Διαχείριση Δικτύων Βασισμένων στο Λογισμικό 2025 (DIT306)

Δρ. Ειρήνη Λιώτου

eliotou@hua.gr

3/4/2025

Chapter 5
Network Layer:
The Control Plane

Network-layer functions

Recall: two network-layer functions:

- *forwarding:* move packets from routers input to appropriate router output — *data plane*
- *routing:* determine route taken by packets from source to destination — *control plane*

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

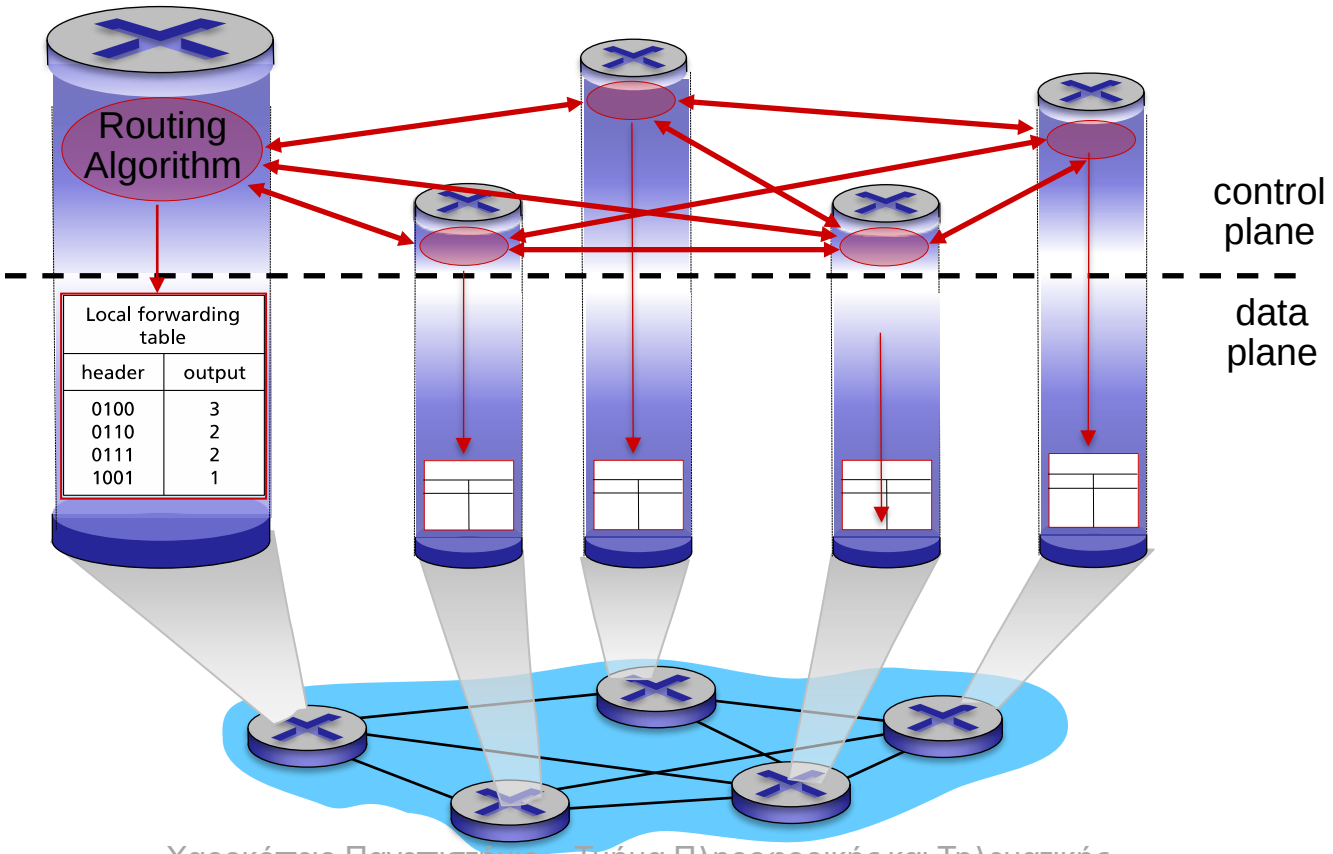
Software defined networking (SDN)

