Lecture 11

SDNs and Link Layer (Start)

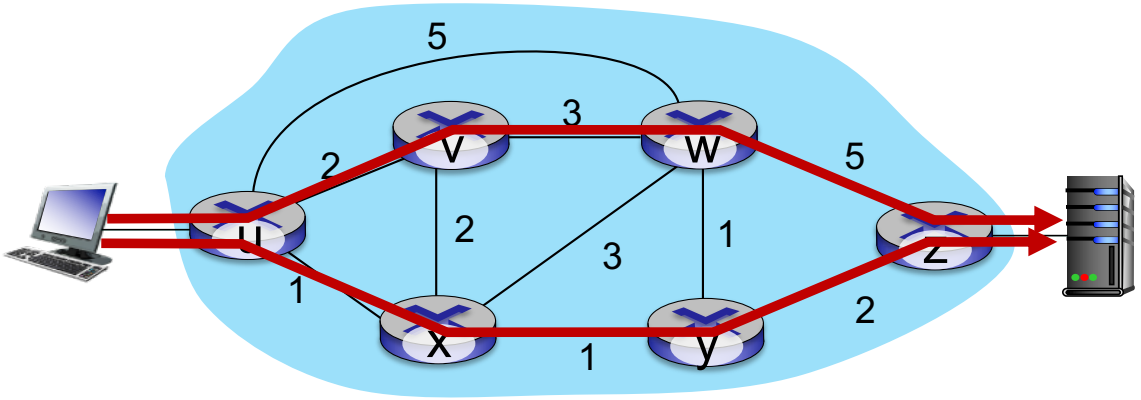
Software defined networking (SDN)

- Big picture: making our networks **programmable**
  - At the data plane level
  - At the control plane level

- Why?

Software defined networking (SDN)

Traffic engineering: difficult with traditional routing

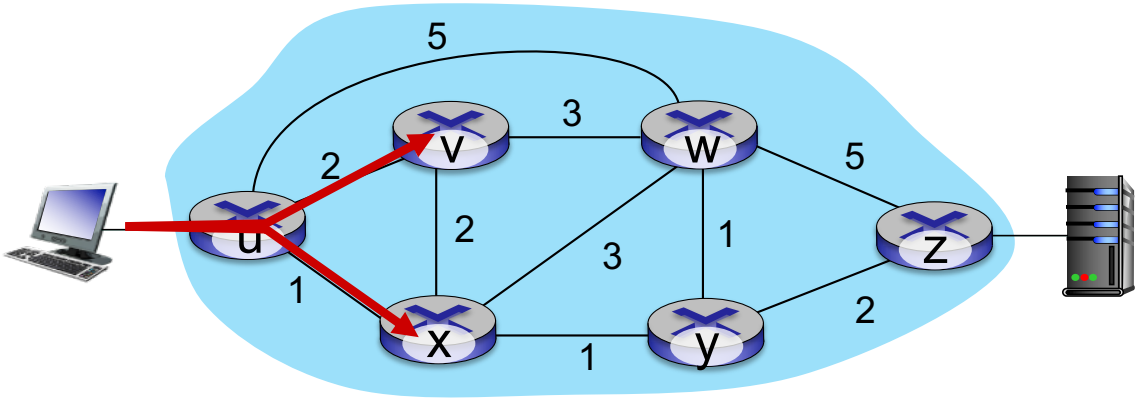


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

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!

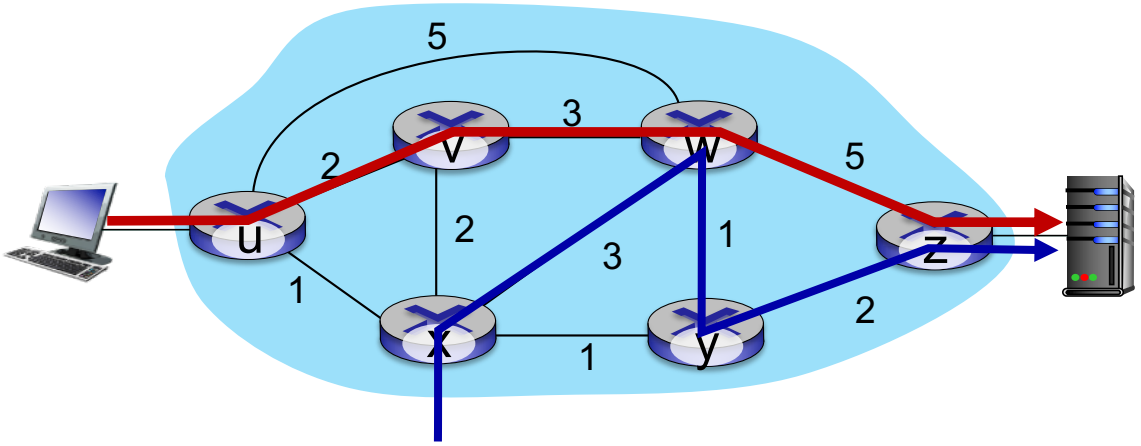
Traffic engineering: difficult with traditional routing



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



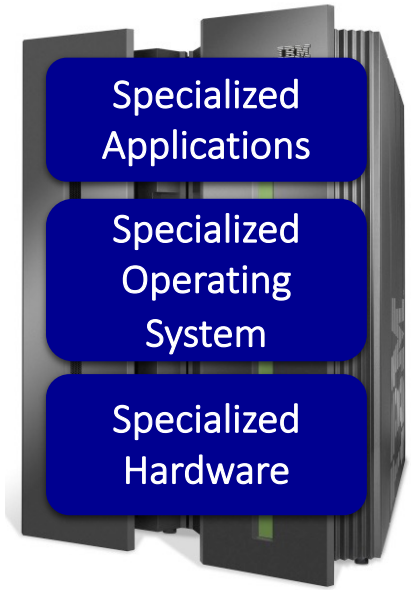
Q: what if w wants to route blue and red traffic differently from w to z?

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

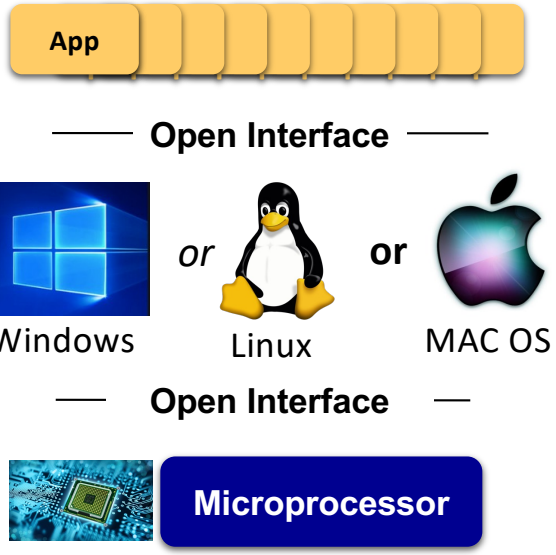
Path to Software defined networking (SDN)

- 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
- ~2013: renewed interest in rethinking network data plane

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

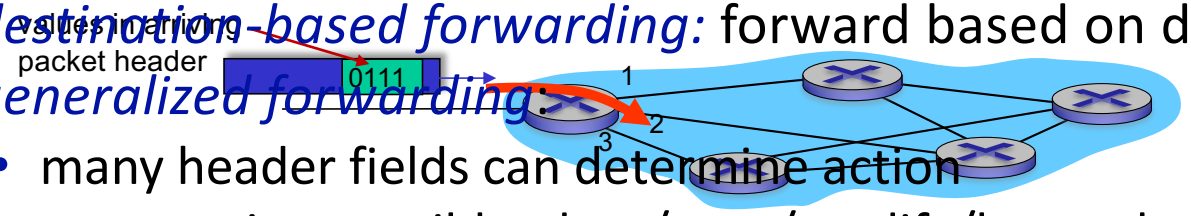
SDN: Some key ideas

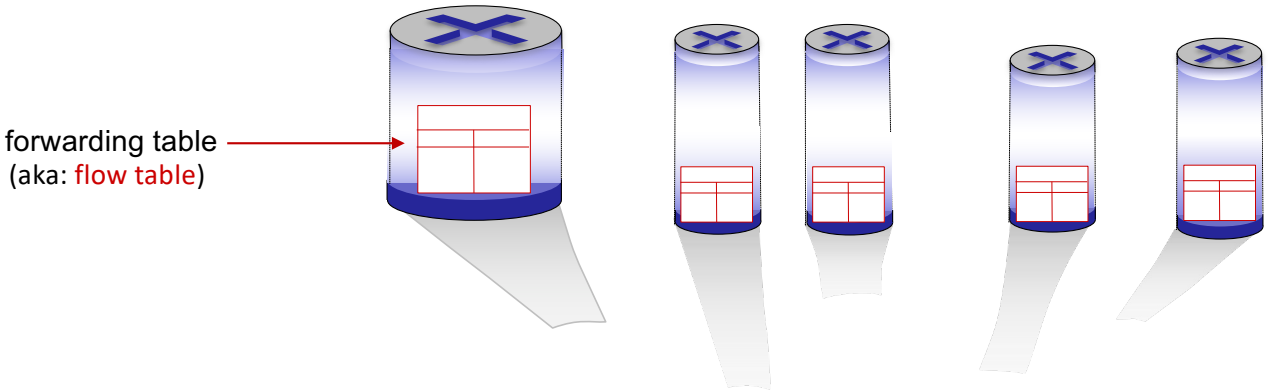
- Data plane: **Generalized Forwarding**
- Control plane: **Logically centralized** control plane (with **open-source**, software controller)

Generalized forwarding: match plus action

Review: each router contains a forwarding table (aka: flow table)

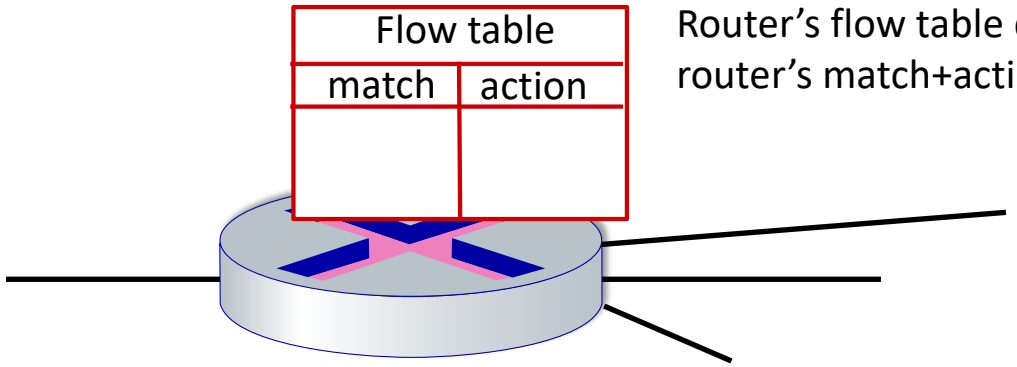
- "match plus action" abstraction: match bits in arriving packet, take action
  - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding:*
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet





