
Link Layer, Course Summary

Qiao Xiang

<https://qiaoxiang.me/courses/cnns-xmuf22/index.shtml>

12/13/2022

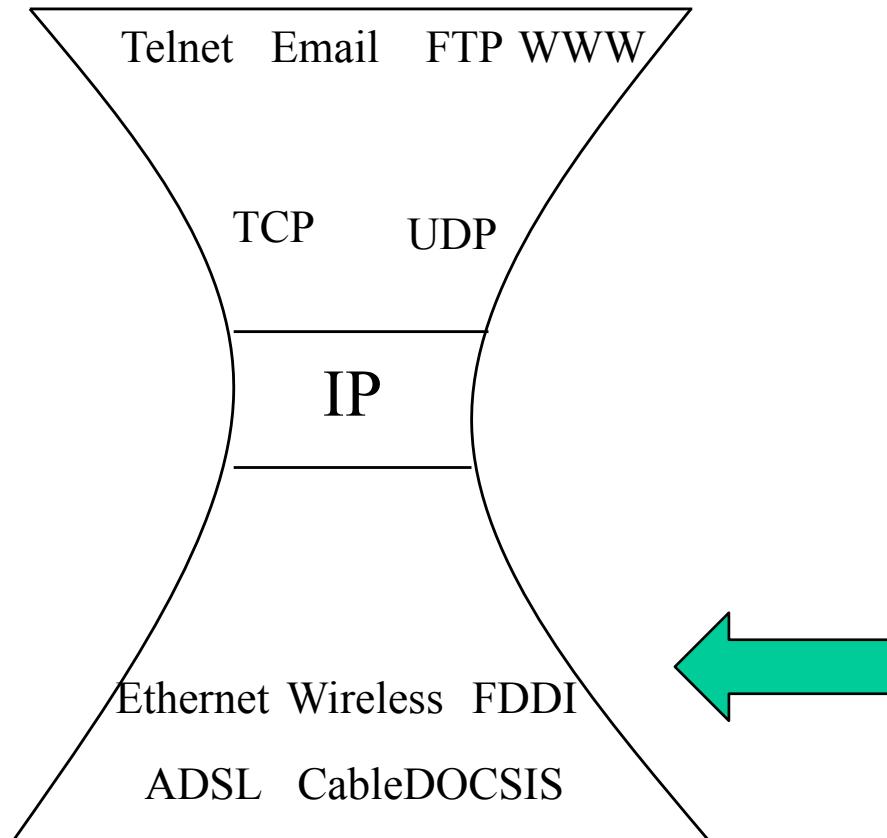
Outline

- Admin and recap
- Link layer
- Course Summary

Admin

- Assignment 5 posted
 - No programming needed
 - Due on Dec. 31
- Guest lecture on Thursday
 - 5G Networks, Dr. Zhenhua Li@Tsinghua University

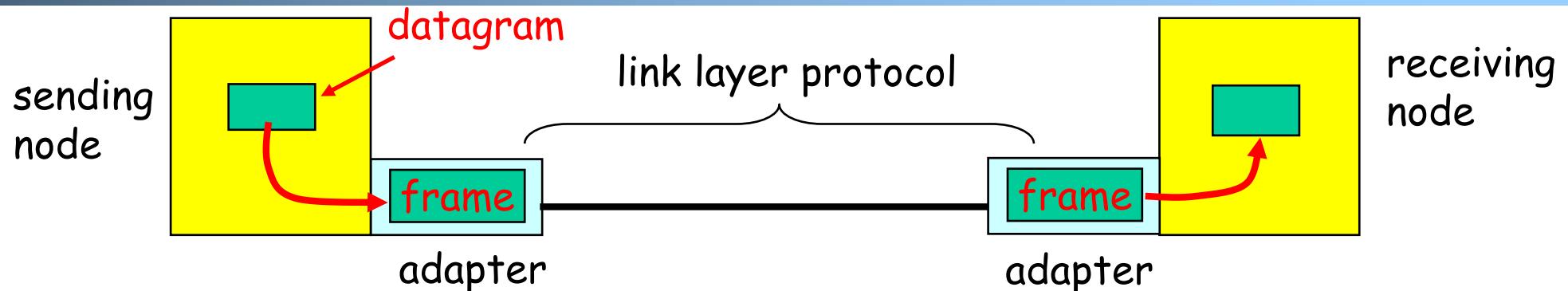
Roadmap: The Hourglass Architecture of the Internet



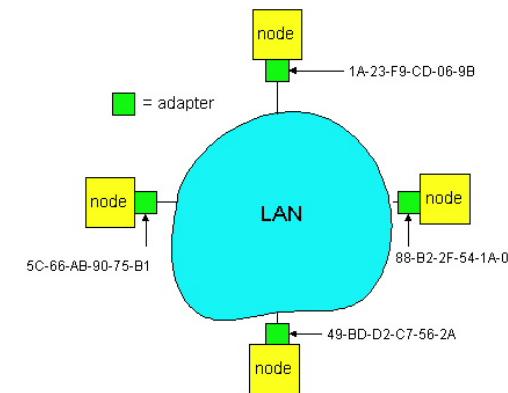
Link Layer Services

- Framing
 - encapsulate datagram into frame, adding header, trailer and error detection/correction
- Multiplexing/demultiplexing
 - frame headers to identify src, dest
- Reliable delivery between adjacent nodes
 - we learned how to do this already !
 - seldom used on low bit error link (fiber, some twisted pair)
 - common for wireless links: high error rates
- Media access control
- Forwarding/switching with a link-layer (Layer 2) domain

Adaptors Communicating



- link layer typically implemented in “adaptor” (aka NIC)
 - Ethernet card, modem, 802.11 card, cloud virtual switch
- adapter is semi-autonomous, implementing link & physical layers
- in most link-layer, each adapter has a unique link layer address (also called MAC address)

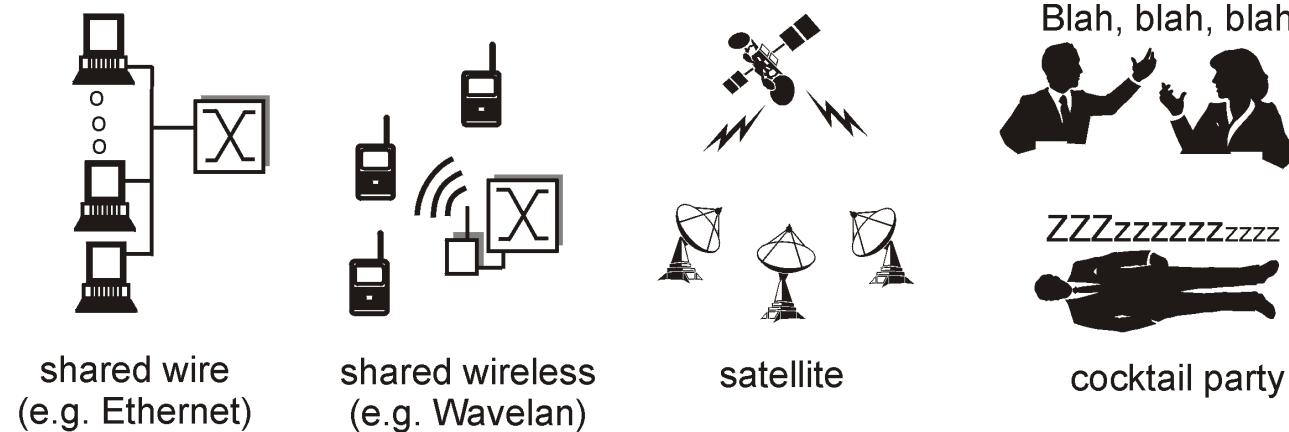


Outline

- Admin and recap
- Network layer
- Link layer
 - Overview
 - Media access
 - Link layer forwarding

Multiple Access Links and Protocols

- Many link layers use **broadcast** (shared wire or medium)
 - traditional Ethernet; Cable networks
 - 802.11 wireless LAN; cellular networks
 - satellite



- Problem: if two or more simultaneous transmissions, due to **interference**, only one node can send successfully at a time (see CDMA later for an exception)

Multiple Access Protocol

- Protocol that determines how nodes share channel, i.e., determines when nodes can transmit
- Communication about channel sharing must use channel itself !