- 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

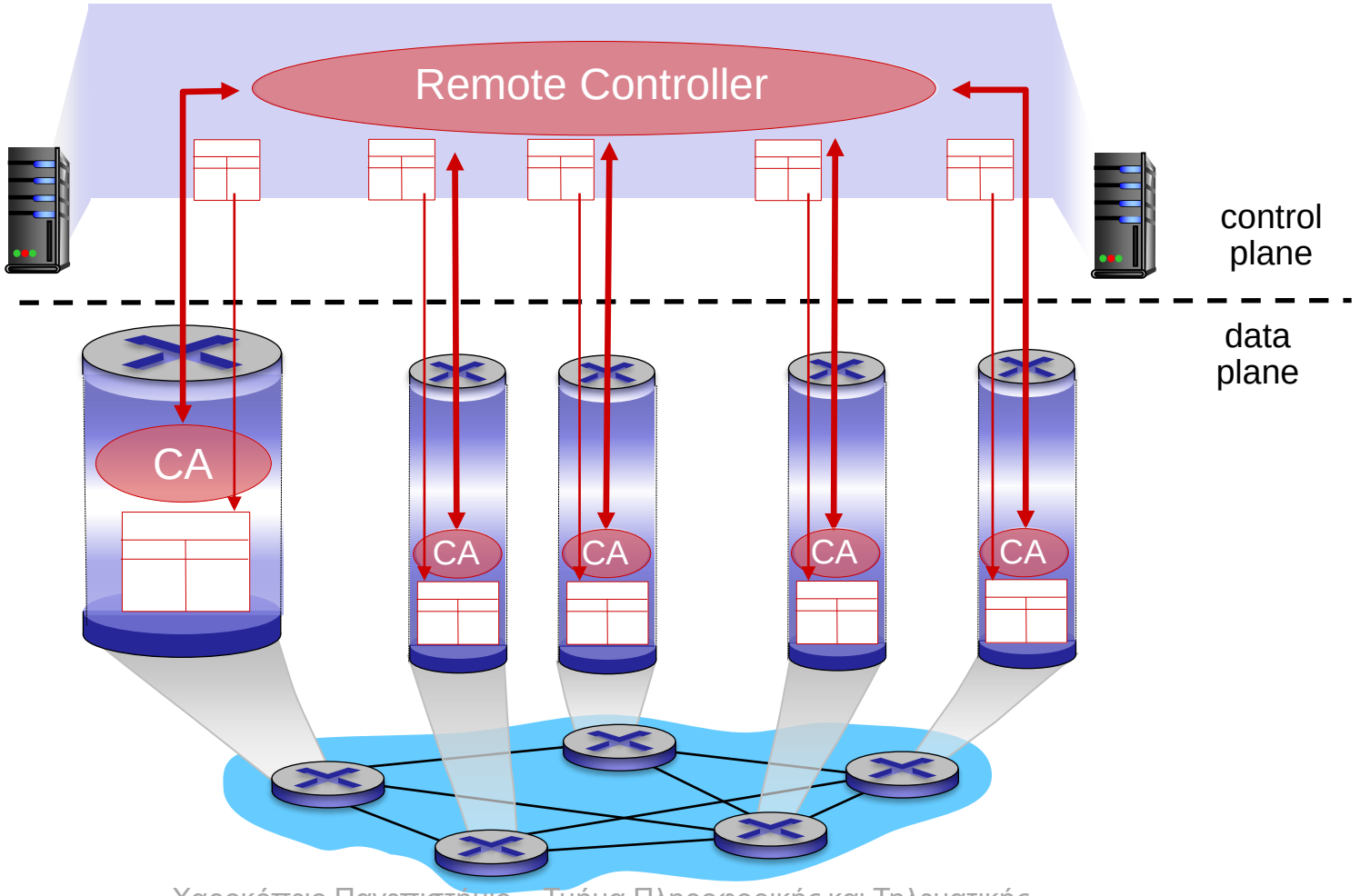
Recall: per-router control plane

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

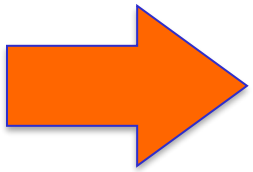


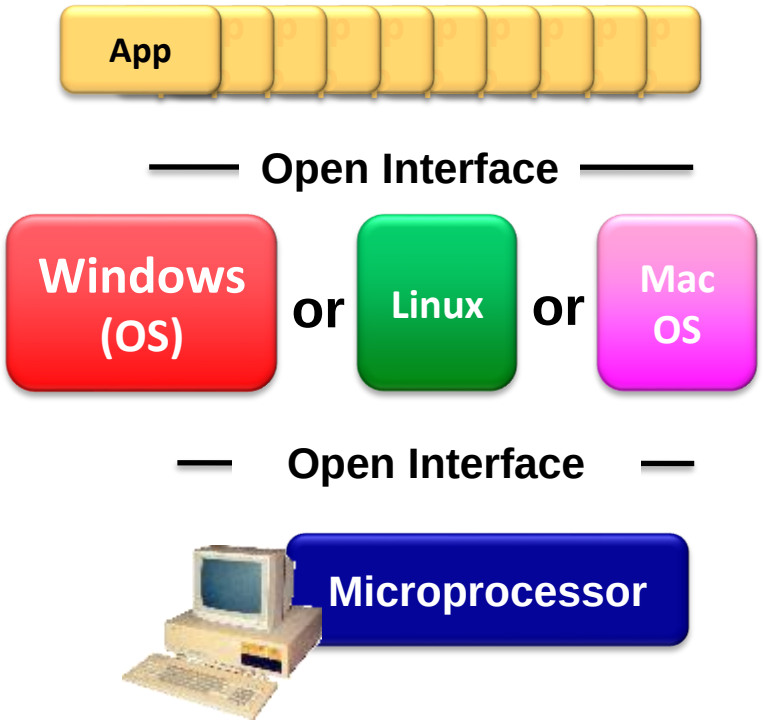
Software defined networking (SDN)

Why a *logically centralized* control plane?

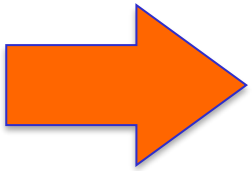
- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows "programming" routers
  - distributed "programming" more difficult: compute tables as result of distributed algorithm (protocol) implemented in each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

Analogy: mainframe to PC evolution*





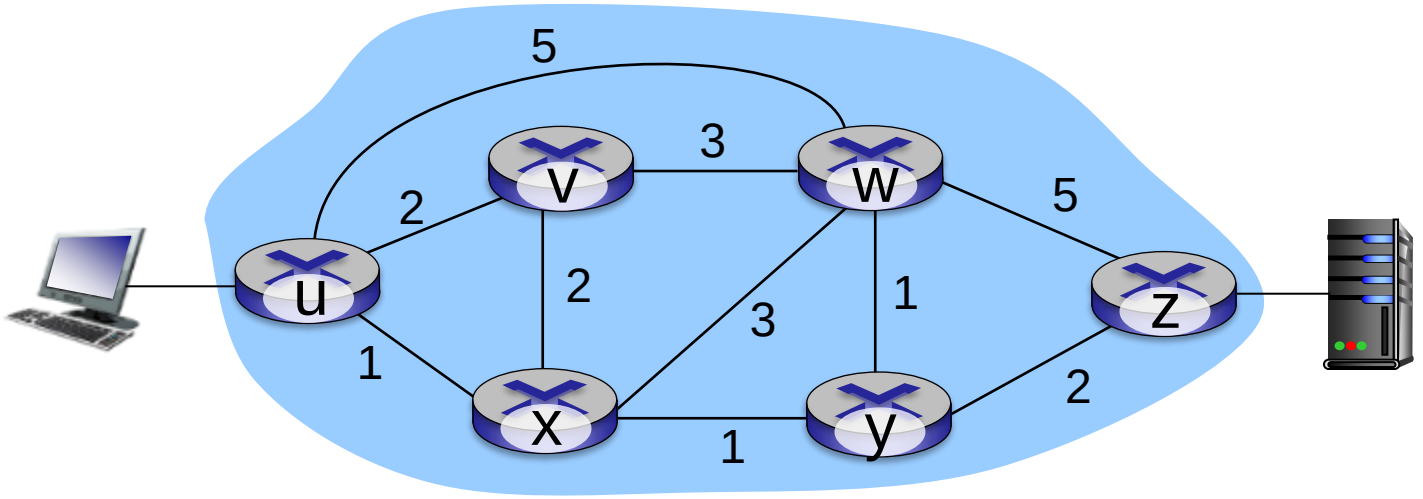
Vertically integrated
Closed, proprietary
Slow innovation
Small industry



