

Chapter 5

Network Layer: Control Plane

Part 2

Instructor: Zhuozhao Li

Lab: Qing Wang

Department of Computer Science and Engineering

Miscellaneous

- PA3: modified requirements on Sakai

Quick review

- Network-layer – Control plane
 - Distance vector
 - Good news
 - Bad news
 - Intra-AS routing: OSPF
 - Link state
 - Public
 - Hierarchical OSPF
 - Inter-AS routing: BGP
 - iBGP and eBGP
 - Policy
 - ICMP
 - ping and traceroute

Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- **SDN control plane**
- Internet Control Message Protocol



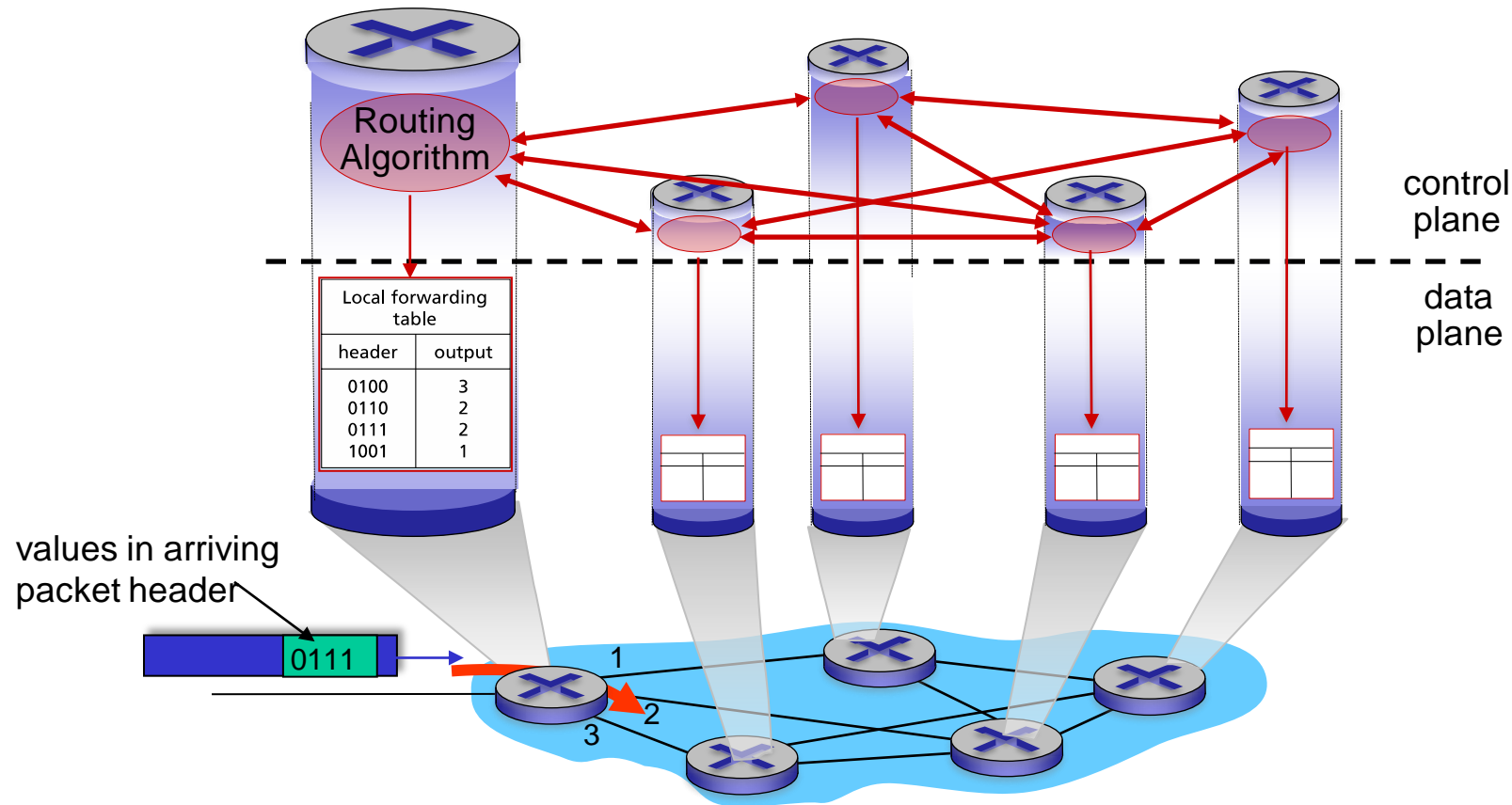
- network management, configuration
 - SNMP

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

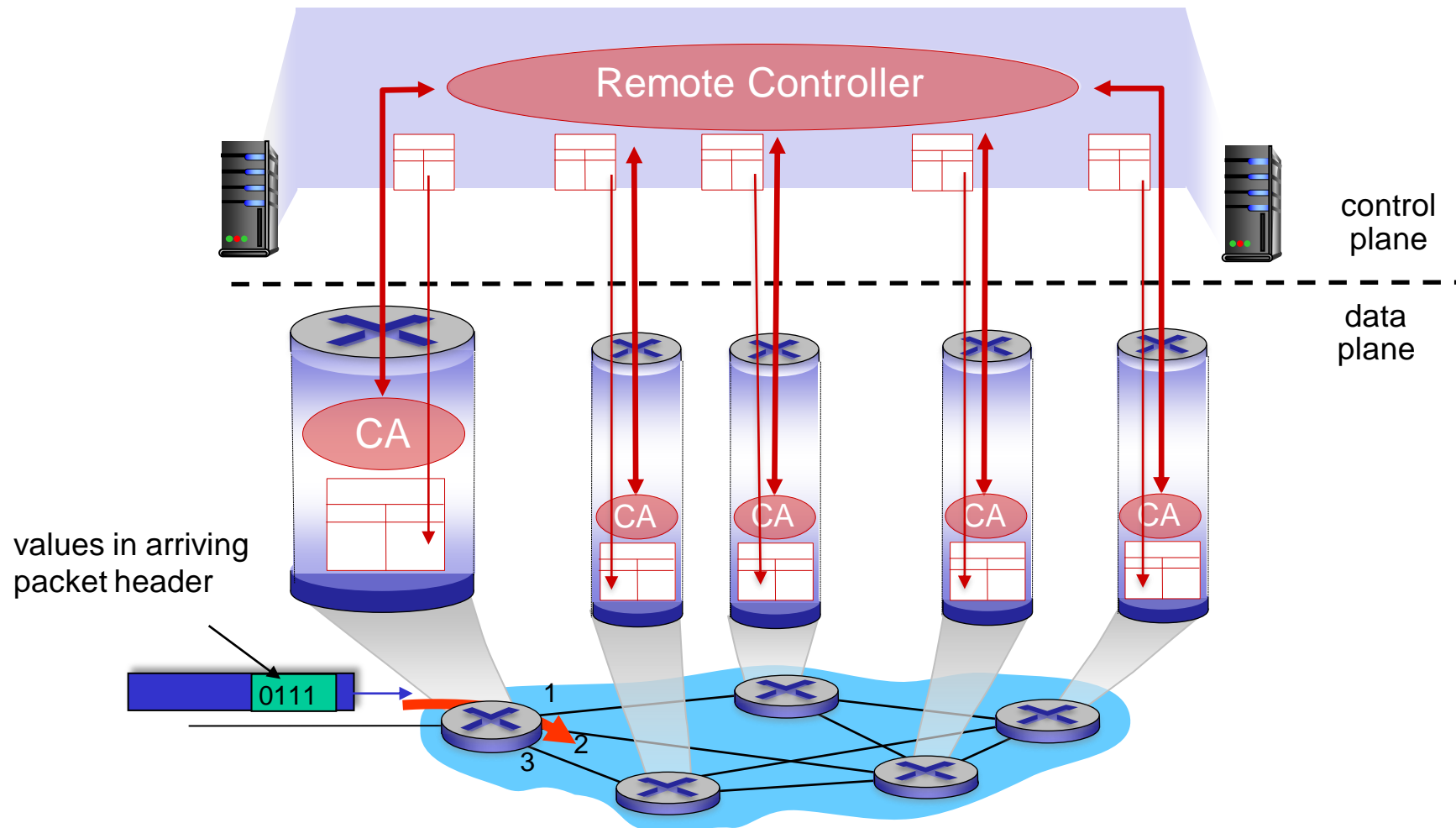
Per-router control plane

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



Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers

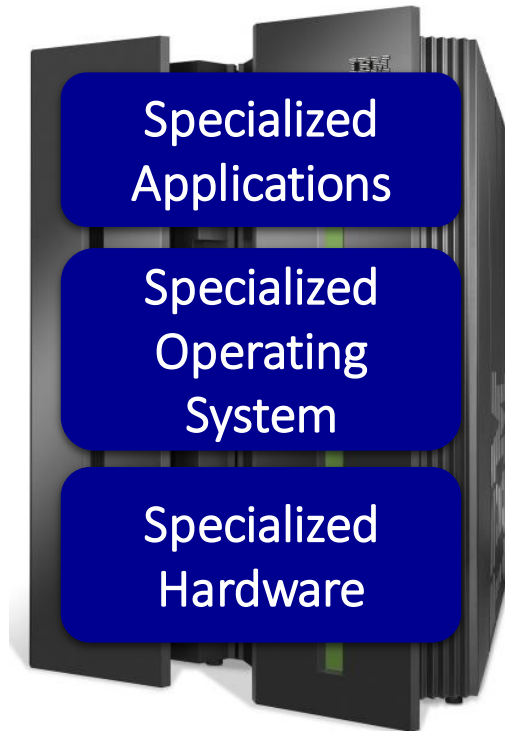


Software defined networking (SDN)

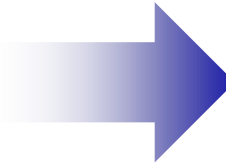
Why a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows “programming” routers
 - 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



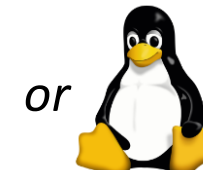
Vertically integrated
Closed, proprietary
Slow innovation
Small industry



—— Open Interface ——



Windows



Linux

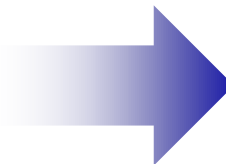


MAC OS

—— Open Interface ——

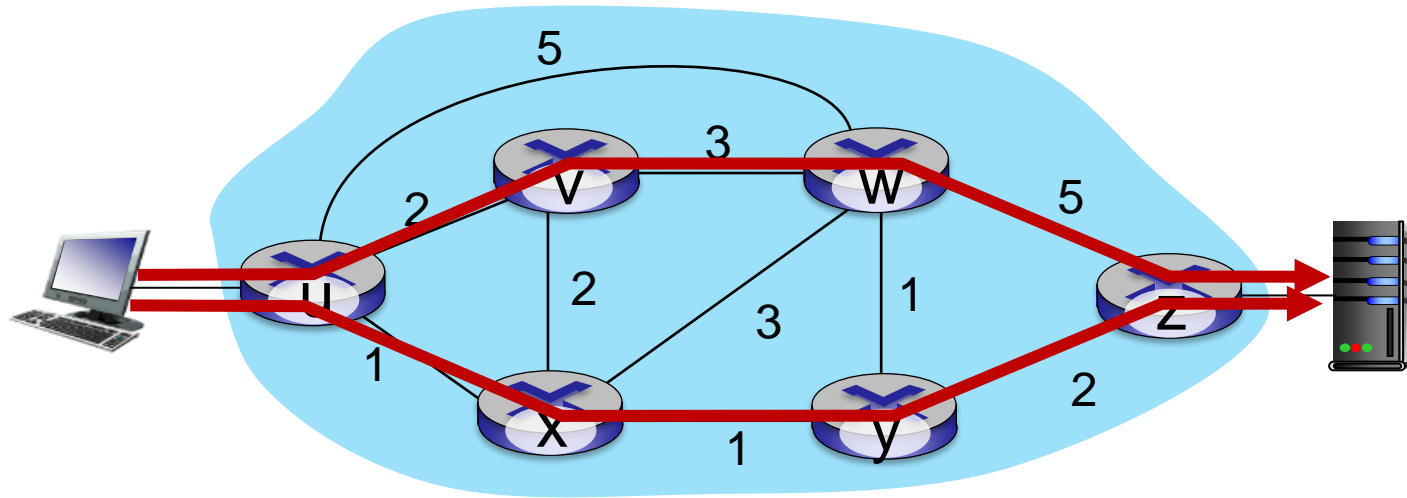


Microprocessor



Horizontal
Open interfaces
Rapid innovation
Huge industry

Traffic engineering: difficult with traditional routing

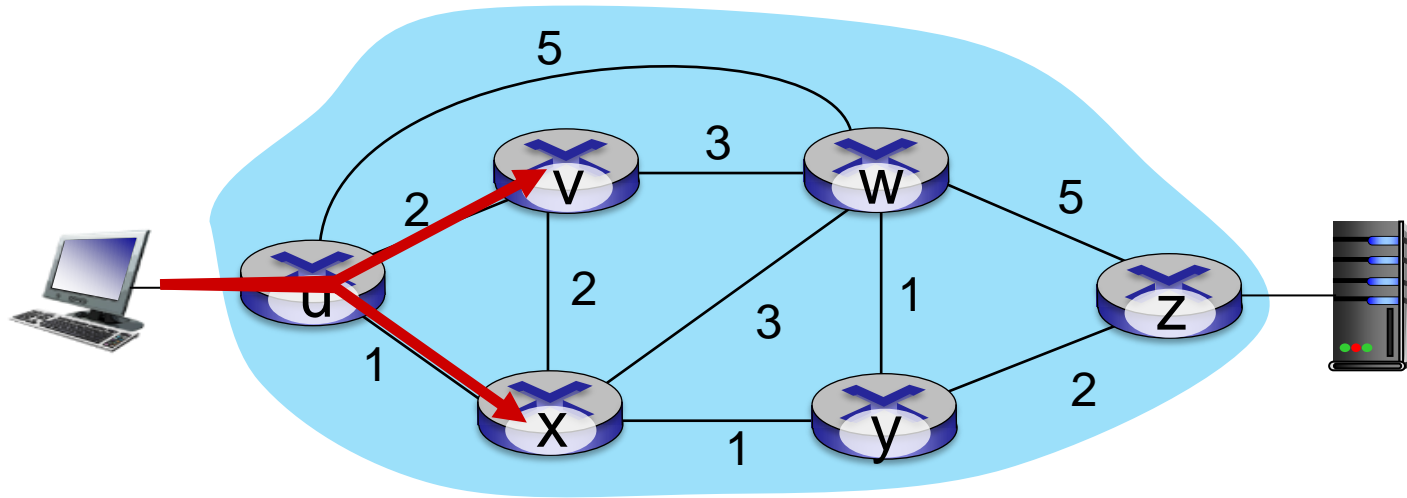


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

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!

