Computer Networks

Jiaqi Zheng

Material with thanks Mosharaf Chowdhury, and many other colleagues.

Agenda

- Software-defined networking

The field of networking

- We have built a great artifact – the Internet
  - It grew mostly unrelated to the academic research, which came later
- CS networking today is largely the study of the Internet
- BUT we do not really have an academic discipline

Building an artifact, not a discipline

- Other fields in “systems”: OS, DB, etc.
  - Teach basic principles
  - Are easily managed
  - Continue to evolve
- Networking:
  - Teach big bag of protocols
  - Notoriously difficult to manage
  - Evolves very slowly
- Networks are much more primitive and less understood than other computer systems

A tale of two planes

- **Data plane**: forwarding packets
  - Based on local forwarding state
- **Control plane**: computing that forwarding state
  - Involves coordination with rest of system

Original goals for the control plane

- **Basic connectivity**: route packets to destination
  - Local state computed by routing protocols
  - Globally distributed algorithms
- **Inter-domain policy**: find policy-compliant paths
  - Done by globally distributed BGP
- What other goals are there in running a network?

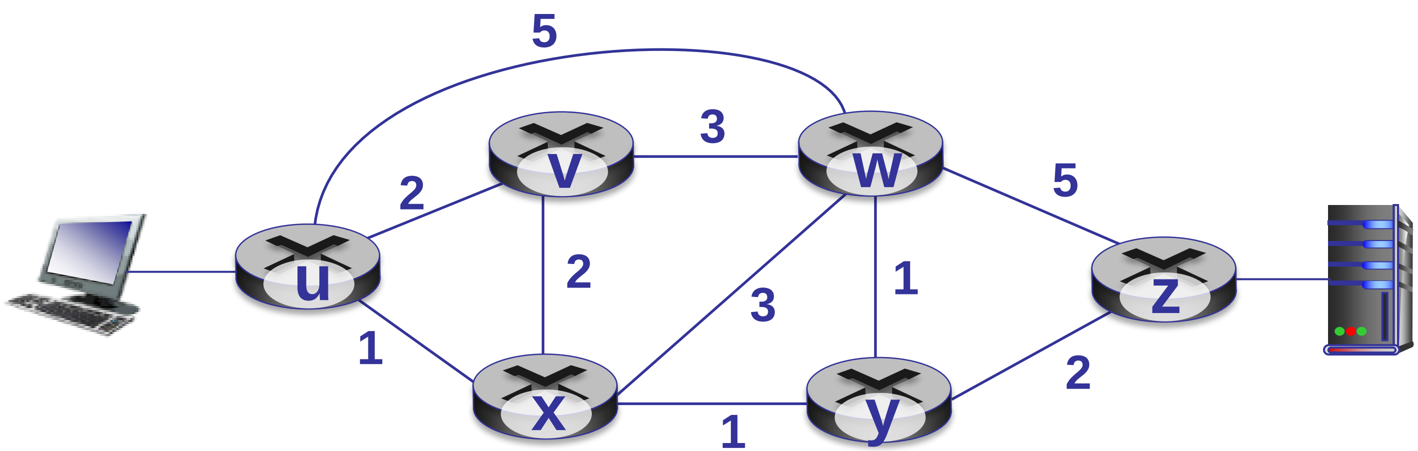
Extended roles of the control plane

- Performs various network management tasks
  - For example,
    - »Where to route?
    - »How much to route?
    - »At what rate to route?
    - »Should we route at all?
    - »…

Traffic engineering

- Want to avoid persistent overloads on links
- Choose routes to spread traffic load across links