Horizontal
Open interfaces
Rapid innovation
Huge industry

* Slide courtesy: N. McKeown

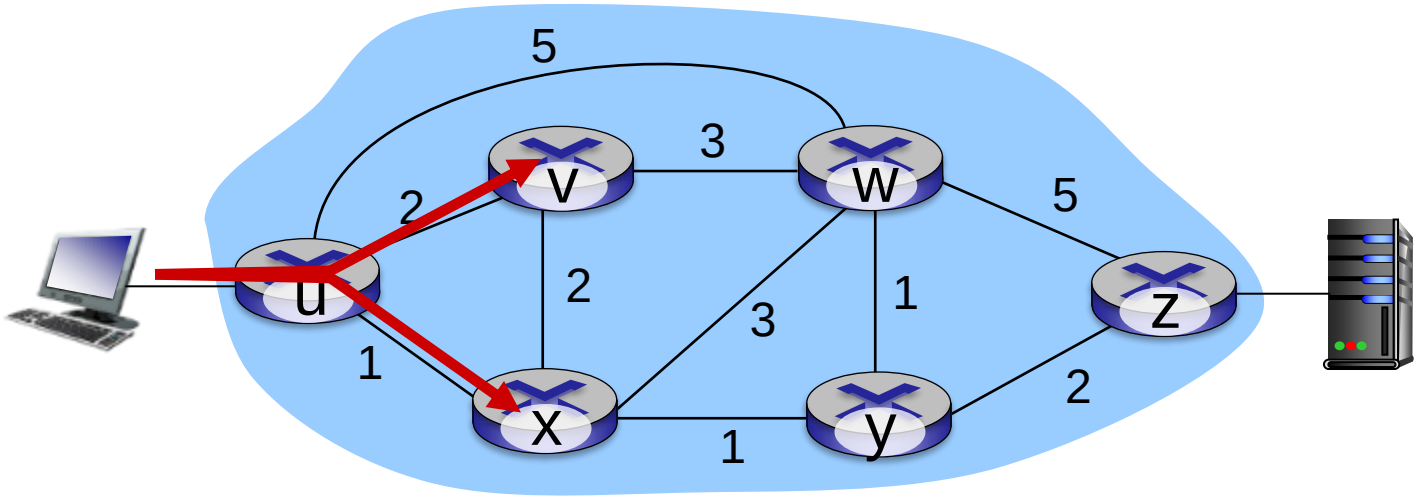
Traffic engineering: difficult traditional routing



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)!

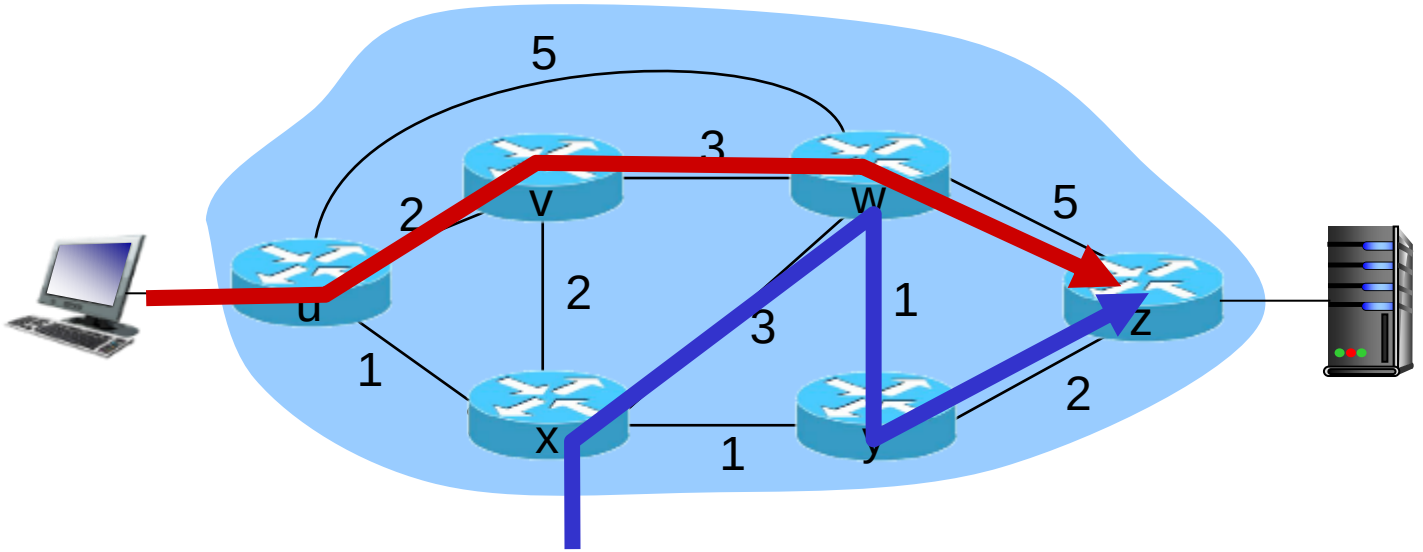
Traffic engineering: difficult



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)

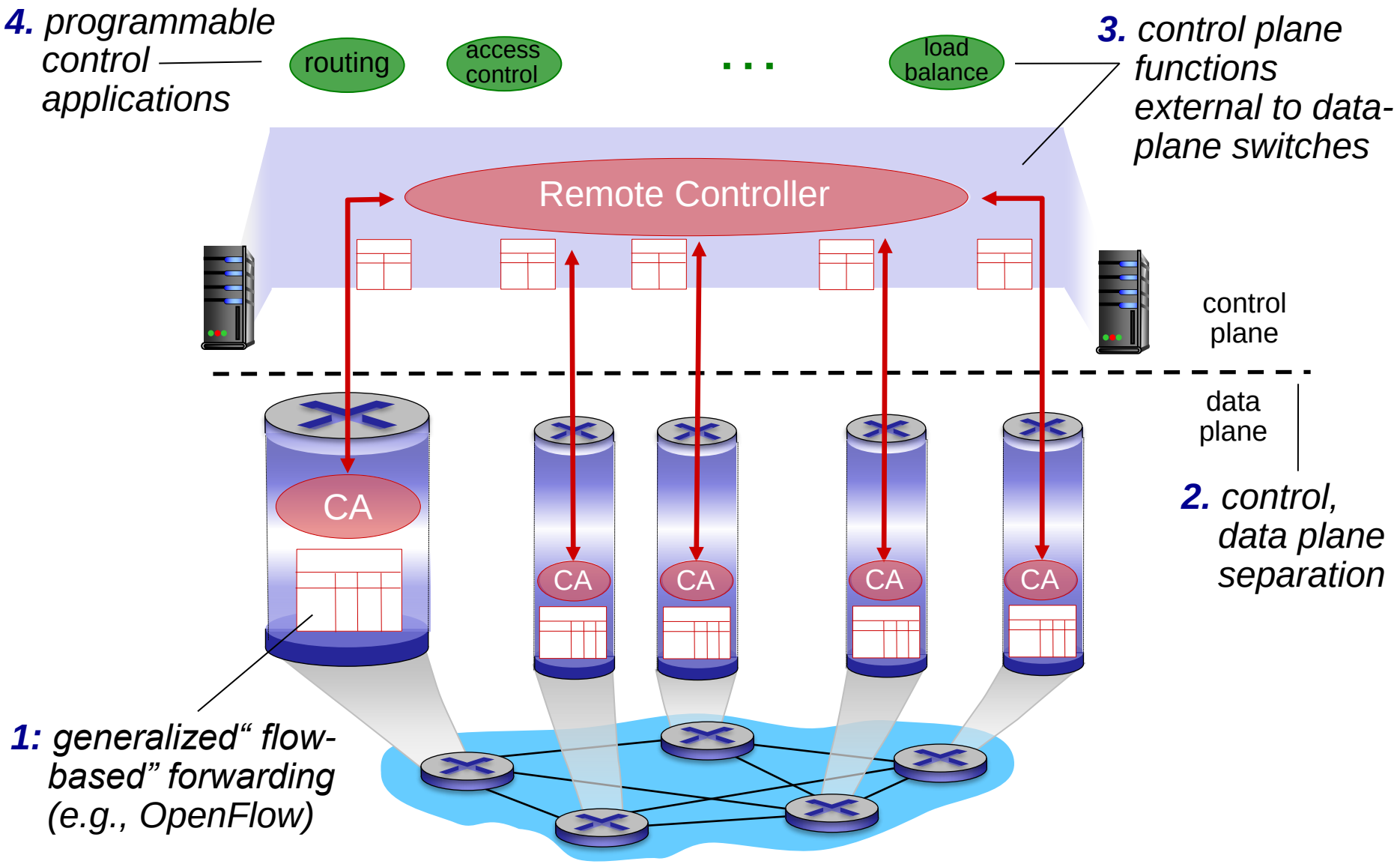
Traffic engineering: difficult



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

A: can't do it (with destination-based forwarding, and Link State, Distance Vector routing)