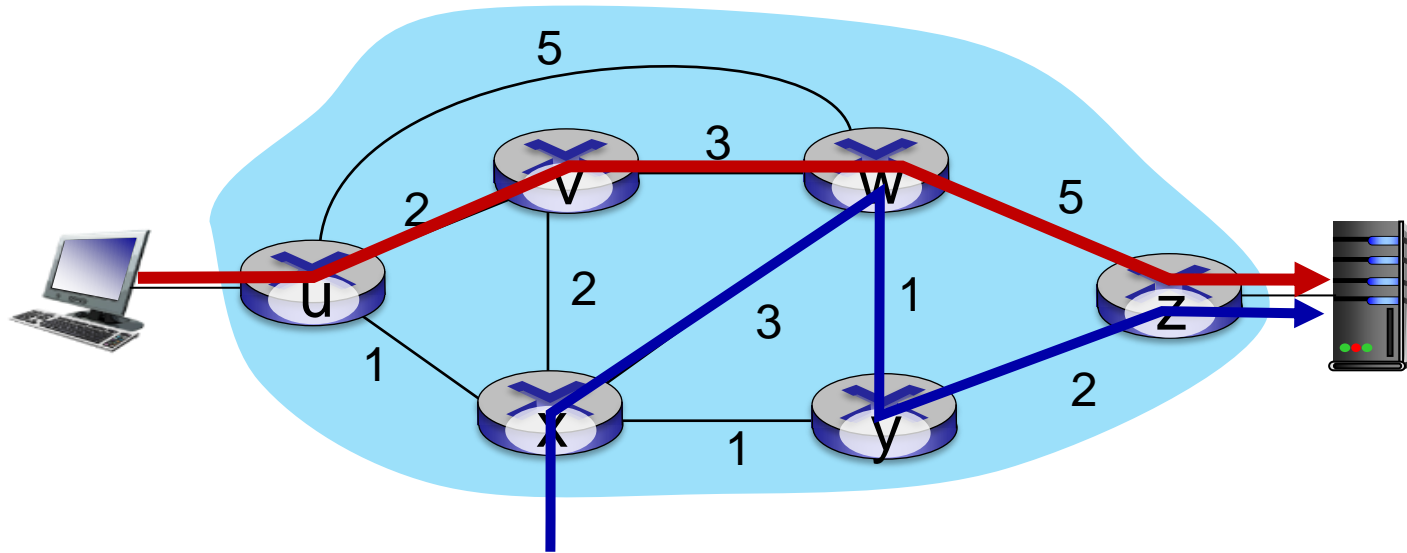
Traffic engineering: difficult with traditional routing



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

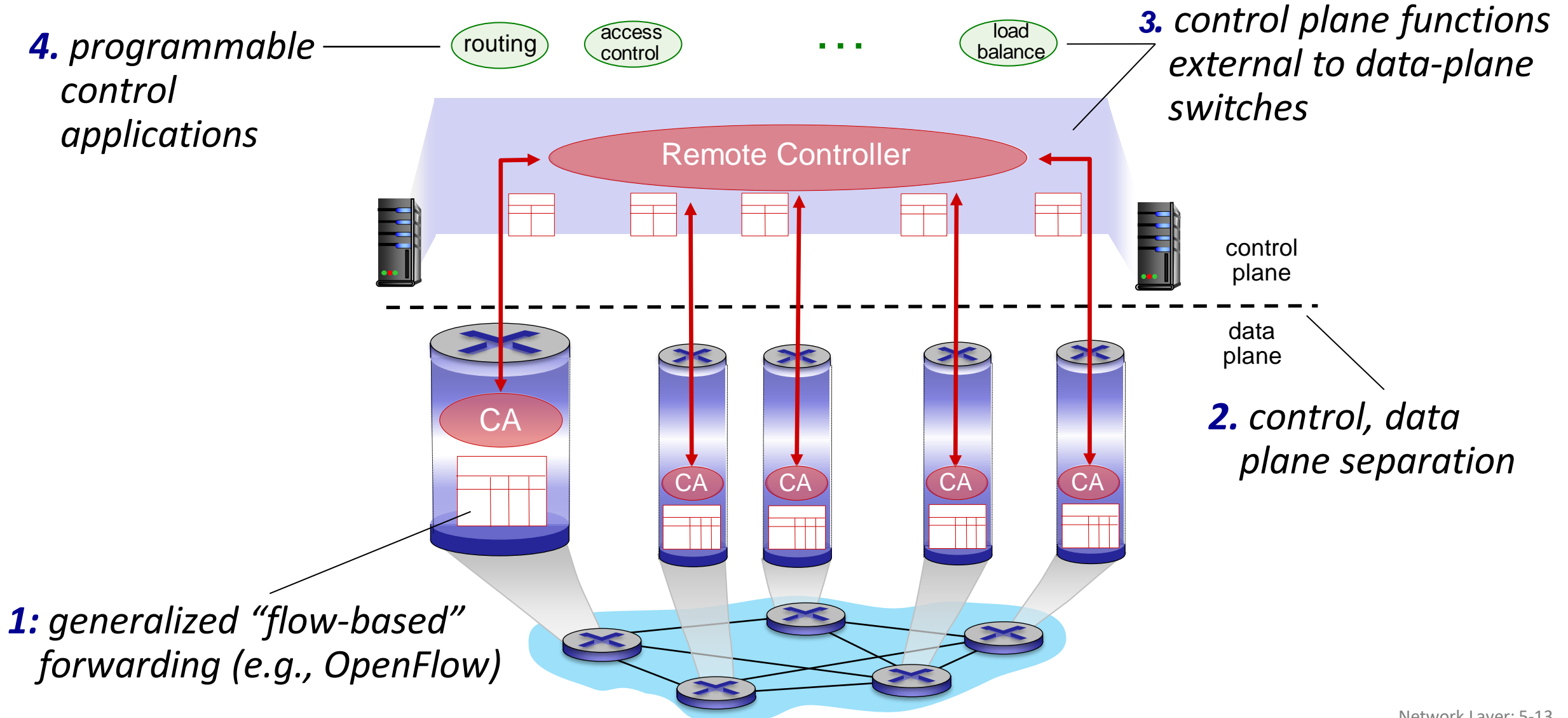


Q: what if w wants to route blue and red traffic differently from w to z?

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

We learned in Chapter 4 that generalized forwarding and SDN can be used to achieve *any* routing desired

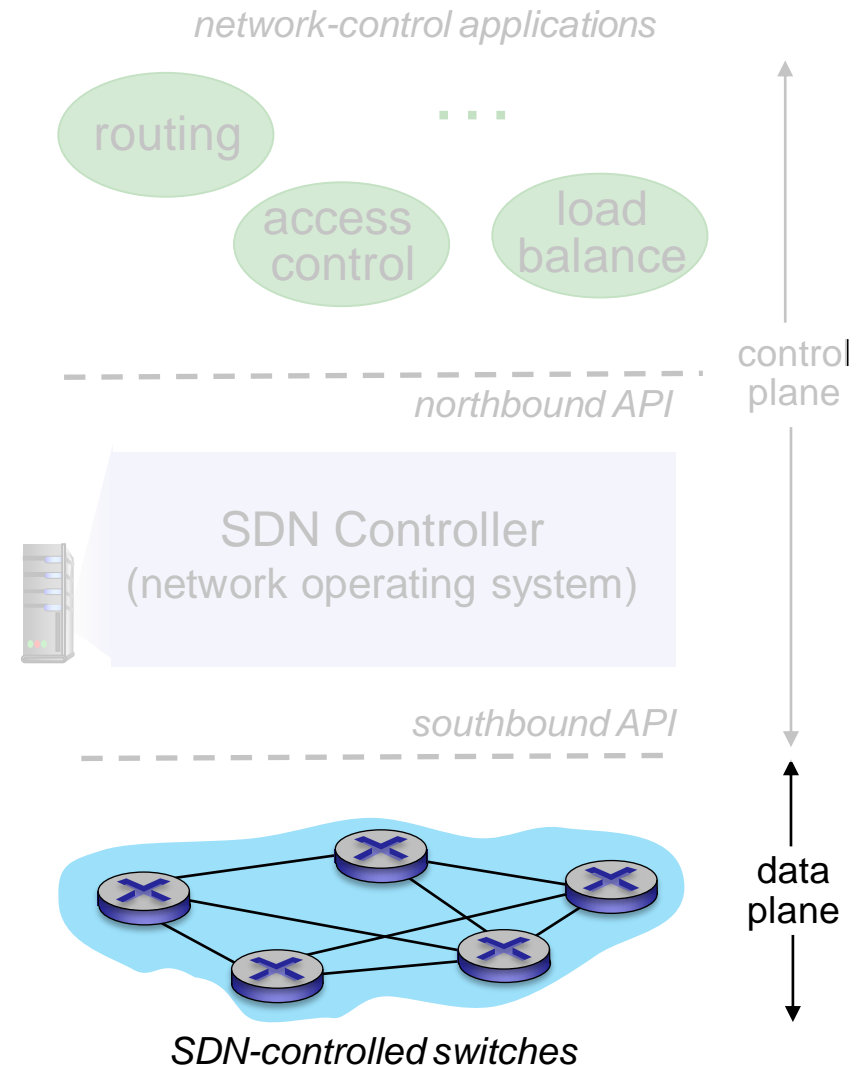
Software defined networking (SDN)



Software defined networking (SDN)

Data-plane switches:

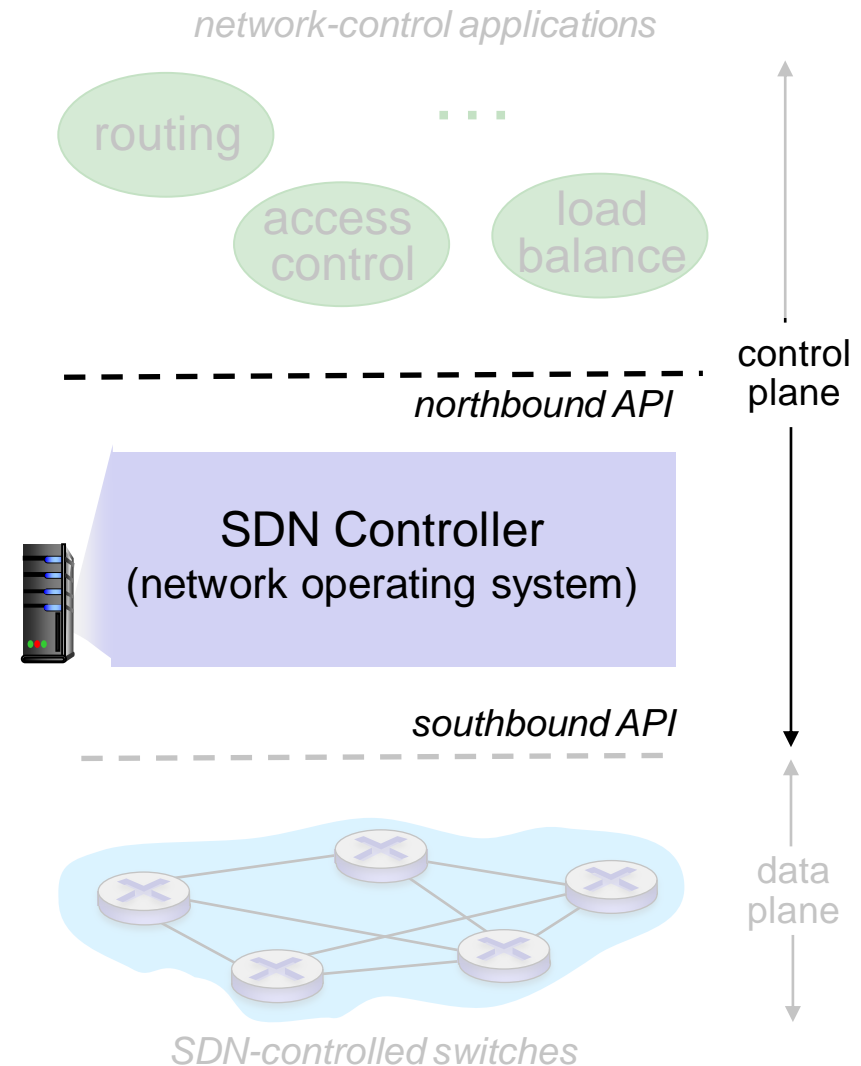
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable, what is not
- protocol for communicating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



Software defined networking (SDN)

SDN controller (network OS):

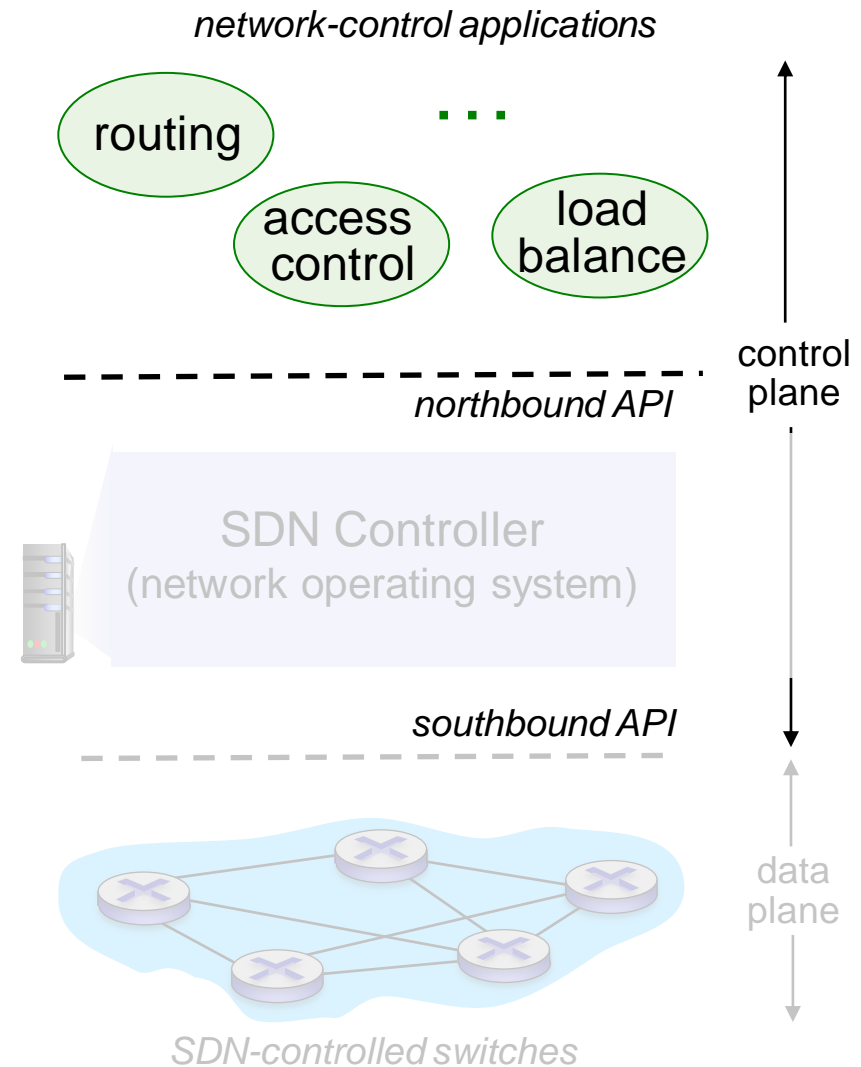
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