Flow table abstraction

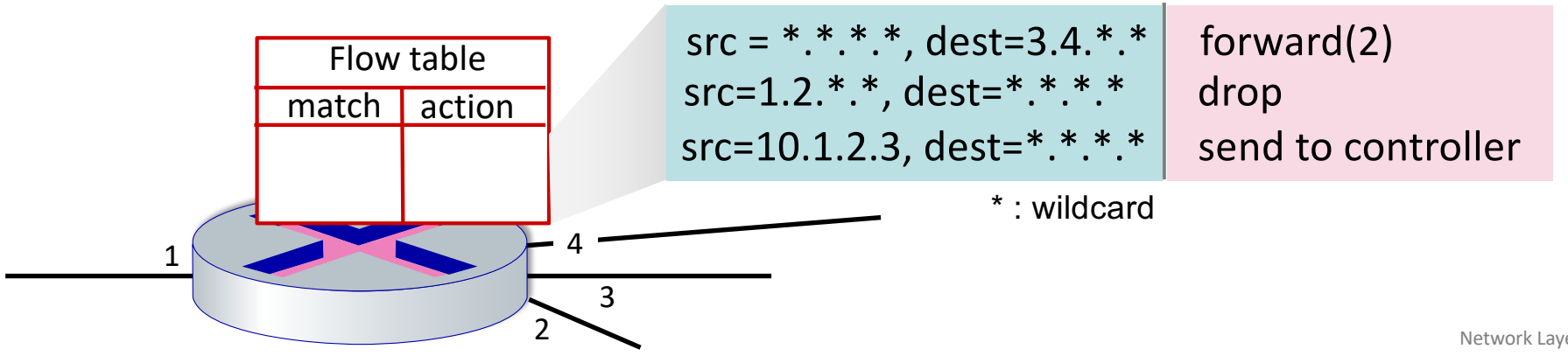
- flow: defined by header field values (in link-, network-, transport-layer fields)
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  -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



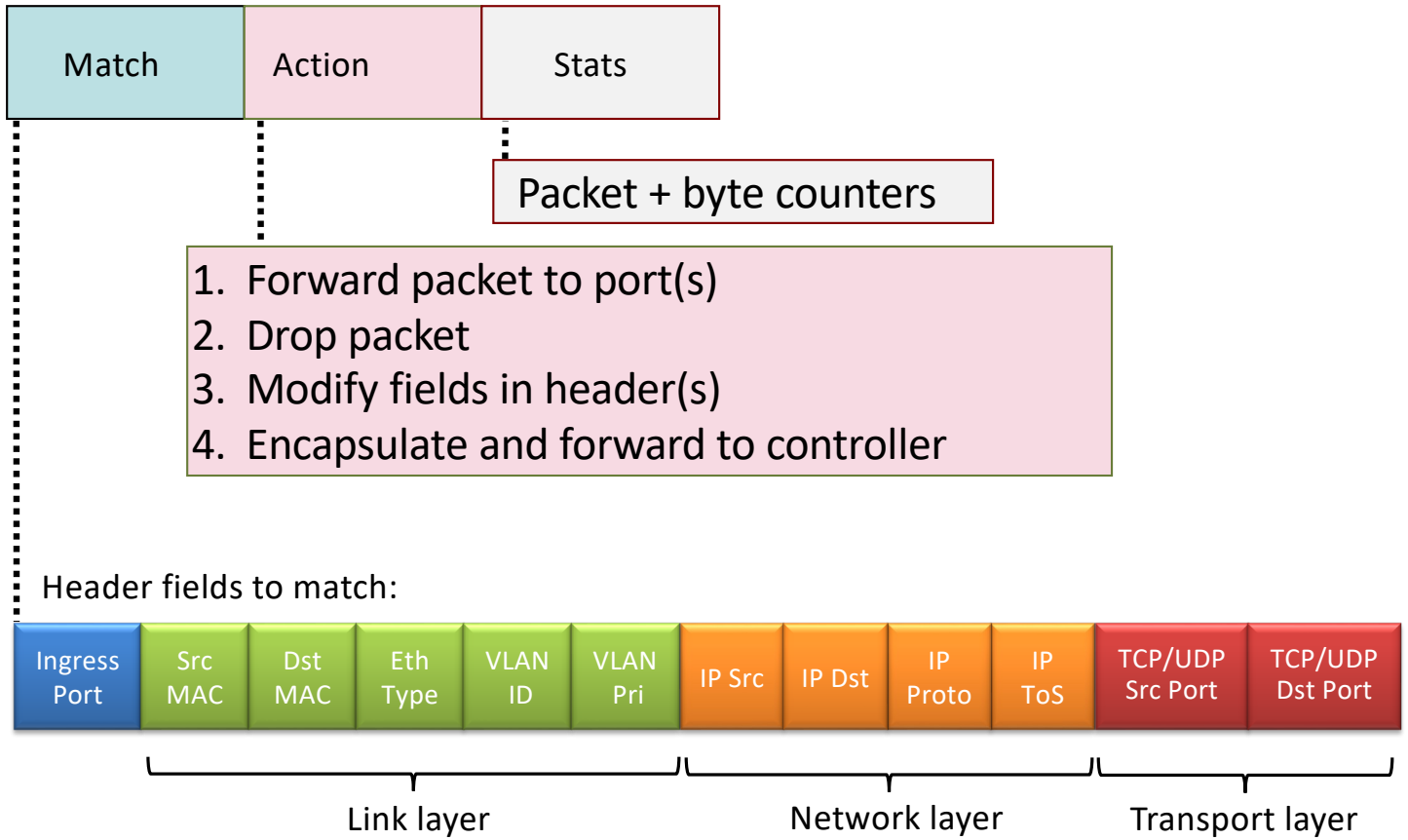
Router's flow table define router's match+action rules

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 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

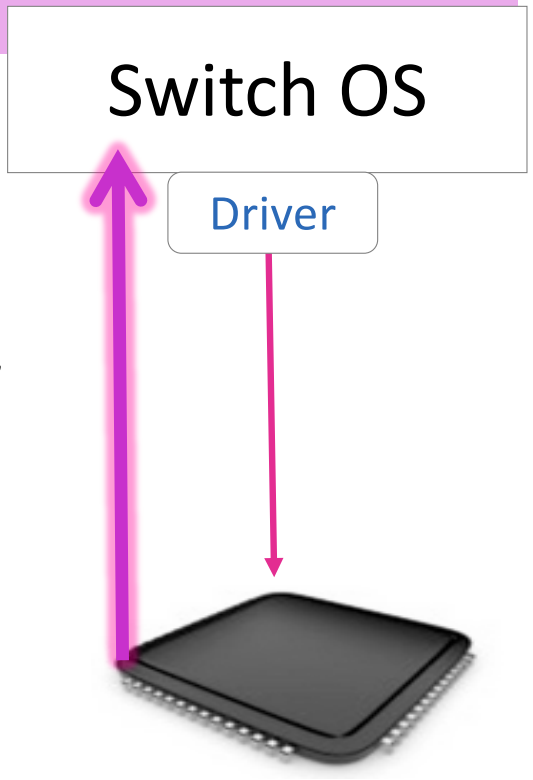
Generalized forwarding: summary

- "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"
  - *historical roots:* active networking
  - *today:* more generalized programming: P4 (see p4.org).

Toward General Data Plane Programming: P4

- Open Flow is still limited to headers that the switch *knows how to process*

"This is how I process packets …"



Fixed-function switch

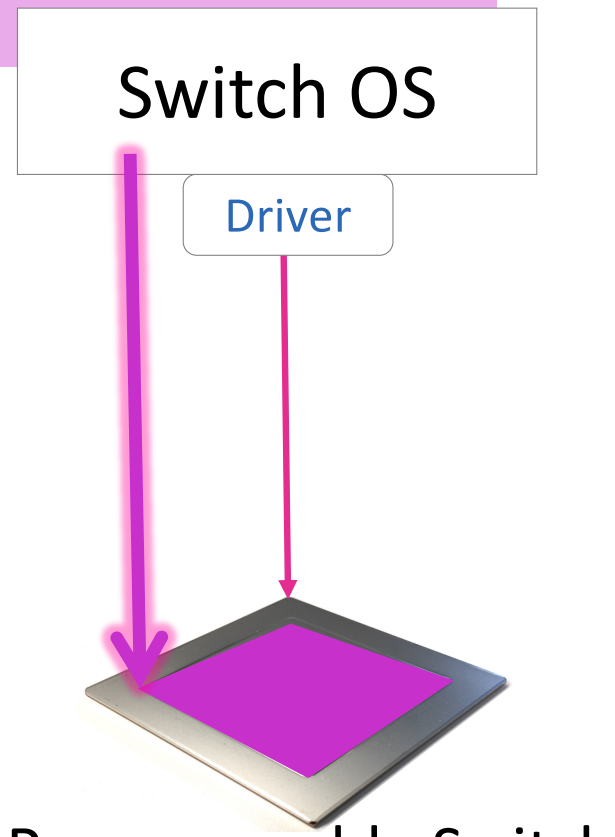
Toward General Data Plane Programming: P4

"This is precisely how you must process packets"

```
table int_table {
  reads {
    ip.protocol;
  }
  actions {
    export_queue_latency;
  }
}
```

```
action export_queue_latency (sw_id) {
  add_header(int_header);
  modify_field(int_header.kind, TCP_OPTION_INT);
  modify_field(int_header.len, TCP_OPTION_INT_LEN);
  modify_field(int_header.sw_id, sw_id);
  modify_field(int_header.q_latency,
               intrinsic_metadata.deq_timedelta);
  add_to_field(tcp.dataOffset, 2);
  add_to_field(ipv4.totalLen, 8);
  subtract_from_field(ingress_metadata.tcpLength,
                      12);
}
```