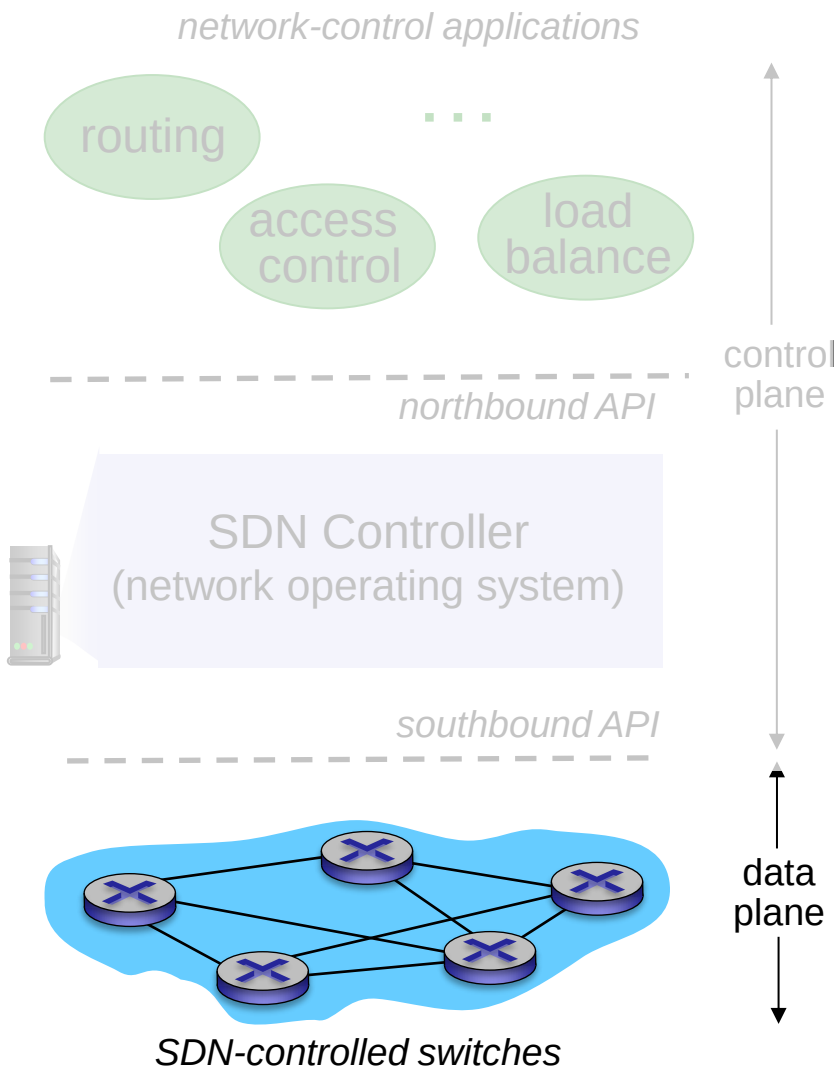
Software defined networking (SDN)



SDN perspective: data plane switches

Data plane switches

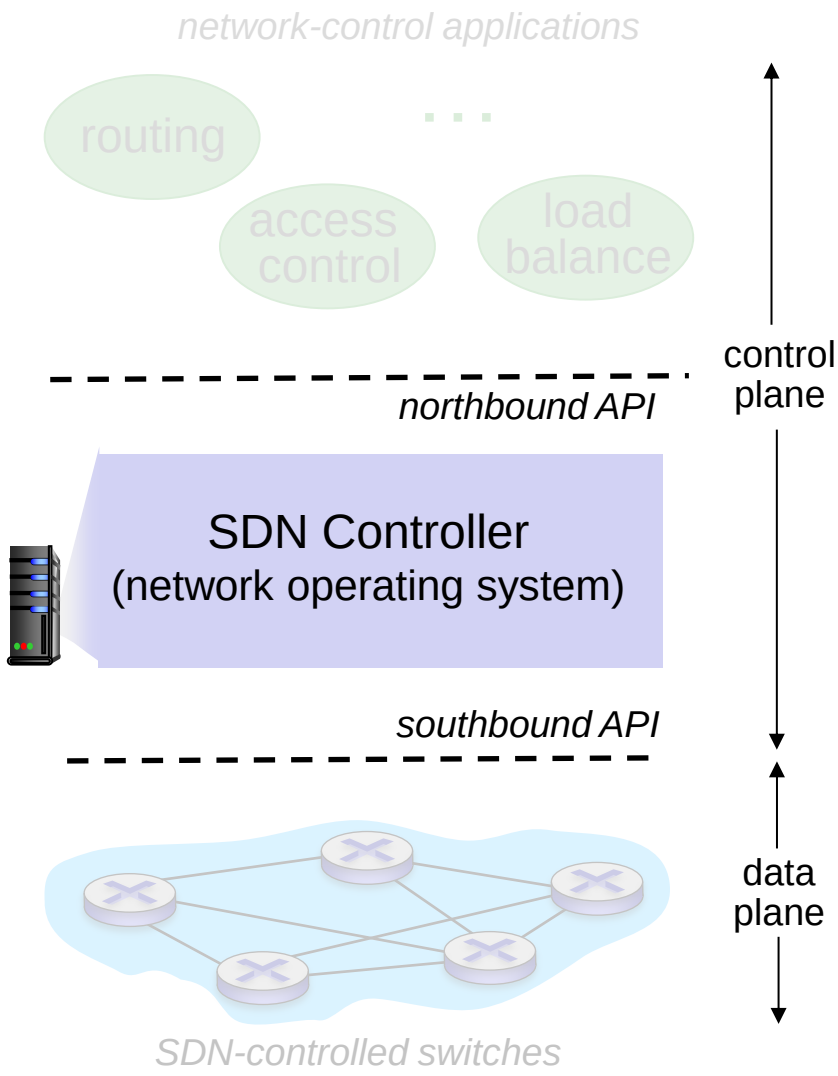
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- switch flow table computed, installed by controller
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable and what is not
- protocol for communicating with controller (e.g., OpenFlow)



SDN perspective: SDN controller

SDN controller (network OS):