- Discussion: properties of an ideal multiple access protocol.

Ideal Multiple Access Protocol

Broadcast channel of rate R bps

- Efficiency: when only one node wants to transmit, it can send at full rate R
- Rate allocation:
 - simple fairness: when N nodes want to transmit, each can send at average rate R/N
 - we may need more complex rate control
- Decentralized:
 - no special node to coordinate transmissions
 - no synchronization of clocks
- Simple

MAC Protocols

Goals

- efficient, fair, decentralized, simple

Three broad classes:

- non-partitioning
 - random access
 - allow collisions
 - “taking-turns”
 - a token coordinates shared access to avoid collisions
- channel partitioning
 - divide channel into smaller “pieces”
(time slot, frequency, code)

Focus: Random Access Protocols

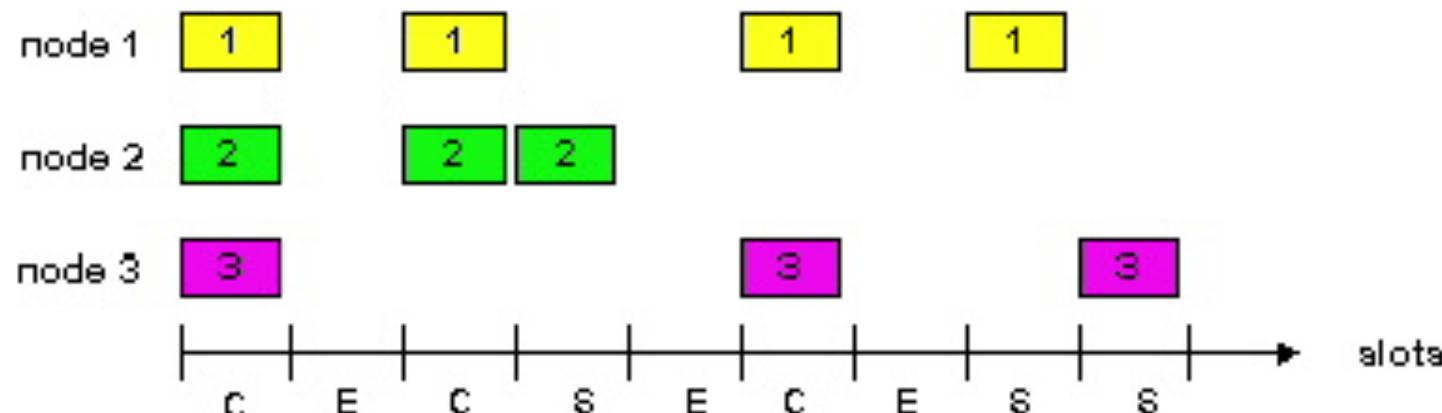
- Examples of random access MAC protocols:
 - slotted ALOHA and pure ALOHA
 - CSMA and CSMA/CD, CSMA/CA
 - Ethernet, WiFi 802.11

- Key design points:
 - when to access channel?
 - how to detect collisions?
 - how to recover from collisions?

Slotted Aloha [Norm Abramson]



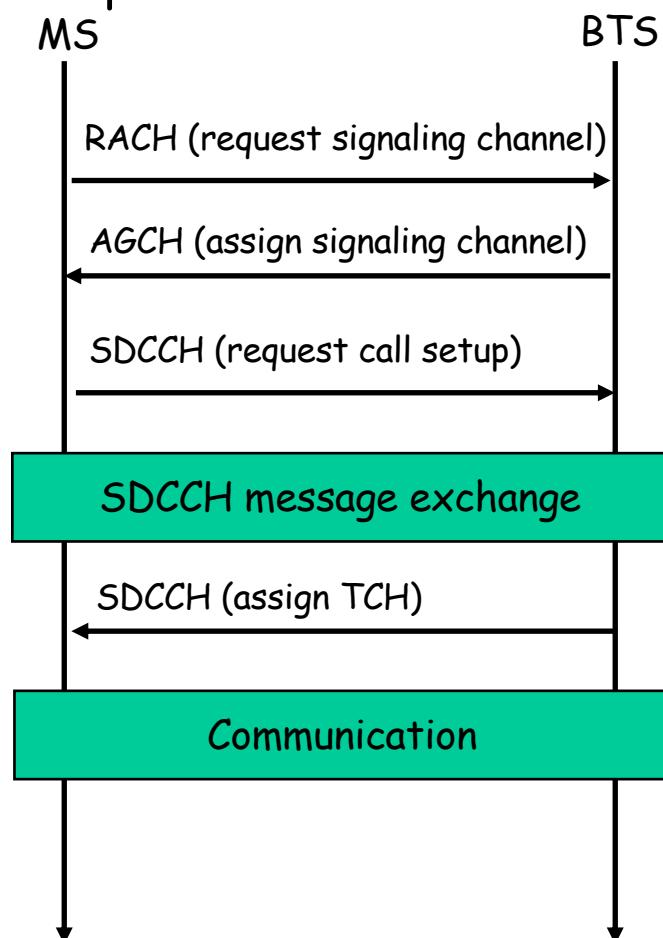
- Time is divided into equal size slots (= pkt trans. time)
- Node with new arriving pkt: transmit at beginning of next slot
- If collision: retransmit pkt in future slots with probability **p**, until successful.



Success (S), Collision (C), Empty (E) slots

Slotted Aloha in Real Life

- call setup in GSM



- Notations:

- Broadcast control channel (BCCH): from base station, announces cell identifier, synchronization
- Random access channel (RACH): MSs for initial access, **slotted Aloha**
- access grant channel (AGCH): BTS informs an MS its allocation
- standalone dedicated control channel (SDCCH): signaling and short message between MS and an MS
- Traffic channels (TCH)

Slotted Aloha Efficiency

Q: What is the fraction of successful slots?