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

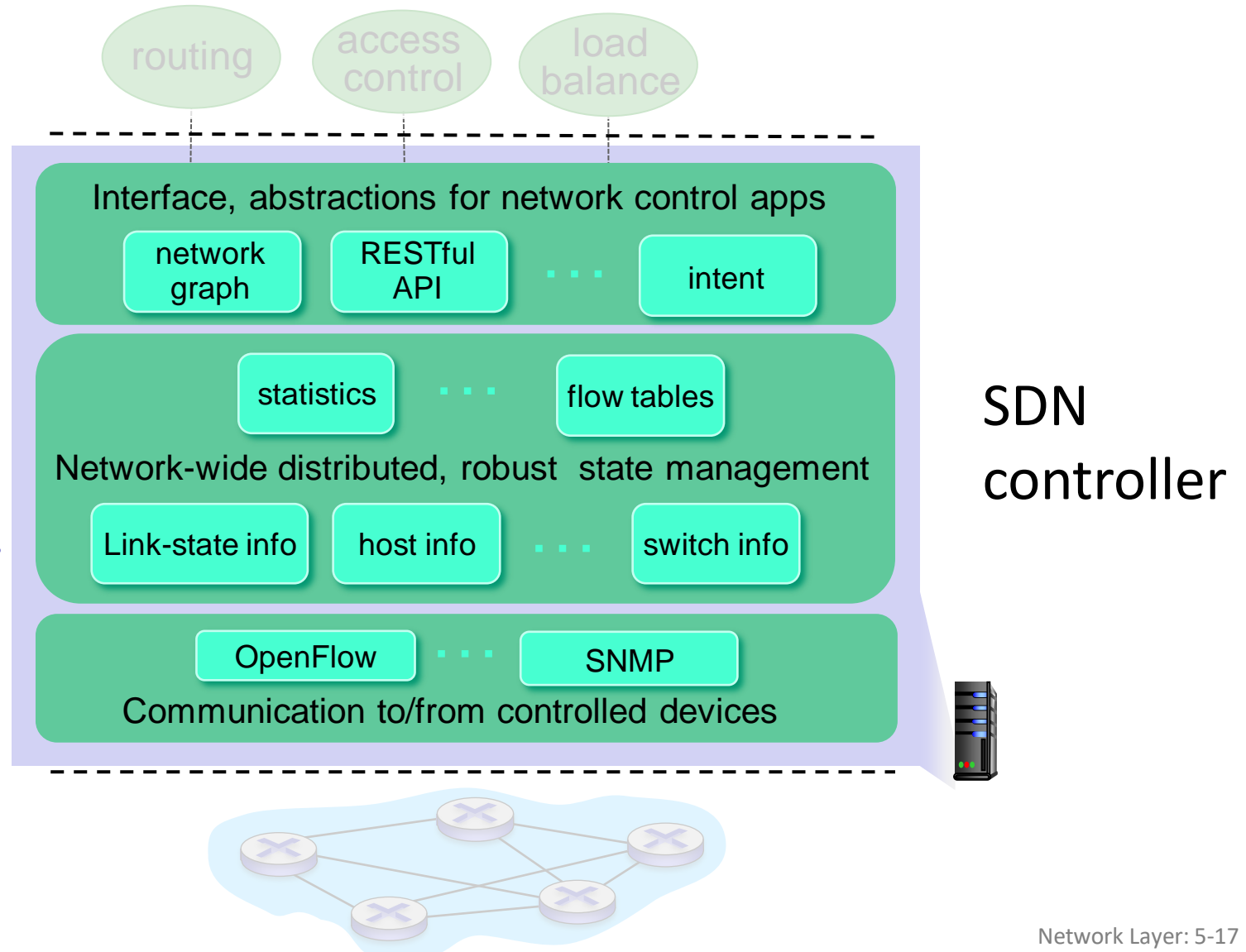


Components of SDN controller

interface layer to network control apps: abstractions API

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

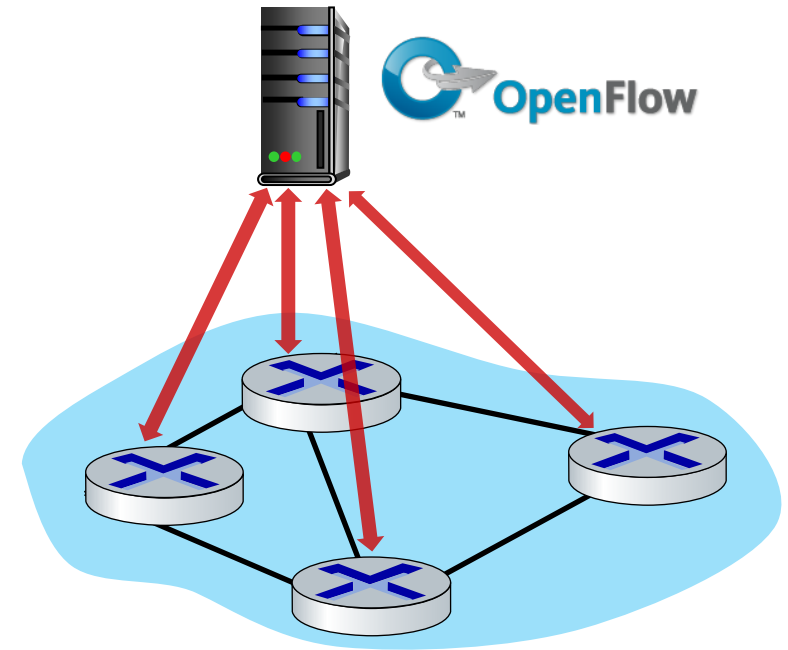
communication: communicate between SDN controller and controlled switches



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.)

OpenFlow Controller

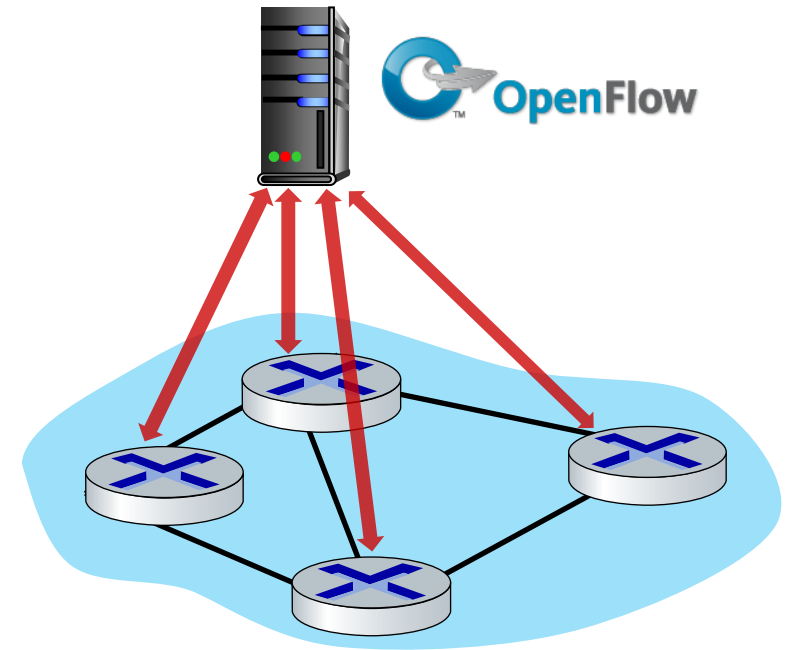


OpenFlow: controller-to-switch messages

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

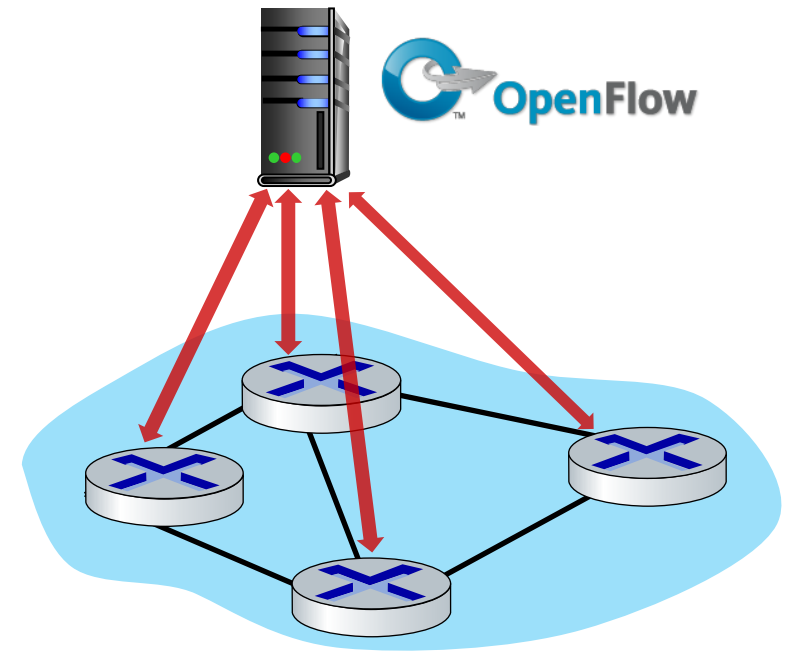


OpenFlow: switch-to-controller messages

Key switch-to-controller messages

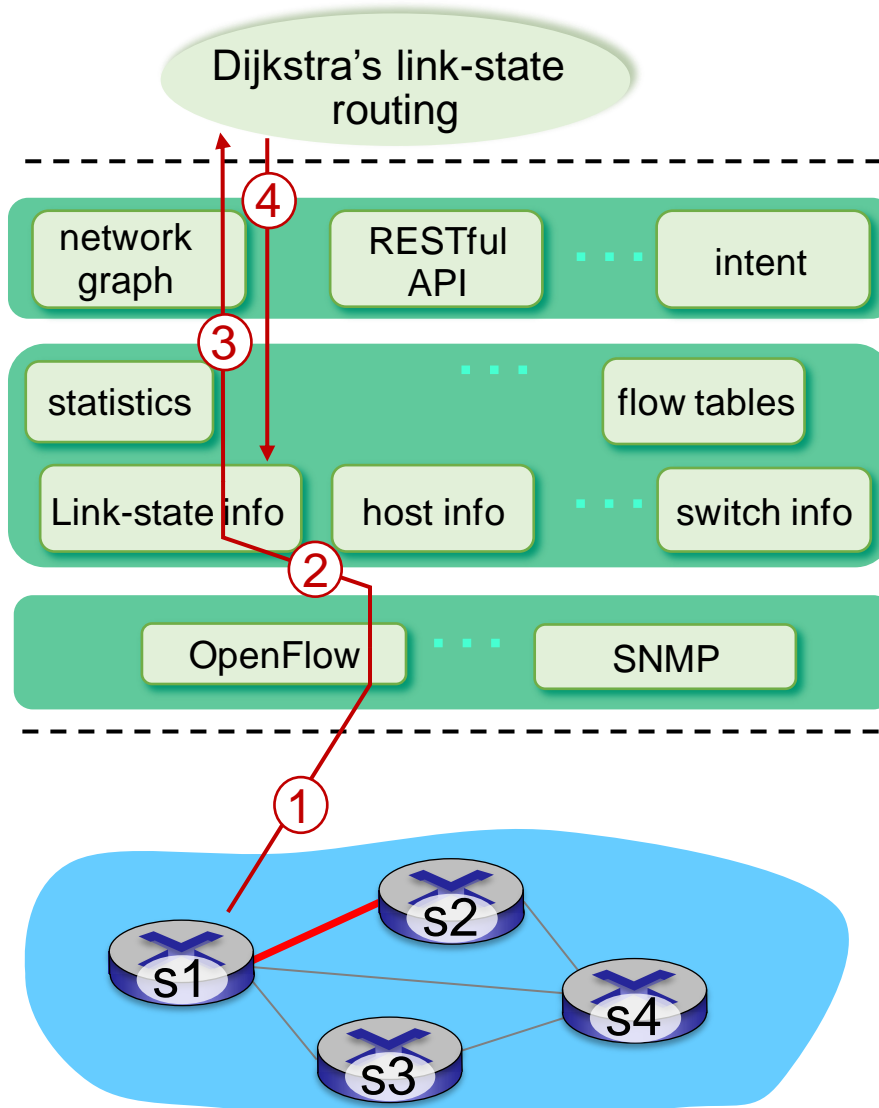
- *packet-in*: transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed*: flow table entry deleted at switch
- *port status*: inform controller of a change on a port.

OpenFlow Controller



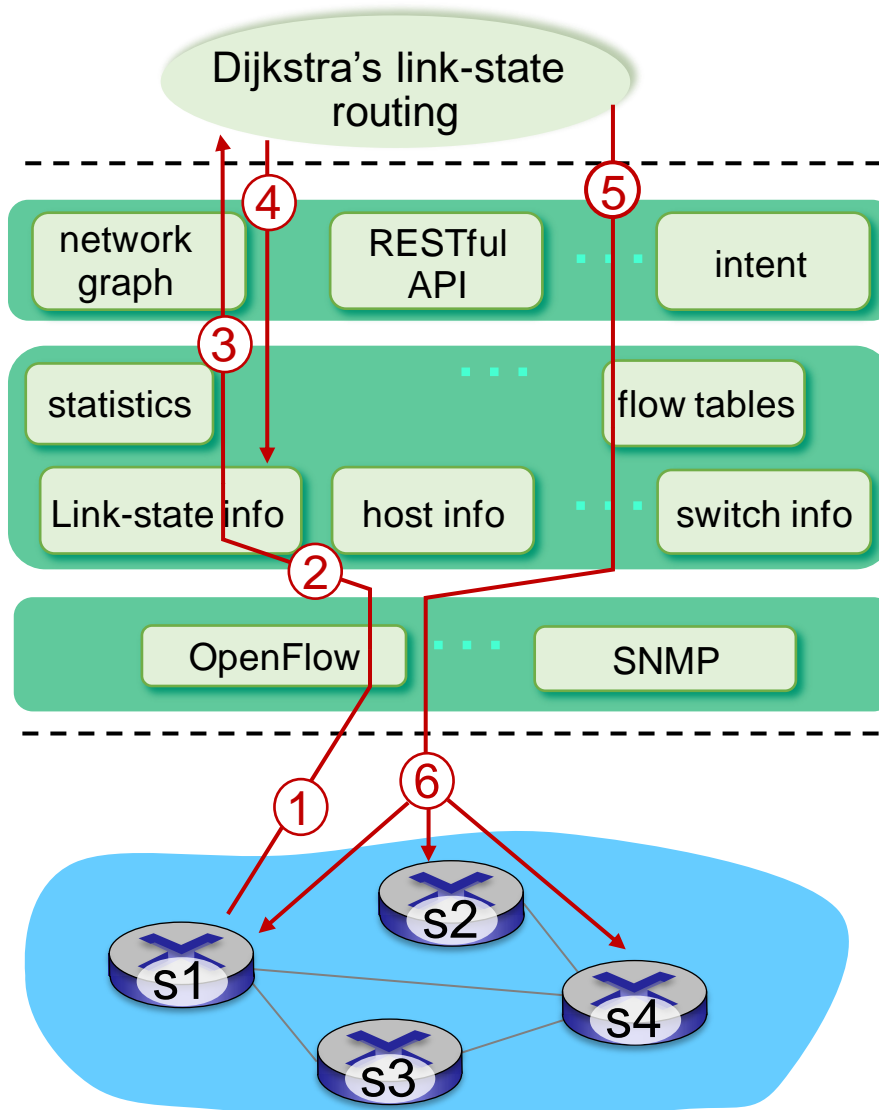
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

SDN: control/data plane interaction example



- ① S1, experiencing link failure 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

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

Network layer: “control plane” roadmap

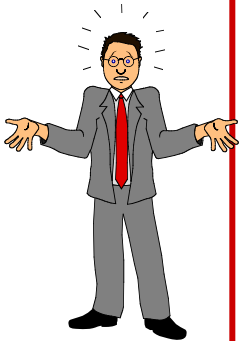
- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
 - SNMP

What is network management?