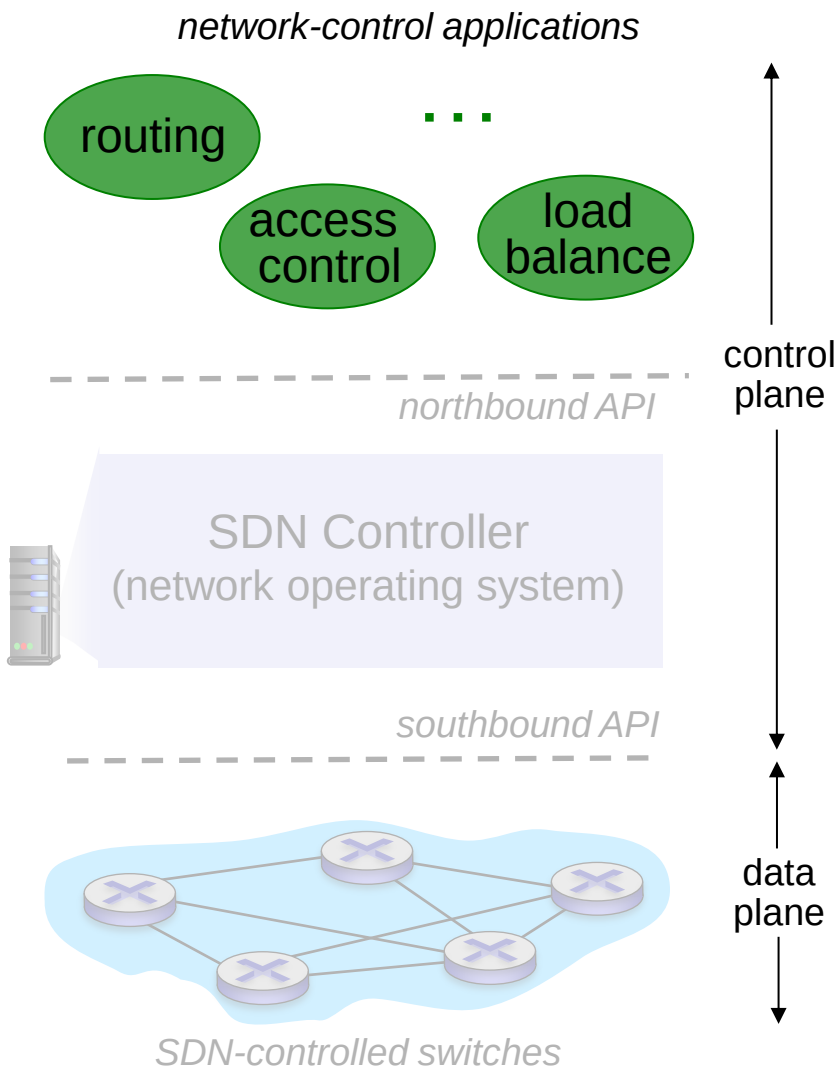
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



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
- Apps can express their requirements, constraints and intents without being affected and constrained by the complexities of the underlying network