suppose n stations have packets to send

suppose each transmits in a slot with probability p

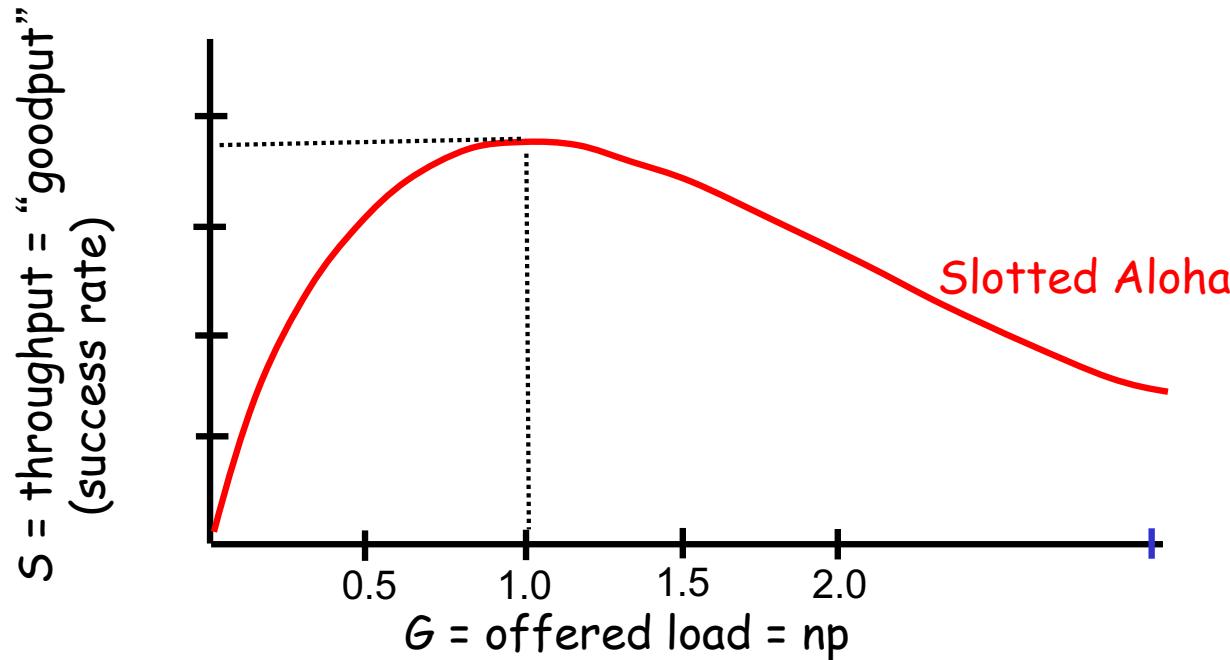
- prob. of succ. by a specific node: $p (1-p)^{(n-1)}$

- prob. of succ. by any one of the N nodes

$$S(p) = n * \text{Prob}(\text{only one transmits})$$

$$= n p (1-p)^{(n-1)}$$

Goodput vs. Offered Load for Slotted Aloha



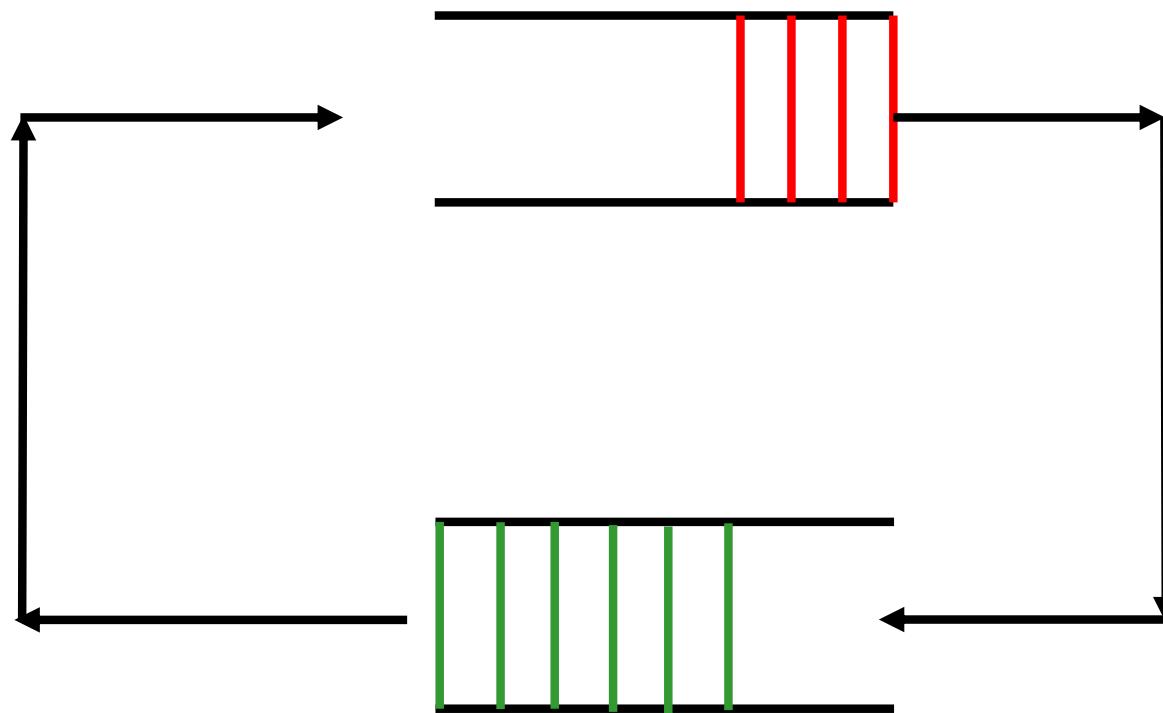
- when $p n < 1$, as p (or n) increases
 - probability of empty slots reduces
 - probability of collision is still low, thus goodput increases
- when $p n > 1$, as p (or n) increases,
 - probability of empty slots does not reduce much, but
 - probability of collision increases, thus goodput decreases
- goodput is optimal when $p n = 1$, $n \rightarrow \infty$, $S \rightarrow 1/e \sim 37\%$

Dynamics of (Slotted) Aloha

- Slotted Aloha has maximum throughput when $np = 1$
 - Implies we need to adjust p as the number of backlog stations varies.
- Early design question: what is the effect if we do not change p --use a fixed p
 - Assume we have a total of m stations (the machines on a LAN):
 - n of them are currently backlogged, each tries with a (fixed) probability p
 - the remaining $m-n$ stations are not backlogged. They may start to generate packets with a probability p_a , where p_a is much smaller than p

