Software Defined Networking

Prof. T. Venkatesh
Dept of CSE, IIT Guwahati

Middleboxes

Middlebox (RFC 3234)

"any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host"

Middleboxes everywhere!

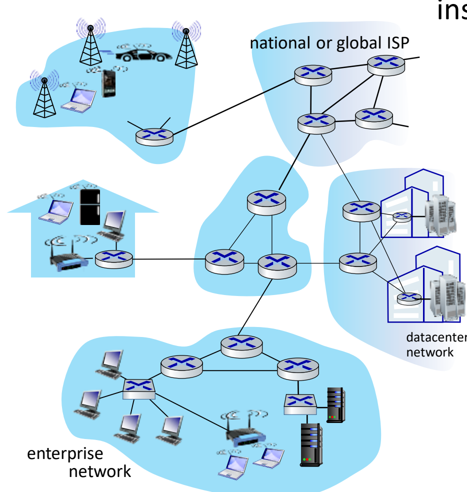
NAT: home, cellular, institutional

Application-specific: service providers, institutional, CDN

Firewalls, IDS: corporate, institutional, service providers, ISPs

Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs



Middleboxes

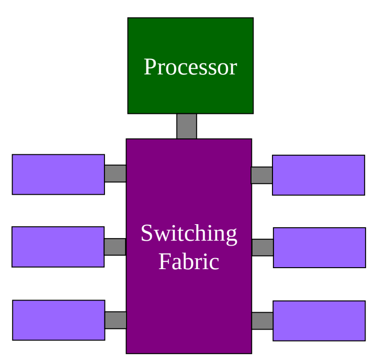
- initially: proprietary (closed) hardware solutions
- move towards "whitebox" hardware implementing open API
  - move away from proprietary hardware solutions
  - programmable local actions via match+action
  - move towards innovation/differentiation in software
- SDN: (logically) centralized control and configuration management often in private/public cloud
- network functions virtualization (NFV): programmable services over white box networking, computation, storage

Data, Control and Management Planes

	Data	Control	Management
Time-scale	Packet (nsec)	Event (10 msec to sec)	Human (min to hours)
Tasks	Forwarding, buffering, filtering, scheduling	Routing, circuit set-up	Analysis, configuration
Location	Line-card hardware	Router software	Humans or scripts

Data Plane

- Streaming algorithms on packets
  - Matching on some bits
  - Perform some actions
- Wide range of functionality
  - Forwarding
  - Access control
  - Mapping header fields
  - Traffic monitoring
  - Buffering and marking
  - Rate Shaping and scheduling
  - Deep packet inspection



6

Control and Management Planes

- Broad definition of “network management”:
  - Everything having to do with the control plane
- **Basic connectivity**: route packets to destination
  - Local state computed by routing protocols
  - Globally distributed algorithms
- **Interdomain policy**: find policy-compliant paths
- Want multiple LANs on single physical network - VLANs
- Operators want to limit access to various hosts – Access Control Lists
- Choose routes to spread traffic load across links
  - Setting up MPLS tunnels
  - Adjusting weights in OSPF
  - Often done centrally

Networks are Hard to Manage

- Operating a network is expensive
  - More than half the cost of a network
  - Yet, operator error causes most outages
- Buggy software in the equipment
  - Routers with 20+ million lines of code
  - Cascading failures, vulnerabilities, etc.
- The network is all over
  - Huge problem in data centers, 100,000s machines & 10,000s switches
  - Large datacenters can host many customers
    - Each customer gets their own logical network

Datacenter networks

10's to 100's of thousands of hosts, often closely coupled, in close proximity:

