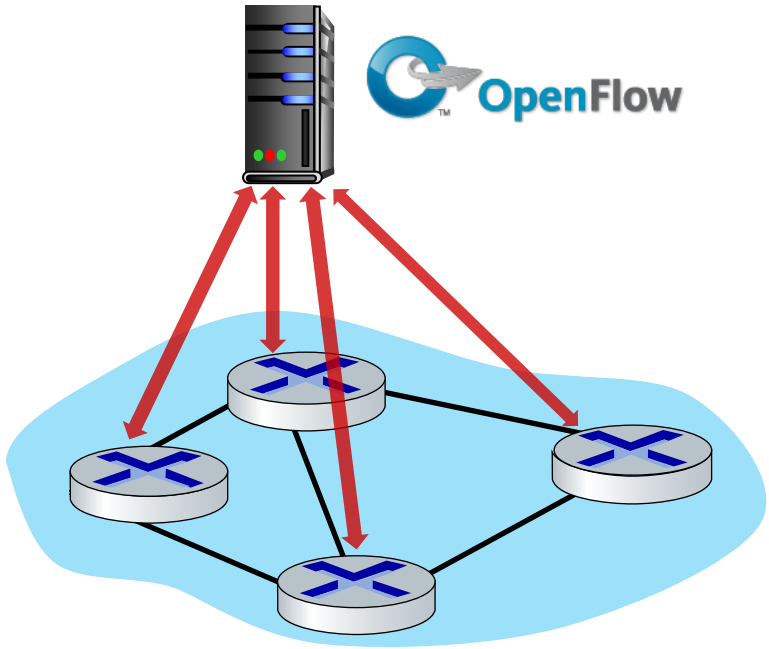


OpenFlow: symmetric messages

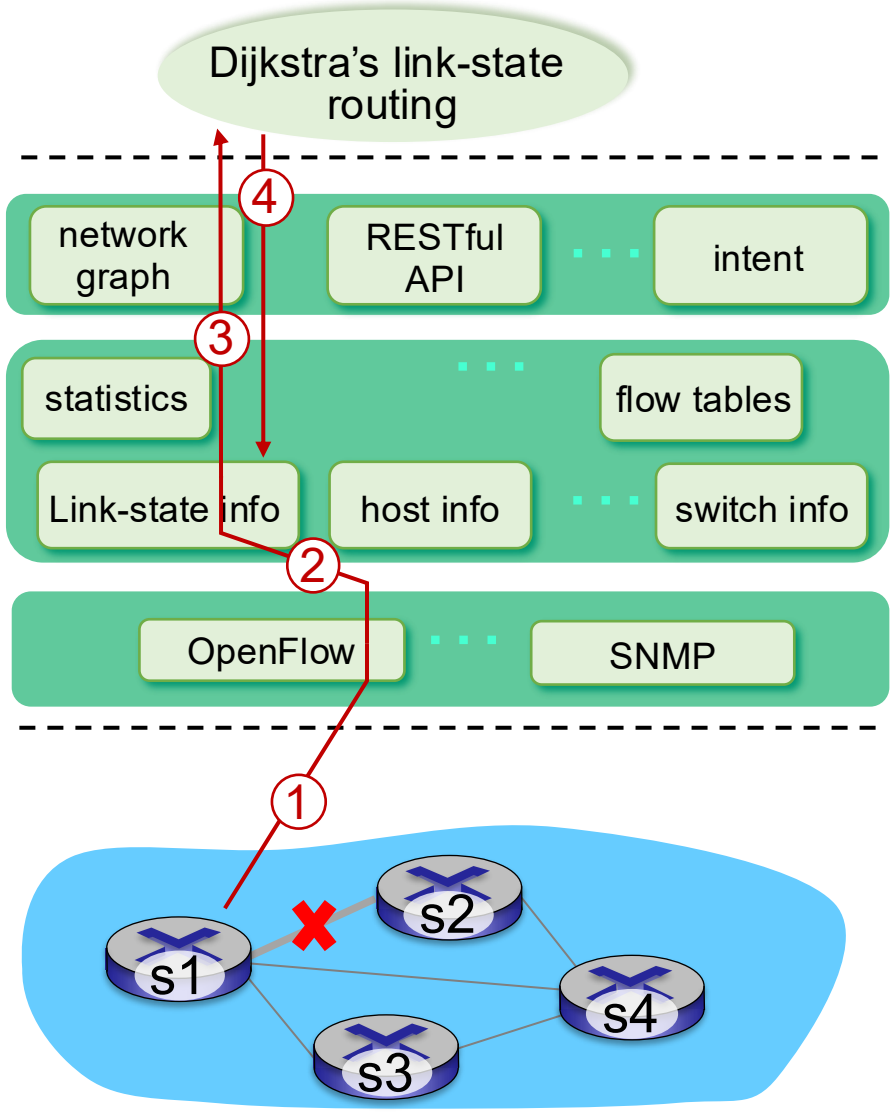
Key symmetric messages (switch <--> controller in either direction)