- Fast *programmable* switch chips are now available
  - PISA: Protocol Independent Switch Architecture
- P4 is a programming language for specifying header formats, match rules, and actions
  - Can compile to any PISA chip

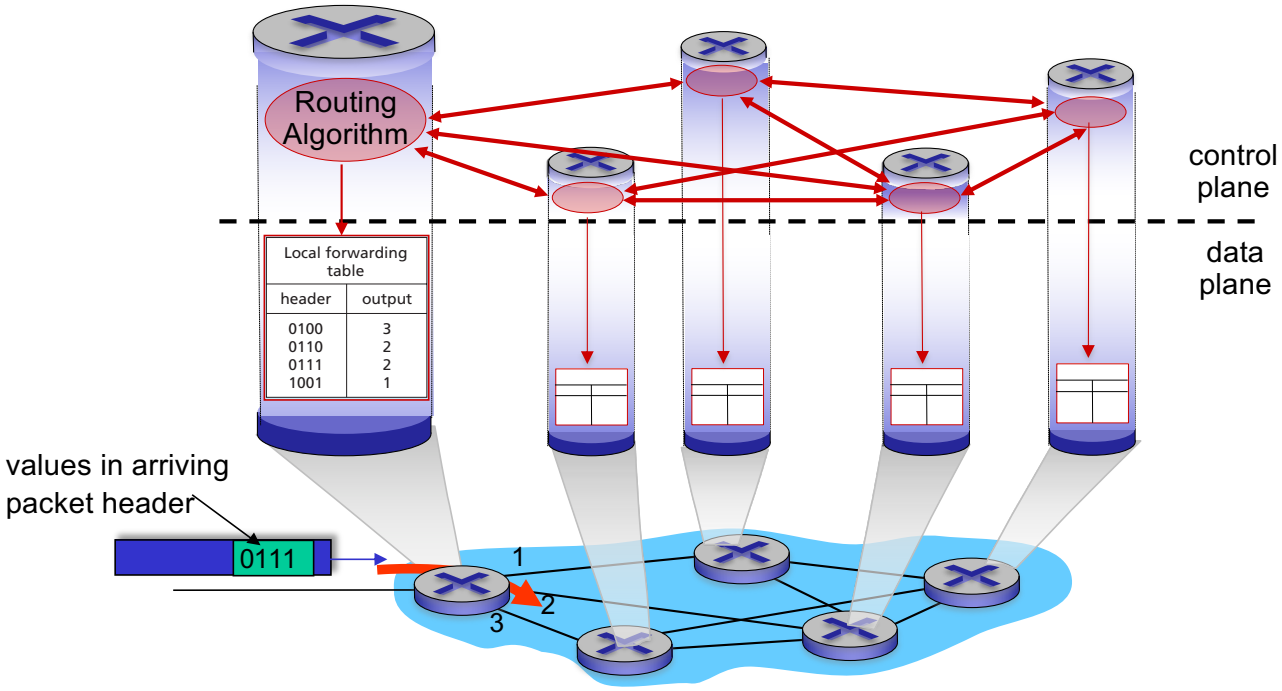


SDN: Some key ideas

- Data plane: Generalized Forwarding
- **Control plane: Logically centralized control plane (with open-source, software controller)**

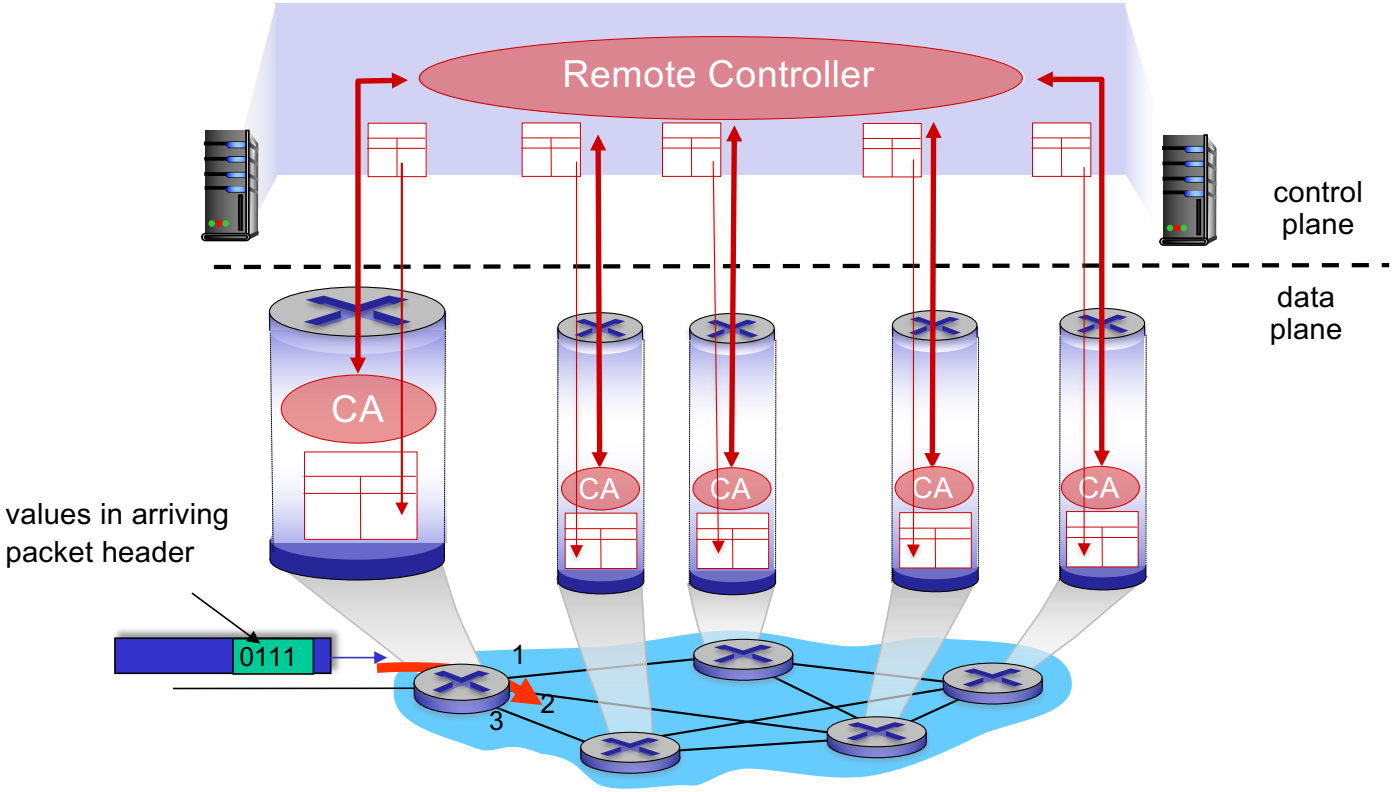
Traditional Approach: Per-router control plane

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



Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers

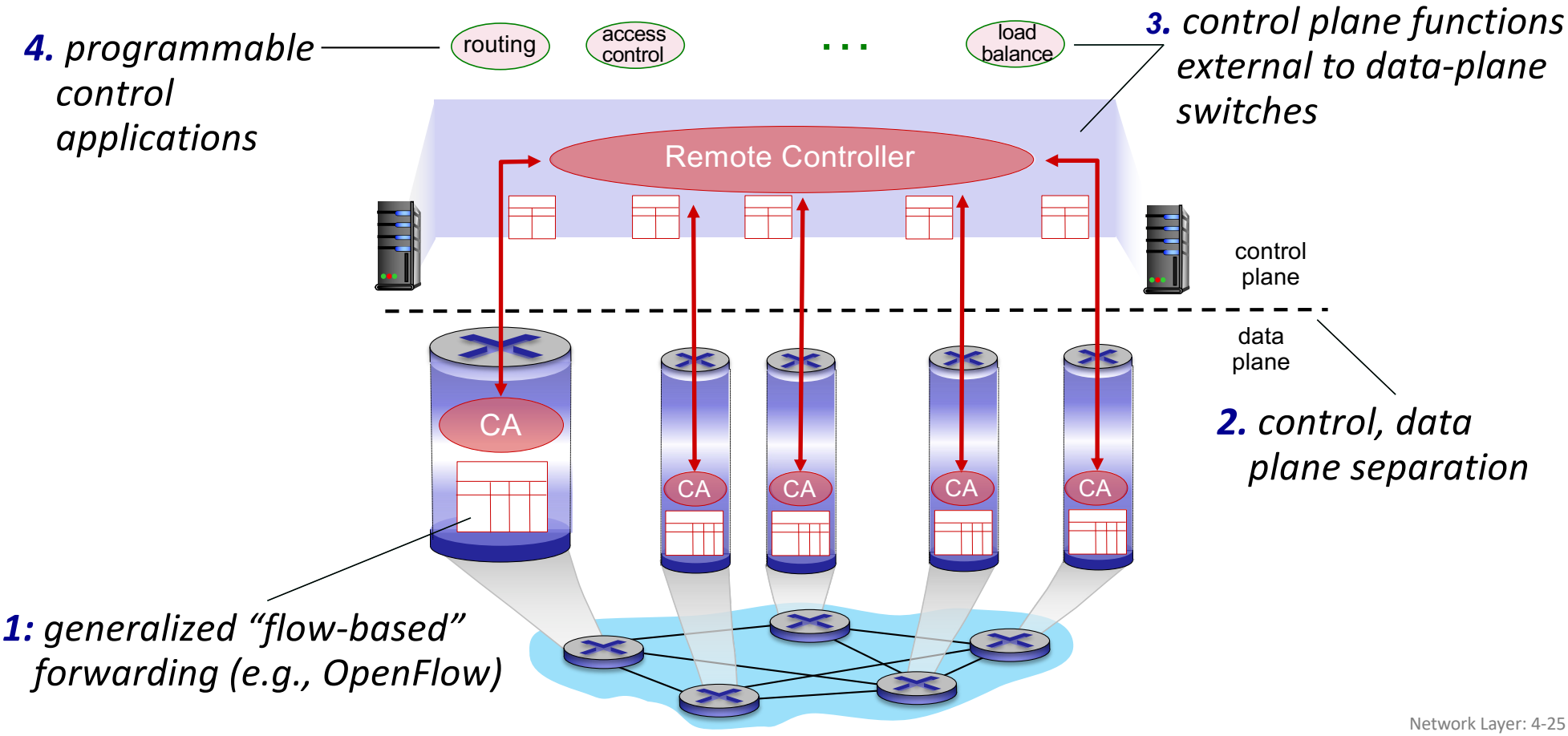


Software defined networking (SDN)