- autonomous systems (aka “network”): 1000s of interacting hardware/software components
- other complex systems requiring monitoring, configuration, control:
 - jet airplane, nuclear power plant, others?



"**Network management** includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

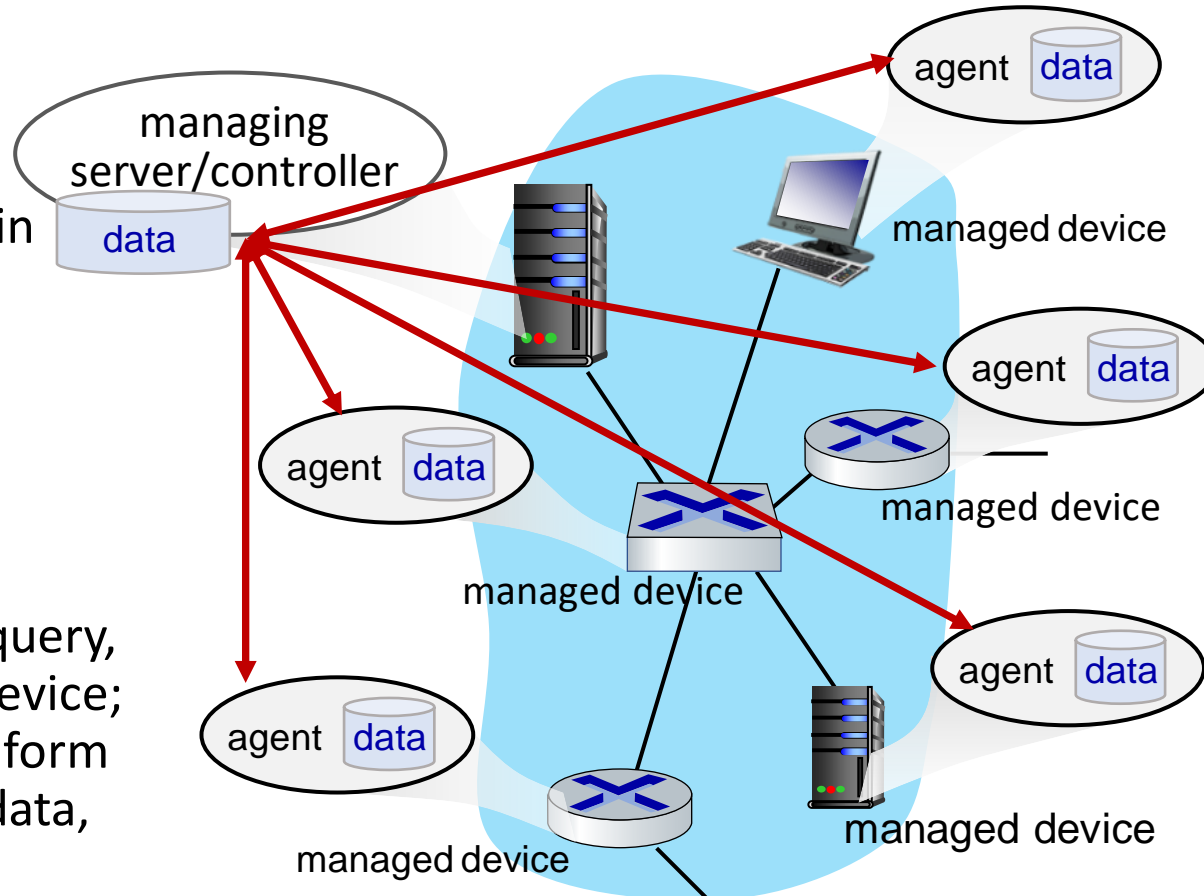
Components of network management

Managing server:

application, typically with network managers (humans) in the loop

Network management protocol:

used by managing server to query, configure, manage device; used by devices to inform managing server of data, events.



Managed device:

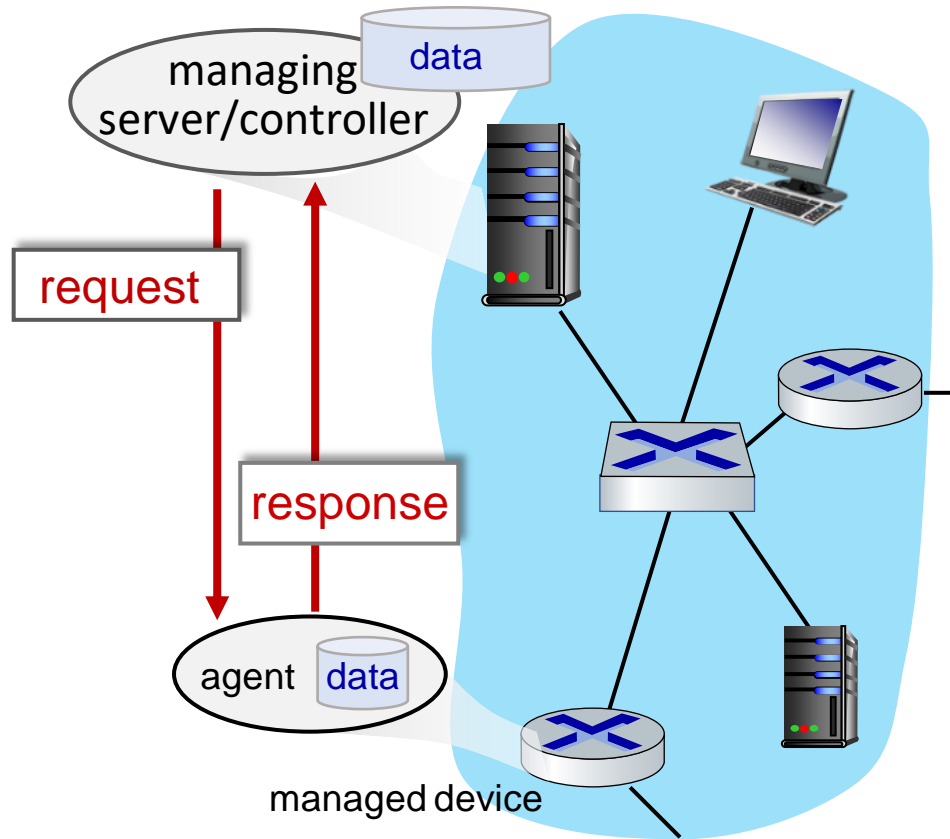
equipment with manageable, configurable hardware, software components

Data:

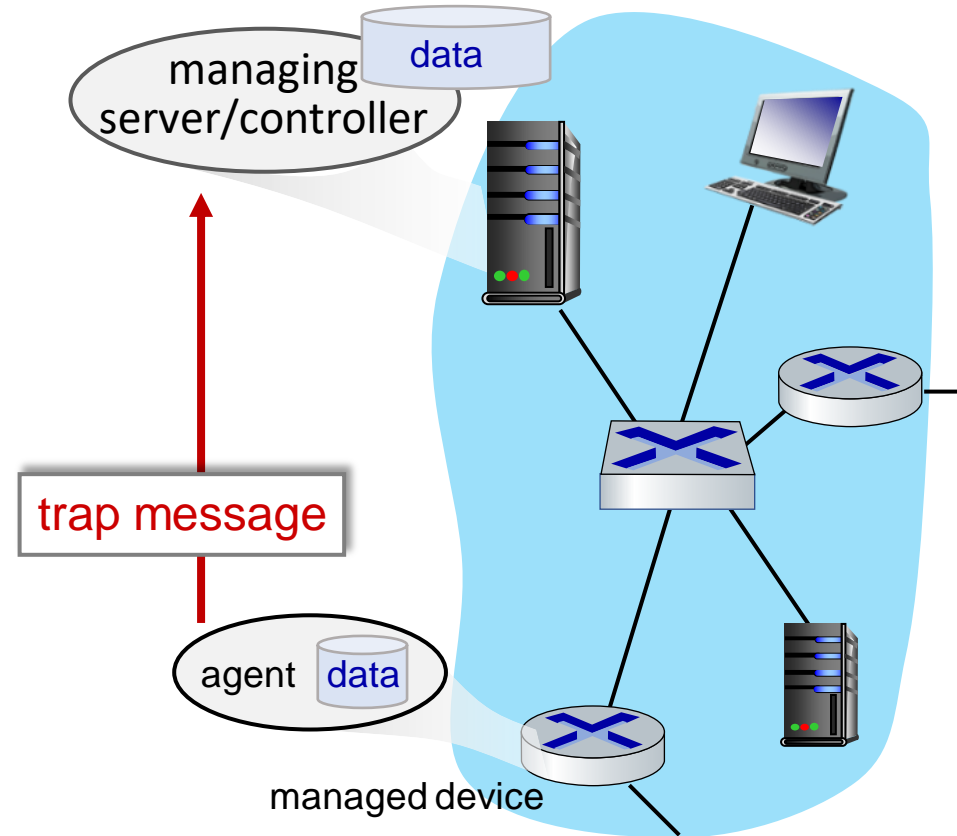
device “state”
configuration data,
operational data,
device statistics

Simple Network Management Protocol (SNMP)

Two ways to convey MIB info, commands:



request/response mode



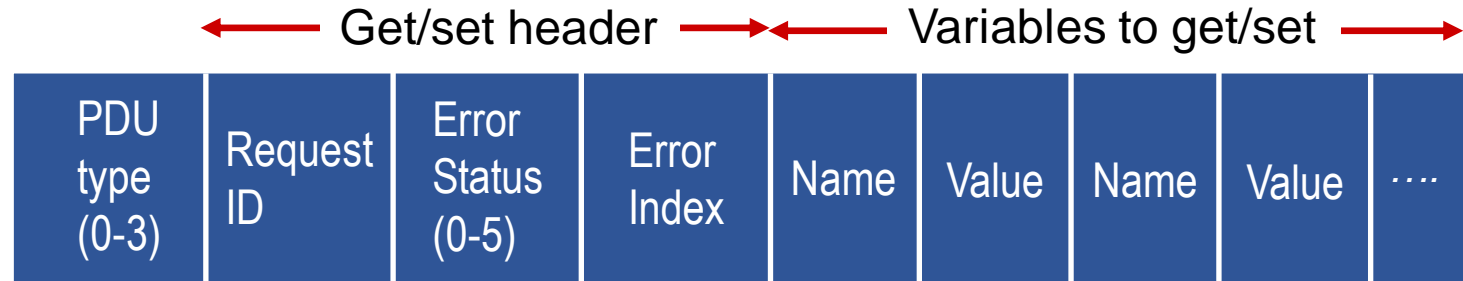
trap mode

SNMP protocol: message types

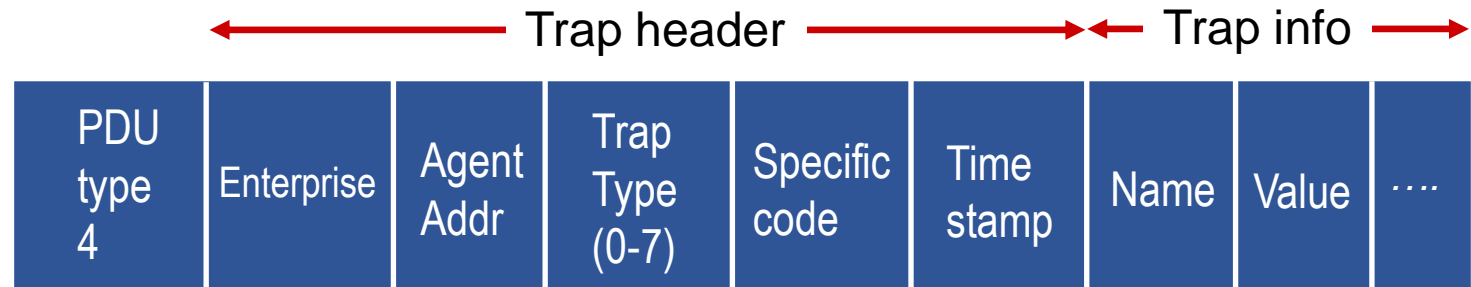
Message type	Function
GetRequest GetNextRequest GetBulkRequest	manager-to-agent: “get me data” (data instance, next data in list, block of data).
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

SNMP protocol: message formats

message types 0-3



message type 4



Network layer: Summary

we've learned a lot!

- approaches to network control plane
 - per-router control (traditional)
 - logically centralized control (software defined networking)
- traditional routing algorithms
 - implementation in Internet: OSPF , BGP
- SDN controllers
- Internet Control Message Protocol
- network management

next stop: link layer!