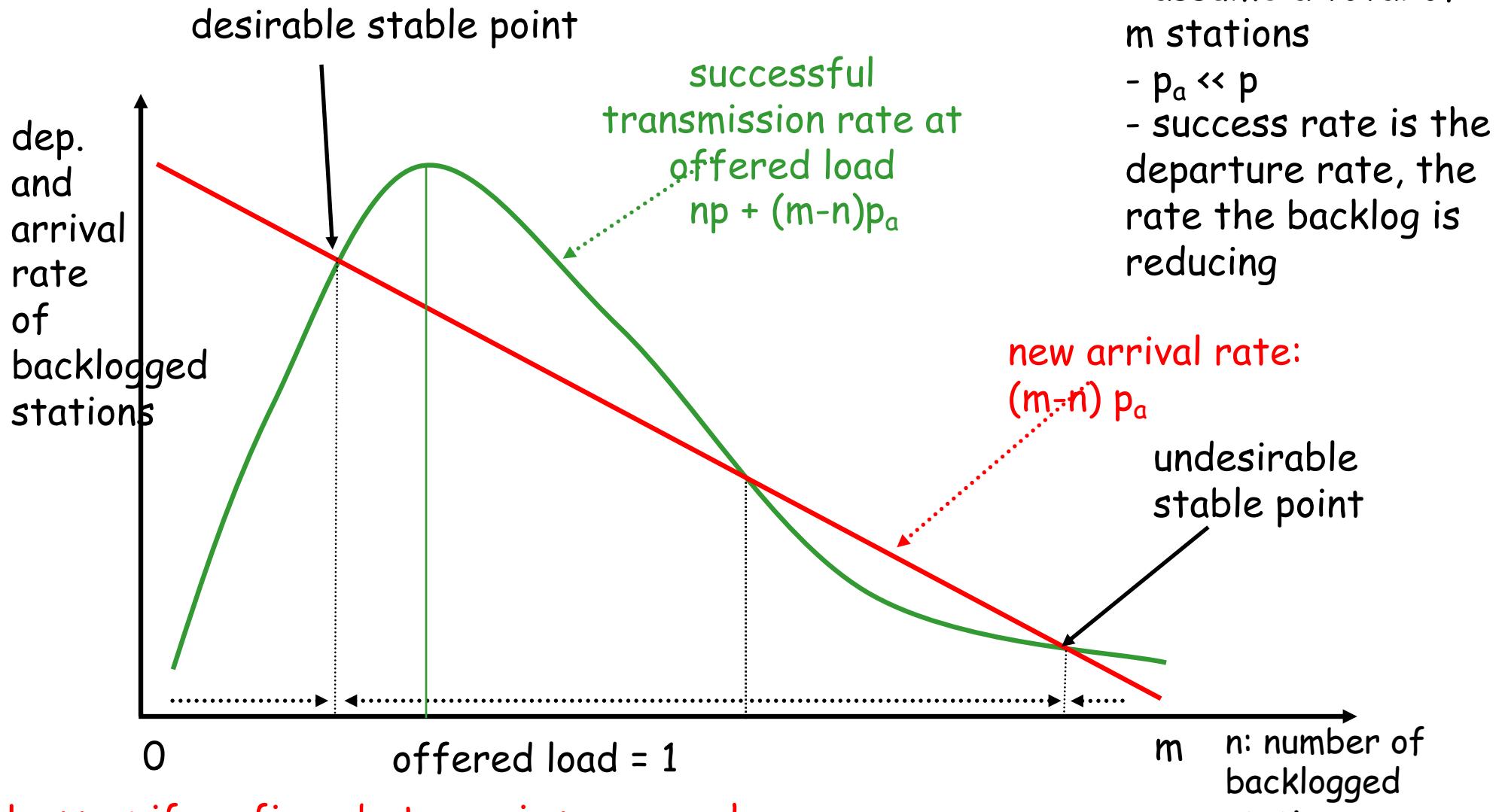
Model

n backlogged
each transmits with prob. p



$m-n$: unbacklogged
each transmits with prob. p_a

Dynamics of Aloha: Effects of Fixed Probability



Lesson: if we fix p , but n varies, we may have an undesirable stable point

Summary of Problems of Aloha Protocols

□ Problems

- slotted Aloha has better efficiency than pure Aloha but clock synchronization is hard to achieve
- Aloha protocols have low efficiency due to collision or empty slots
 - when offered load is optimal ($p = 1/N$), the goodput is only about 37%
 - when the offered load is not optimal, the goodput is even lower
- undesirable steady state at a fixed transmission rate, when the number of backlogged stations varies

□ Ethernet design: address the problems:

- optimal transmission rate

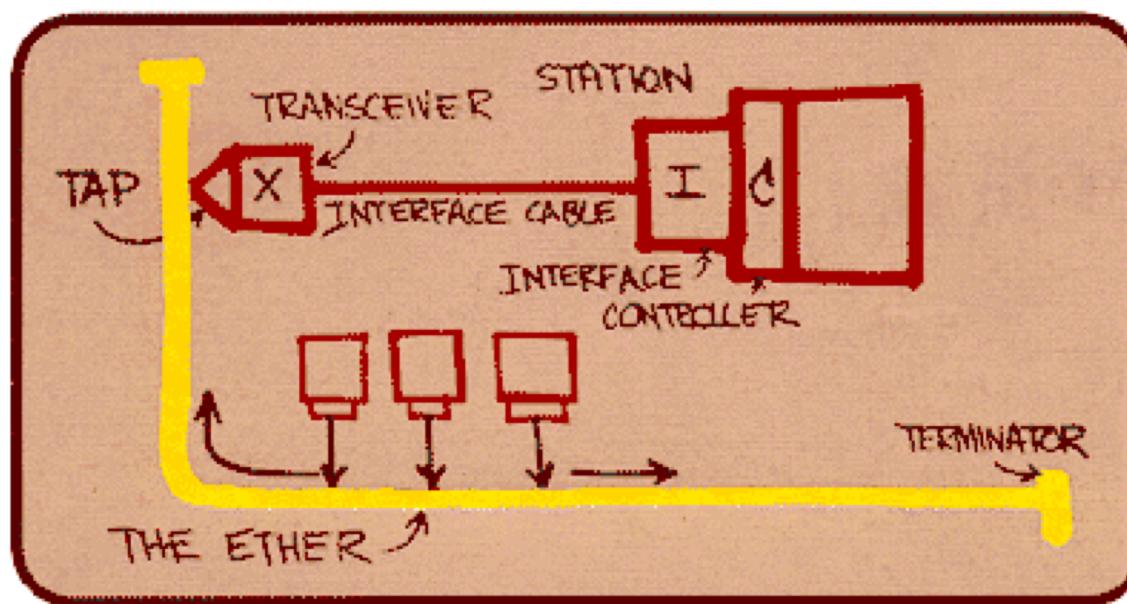
The Basic MAC Mechanisms of Ethernet

```
get a packet from upper layer;  
K := 0; n := 0; // K: control wait time; n: no. of collisions  
repeat:  
    wait for K * 512 bit-time;  
    while (network busy) wait;  
    wait for 96 bit-time after detecting no signal;  
    transmit and detect collision;  
    if detect collision  
        stop and transmit a 48-bit jam signal;  
        n ++;  
        m:= min(n, 10), where n is the number of collisions  
        choose K randomly from {0, 1, 2, ..., 2m-1}.  
    if n < 16 goto repeat  
else give up
```

Ethernet

“Dominant” LAN technology:

- First widely used LAN technology
- Kept up with speed race: 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps



Metcalfe's Ethernet sketch

Outline

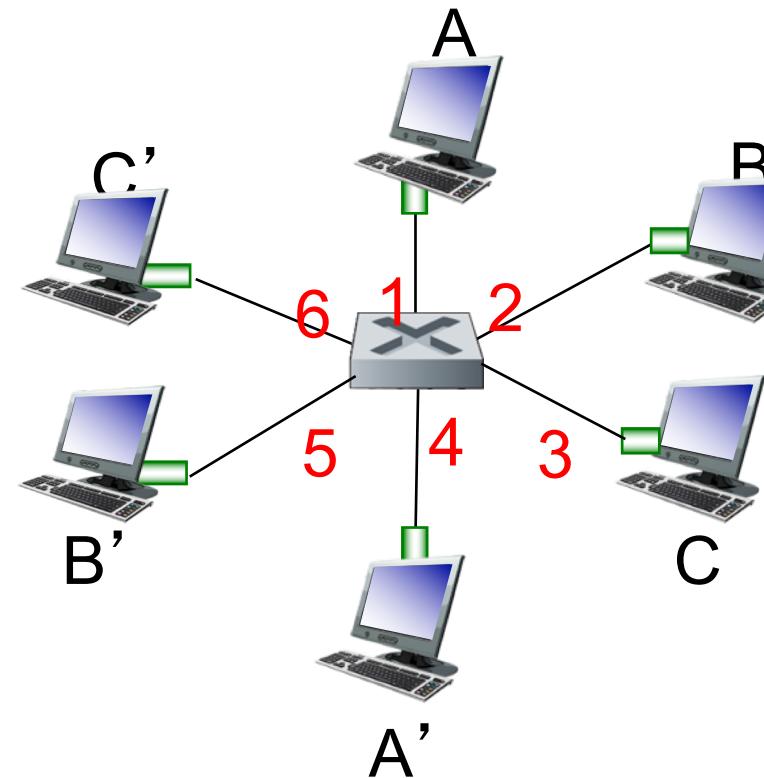
- Admin and recap
- Link layer
 - Ethernet switch