Traffic engineering: Difficult

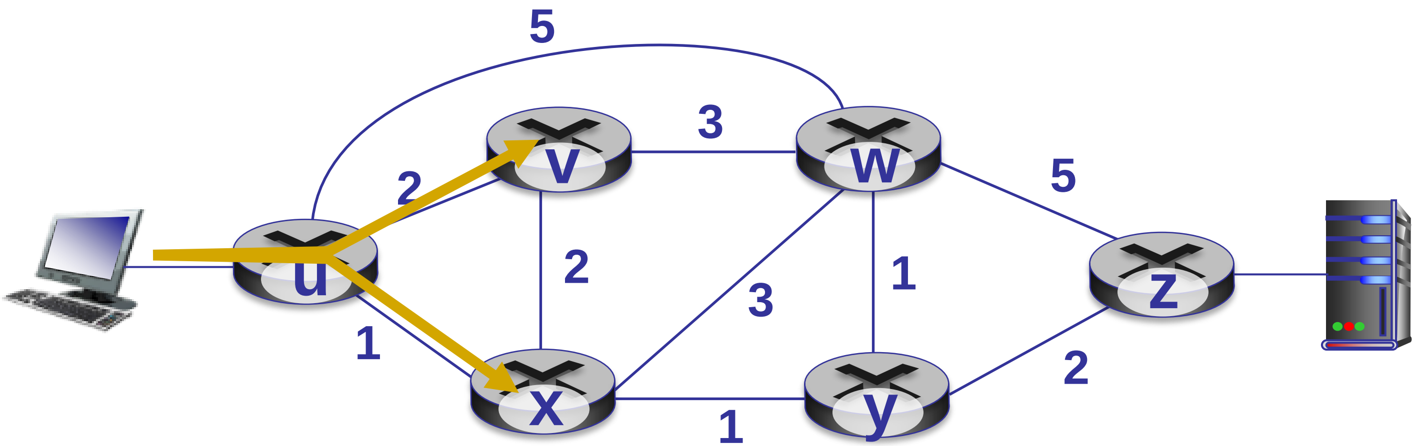


Q: What if network operator wants u-to-z traffic to flow along uvwz, x-to-z traffic to flow xwyz?

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

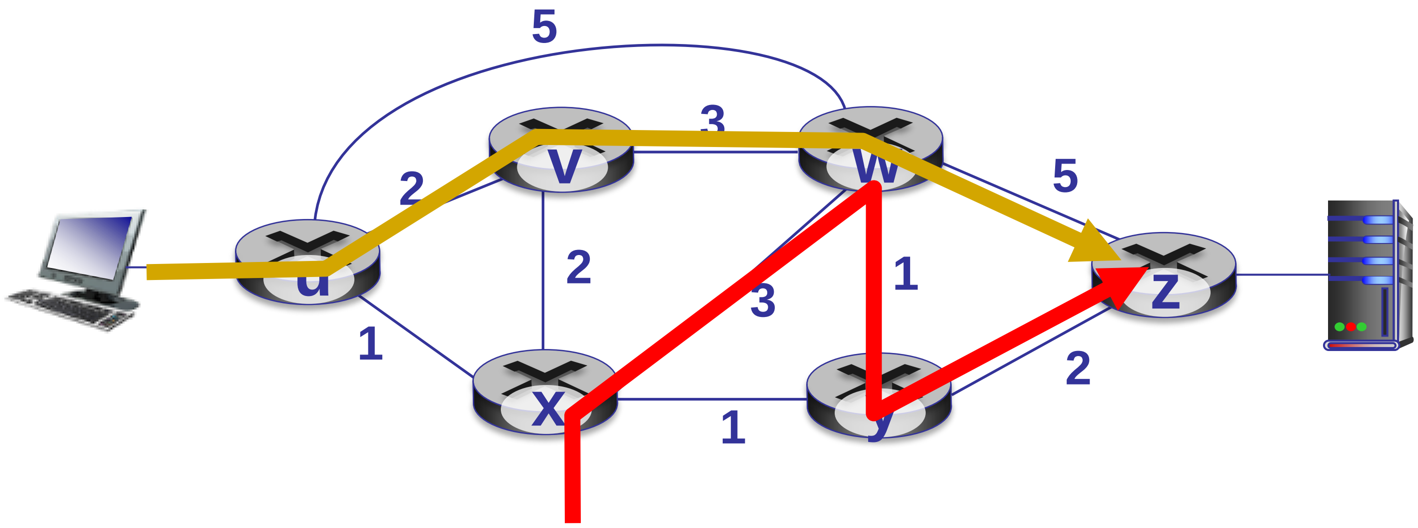
Link weights are only control "knobs"

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 the two flows differently?