Chapter 6

The Link Layer and LANs

Part 1

Instructor: Zhuozhao Li

Lab: Qing Wang

Department of Computer Science and Engineering

Link layer, LANs: roadmap

- introduction
- error detection, correction
- multiple access protocols
- LANs
 - addressing, ARP
 - Ethernet
 - switches
 - VLANs
- link virtualization: MPLS
- data center networking



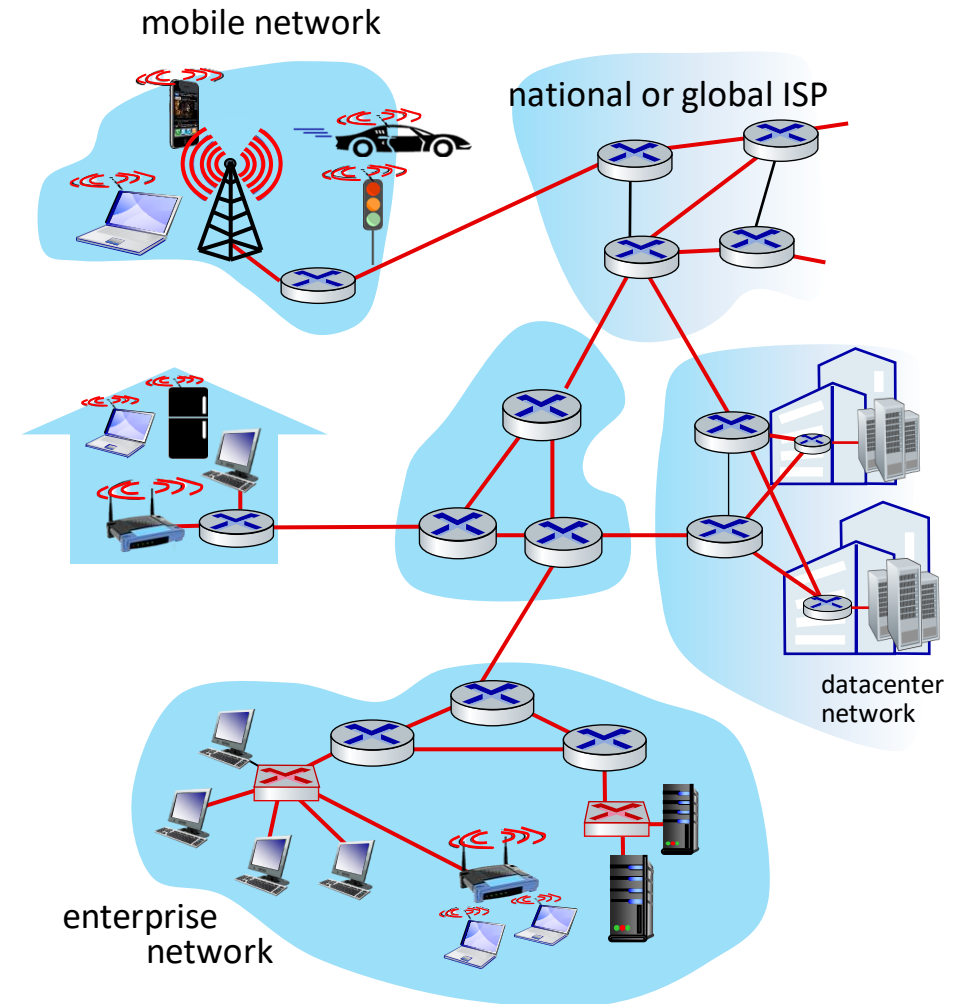
- a day in the life of a web request

Link layer: introduction

terminology:

- hosts and routers: nodes
- communication channels that **directly** connect **physically adjacent nodes** along communication path: **links**
 - wired, wireless
 - LANs
- layer-2 packet: **frame**, encapsulates datagram

*link layer has responsibility of transferring datagram from one node to **physically adjacent** node over a link*



Link layer: context

- datagram transferred by different link protocols over different links:
 - e.g., WiFi on first link, Ethernet on next link
- each link protocol provides different services
 - e.g., may or may not provide reliable data transfer over link

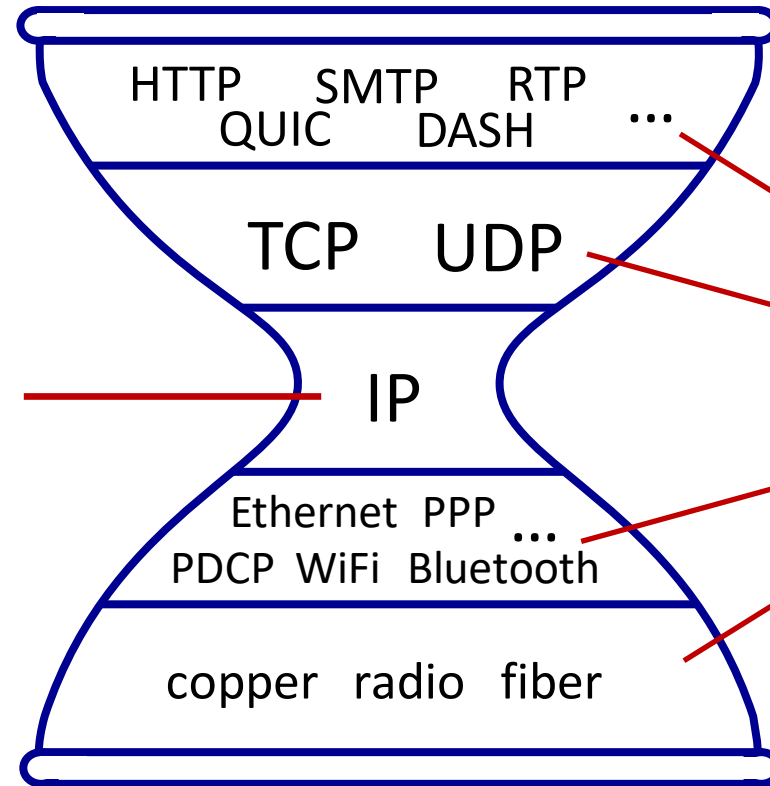
transportation analogy:

- trip from SUSTech to Tsinghua
 - metro: SUSTech to SZ North
 - High speed train: SZ North to Beijing West
 - taxi: Beijing West to Tsinghua
- tourist = **datagram**
- transport segment = **communication link**
- transportation mode = **link layer protocol**
- travel agent = **routing algorithm**

The IP hourglass

Internet's "thin waist":

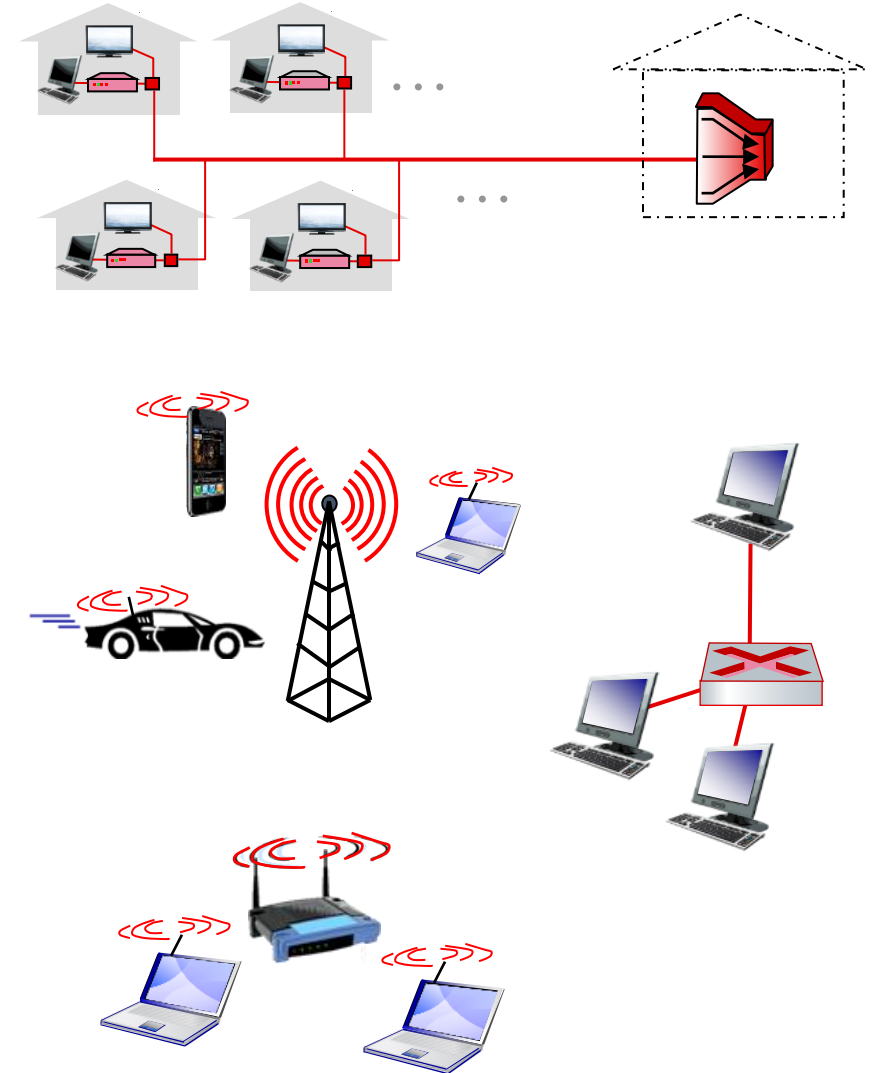
- *one* network layer protocol: IP
- *must* be implemented by every (billions) of Internet-connected devices



many protocols in physical, link, transport, and application layers

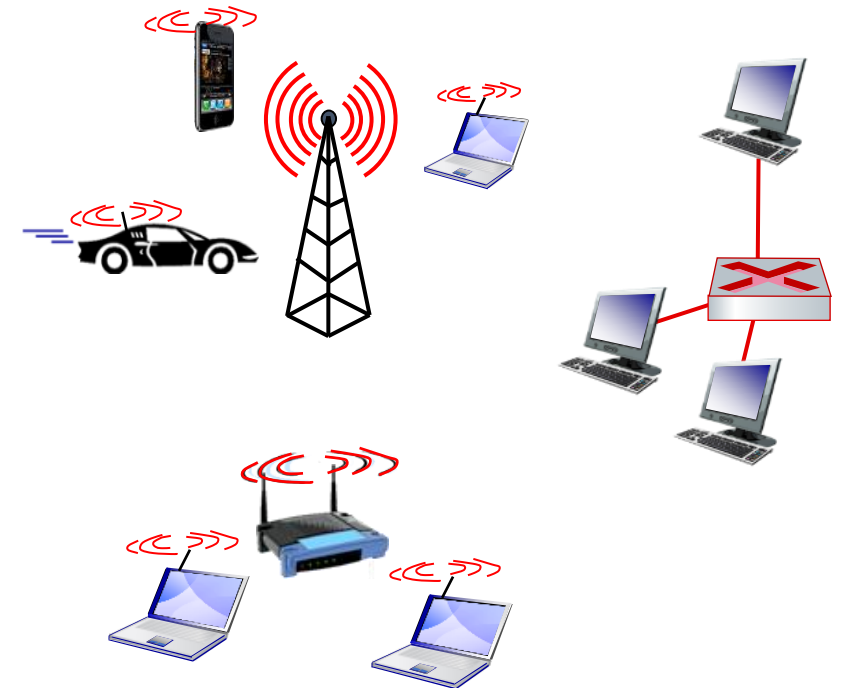
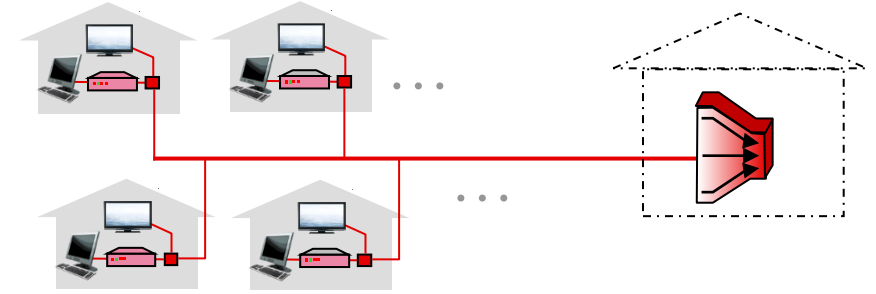
Link layer: services

- **framing, link access:**
 - encapsulate datagram into frame, adding header, trailer
 - channel access if shared medium
 - “MAC” addresses in frame headers identify source, destination (different from IP address!)
- **reliable delivery between adjacent nodes**
 - we already know how to do this!
 - seldom used on low bit-error links
 - wireless links: high error rates
 - Q: why both link-level and end-end reliability?

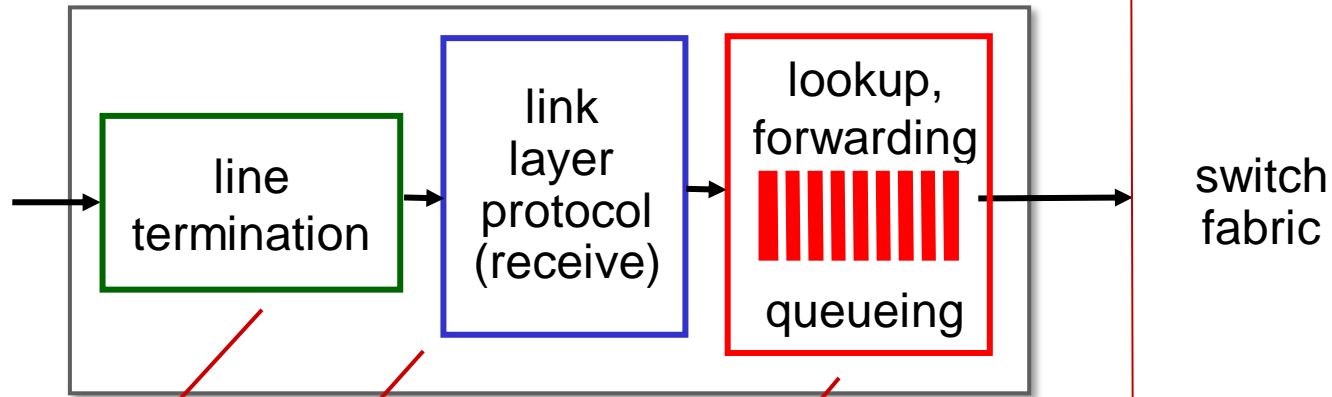


Link layer: services (more)

- **flow control:**
 - pacing between adjacent sending and receiving nodes
- **error detection:**
 - errors caused by signal attenuation, noise.
 - receiver detects errors, signals retransmission, or drops frame
- **error correction:**
 - receiver identifies *and corrects* bit error(s) without retransmission
- **half-duplex and full-duplex:**
 - with half duplex, nodes at both ends of link can transmit, but not at same time



Where is the link layer implemented?