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

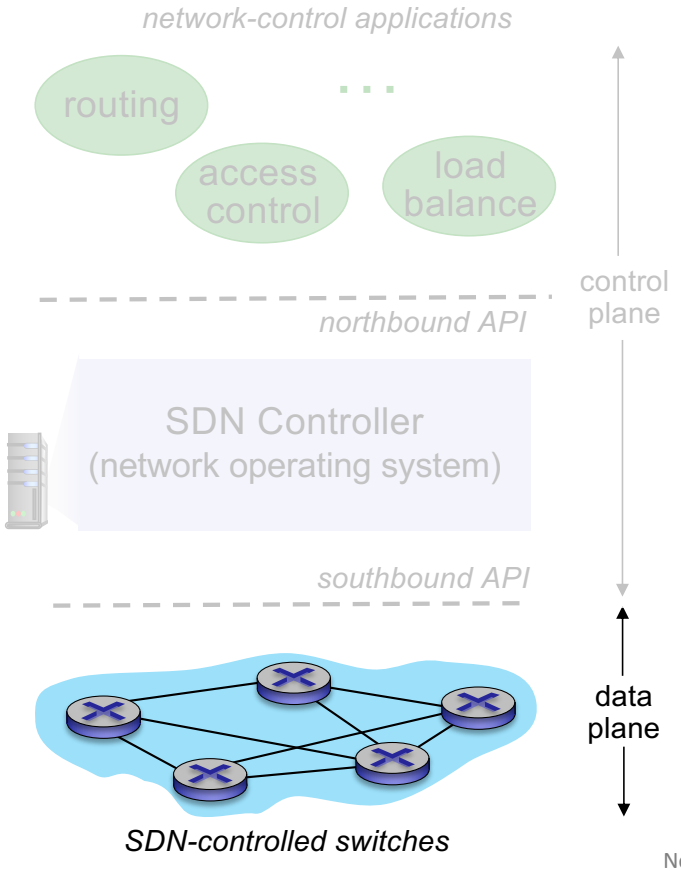
Software defined networking (SDN)



Software defined networking (SDN)

Data-plane switches:

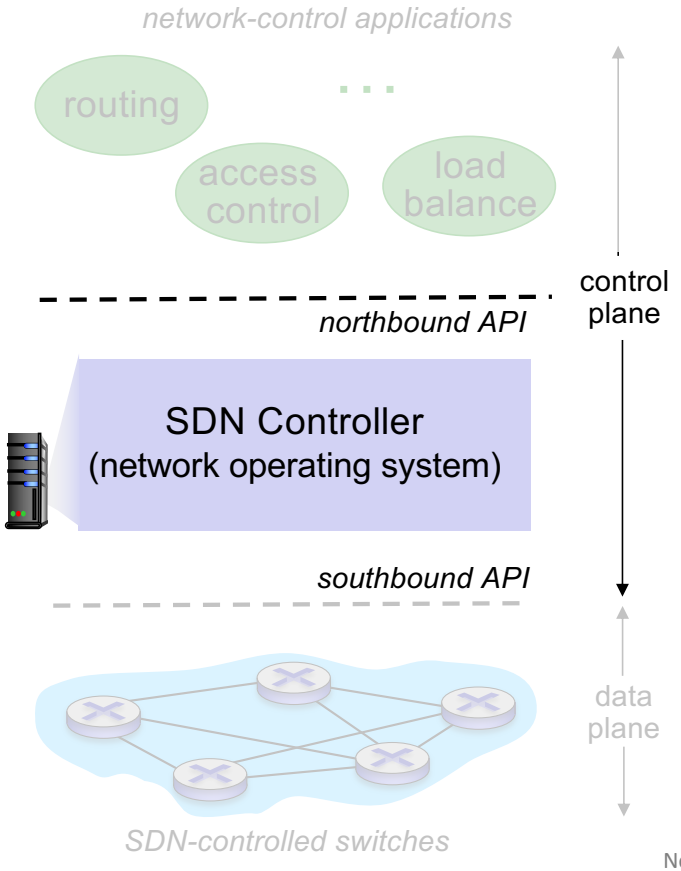
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with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



Software defined networking (SDN)

SDN controller (network OS):

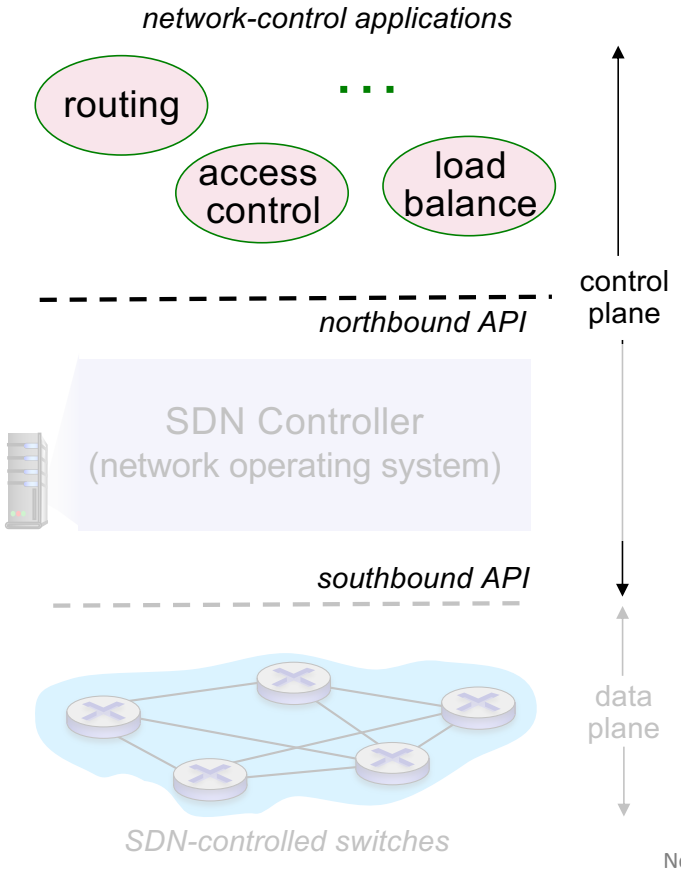
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



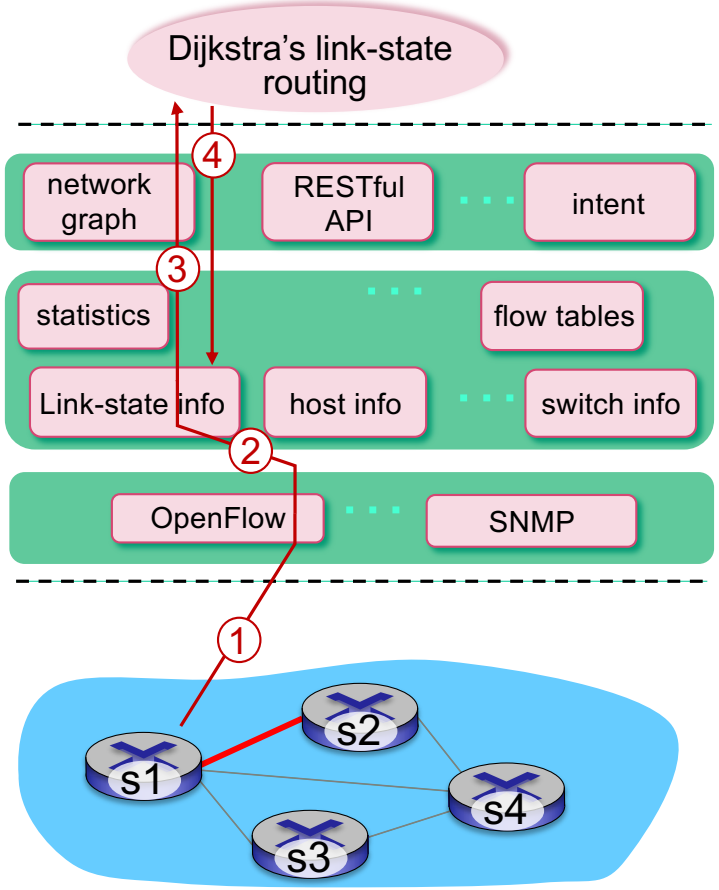
Software defined networking (SDN)

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



SDN: control/data plane interaction example



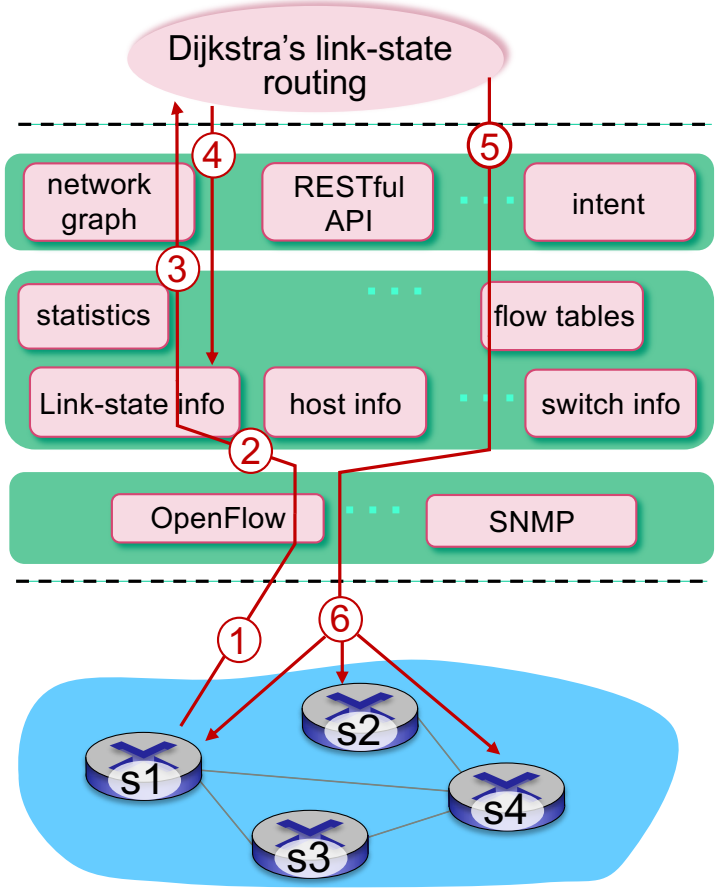
① S1, experiencing link failure uses OpenFlow port status message to notify controller