A: Can’t do it (with LS or DV)

Network management has many goals

- Achieving these goals is job of the control plane…
- …which currently involves many mechanisms
- **Globally distributed:** Routing algorithms
- **Manual/scripted configuration:** ACLs
- **Centralized computation:** Traffic engineering

Bottom line

- Many different control plane mechanisms
- Each designed from scratch for their intended goal
- Encompassing a wide variety of implementations
  - Distributed, manual, centralized,…
- None of them particularly well designed
- Network control plane is a complicated mess!

"The Power of Abstraction"

- "Modularity based on abstraction is the way things get done"
  - Barbara Liskov
- Abstractions → Interfaces → Modularity

Analogy: Mainframe to PC evolution

Vertical integration, closed

- Specialized application
- Specialized operating system
- Specialized hardware

Open interfaces

- Arbitrary applications
- Commodity operating systems
- Microprocessor

We want the same for networking!

Many control plane mechanisms

- Variety of goals, no modularity
  - Routing: distributed routing algorithms
  - Isolation: ACLs, Firewalls,…
  - Traffic engineering: adjusting weights,…
- Control Plane: mechanism without abstraction
  - Too many mechanisms, not enough functionality

Task: Compute forwarding state

- Consistent with low-level hardware/software
  - Which might depend on particular vendor
- Based on entire network topology
  - Because many control decisions depend on topology
- For all routers/switches in network
  - Every router/switch needs forwarding state

Our current approach

- Design one-off mechanisms that solve all three
- Other fields would define abstractions for each subtask
- …and so should we!

Separate concerns with abstractions

- Be compatible with low-level hardware/software
  - Need an abstraction for general forwarding model
- Make decisions based on entire network
  - Need an abstraction for network state
- Compute configuration of each physical device
  - Need an abstraction that simplifies configuration

#1: Forwarding abstraction

- Express intent independent of implementation
  - Don’t want to deal with proprietary HW and SW
- OpenFlow is current proposal for forwarding
  - Standardized interface to switch
  - Configuration in terms of flow entries: <header, action>
- Design details concern exact nature of:
  - Header matching
  - Allowed actions

Two important facets to OpenFlow

- Switches accept external control messages
  - Not closed, proprietary boxes
- Standardized flow entry format
  - So switches are interchangeable

Separate concerns with abstractions

- Be compatible with low-level hardware/software
  - Need an abstraction for general forwarding model
- Make decisions based on entire network
  - Need an abstraction for network state
- Compute configuration of each physical device
  - Need an abstraction that simplifies configuration

#2: Network state abstraction

- Abstract away various distributed mechanisms
- Abstraction: **global network view**
  - Annotated network graph provided through an API
- Creates a logically centralized view of the network (Network Operating System)
  - Runs on replicated servers in network (“controllers”)
- Information flows both ways
  - Information *from* routers/switches to form “view”
  - Configurations *to* routers/switches to control forwarding

Network Operating System

- Think of it as a centralized link-state algorithm
- 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”
- Complicated protocols replaced with simple graph algorithms

Separate concerns with abstractions

- Be compatible with low-level hardware/software
  - Need an abstraction for general forwarding model
- Make decisions based on entire network
  - Need an abstraction for network state
- Compute configuration of each physical device
  - Need an abstraction that simplifies configuration