physical layer:
bit-level reception

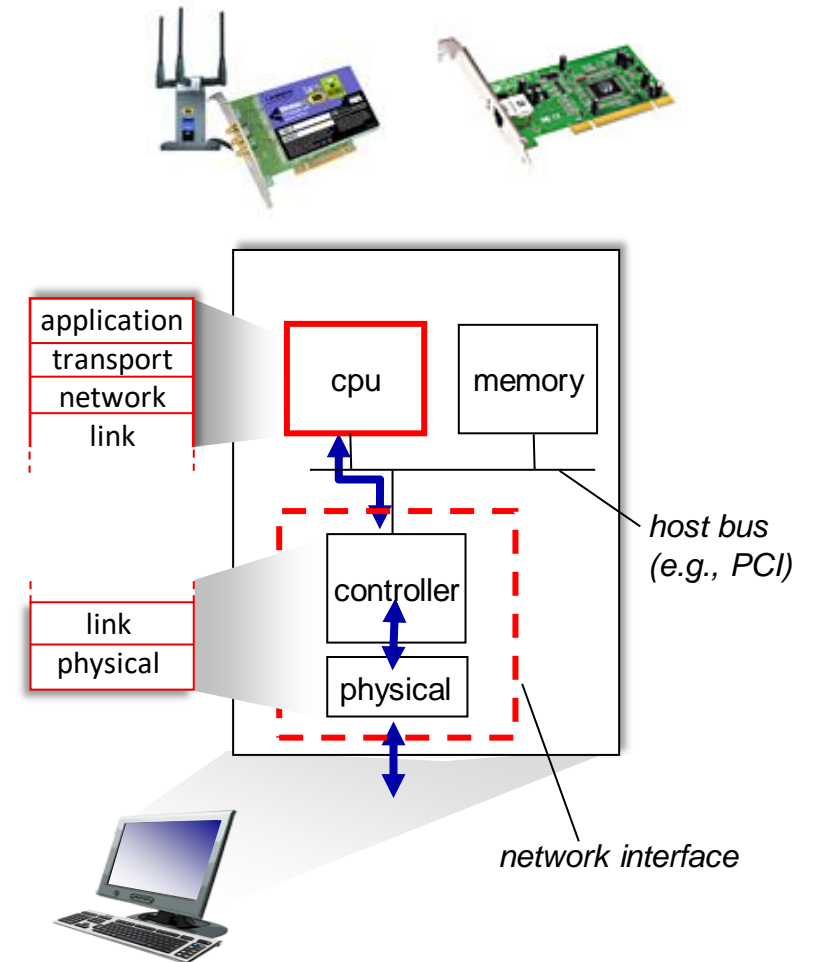
link layer:
e.g., Ethernet
(chapter 6)

decentralized switching:

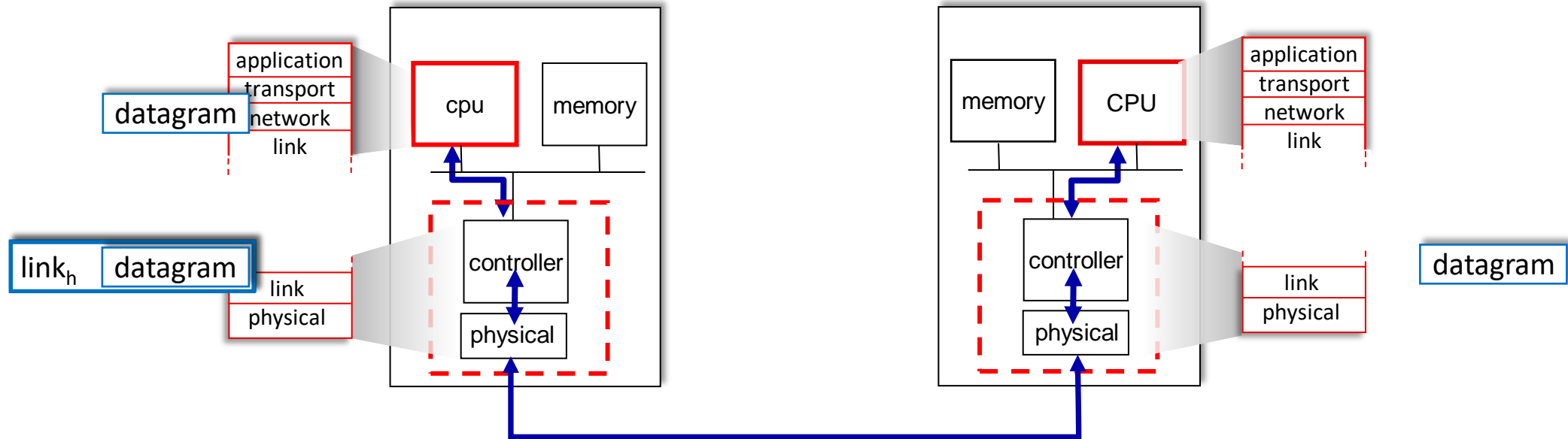
- using header field values, lookup output port using forwarding table in input port memory (*"match plus action"*)
- goal: complete input port processing at 'line speed'
- **input port queueing:** if datagrams arrive faster than forwarding rate into switch fabric

Where is the link layer implemented?

- in each-and-every host
- link layer implemented in *network interface card* (NIC) or on a chip
 - Ethernet, WiFi card or chip
 - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Interfaces communicating



sending side:

- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

- looks for errors, reliable data transfer, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

Link layer, LANs: roadmap

- introduction
- **error detection, correction**
- multiple access protocols
- LANs
 - addressing, ARP
 - Ethernet
 - switches
 - VLANs
- link virtualization: MPLS
- data center networking

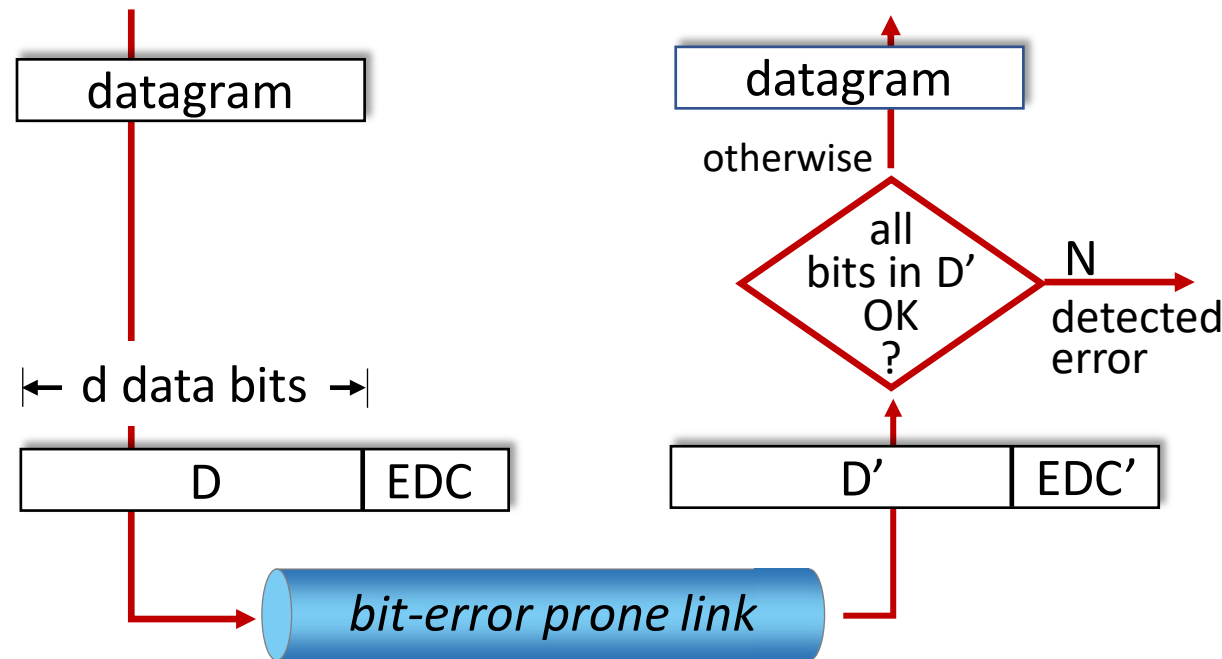


- a day in the life of a web request

Error detection

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



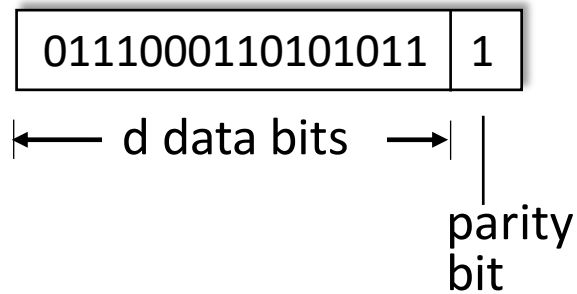
Error detection not 100% reliable!

- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

Parity checking

single bit parity:

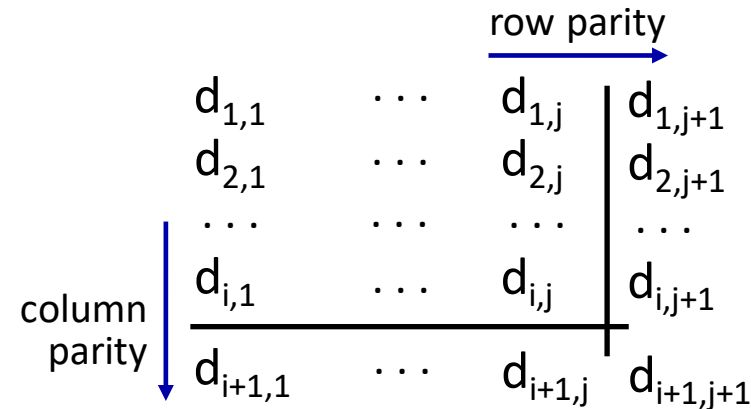
- detect single bit errors



Even parity: set parity bit so there is an even number of 1's

two-dimensional bit parity:

- detect *and correct* single bit errors



no errors:

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
1	0	1	0	1	0

detected and correctable single-bit error:

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
1	0	1	0	1	0

\downarrow parity error
 \rightarrow parity error

Internet checksum (review)

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment

sender:

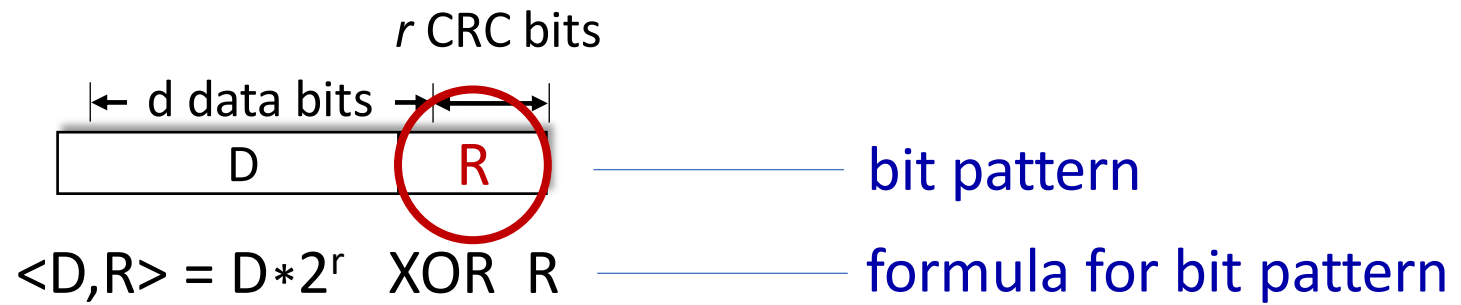
- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- **checksum:** addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
 - not equal - error detected
 - equal - no error detected. *But maybe errors nonetheless? More later*

Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- **D**: data bits (given, think of these as a binary number)
- **G**: bit pattern (generator), of $r+1$ bits (given)



goal: choose r CRC bits, **R**, such that $\langle D, R \rangle$ exactly divisible by $G \pmod{2}$

- receiver knows G , divides $\langle D, R \rangle$ by G . If non-zero remainder: error detected!
- can detect all burst errors less than $r+1$ bits
- widely used in practice (Ethernet, 802.11 WiFi)

Cyclic Redundancy Check (CRC): example

We want:

$$D \cdot 2^r \text{ XOR } R = nG$$

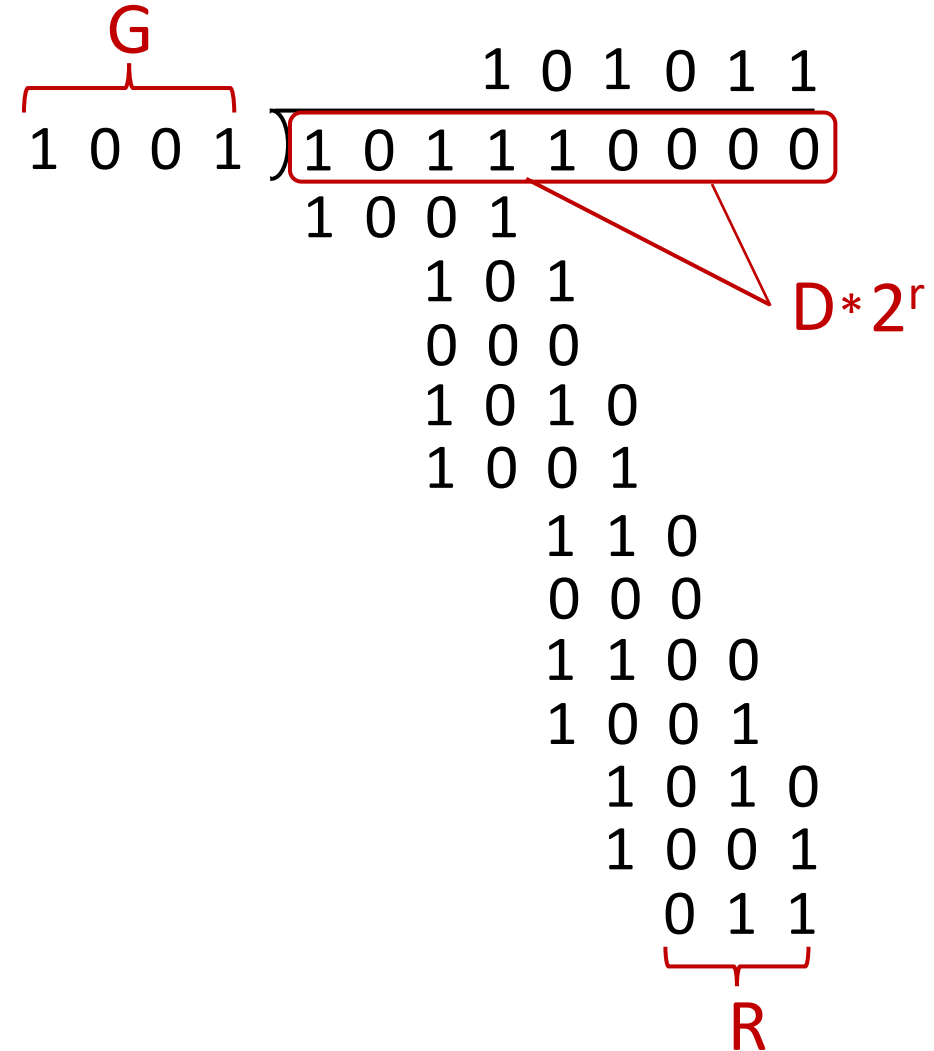
or equivalently:

$$D \cdot 2^r = nG \text{ XOR } R$$

or equivalently:

if we divide $D \cdot 2^r$ by G , want remainder R to satisfy:

$$R = \text{remainder} \left[\frac{D \cdot 2^r}{G} \right]$$



Link layer, LANs: roadmap

- introduction
- error detection, correction
- **multiple access protocols**
- LANs
 - addressing, ARP
 - Ethernet
 - switches
 - VLANs
- link virtualization: MPLS
- data center networking