Ethernet Switch

- link-layer device: takes an *active* role
 - store, forward Ethernet frames
 - examine incoming frame's MAC address,
selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- *transparent*
 - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
 - switches do not need to be configured

Switch: Multiple Simultaneous Transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
 - each link is its own collision domain
- *switching*: A-to-A' and B-to-B' can transmit simultaneously, without collisions



*switch with six interfaces
(1,2,3,4,5,6)*

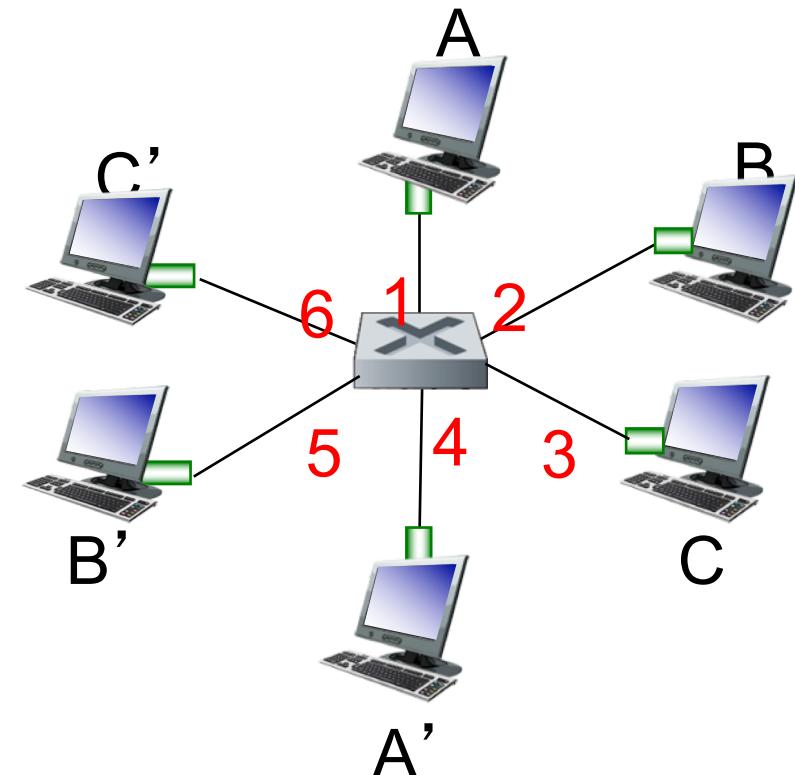
Switch Forwarding Table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- A: each switch has a **switch table**, each entry:
 - (MAC address of host, interface to reach host, time stamp)
 - looks like a routing table!

Q: how are entries created, maintained in switch table?

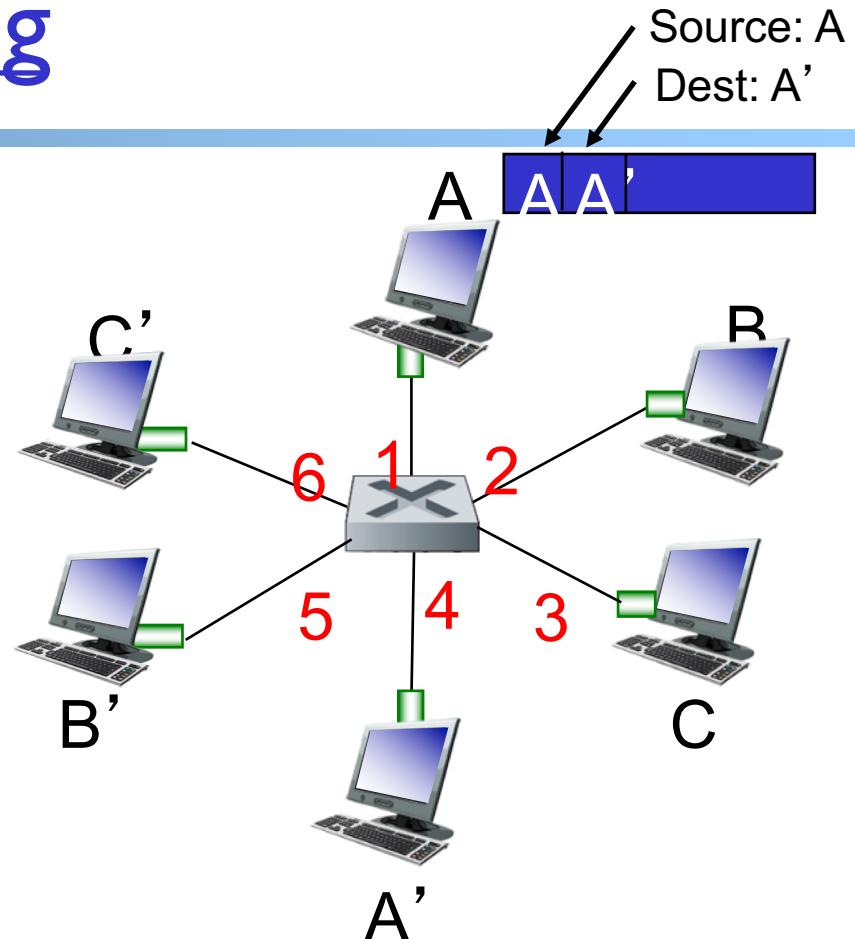
- something like a routing protocol?



*switch with six interfaces
(1,2,3,4,5,6)*

Switch: Self-Learning

- switch *learns* which hosts can be reached through which interfaces
 - when frame received, switch “learns” location of sender: incoming LAN segment
 - records sender/location pair in switch table



MAC addr	interface	TTL
A	1	60

*Switch table
(initially empty)*

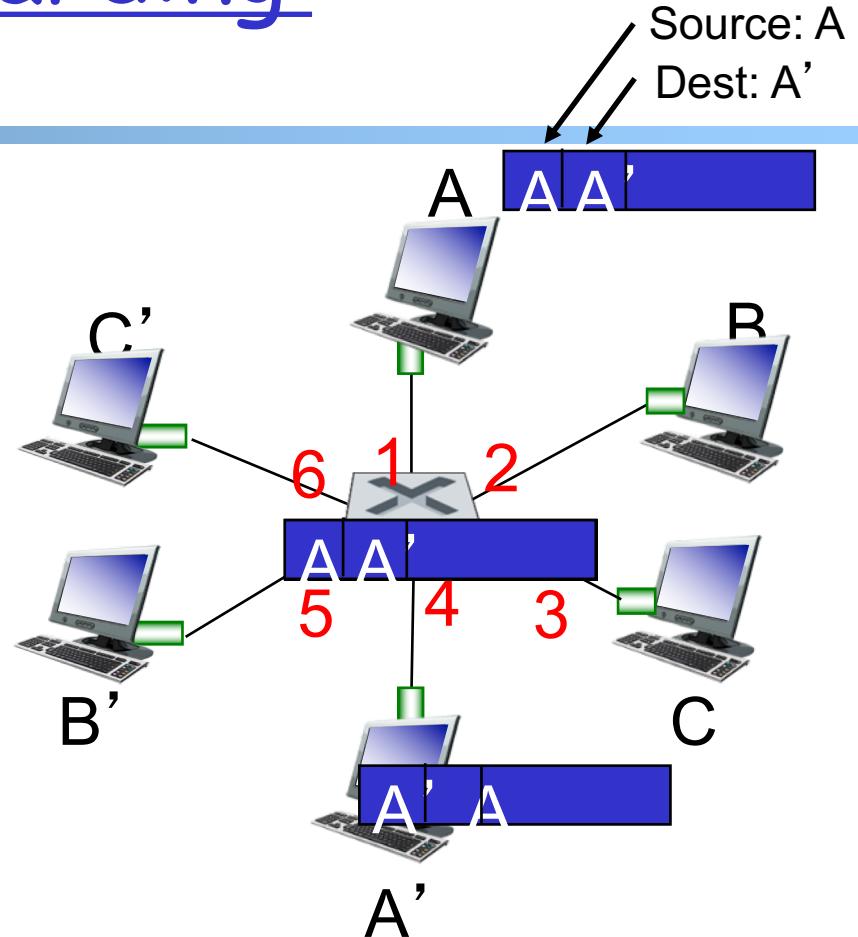
Switch: Frame Filtering /Forwarding

when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
 - then {
 - if destination on segment from which frame arrived
 - then drop frame
 - else forward frame on interface indicated by entry
 - }
- else flood /* forward on all interfaces except arriving interface */