#3: Specification abstraction

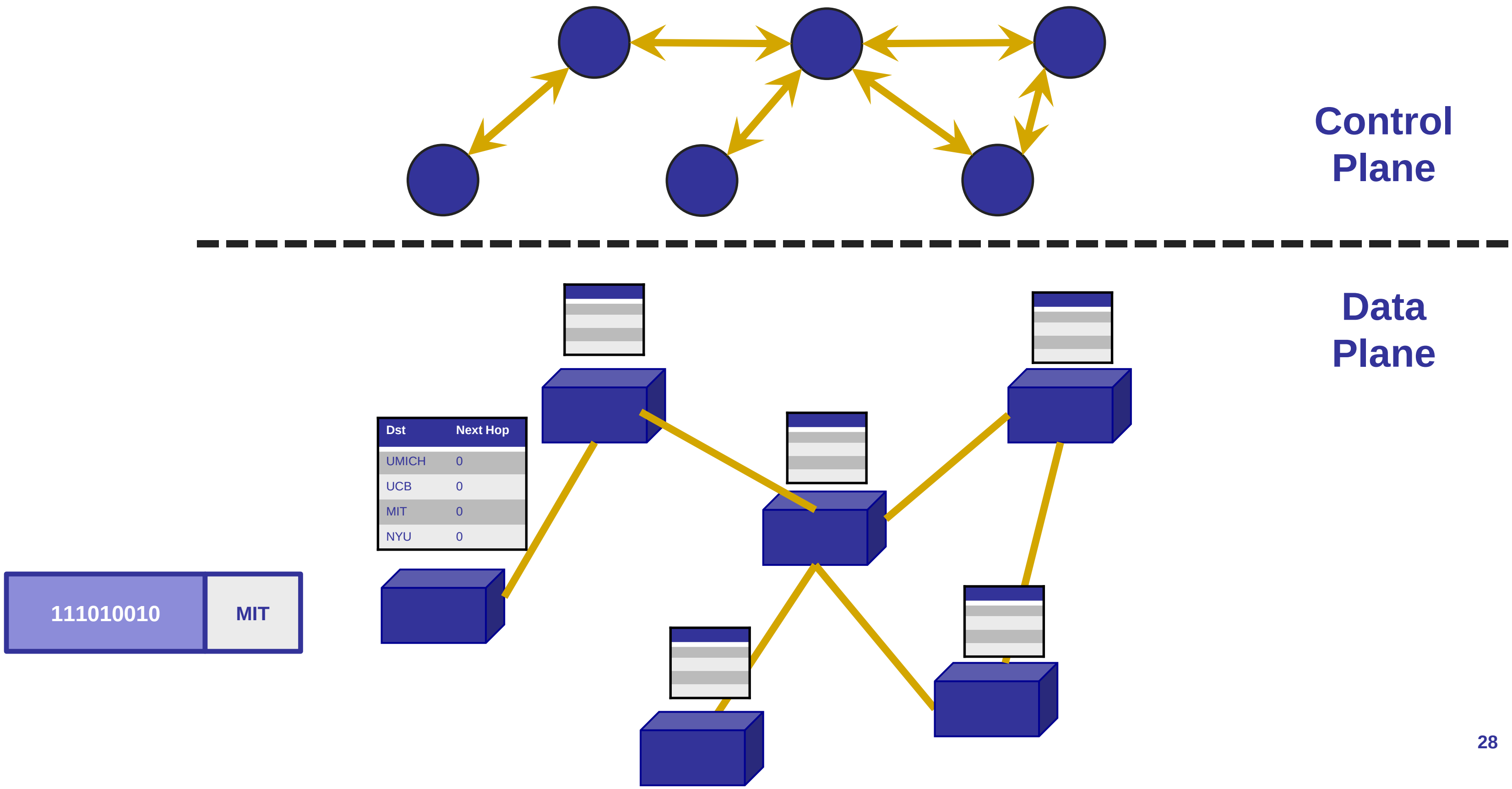
- Control mechanism expresses desired behavior
  - Whether it be isolation, access control, or QoS
- It should not be responsible for implementing that behavior on physical network infrastructure
  - Requires configuring the forwarding tables in each switch
- Abstract view of network
  - Models only enough detail to specify goals
  - Will depend on task semantics