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

SDN: selected challenges

- 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: beyond a single AS
- Wireless world: SDN and 5G cellular networks

Reviewing the Network Layer

- Data Plane: Forwarding
- Control Plane: Routing

Data Plane: Forwarding

- **Routers** map **incoming packets** to **output ports**
  - Look up destination address in **forwarding table** and select output port based on **longest prefix match**

- Router **switching fabric** moves packet from input port to output port
  - Lookup + switching needs to be fast! (processing delay, limits supported throughput)

- **Queuing delay** occurs when packets for a given output port arrive faster than they can be sent out
  - **Buffer management** and **packet scheduling** policies determine which specific packets are dropped/sent when queuing occurs

Data Plane: IP addresses

- Forwarding is traditionally based on **destination IP address**
- IPv4 address is a 32-bit number
- IP addresses have **structure** that reflects network organization
  - Organized into **subnets**
  - Each address has **subnet part** and **host part**
  - Consider a subnet 223.1.1.0/24
    - high-order 24 bits represent the subnet and will be the same for all hosts in that subnet: 223.1.1.
    - remaining 32-24 = 8 bits identify individual host interfaces: IP addresses can range from 223.1.1.0 to 223.1.1.255
- IP address blocks are allocated to ISPs by ICANN (or registrars they delegate to) and then subdivided and allocated to customers by the ISP

Data Plane: IP addresses (continued)

- Individual hosts in a network can have statically configured IP addresses, or can get them dynamically when they join through **DHCP**
  - DHCP messages are **broadcasted**, because the joining host doesn't know where the DHCP server is yet, and doesn't have its own address for the DHCP server to send to yet
- IPv4 address space exhaustion: 32 bits is no longer enough!
  - Network Address Translation (NAT)
    - All hosts in a local network share one public IPv4 address
    - Use private IP addresses internally
    - Router modifies packets to use single public IP address, and some new source port that it records in translation table. Uses translation table to correctly map incoming response to correct host
  - IPv6
    - Increases address size to 128 bits (additionally simplifies header for faster processing, adds "flow" concept)
    - Adoption *still* in progress: **tunneling** is used in the meantime (IPv6 packets encapsulated in IPv4 packets to traverse routers that don't support IPv6 yet)

Control Plane: Routing

- Abstractly, look at network as a graph. Goal is to find "best" path from source node to all destinations (to populate its forwarding table)
  - Usually "best" = least-cost
- Link state routing
  - Each node has **full view** of the network topology and link costs
  - Runs **Dijkstra's algorithm** to compute least-cost paths to every other node
  - When a node's link costs change, it **broadcasts update** to all nodes so they can recompute
- Distance vector routing
  - **Bellman-Ford equation**: My distance to node X is the minimum over all my neighbors of the neighbor's distance to X + my distance to that neighbor
  - Each node maintains a **vector with its (estimated) distance/cost** from itself to each other node
  - **Neighboring nodes exchange vectors** (and store each others vectors) and update distance estimates based on received vectors
  - When its link costs change, a node **only sends update to its neighbors** (who will recompute and potentially send their own update)

Control Plane: Internet Routing

- **Hierarchy** is used to make routing scalable, and allow operators to control their own networks (network-of-networks)**: organized into Autonomous Systems**
- Intra-AS Routing
  - Within a single AS (different ASes can use different protocols)
  - **OSPF** is commonly used
    - Classic **Link-State** protocol
    - Some advances: security, load balancing, hierarchy for scalability
- Inter-AS Routing
  - Across different ASes (connected by **gateway routers**)
  - **BGP** is the de facto protocol
    - Distance-vector-like, but some differences
    - Advertises **path**, not just distance (**loop avoidance**)
    - **Policy**, not distance/performance is primary consideration: generally, providers only want to **carry traffic to/from their customers**

The Link Layer

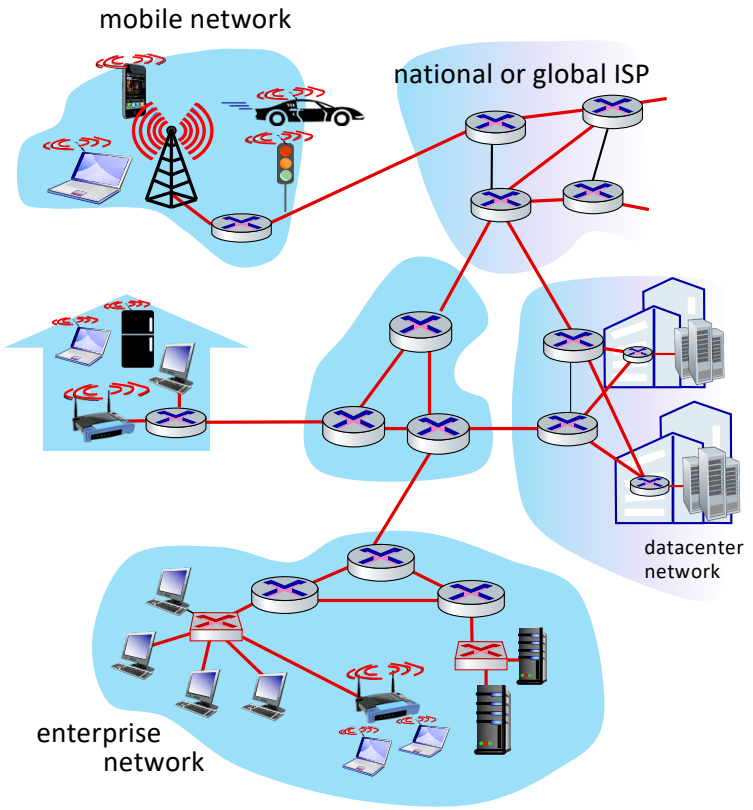
Local Connectivity

Link layer: introduction

terminology:

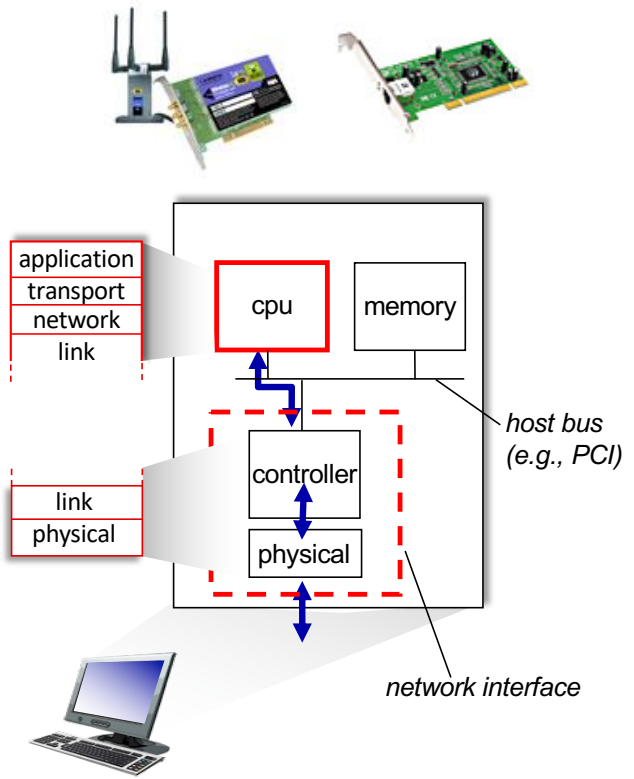
- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired
  - wireless
  - LANs
- layer-2 packet: *frame*, encapsulates datagram

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

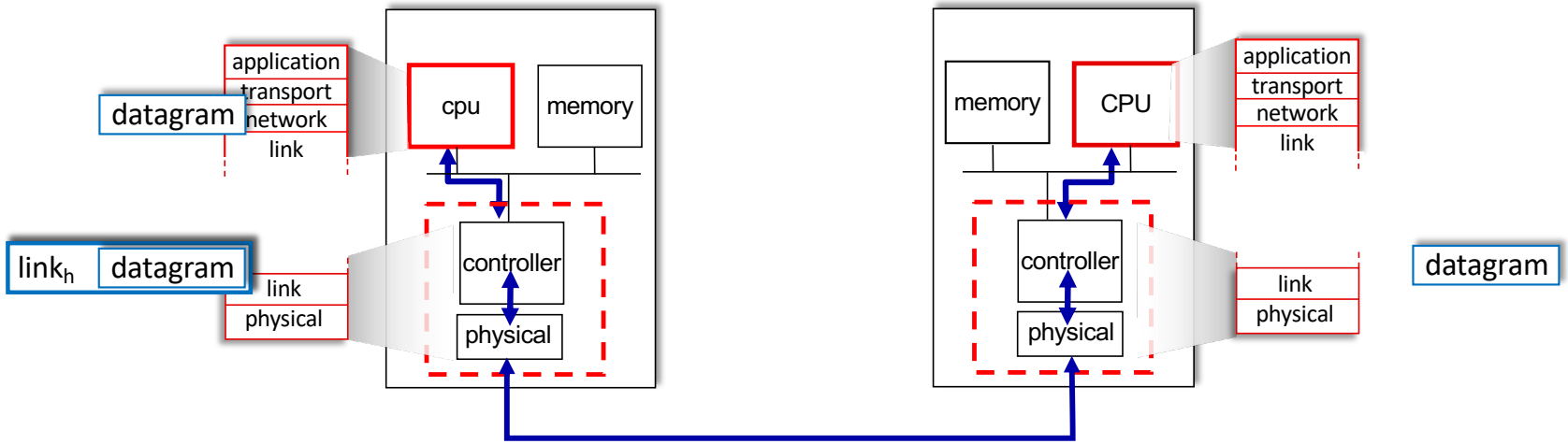


Where is the link layer implemented?