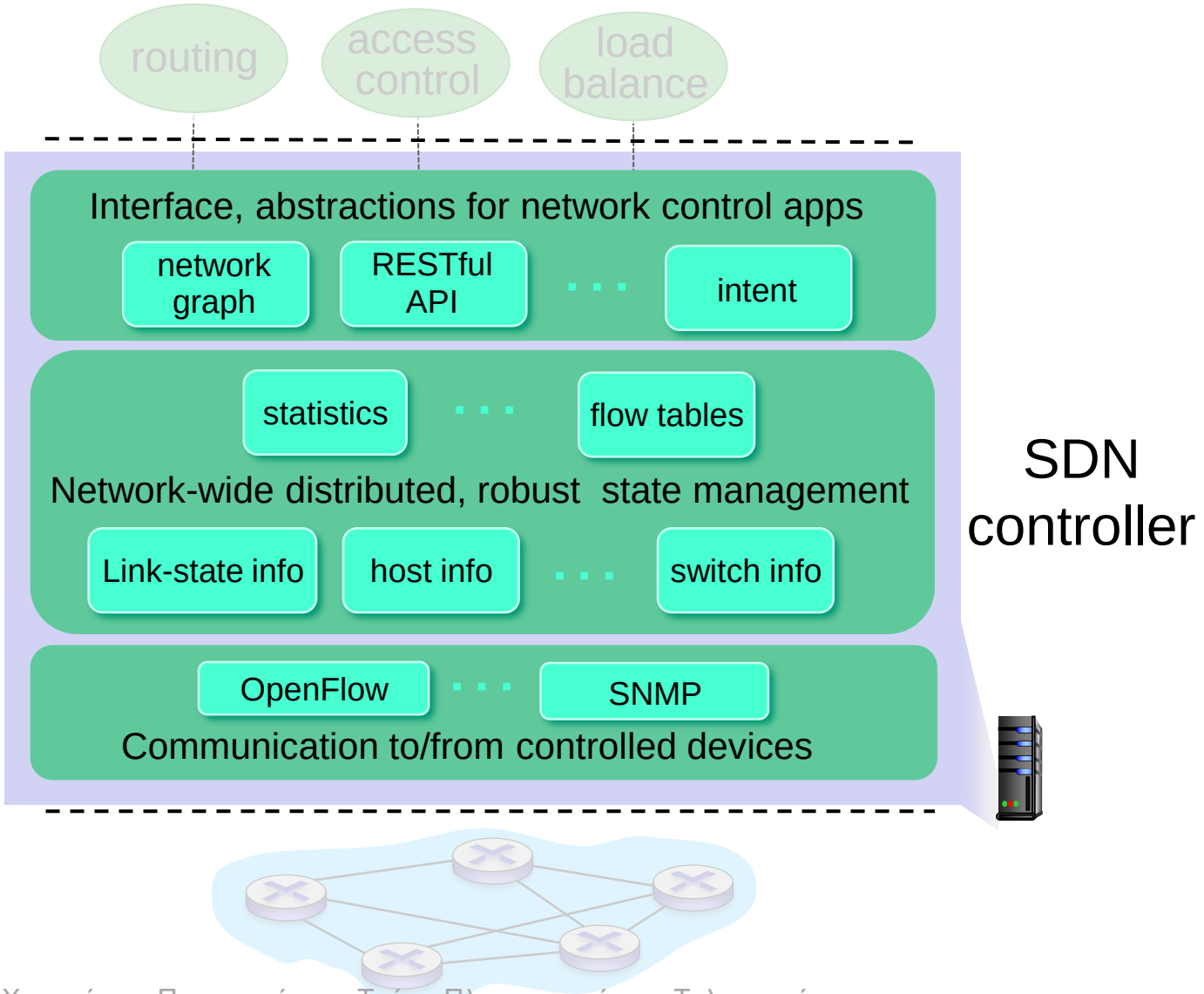


Components of SDN controller

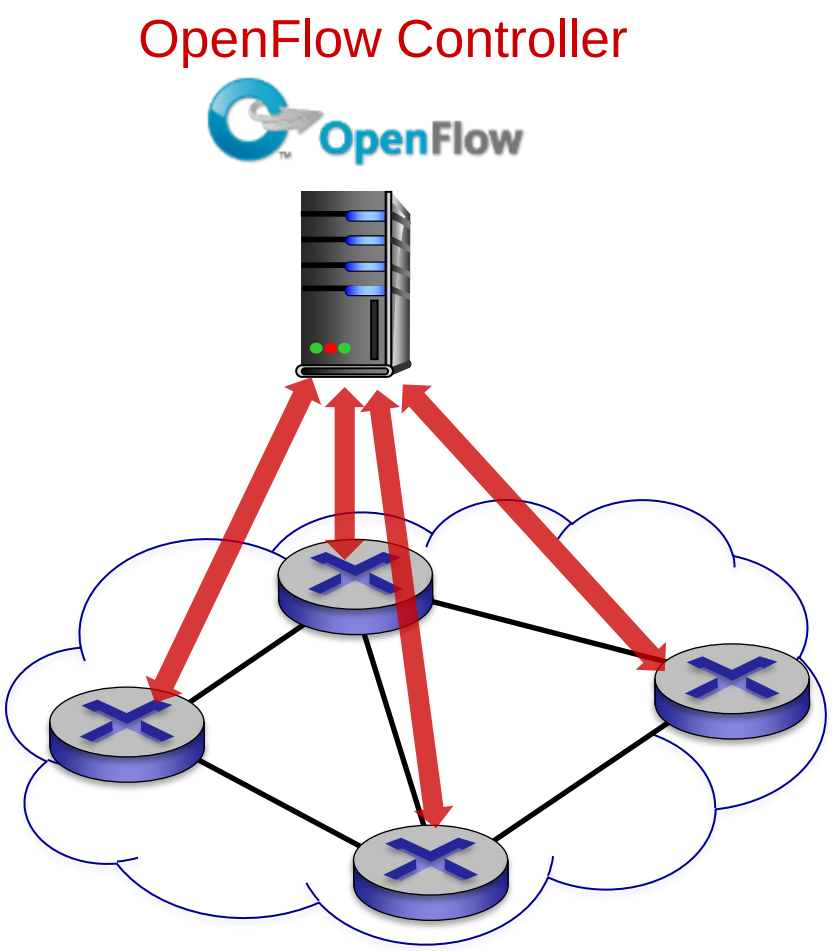
Interface layer to network control apps: abstractions API

Network-wide state management layer: state of networks links, switches, services: a *distributed database (counters&tables)*

Communication layer: communicate between SDN controller and controlled switches



OpenFlow protocol

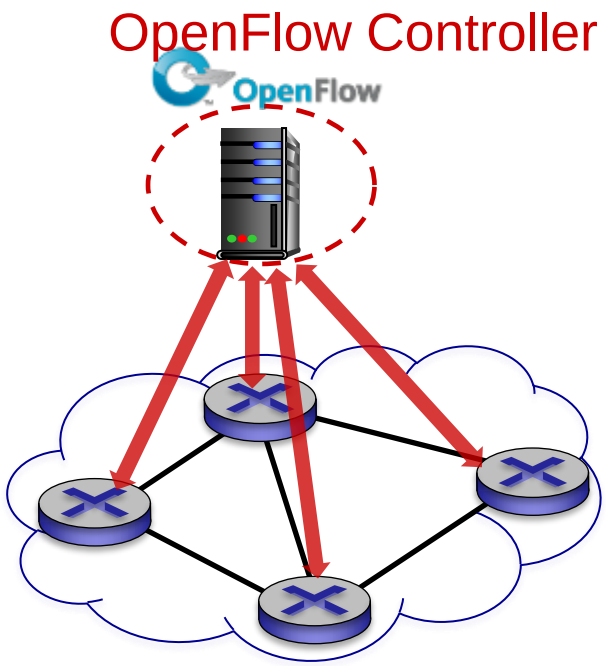


- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

OpenFlow: controller-to-switch messages

Key controller-to-switch messages

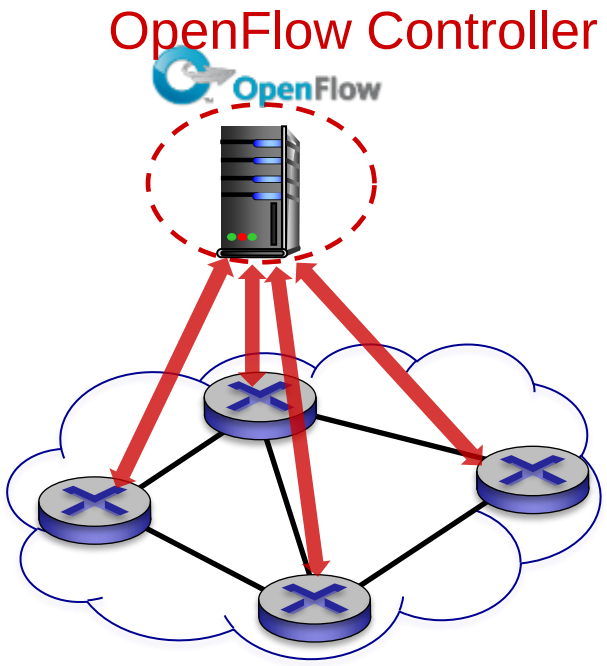
- *configuration:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *features read-state:* controller queries switch features/statistics, switch replies
- *packet-out:* controller can send this packet out of specific switch port