Separate concerns with abstractions

- Be compatible with low-level hardware/software
  - Forwarding abstraction
- Make decisions based on entire network
  - Network state abstraction
- Compute configuration of each physical device
  - Specification abstraction

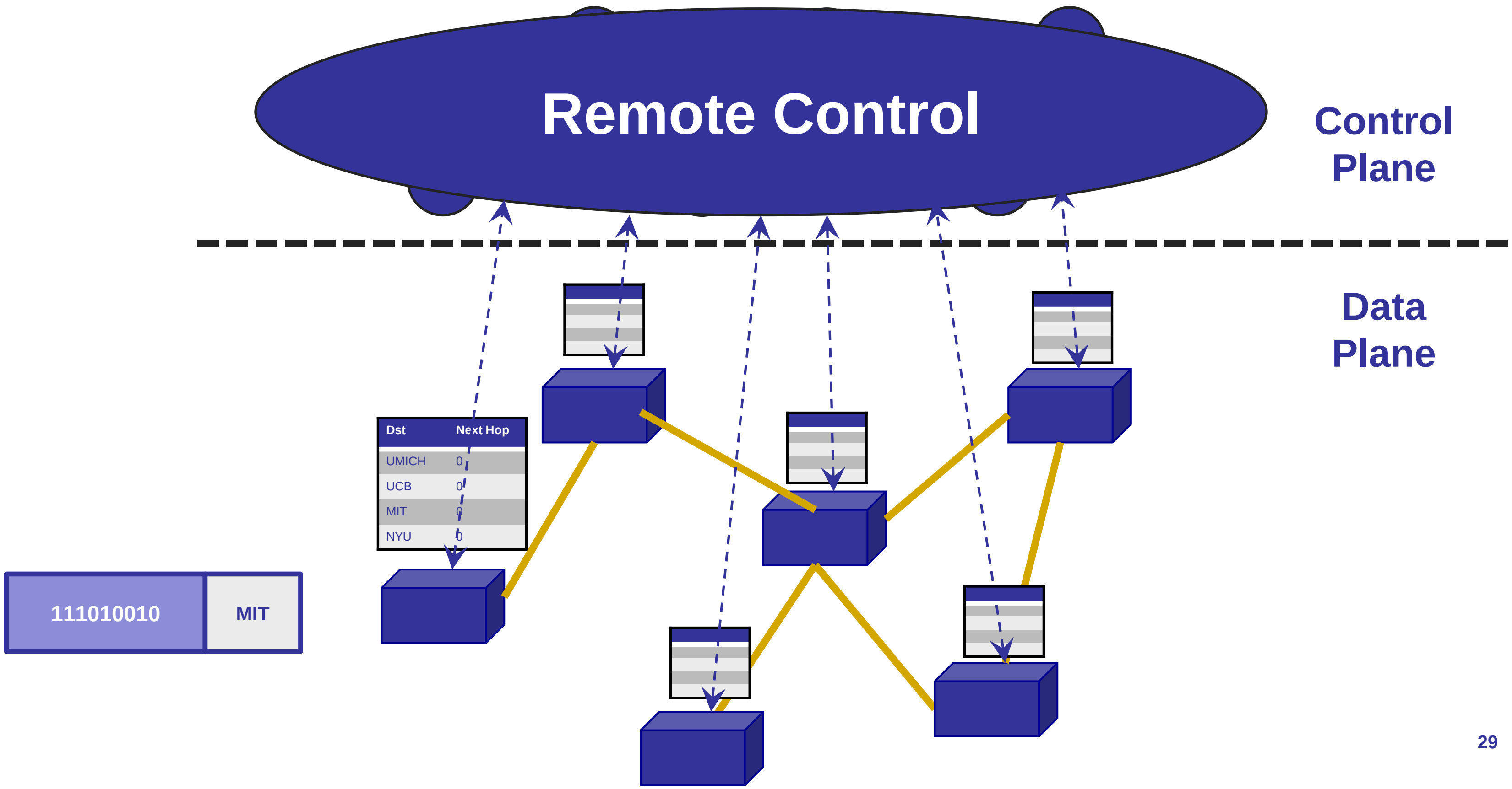
Traditional fully decentralized control plane