- a day in the life of a web request

Multiple access links, protocols

two types of “links”:

- point-to-point
 - point-to-point link between Ethernet switch, host
 - PPP for dial-up access
- broadcast (shared wire or medium)
 - old-fashioned Ethernet
 - upstream HFC in cable-based access network
 - 802.11 wireless LAN, 4G/4G. satellite



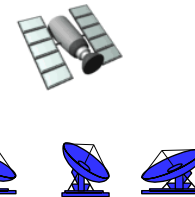
shared wire (e.g.,
cabled Ethernet)



shared radio: 4G/5G



shared radio: WiFi



shared radio: satellite

Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
 - *collision* if node receives two or more signals at the same time

multiple access protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
 - no out-of-band channel for coordination

An ideal multiple access protocol

given: multiple access channel (MAC) of rate R bps

desiderata:

1. when one node wants to transmit, it can send at rate R .
2. when M nodes want to transmit, each can send at average rate R/M
3. fully decentralized:
 - no special node to coordinate transmissions
 - no synchronization of clocks, slots
4. simple

MAC protocols: taxonomy

three broad classes:

- **channel partitioning**

- divide channel into smaller “pieces” (time slots, frequency, code)
- allocate piece to node for exclusive use

- ***random access***

- channel not divided, allow collisions
- “recover” from collisions

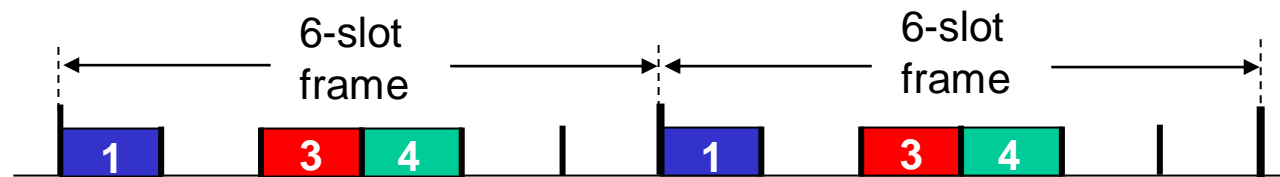
- **“taking turns”**

- nodes take turns, but nodes with more to send can take longer turns

Channel partitioning MAC protocols: TDMA

TDMA: time division multiple access

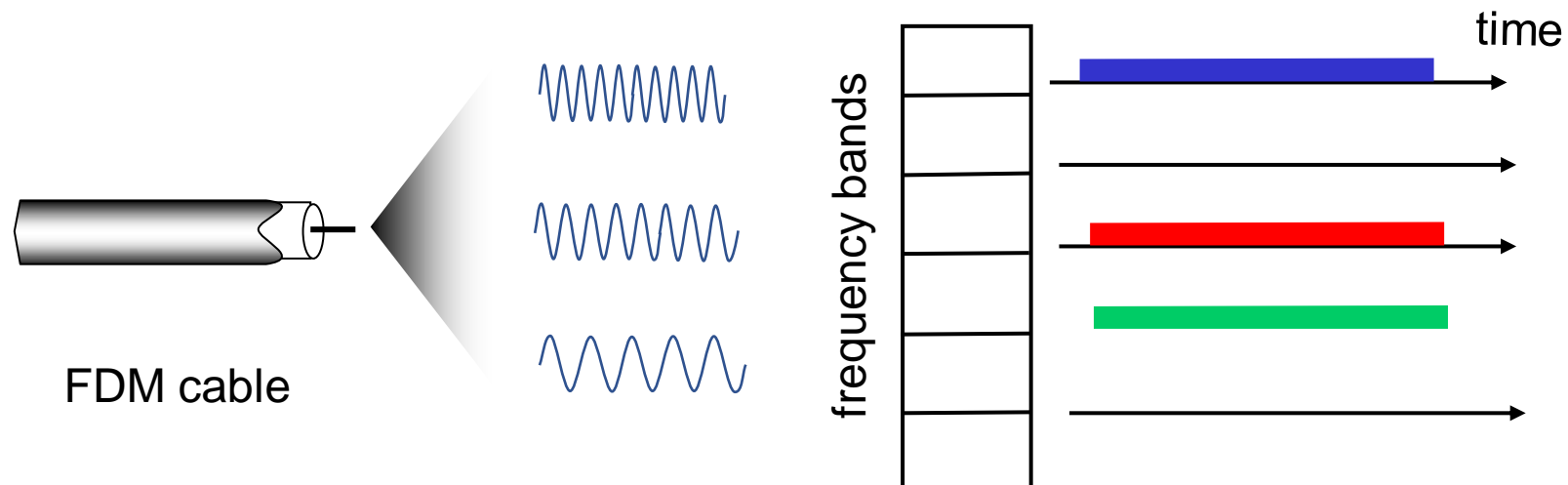
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random access protocols

- when node has packet to send
 - transmit at full channel data rate R .
 - no *a priori* coordination among nodes
- two or more transmitting nodes: “collision”
- random access MAC protocol specifies:
 - how to detect collisions
 - how to recover from collisions (e.g., via delayed retransmissions)
- examples of random access MAC protocols:
 - ALOHA, slotted ALOHA
 - CSMA, CSMA/CD, CSMA/CA

Slotted ALOHA

assumptions:

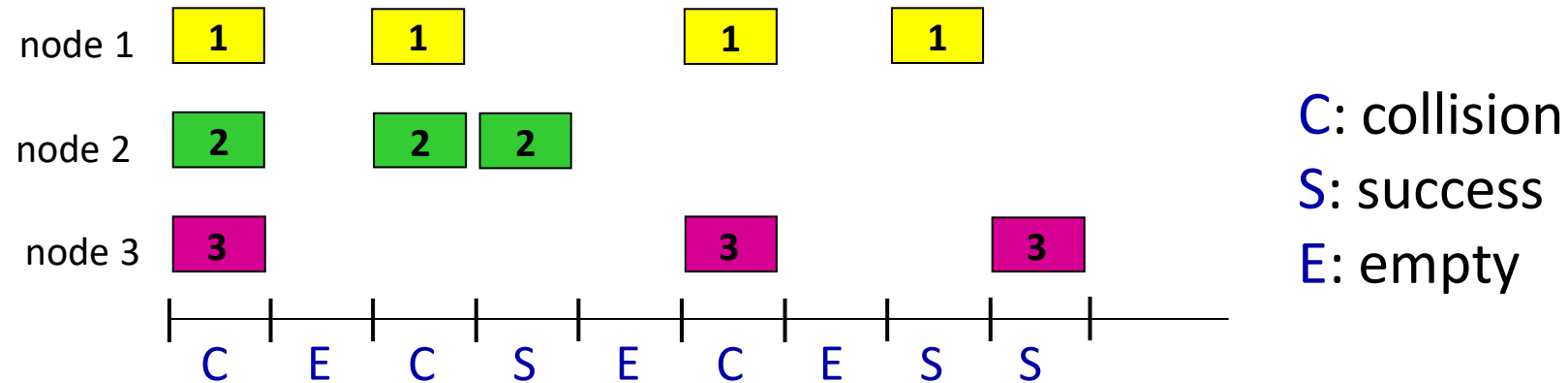
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

- when node obtains fresh frame, transmits in next slot
 - *if no collision*: node can send new frame in next slot
 - *if collision*: node retransmits frame in each subsequent slot with probability p until success

randomization – *why?*

Slotted ALOHA



Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:

- collisions, wasting slots
- idle slots
- clock synchronization

Slotted ALOHA: efficiency

efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)

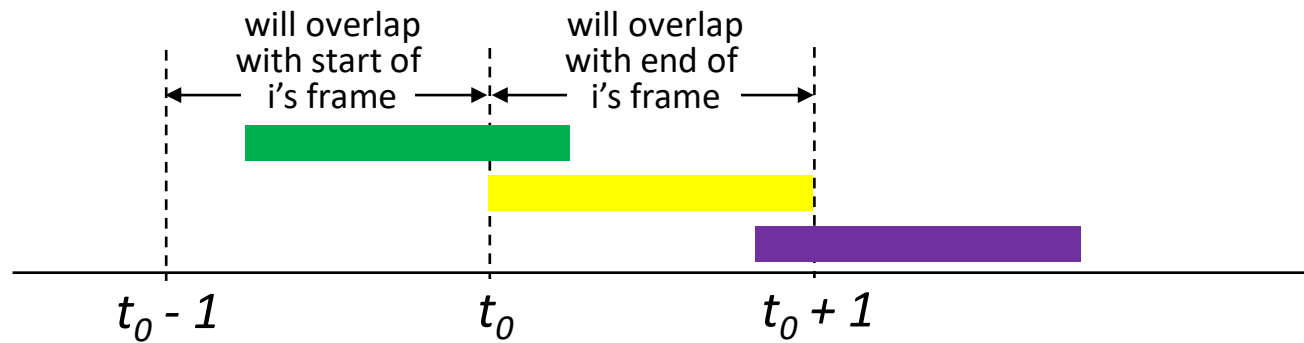
- *suppose:* N nodes with many frames to send, each transmits in slot with probability p
 - prob that given node has success in a slot $= p(1-p)^{N-1}$
 - prob that *any* node has a success $= Np(1-p)^{N-1}$
 - max efficiency: find p^* that maximizes $Np(1-p)^{N-1}$
 - for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives:

max efficiency = $1/e = .37$

- *at best:* channel used for useful transmissions 37% of time!

Pure ALOHA

- unslotted Aloha: simpler, no synchronization
 - when frame first arrives: transmit immediately
- collision probability increases with no synchronization:
 - frame sent at t_0 collides with other frames sent in $[t_0-1, t_0+1]$



- pure Aloha efficiency: 18% !

CSMA (carrier sense multiple access)

simple **CSMA**: listen before transmit:

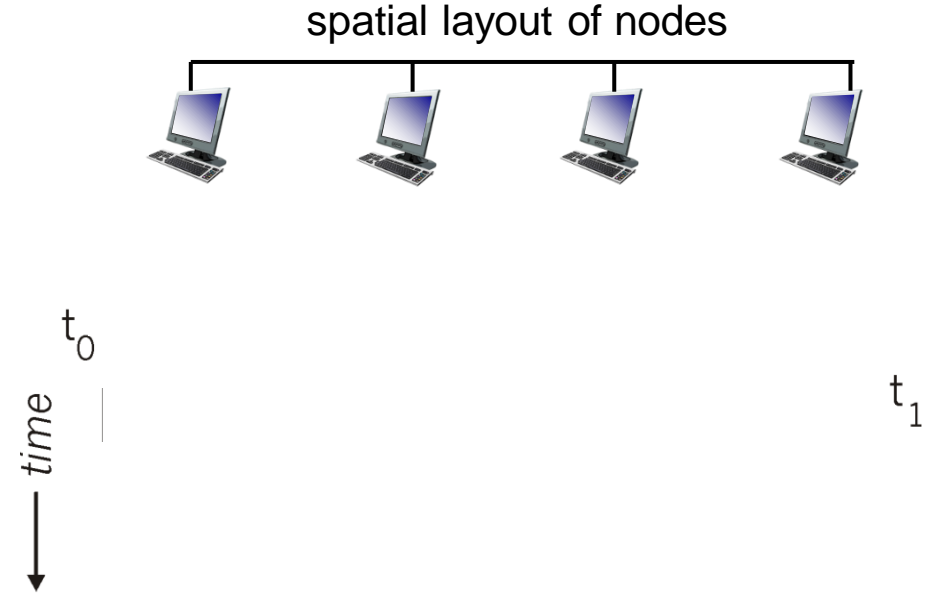
- if channel sensed idle: transmit entire frame
 - if channel sensed busy: defer transmission
- human analogy: don't interrupt others!

CSMA/CD: CSMA with *collision detection*

- collisions *detected* within short time
 - colliding transmissions aborted, reducing channel wastage
 - collision detection easy in wired, difficult with wireless
- human analogy: the polite conversationalist

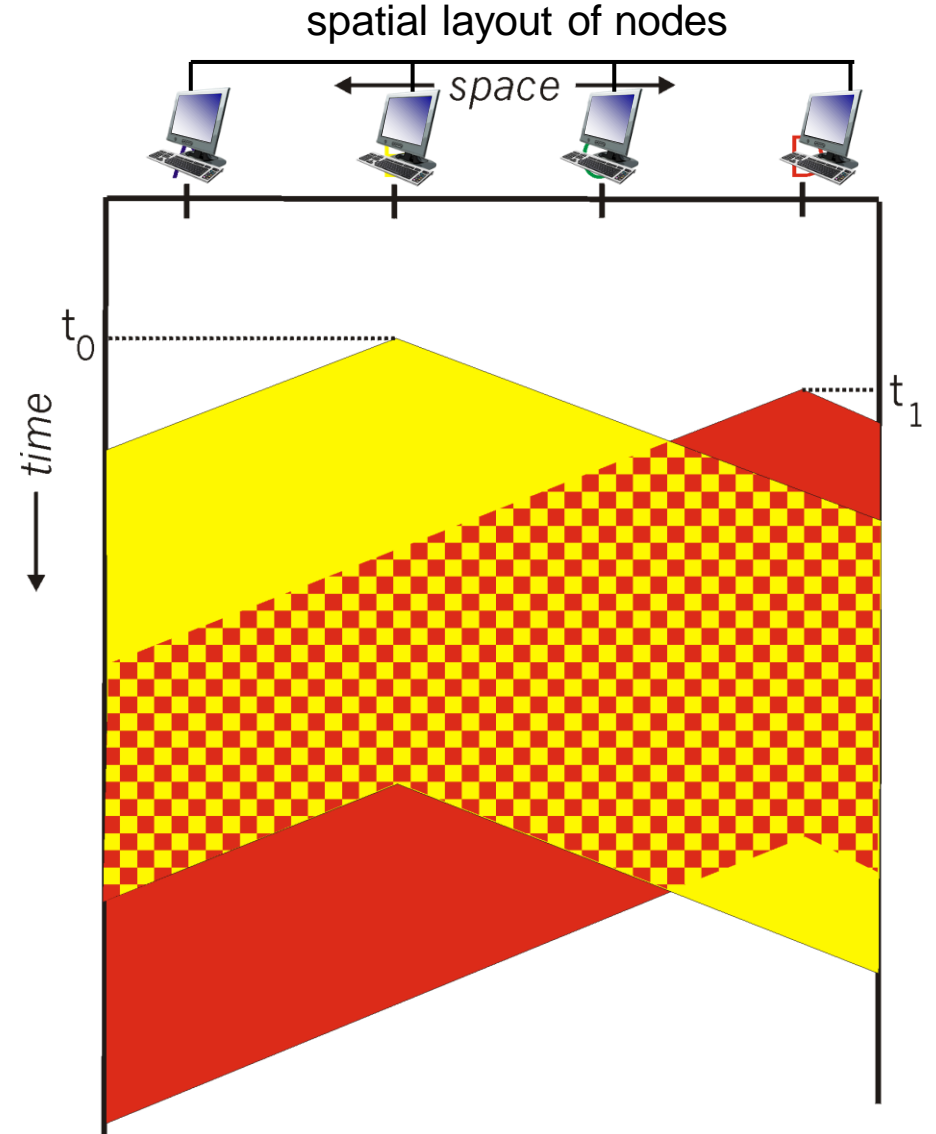
CSMA: collisions

- collisions *can* still occur with carrier sensing:
 - propagation delay means two nodes may not hear each other's just-started transmission
- **collision**: entire packet transmission time wasted
 - distance & propagation delay play role in determining collision probability