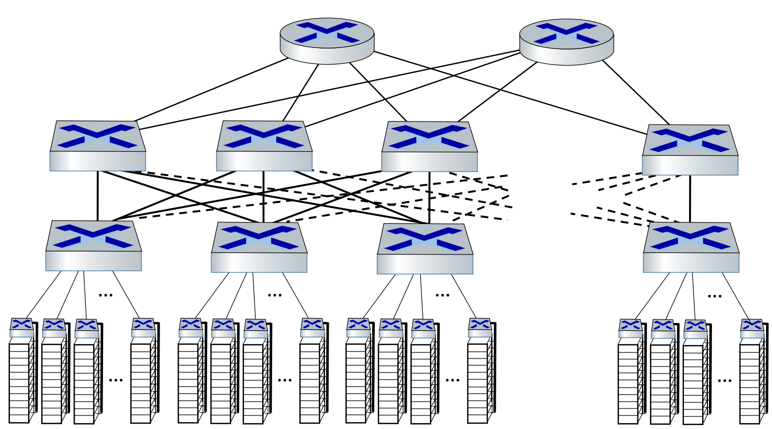
- e-business (e.g. Amazon)
- content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
- search engines, data mining (e.g., Google)

challenges:

- multiple applications, each serving massive numbers of clients
- reliability
- managing/balancing load, avoiding processing, networking, data bottlenecks

Inside a 40-ft Microsoft container, Chicago data center

Datacenter networks: network elements



Border routers

- connections outside datacenter

Tier-1 switches

- connecting to ~16 T-2s below

Tier-2 switches

- connecting to ~16 TORs below

Top of Rack (TOR) switch

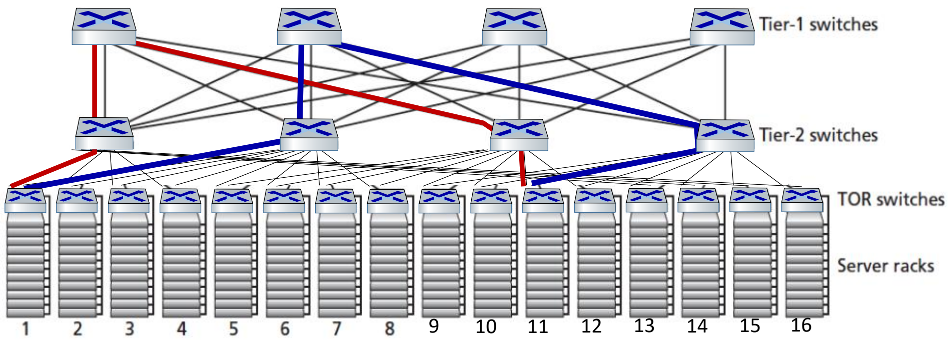
- one per rack
- 40-100Gbps Ethernet to blades

Server racks

- 20- 40 server blades: hosts

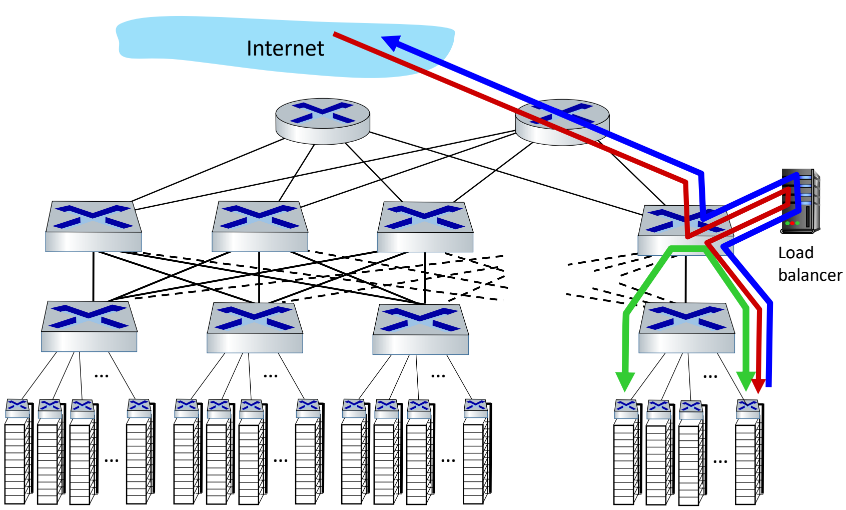
Datacenter networks: multipath

- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy



two disjoint paths highlighted between racks 1 and 11

Datacenter networks: application-layer routing



load balancer: application-layer routing

- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)