- in 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:

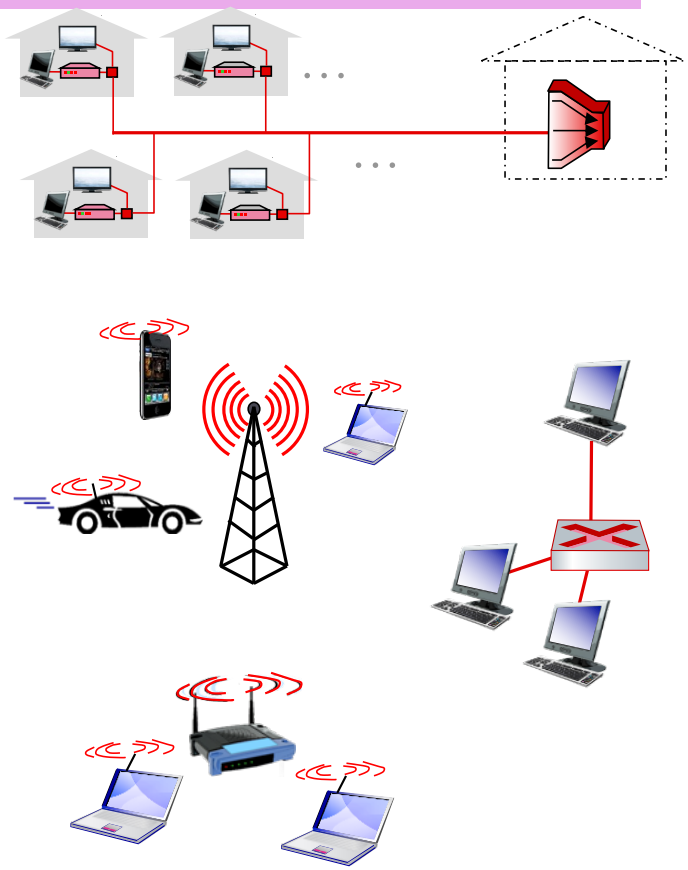
- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

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

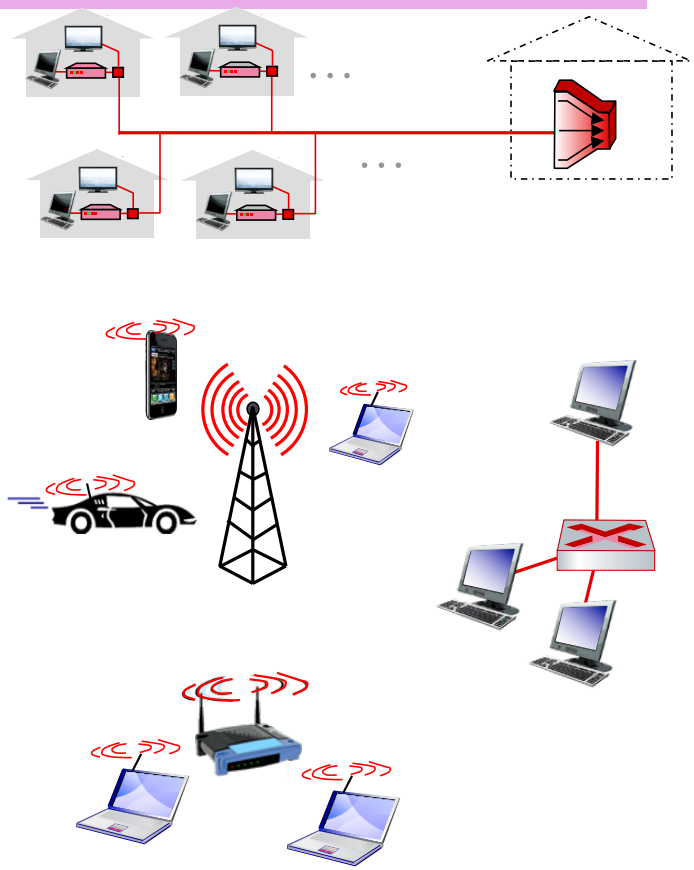
Link layer: services

- **Services** offered by different link layer protocols can vary
- framing
  - encapsulate datagram into frame, adding header
- error detection *and correction*:
  - errors caused by signal attenuation, noise.
  - receiver detects errors, signals retransmission, or drops frame
  - may be able to correct error without retransmissions
- link access
  - coordinate medium access if shared broadcast link



Link layer: services (more)

- reliable delivery between adjacent nodes
  - seldom used on low bit-error links
  - wireless links: high error rates
    - *Q:* why both link-level and end-end reliability?
- flow control:
  - pacing between adjacent sending and receiving nodes
- half-duplex vs full-duplex:
  - with half duplex, nodes at both ends of link can transmit, but not at same time

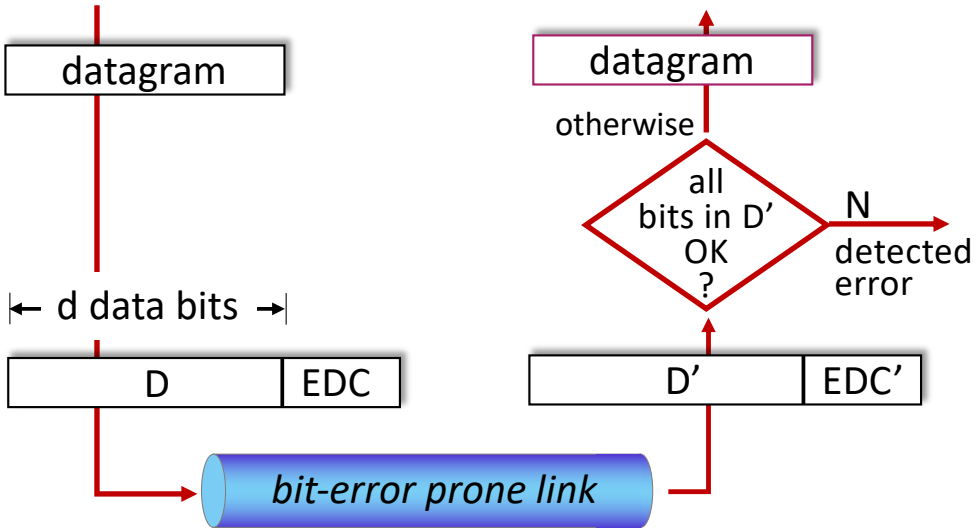


Link Layer Services: Error Detection and Correction

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!

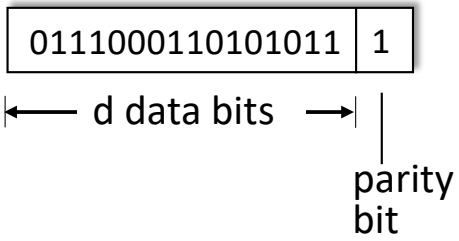
- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

Error detection method 1: Parity checking

what happens if there are multiple bit errors??

single bit parity:

- detect **single bit** errors

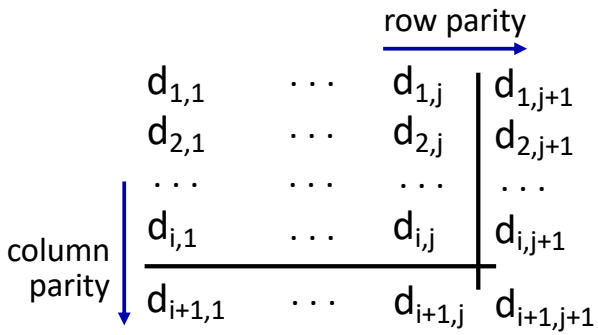


Even parity: set parity bit so there is an even number of 1's

If receiver sees an *odd* number of 1s, it knows a bit error has occurred

two-dimensional bit parity:

- detect *and correct* single bit errors

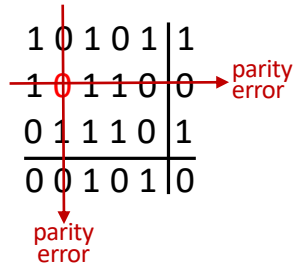


Additionally *detects* (but can't correct) any combination of 2 errors

no errors:

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

detected and correctable single-bit error:



Error detection method 2: Checksumming (review)

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

sender:

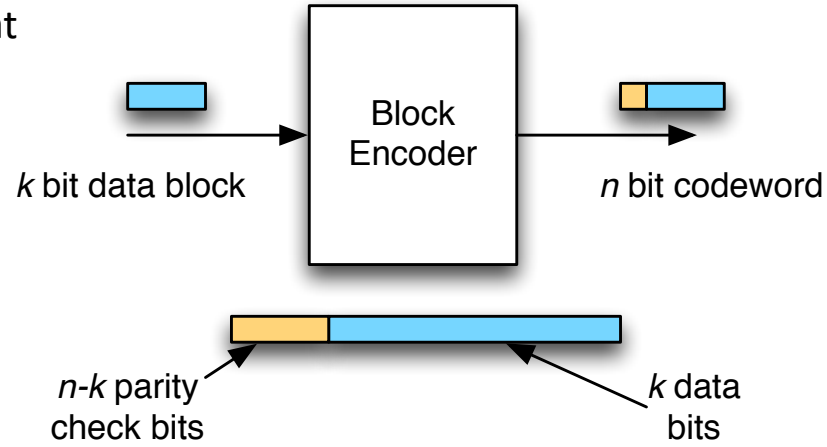
- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected. *BUT undetected errors are still possible…relatively weak protection*

Error control coding

- Coding means something different here!
  - Transmit redundant bits using which you can recover from errors
  - The redundant bits have a pattern that enables this recovery
- Types of coding
  - Block codes (*n*,*k*)
  - Convolutional codes
  - Trellis coded modulation
  - Turbo codes
  - LDPC codes
  - others



Code Rate – *k/n*

Cyclic Codes

- Cyclic or polynomial codes: a subclass of block codes
  - Cyclic shifts of valid codeword is another valid codeword
    - If $C=(c_{n-1},..., c_0)$ is codeword then so is $(c_{n-2},..., c_0, c_{n-1})$
  - Sum of two codewords (modulo 2) is also a codeword
- Advantage of easy encoder/decoder implementation resulting from this structure
- Design of codes is based on algebraic theory
- Most block codes in practice today are cyclic or related
- The data or message “word”, the parity bits, are all written as polynomials
  - Message: u(x);
  - Parity: b(x)