- Individual routing algorithm components in every router interact in the control plane



Logically centralized control plane

- A distinct (typically remote) controller interacts with local control agents (CAs)



Each goal is an app via specification abstraction

- What if an operator wants X?
- What if a customer wants to do weighted traffic splitting?
- …

- There is an app for it!
  - Write your own routing protocol, load balancing algorithm, access control policies

Reason about each app via network state abstraction

- Now that the network is not distributed anymore and is a simple graph, we can verify whether whatever we specified…
  - …makes sense
  - …likely to work
  - …likely to work with the rest
- No more *umm-I don't no-maybe*

Logically centralized control plane

- A distinct (typically remote) controller interacts with local control agents (CAs)
- Each router contains a flow table
- Each entry of the flow table defines a match-action rule
- Entries of the flow table is computed and distributed by the (logically) centralized controller

OpenFlow data plane abstraction

- Flow is 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. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow: Flow table entries

Rule	Action	Stats

Packet + byte counters

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline
5. Modify Fields

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

$\underbrace{\text{VLAN ID, MAC src, MAC dst, Eth type}}_{\text{Link layer}}$ $\underbrace{\text{IP Src, IP Dst, IP Prot}}_{\text{Network layer}}$ $\underbrace{\text{TCP sport, TCP dport}}_{\text{Transport layer}}$