- *Hello:* exchanged upon connection setup
- *Echo:* verify liveness of a connection or measure latency

OpenFlow Controller



SDN: control/data plane interaction example



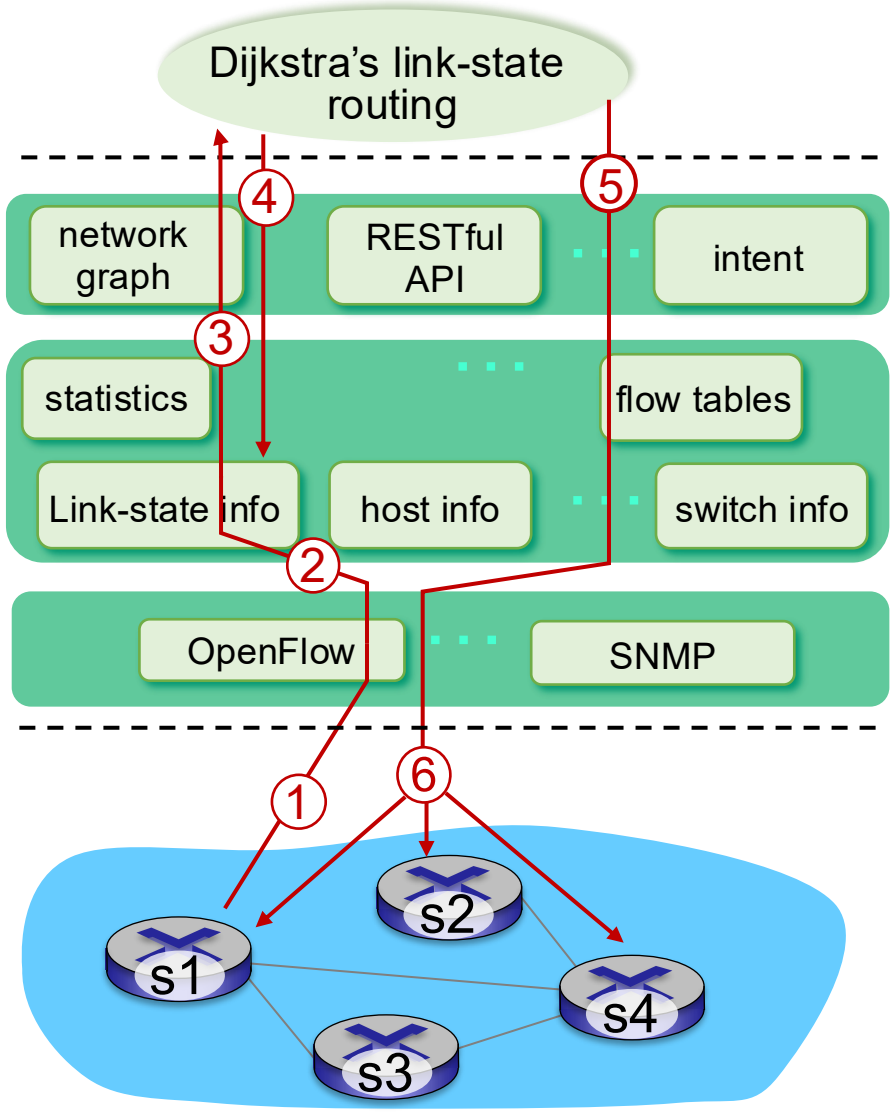
① S1, experiencing link failure uses OpenFlow Port-status message to notify controller.

② SDN controller receives OpenFlow message, updates link status info.

③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.

④ Dijkstra's routing algorithm accesses network graph info, link state info in controller, computes new routes.