Datacenter networks: protocol innovations

- link layer:
  - RoCE: remote DMA (RDMA) over Converged Ethernet
- transport layer:
  - ECN (explicit congestion notification) used in transport-layer congestion control (DCTCP, DCQCN)
  - experimentation with hop-by-hop (backpressure) congestion control
- routing, management:
  - SDN widely used within/among organizations' datacenters
  - placing data as close as possible (e.g., in same rack or nearby rack) to minimize tier-2, tier-1 communication

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

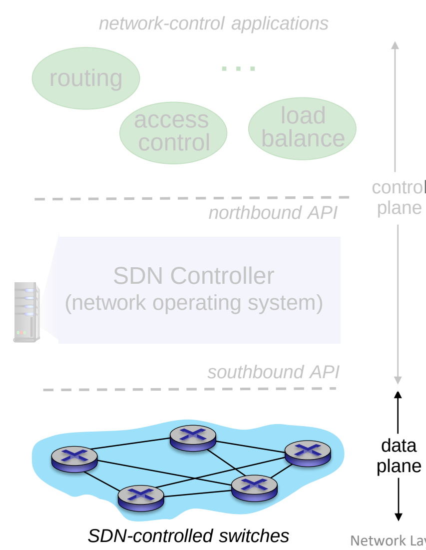
SDN Key Ideas

- Separate Control plane and Data plane entities
  - Network intelligence and state are logically centralized
  - The underlying network infrastructure is abstracted from the applications
- Execute or run Control plane software on general purpose hardware
  - Decouple from specific networking hardware
  - Use commodity servers
- Have programmable data planes
  - Maintain, control and program data plane state from a central entity
- An architecture to control not just a networking device but an entire network

Software defined networking (SDN)

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)



Network Layer: 5-16

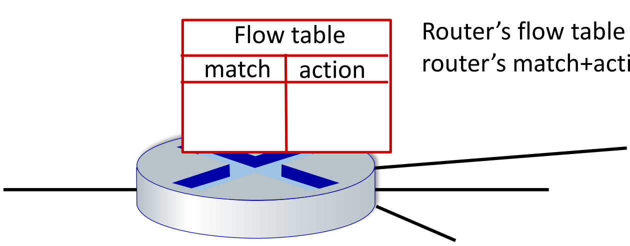
Generalized forwarding: match plus action

Each router contains a forwarding table (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 action possible: drop/copy/modify/log packet

Flow table abstraction

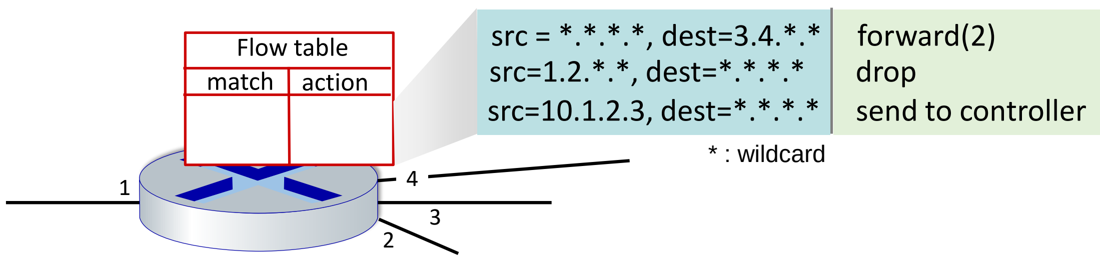
- flow: defined by header field values (in link-, network-, transport-layer fields)
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  -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



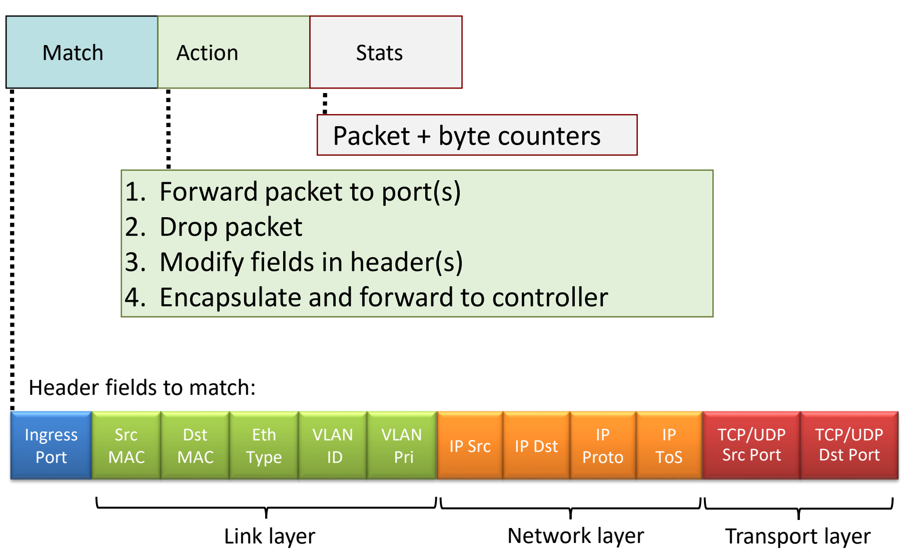
Router's flow table define router's match+action rules