OpenFlow: switch-to-controller messages

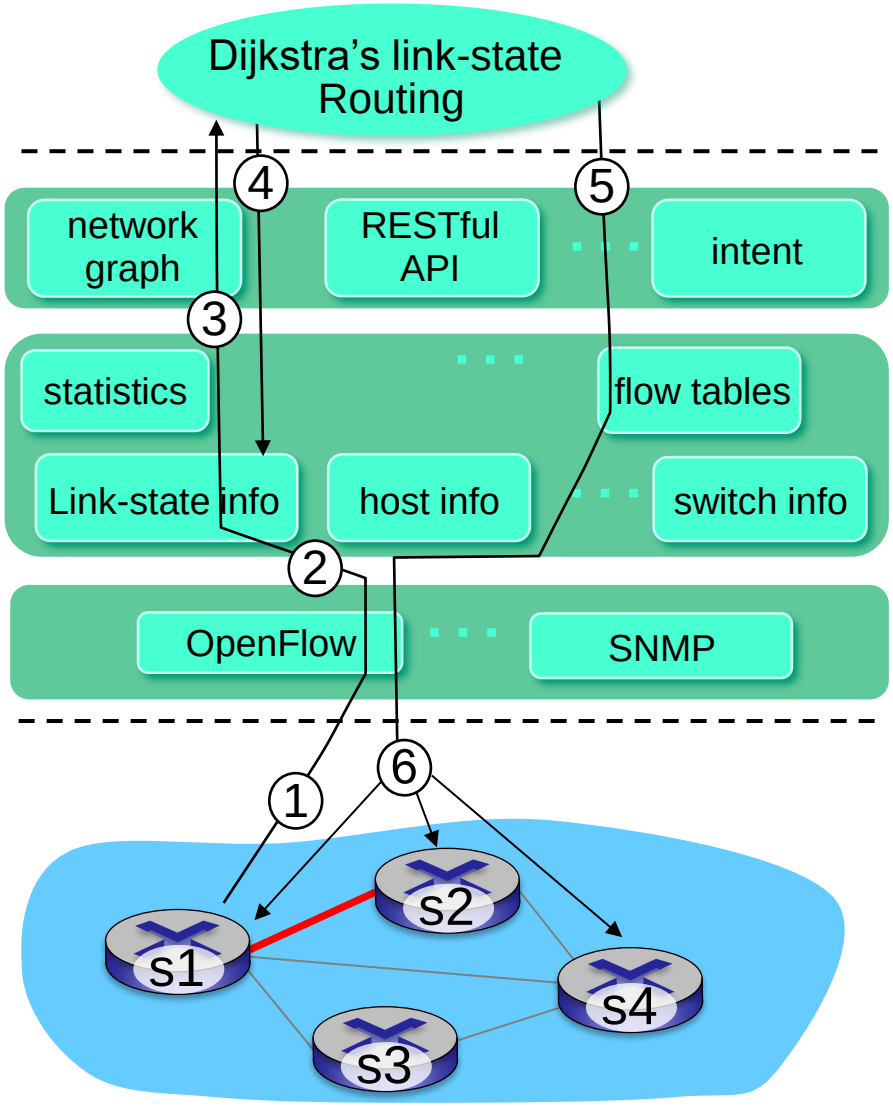
Key switch-to-controller messages

- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch (e.g. expired)
- *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



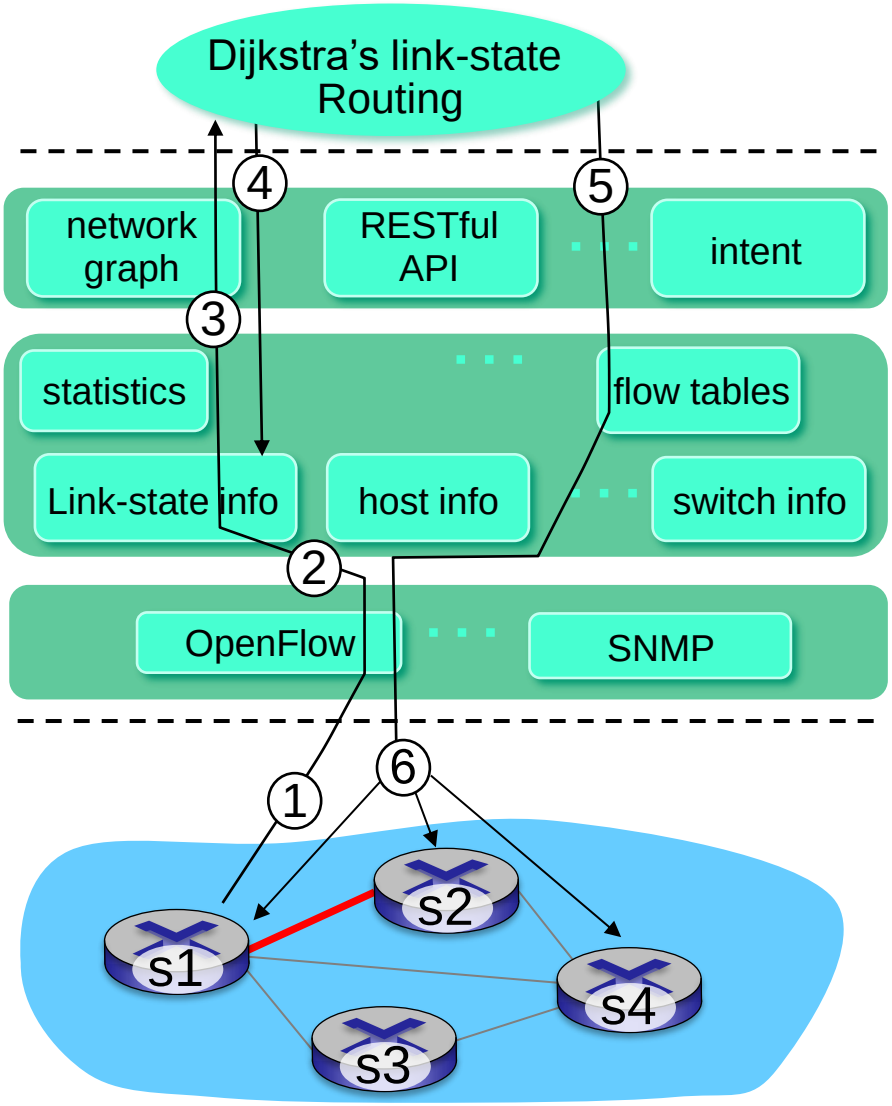
① S1, experiencing link failure using OpenFlow port status message to notify controller