Link layer **Network layer** **Transport layer**

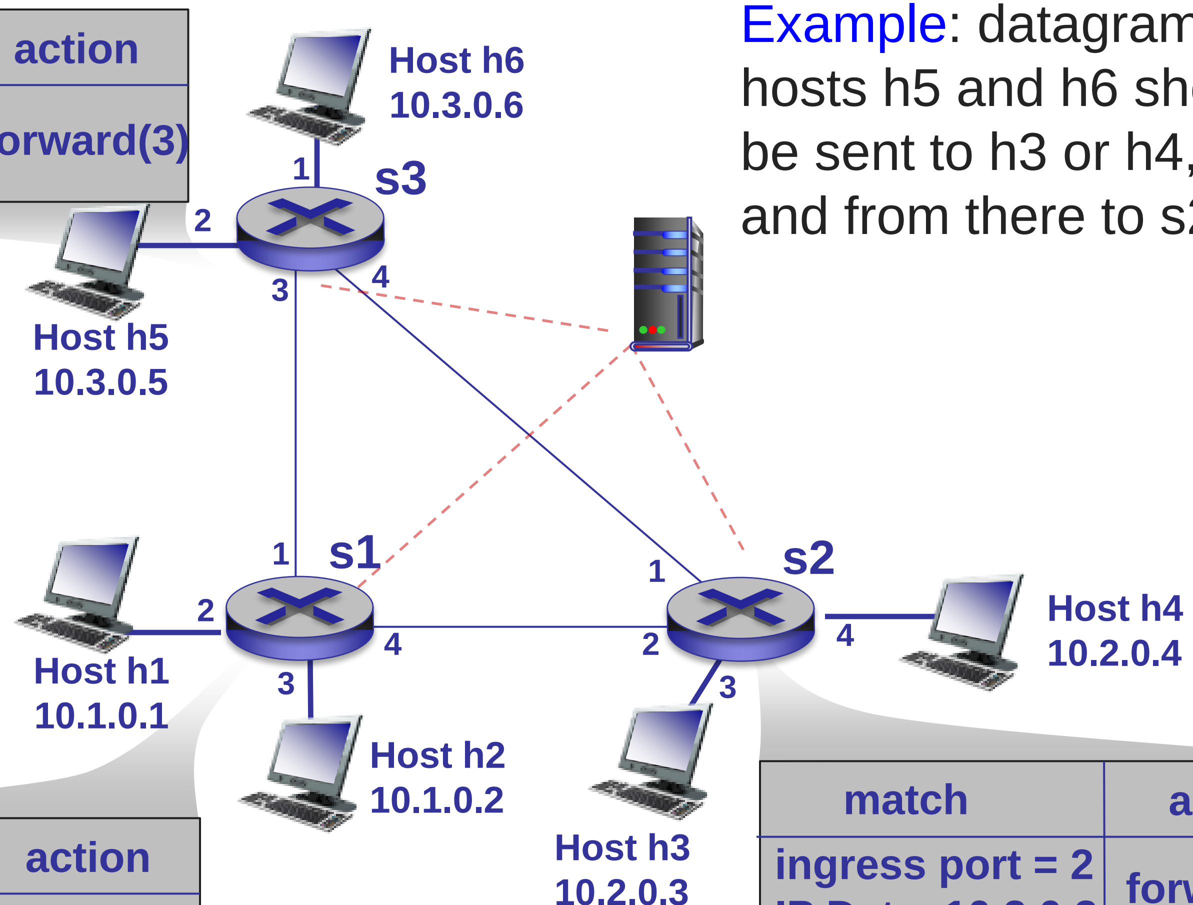
Forwarding abstraction

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

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

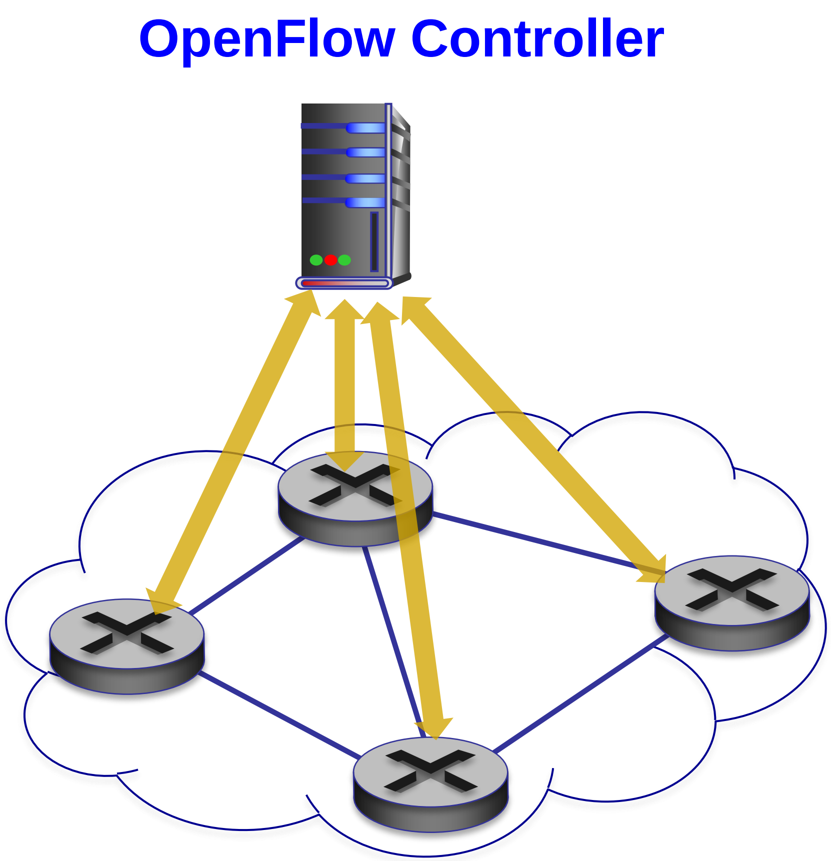


match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)