Self-Learning, Forwarding: Example

- frame destination, A', location unknown: **flood**
- destination A location known: **selectively send on just one link**

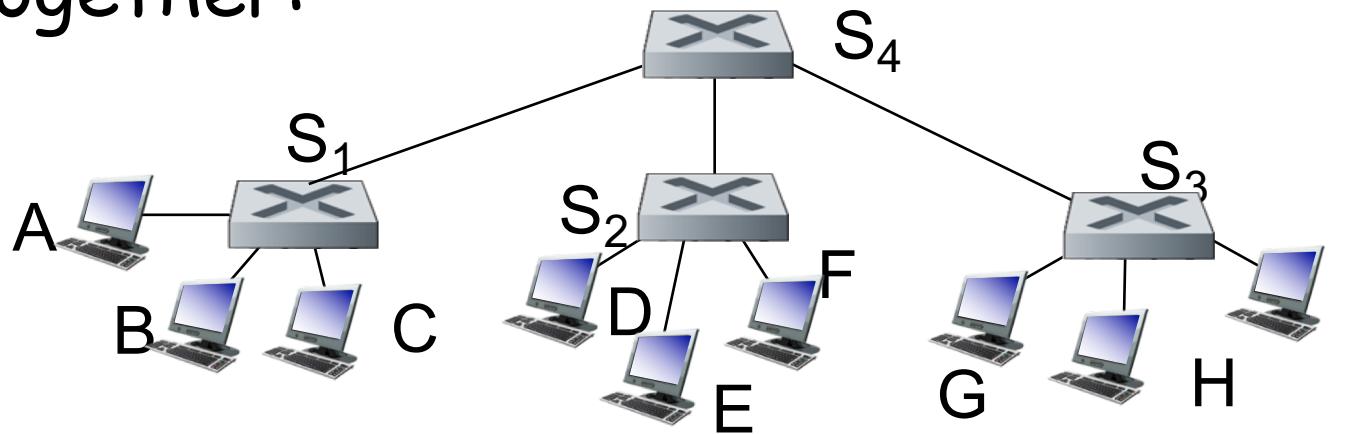


MAC addr	interface	TTL
A, A'	1 4	60 60

*switch table
(initially empty)*

Interconnecting Switches

self-learning switches can be connected together:

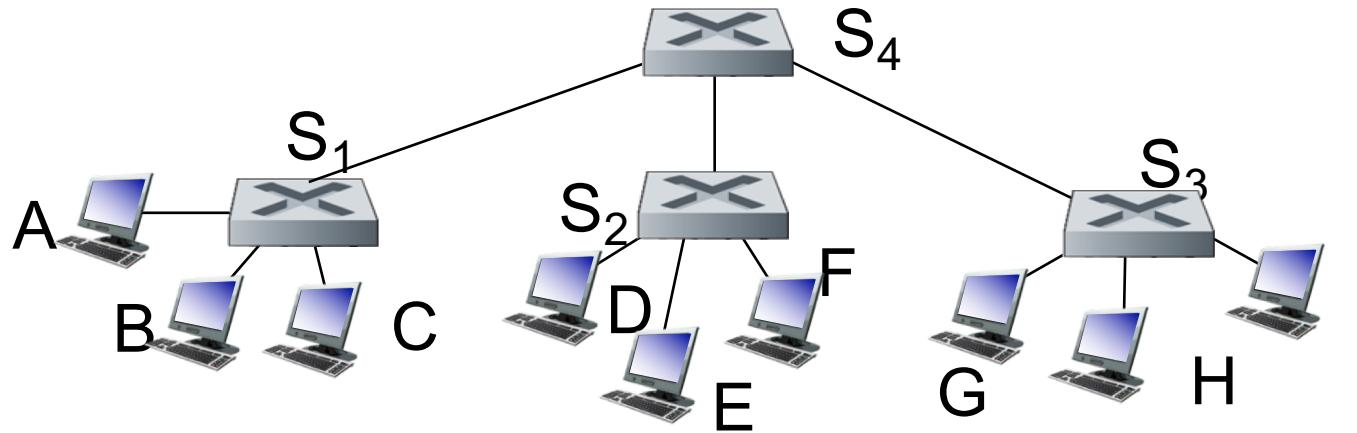


Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3 ?

- A: self learning! (works exactly the same as in single-switch case!)