Properties of cyclic codes

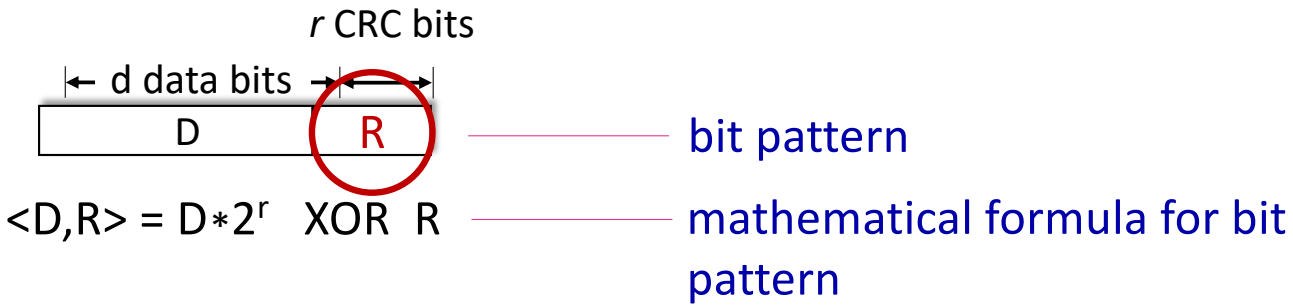
- Each codeword can be represented as a polynomial
  - $[c_0\ c_1\ c_2\ \ldots\ c_{n-1}] = c_0\ x^0 + c_1\ x^1 + \ldots + c_{n-1}\ x^{n-1} = c(x)$
  - Example: $1010001 = 1 + x^2 + x^6$
  - What is the degree of $c(x)$?
- We have a generator polynomial used to create the parity polynomial
  - Generator polynomial is $g(x)$ of degree $n - k$
- Based on Galois Fields

Creating and Decoding Cyclic Codes

- Given the generator polynomial $g(x)$
  - Multiply the message polynomial $u(x)$ by x^{n-k}
  - Divide $x^{n-k}\,u(x)$ by $g(x)$ → the remainder is $b(x)$
  - Add $b(x)$ to $x^{n-k}\,u(x)$ to get the code polynomial $c(x)$
- Let the received polynomial be $r(x)$
  - Divide $r(x)$ by $g(x)$
  - The remainder $s(x)$ is the "syndrome" polynomial
    - What is the degree of $s(x)$?
  - If $s(x) = 0$, there are no errors
  - Otherwise, $s(x)$ can be mapped to "most likely" errors and corrected

Cyclic Redundancy Check (CRC) in binary form

- 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 <D,R> exactly divisible by G (using mod 2 arithmetic)

- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can **detect all burst errors less than r+1 bits** (i.e. errors in up to r consecutive bits)
- widely used in practice (Ethernet, 802.11 WiFi)

Cyclic Redundancy Check (CRC): example

We want:

$D \cdot 2^r \ XOR \ R = nG$

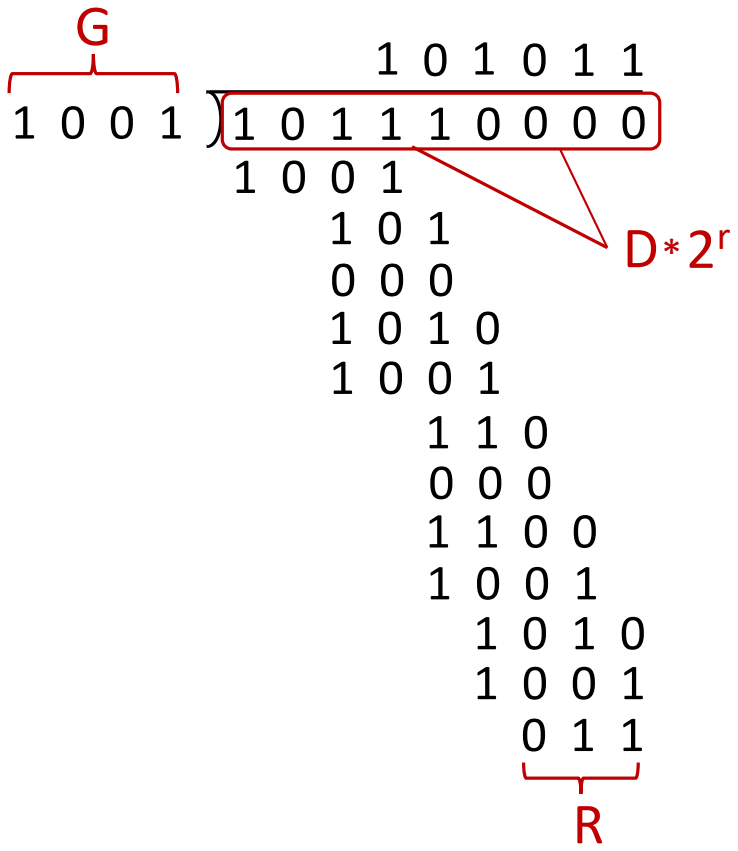
or equivalently:

$D \cdot 2^r = nG \ XOR \ R$

or equivalently:

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

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



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Link Layer Services: Multiple Access Protocols

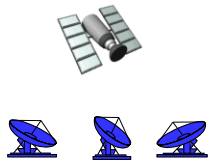
Point-to-point vs broadcast links

- point-to-point: dedicated pairwise communication
  - point-to-point link between Ethernet switch, host
  - long distance fiber link
- broadcast: shared wire or medium
  - old-fashioned Ethernet (pre ~2000)
  - 802.11 wireless LAN, 4G/5G, satellite
  - upstream HFC in cable-based access network

shared wire (e.g., cabled Ethernet)

shared radio: 4G/5G

shared radio: WiFi



shared radio: satellite

Multiple access problem

- Context: single shared **broadcast channel**

- What happens if multiple nodes want to transmit on the link at the same time?
  - **Collision**: a node receives two or more different signals at the same time…data is garbled, and all frames are lost

Coordination is needed!

An ideal multiple access protocol

given: multiple access channel with transmission 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: strategies

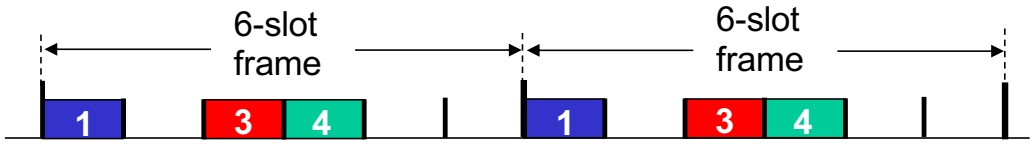
three broad classes:

- 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

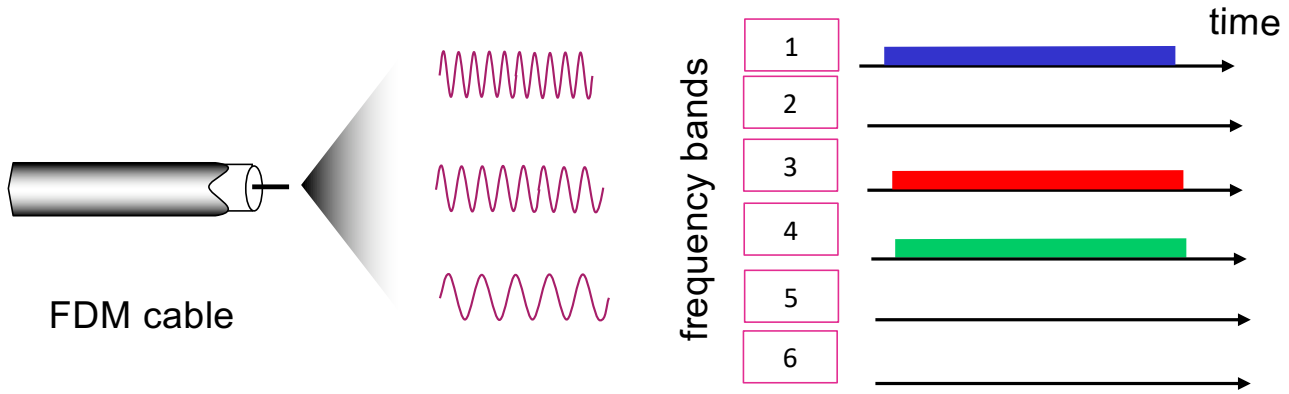
- access to channel in “rounds”
- each node gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: node LAN, 1,3,4 have packets to send, slots 2,5,6 idle



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-node LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random access protocols

- **when a node has packet to send:**
  - it transmits at full channel data rate R (without pre-coordination)
- **two or more transmitting nodes** -> **collision**
  - data is lost

- **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:

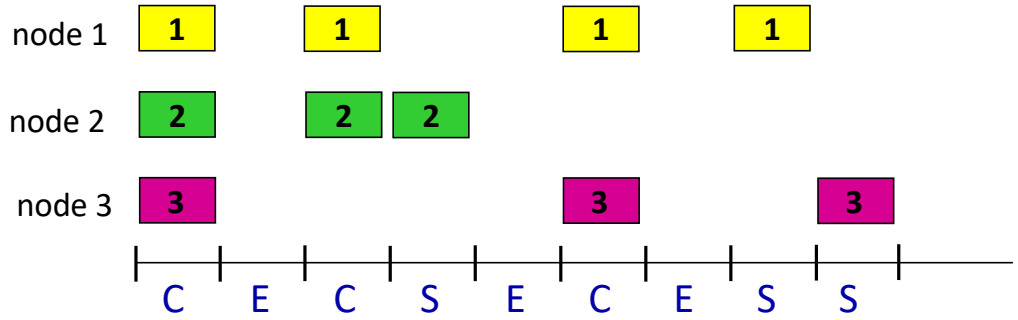
- all frames same size
- time divided into equal size **slots** (time to transmit 1 frame)
- nodes start to transmit only at start of slot
- **nodes are synchronized**
- if 2 or more nodes transmit in slot, all nodes detect collision before slot ends

operation:

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

randomization – *why*?

Slotted ALOHA



C: collision
S: success
E: empty

Pros:

- single active node can continuously transmit at full rate of channel
- highly decentralized: nodes independently decide when to send
- simple

Cons:

- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- *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:

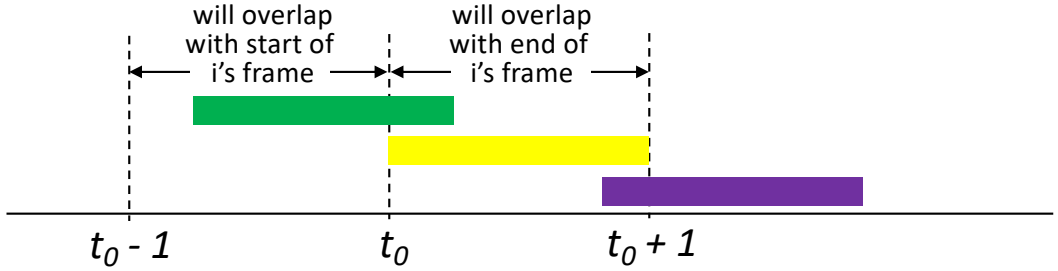
p = probability given node transmits
$(1-p)^{N-1}$ = probability that none of the other N-1 nodes transmits (no collision)

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
  - on collision: retransmit with probability *p* (and wait 1 frame transmission time between attempts)
- 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% !