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)

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
  - 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: 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
- Network operators do not “program” switches by creating/sending OpenFlow messages directly.
  - Instead, they use higher-level abstraction at controller

SDN: Many challenges remain

- Hardening the control plane: 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, protocols meeting mission-specific requirements
  - E.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling

Some progress for SDN in the wide area

- Google and Microsoft use SDN to manage traffic between datacenters
- One centralized controller to rule the entire world (well, their world)

Google's WAN-SDN (B4)

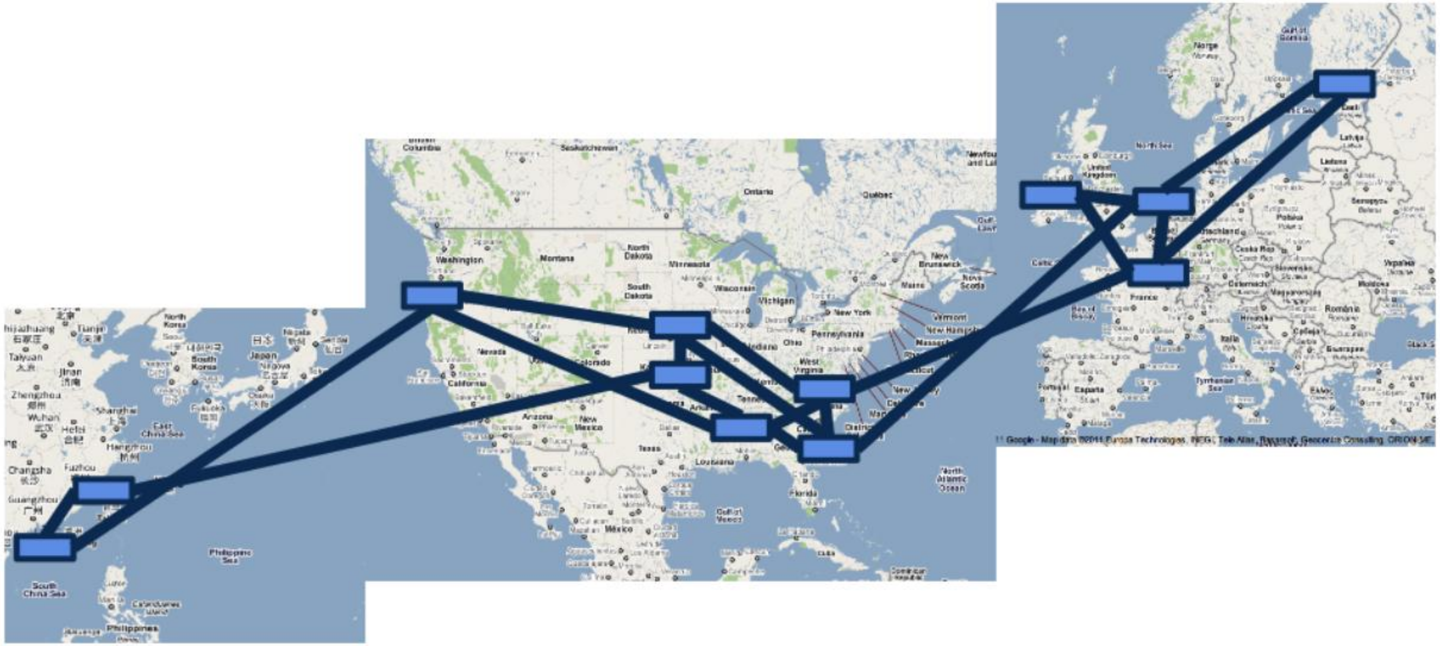


Figure 1: B4 worldwide deployment (2011).

Software-defined IXP (SDX)

- More expressive policies where ASes meet
- https://noise-lab.net/projects/software-defined-networking/sdx/

Summary

- Abstractions beget modularity
  - Modularity is (almost always) good