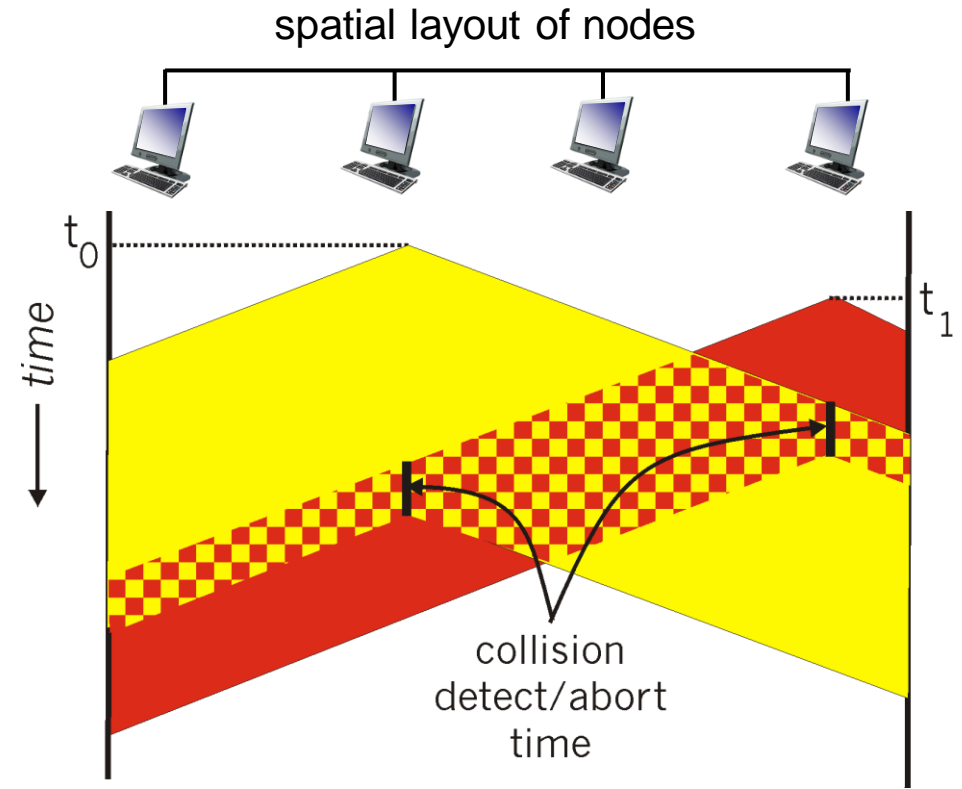
CSMA: collisions

- collisions *can* still occur with carrier sensing:
 - propagation delay means two nodes may not hear each other's just-started transmission
- **collision**: entire packet transmission time wasted
 - distance & propagation delay play role in determining collision probability



CSMA/CD:

- CSMA/CS reduces the amount of time wasted in collisions
 - transmission aborted on collision detection



Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel:
 - if **idle**: start frame transmission.
 - if **busy**: wait until channel idle, then transmit
3. If NIC transmits entire frame without collision, NIC is done with frame !
4. If NIC detects another transmission while sending: abort, send jam signal
5. After aborting, NIC enters *binary (exponential) backoff*:
 - after m th collision, NIC chooses K at random from $\{0, 1, 2, \dots, 2^m - 1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2
 - more collisions: longer backoff interval

CSMA/CD efficiency

- T_{prop} = max prop delay between 2 nodes in LAN
- t_{trans} = time to transmit max-size frame

$$\text{efficiency} = \frac{1}{1 + 5t_{\text{prop}}/t_{\text{trans}}}$$

- efficiency goes to 1
 - as t_{prop} goes to 0
 - as t_{trans} goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

“Taking turns” MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, $1/N$ bandwidth allocated even if only 1 active node!

random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

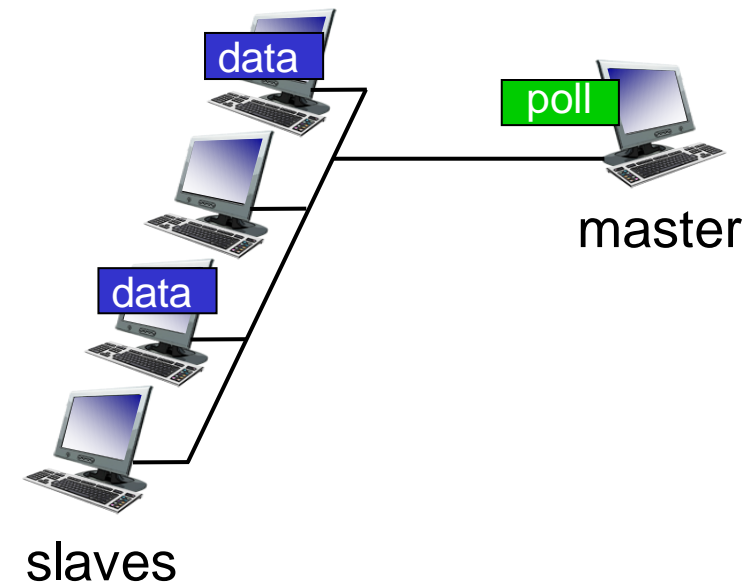
“taking turns” protocols

- look for best of both worlds!

“Taking turns” MAC protocols

polling:

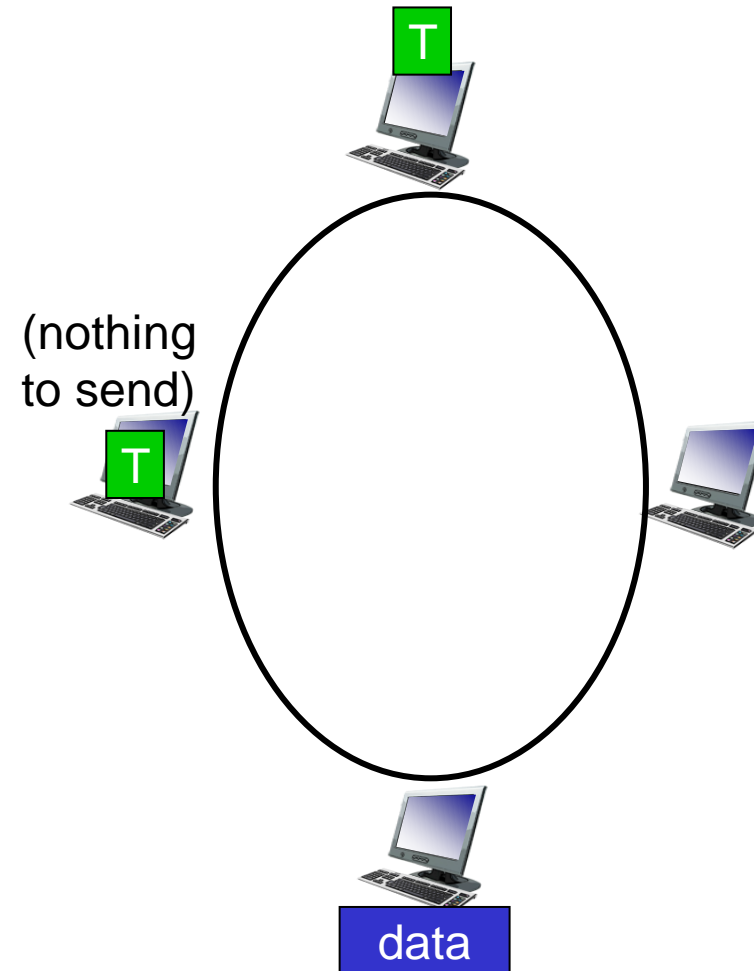
- master node “invites” other nodes to transmit in turn
- typically used with “dumb” devices
- concerns:
 - polling overhead
 - latency
 - single point of failure (master)
- Bluetooth uses polling



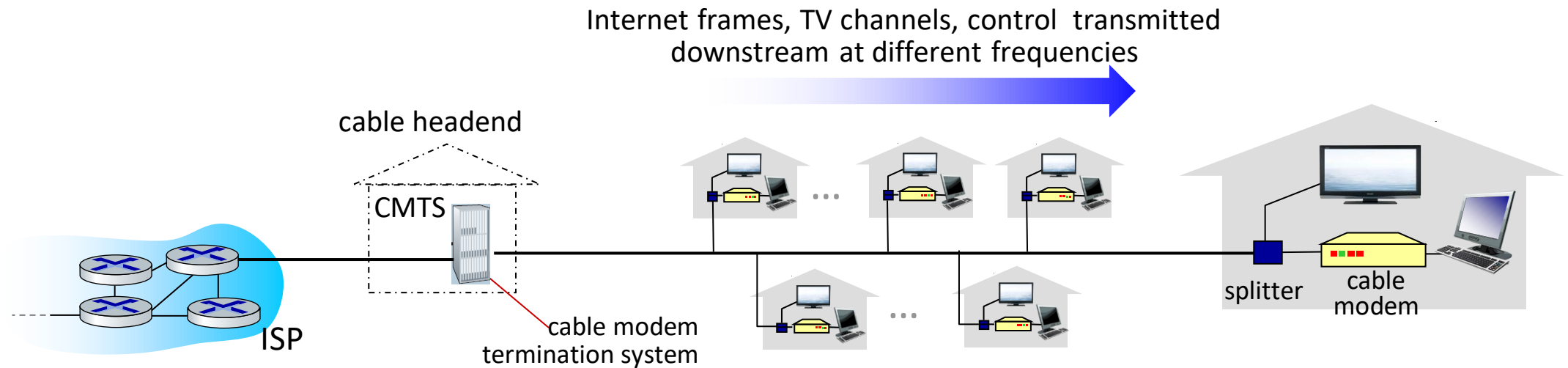
“Taking turns” MAC protocols

token passing:

- control *token* passed from one node to next sequentially.
- token message
- concerns:
 - token overhead
 - latency
 - single point of failure (token)

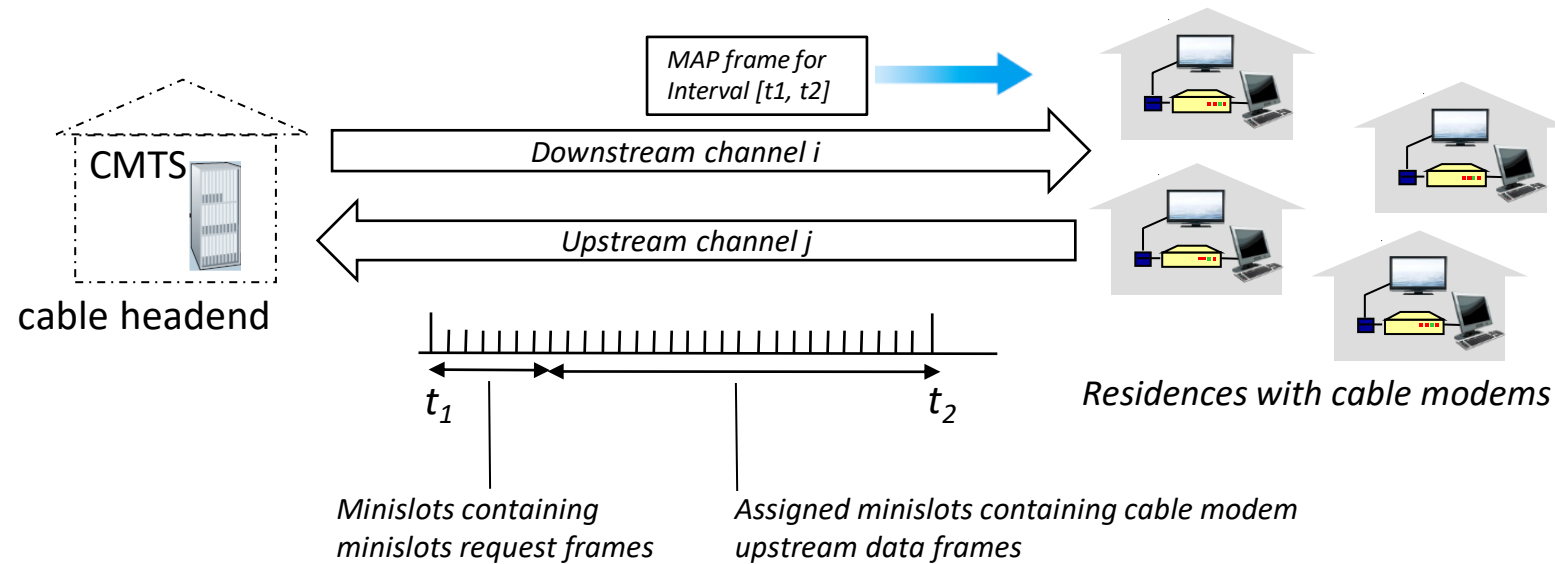


Cable access network: FDM, TDM *and* random access!



- **DOCSIS**: data over cable service interface specification
- **multiple** downstream (broadcast) FDM channels: up to 1.6 Gbps/channel
 - single CMTS transmits into channels
- **multiple** upstream channels (up to 1 Gbps/channel)
 - **multiple access**: all users contend (random access) for certain upstream channel time slots; others assigned TDM

Cable access network:



DOCSIS: data over cable service interface specification

- FDM over upstream, downstream frequency channels
- TDM upstream: some slots assigned, some have contention
 - downstream MAP frame: assigns upstream slots
 - request for upstream slots (and data) transmitted random access (binary backoff) in selected slots

Summary of MAC protocols

- **channel partitioning**, by time, frequency or code
 - Time Division, Frequency Division
- **random access** (dynamic),
 - ALOHA, S-ALOHA, CSMA, CSMA/CD
 - carrier sensing: easy in some technologies (wire), hard in others (wireless)
 - CSMA/CD used in Ethernet
 - CSMA/CA used in 802.11
- **taking turns**
 - polling from central site, token passing
 - Bluetooth, FDDI, token ring