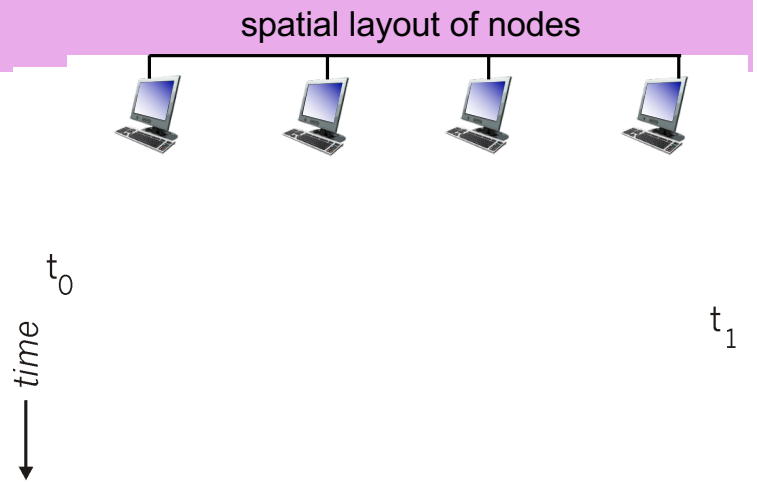
CSMA (carrier sense multiple access)

simple CSMA: **listen** (carrier sense) before transmit:

- if channel sensed idle: transmit entire frame
- if channel sensed busy: wait some random time and try (listen) again
- **Be polite ☺ (don't talk while someone else is talking)**

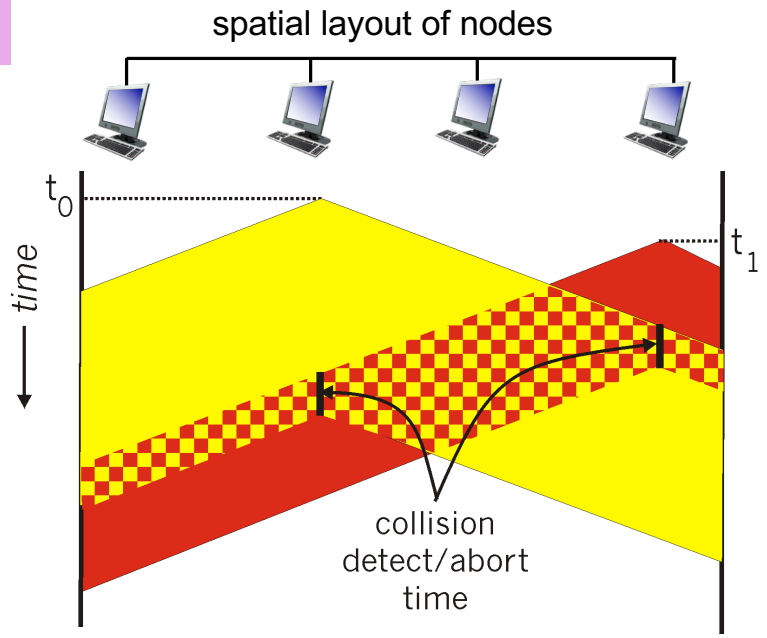
CSMA: collisions

- 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 affect collision probability



CSMA/CD

- CSMA/CD: CSMA with *collision detection*
- CSMA/CD reduces the amount of time wasted in collisions
  - transmission **aborted** on collision detection
  - frees up the channel instead of wasting time on a transmission that we already know will not succeed



CSMA/CD : Collision Detection

CSMA/CD: CSMA with *collision detection*

- Listen before transmitting, as in simple CSMA
- Also listen *while* transmitting to detect simultaneous transmissions (collisions)
- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless
  - receiving while transmitting is difficult in wireless

Ethernet

- Invented as a broadcast technology
  - Hosts share channel
  - Each packet received by all attached hosts
  - CSMA/CD for media access control
- Modern Ethernets are "switched" (more later)
  - Point-to-point links between switches and between a host and switch
  - No sharing $\Rightarrow$ no CSMA/CD

(Traditional) Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. 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 $\{0,1,2, \ldots, 2^m-1\}$
   - NIC waits $K \cdot 512$ bit times, returns to Step 2
   - **more collisions: longer backoff interval**

e.g. After 3 collisions, $2^3-1 = 7$, so choose with equal probability from {0,1,2,3,4,5,6,7}

CSMA/CD efficiency

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

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

- 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

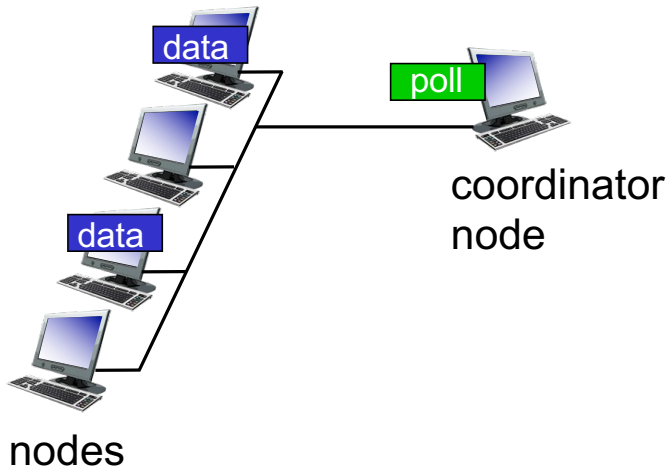
“taking turns” protocols

- look for best of both worlds!

“Taking turns” MAC protocols

polling:

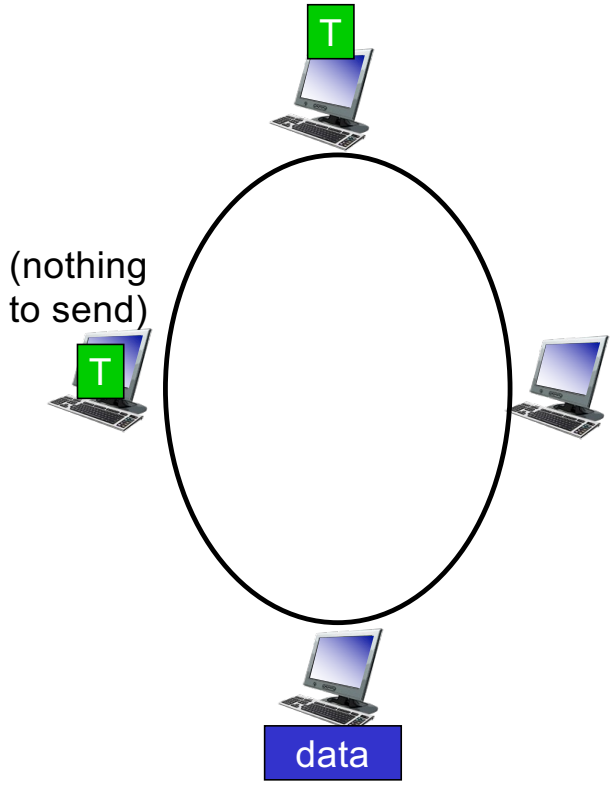
- coordinator node “invites” other nodes to transmit in turn
- typically used with “dumb” devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (coordinator)



“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)

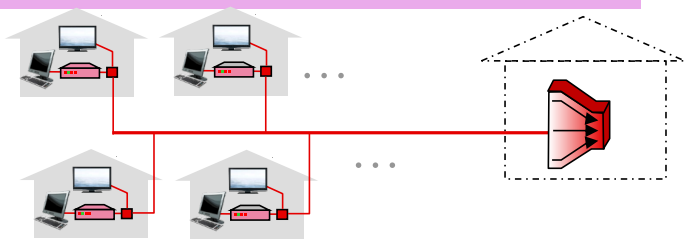


Summary of MAC protocols

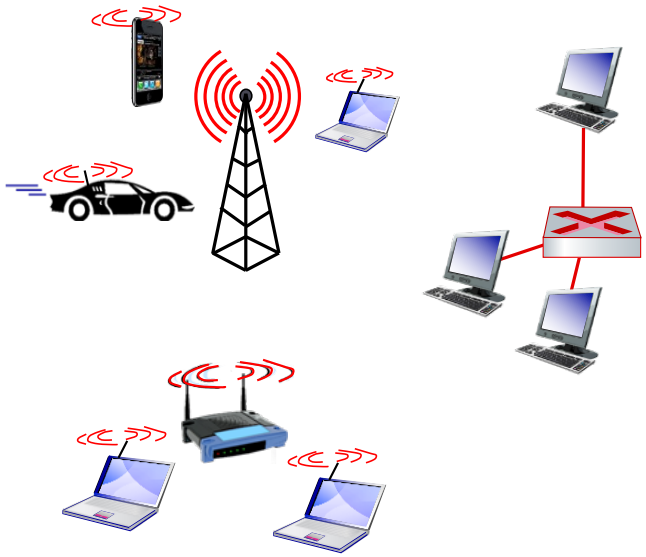
- channel partitioning, by time, frequency
  - Time Division, Frequency Division
- random access (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in (older) Ethernet
  - CSMA/CA used in 802.11
- taking turns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

Link layer: review

- **Link layer is responsible for delivering datagrams between *physically adjacent* nodes (same subnet)**



- **Services** offered by different link layer protocols can vary
  - framing: encapsulate datagram into frame
  - error detection *and correction*
    - link access: coordinate medium access if shared broadcast link



Multiple access problem - review

- Context: single shared **broadcast channel**

- What happens if multiple nodes want to transmit on the link at the same time?
  - **Collision**: a node receives two or more different signals at the same time…data is garbled, and all frames are lost

Coordination is needed!

Summary of MAC protocols

- channel partitioning, by time, frequency
  - 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 (older) Ethernet
  - CSMA/CA used in 802.11
- taking turns
  - polling from central site, token passing
  - Bluetooth, token ring