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



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 abstraction

- match+action: abstraction unifies different kinds of devices

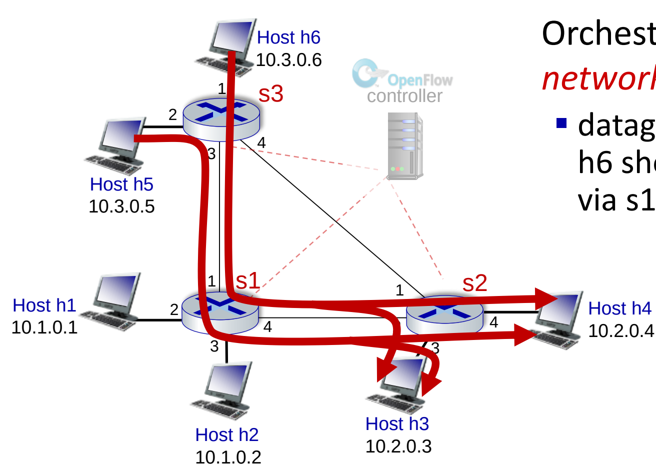
Router
- *match:* longest destination IP prefix
- *action:* forward out a link

Switch
- *match:* destination MAC address
- *action:* forward or flood

Firewall
- *match*: IP addresses and TCP/UDP port numbers
- *action:* permit or deny

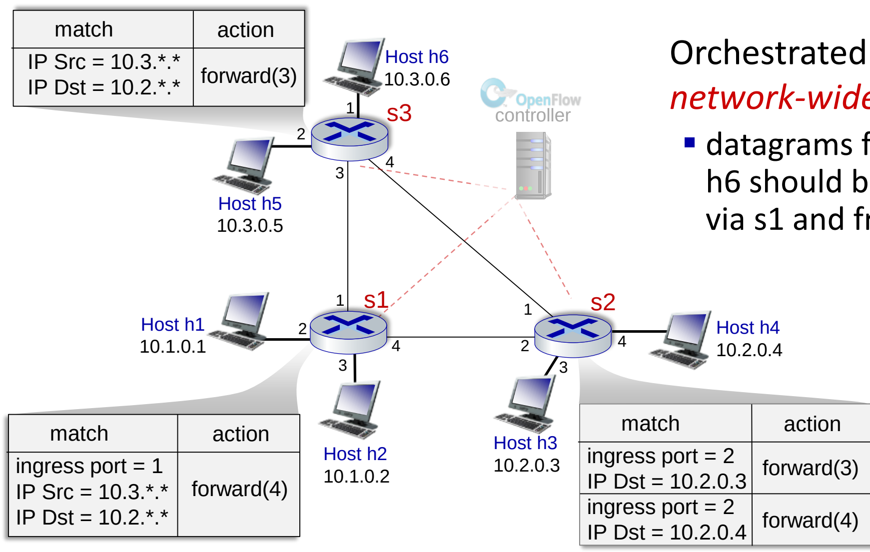
NAT
- *match:* IP address and port
- *action:* rewrite address and port

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



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

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 = 10.2.0.4	forward(4)