② SDN controller receives OpenFlow message, updates link status info

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

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

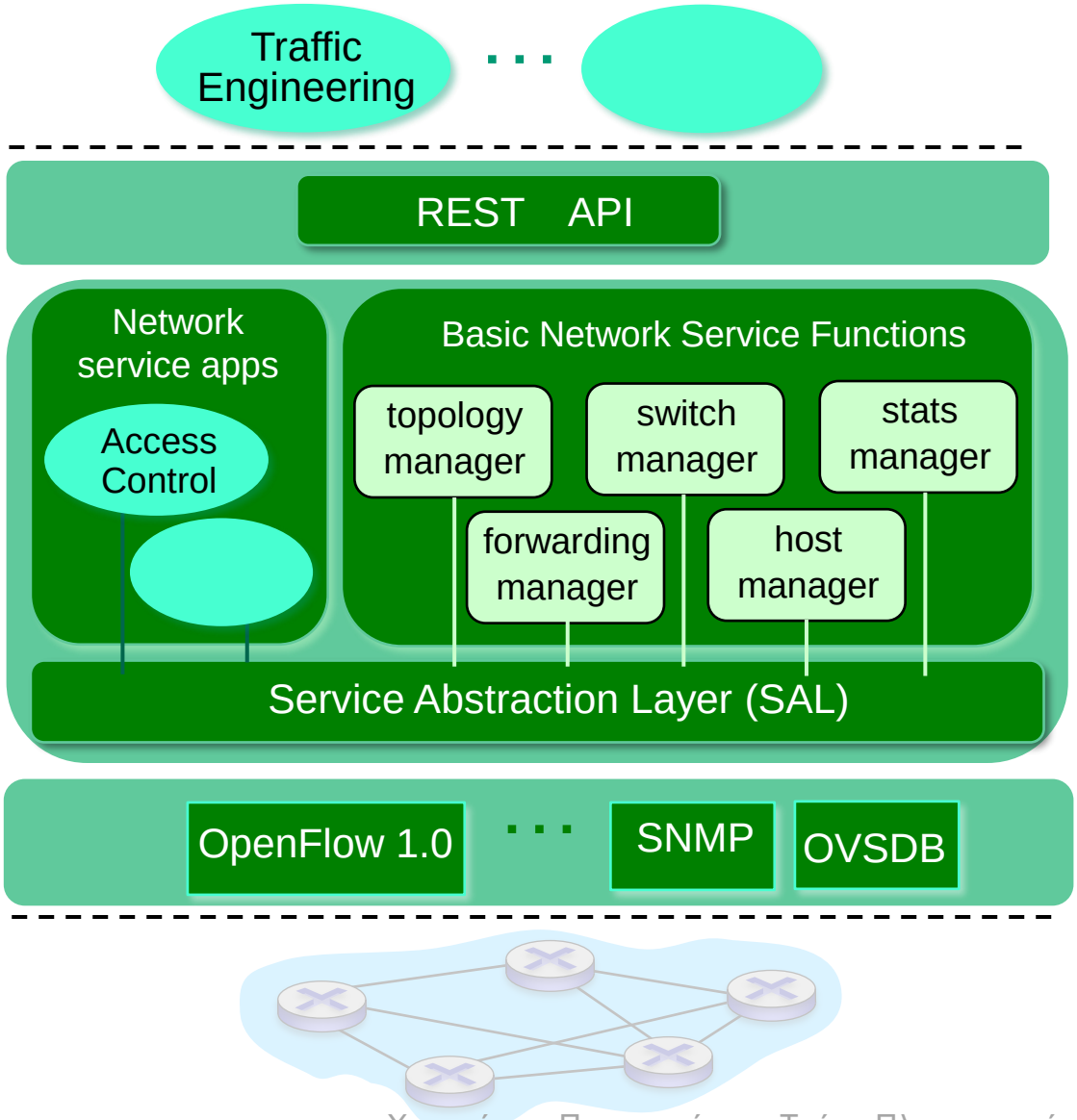
SDN: control/data plane interaction example



⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

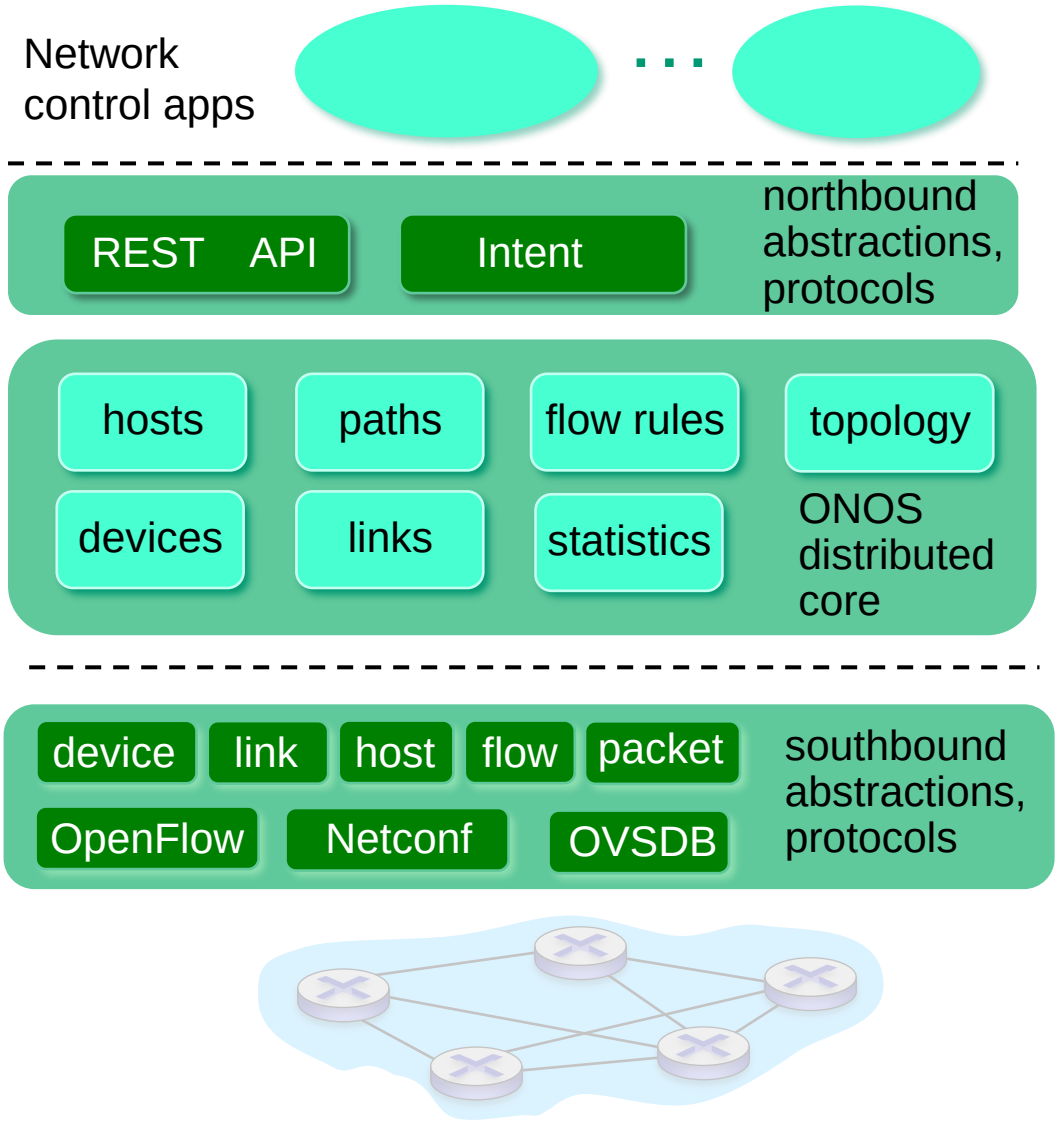
⑥ Controller uses OpenFlow to install new tables in switches that need updating

OpenDaylight (ODL) controller



- ODL Lithium controller
- network apps may be contained within, or be external to SDN controller
- Service Abstraction Layer: interconnects internal, external applications and services

ONOS controller



- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

Various Controllers

Name	Language	Original Developers	Description
Ovs	C	Stanford/Nicira	A reference controller. Act as a learning switch
NOX	C++	Nicira	The first OpenFlow controller
POX	Python	Nicira	Open source SDN controller
Beacon	Java	Stanford	A cross platform, modular OpenFlow controller
Maestro	Java	Rice	Network operating system
Trema	Ruby, C	NEC	A framework for developing OpenFlow controller
Floodlight	Java	BigSwitch	OpenFlow controller that work with physical and virtual OpenFlow switches
Flowvisor	C	Stanford/Nicira	Special purpose controller
Ryu	Python	NTT Labs	Ryu is a component based SDN framework
OpenDayLight	Java	ONF	It is open source project