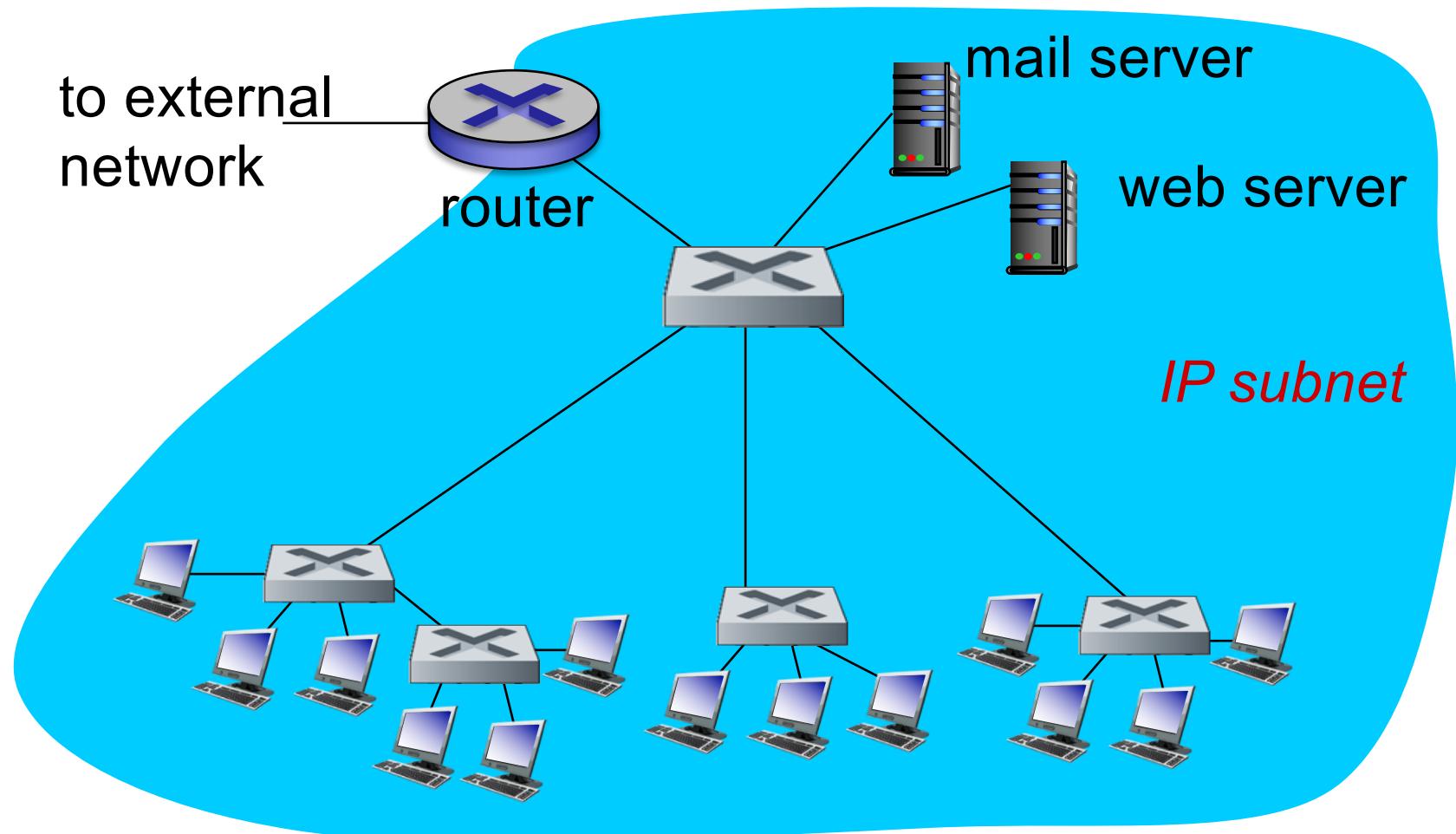
Self-Learning Multi-switch Example

Suppose C sends frame to I, I responds to C



- Offline Exercise: show switch tables and packet forwarding in S_1, S_2, S_3, S_4

Institutional Network



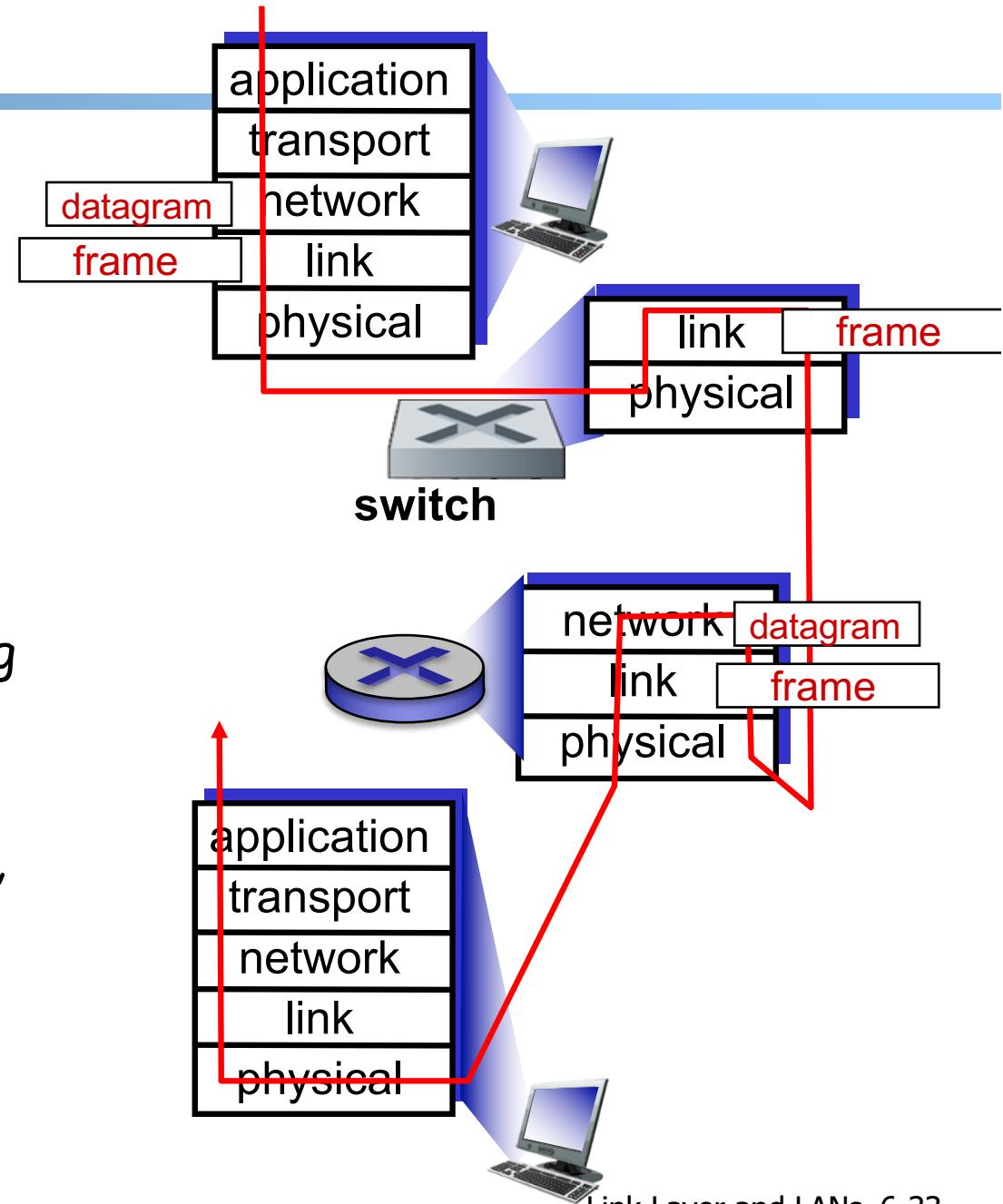
Switches vs. Routers

both are store-and-forward:

- **routers**: network-layer devices (examine network-layer headers)
- **switches**: link-layer devices (examine link-layer headers)

both have forwarding tables:

- **routers**: compute tables using routing algorithms, IP addresses
- **switches**: learn forwarding table using flooding, learning, MAC addresses