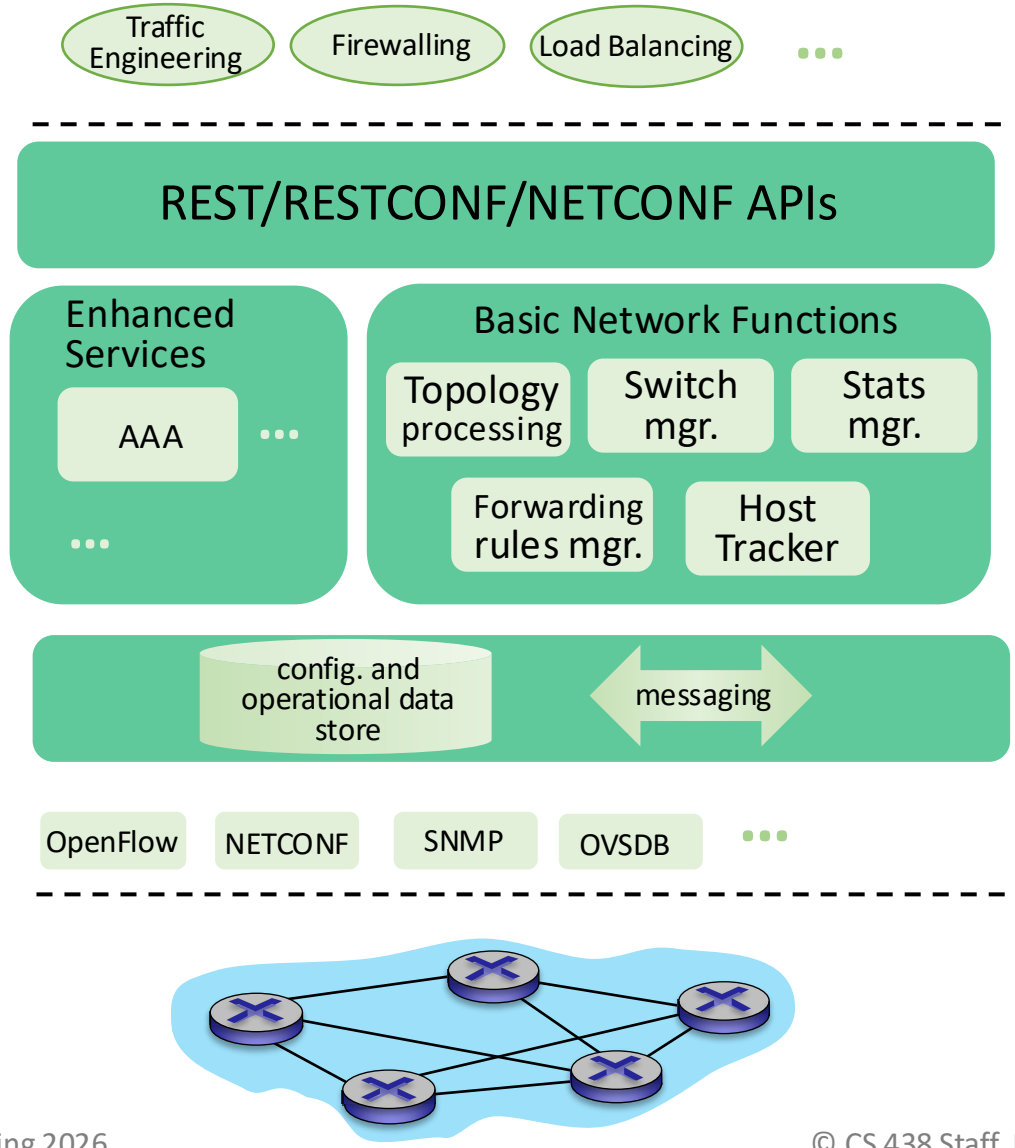
SDN: control/data plane interaction example



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.