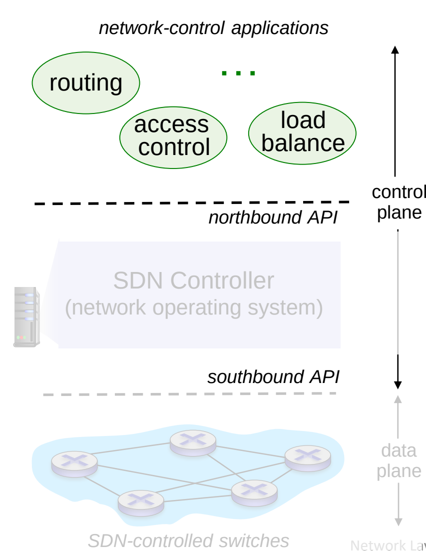
Orchestrated tables can create *network-wide* behavior, e.g.,:

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

Software defined networking (SDN)

network-control apps:

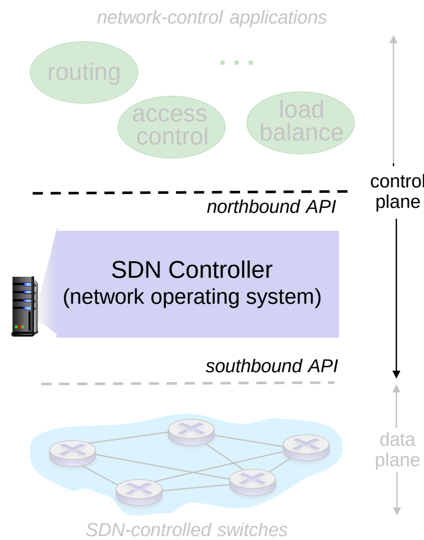
- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller



Software defined networking (SDN)

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

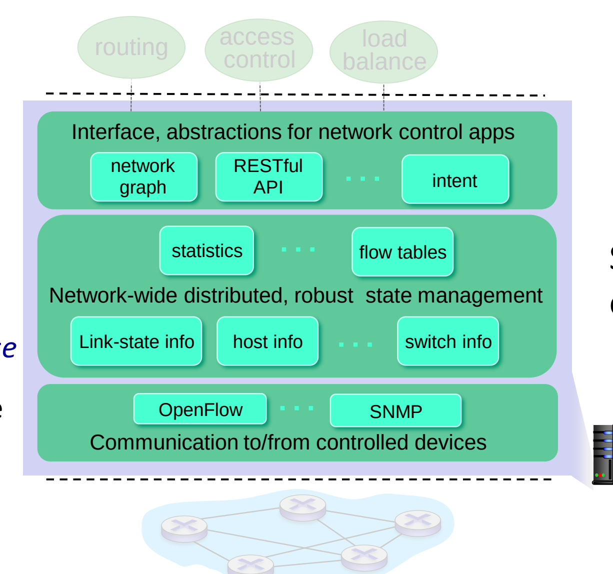


Components of SDN controller

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



Network Operating System

- Switches send connectivity info to controller
- Controller computes forwarding state
  - Some control program that uses the topology as input
- Controller sends forwarding state to switches
- Controller is replicated for resilience
  - System is only “logically centralized”

29

What is OpenFlow

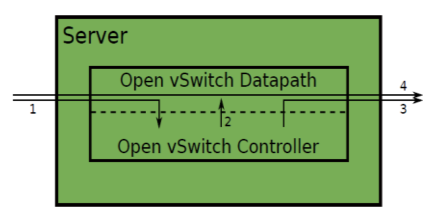
- OpenFlow is similar to an x86 instruction set for the network
- Provide open interface to “black box” networking node
  - (ie. Routers, L2/L3 switch) to enable visibility and openness in network
- OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries
- OpenFlow API used at controller to specify generalized rules in high-level language
- OpenFlow protocol messages for communication between entities

Packets are Managed as Flows

- A flow may be identied by any combination of
  - Input port
  - VLAN ID (802.1Q)
  - Ethernet Source MAC address
  - Ethernet Destination MAC address
  - IP Source MAC address
  - IP Destination MAC address
  - TCP/UDP/... Source Port
  - TCP/UDP/... Destination Port

Packets are Managed as Flows

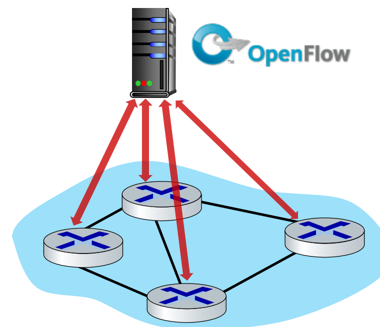
- The 1st packet of a flow is sent to the controller
- The controller programs the datapath's actions for a flow
  - Usually one, but may be a list
  - Actions include:
    - Forward to a port or ports
    - Encapsulate and forward to controller
    - Drop
- And returns the packet to the datapath
- Subsequent packets are handled directly by the datapath



OpenFlow protocol

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - switch to controller
  - miscellaneous

OpenFlow Controller

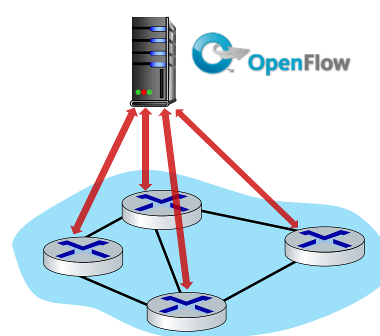


OpenFlow: controller-to-switch messages

Key controller-to-switch messages

- *features:* controller queries switch features, switch replies
- *configure:* 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

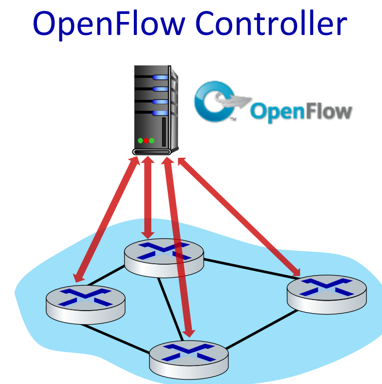
OpenFlow Controller



OpenFlow: switch-to-controller messages

Key switch-to-controller messages

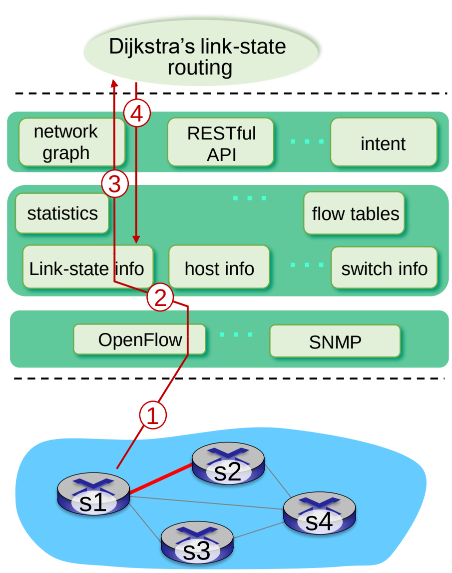
- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.



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

Network Layer: 5-36

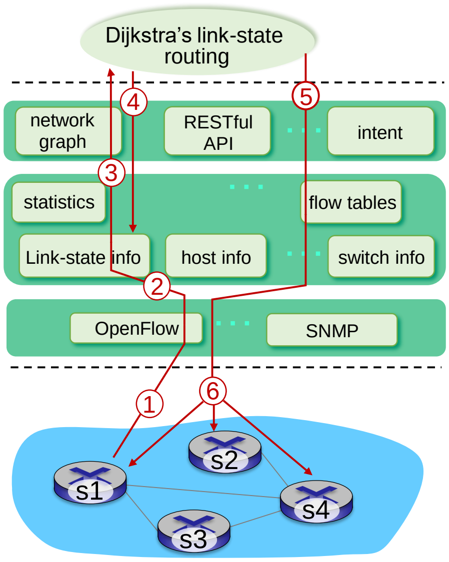
SDN: control/data plane interaction example



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 access network graph info, link state info in controller, computes new routes

Network Layer: 5-37

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

Google B4 Wide Area Network

- Interconnects data centers and server clusters
- SDN control plane built on OpenFlow
  - Maintain 70% utilization across network
  - Traffic engineering: load balancing, re-routing traffic, priority routing
- Custom built switches with OF agent
- Switches interact with control server using out-of-band network
- Controller replicated to handle failures
- Application running above control plane handles traffic engineering