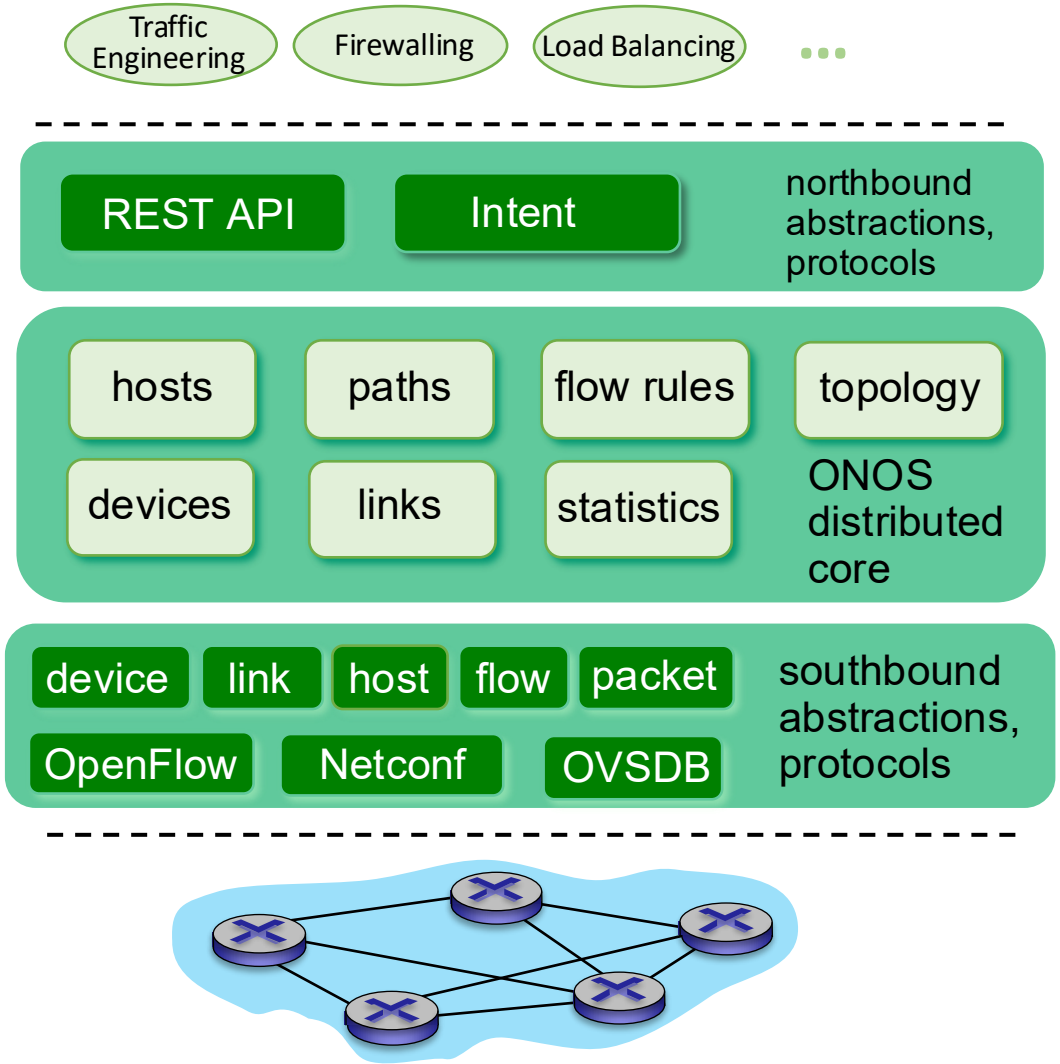
OpenDaylight (ODL) controller



Service Abstraction Layer:

- interconnects internal, external applications and services
  - e.g., access config and operational data, subscribe to events
- evolved from API-driven to model-driven
  - using a data modeling language to define models of device, protocol, network config, and operational data

ONOS controller



- considerable emphasis on distributed core
  - maintains network state with high availability, service replication, performance scaling
- southbound abstractions
  - mask heterogeneity of underlying devices and protocols at a more abstract level
- intent framework
  - high-level specification of service (what rather than how), e.g., to set up a connection between A and B without specifying details

SDN killer app: traffic engineering

- Traditional traffic engineering is rigid and inefficient
  - Poor global visibility → suboptimal link utilization
- SDN introduces centralized, global optimization
  - Controller has network-wide view of topology + demand
  - Computes optimal paths for traffic flows
- Success story: Google B4 and Microsoft SWAN
  - Transformed cloud WAN (wide-area network) traffic engineering

SDN killer app: network virtualization

- Cloud platforms enable massive multi-tenancy at scale
  - Each tenant needs isolated virtual networks with custom topology, addressing, security, and policies
- vSwitch (virtual switch) enables virtualization in the data plane
  - Runs on every node (e.g., hypervisor) and implements flow tables
- SDN provides the control plane for network virtualization
  - Programs vSwitches dynamically
- Success story: Open vSwitch (OVS)
  - Originally developed by SDN pioneers at Nicira (acquired by VMware)

SDN: challenges

- control plane requirements: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: “baked in” from day one?
- networks and protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling: beyond a single AS

SDN and the future of traditional network protocols

- SDN-computed versus router-computer forwarding tables:
  - just one example of logically-centralized-computed versus protocol computed
- How does SDN interoperate with routing protocols?
- How will implementation of network functionality evolve?