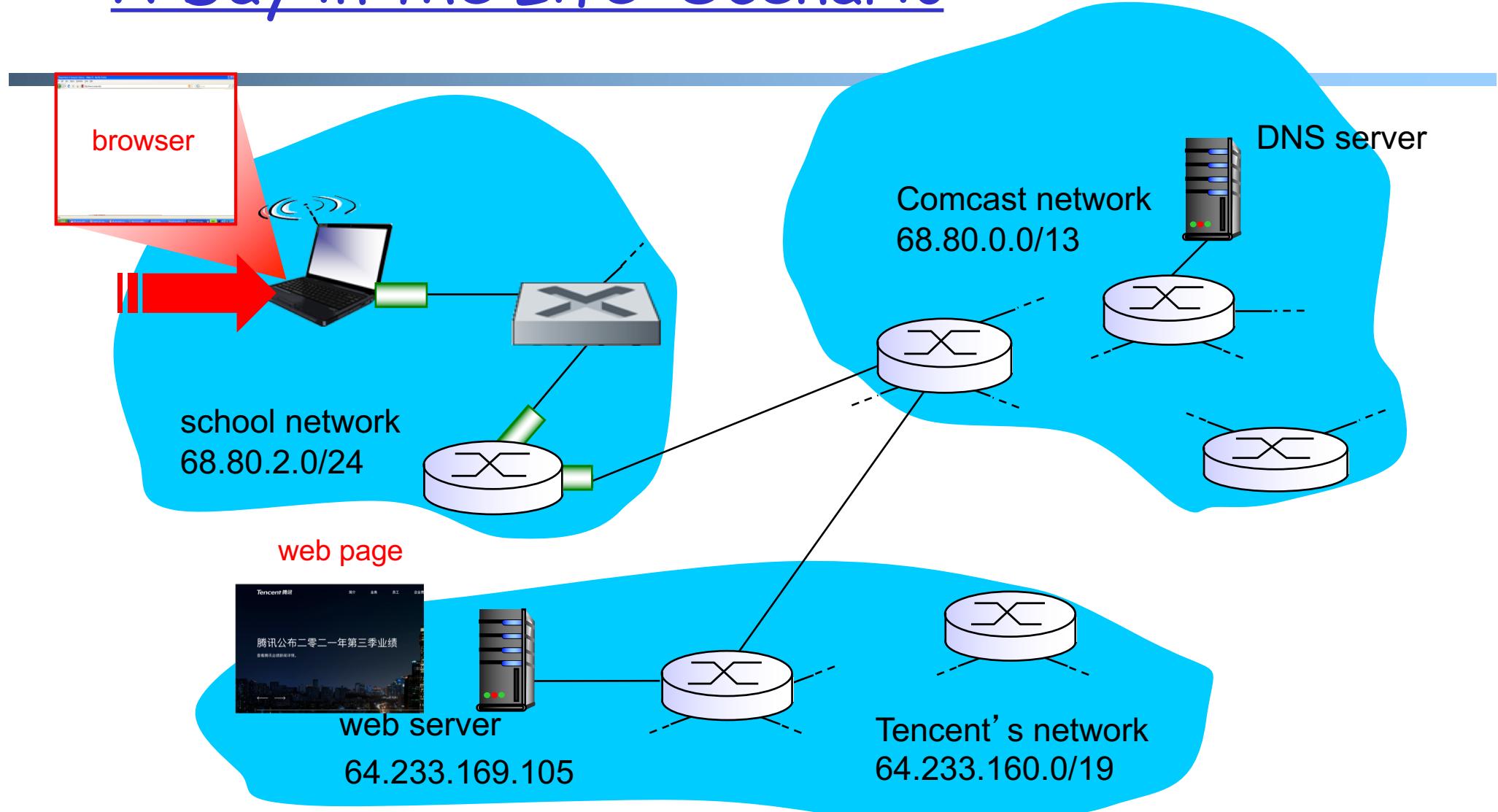
Outline

- Admin and recap
- Link layer
- Course Summary

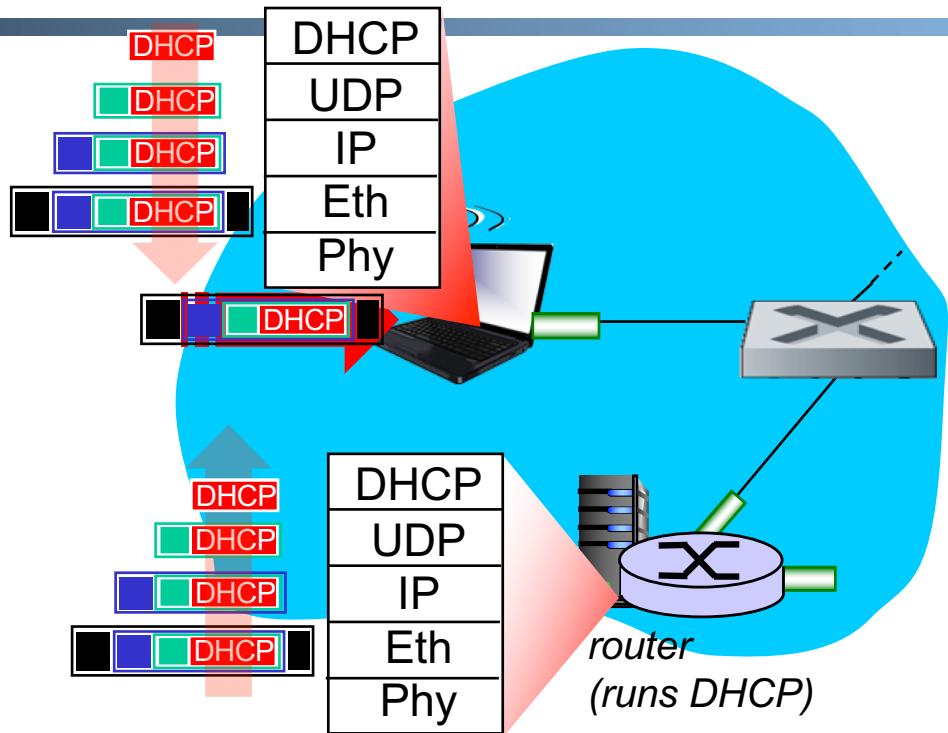
Synthesis: A Day in the Life of a Web Request

- journey down protocol stack complete!
 - application, transport, network, link
- putting-it-all-together: synthesis!
 - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
 - *scenario:* student attaches laptop to campus network, requests/receives www.tencent.com

A Day in the Life: Scenario

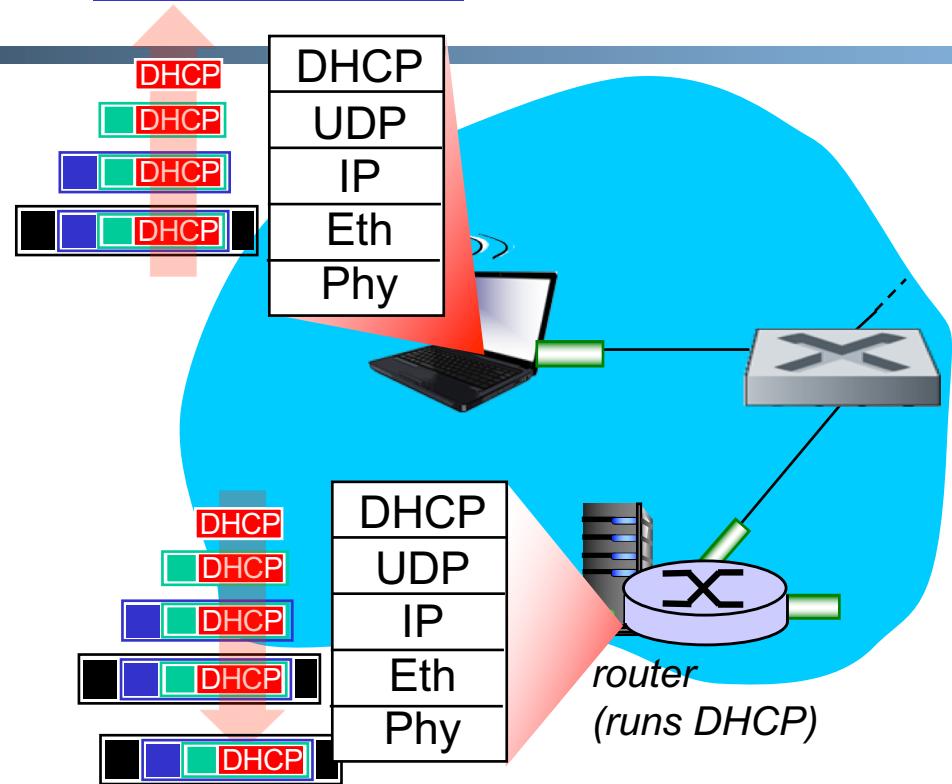


A Day in the Life... Connecting to the Internet



- ❑ connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use **DHCP**
- ❑ DHCP request **encapsulated** in **UDP**, encapsulated in **IP**, encapsulated in **802.3 Ethernet**
- ❑ Ethernet frame **broadcast** (dest: FFFFFFFFFFFF) on LAN, received at router running **DHCP** server
- ❑ Ethernet **demuxed** to IP demuxed, UDP demuxed to DHCP

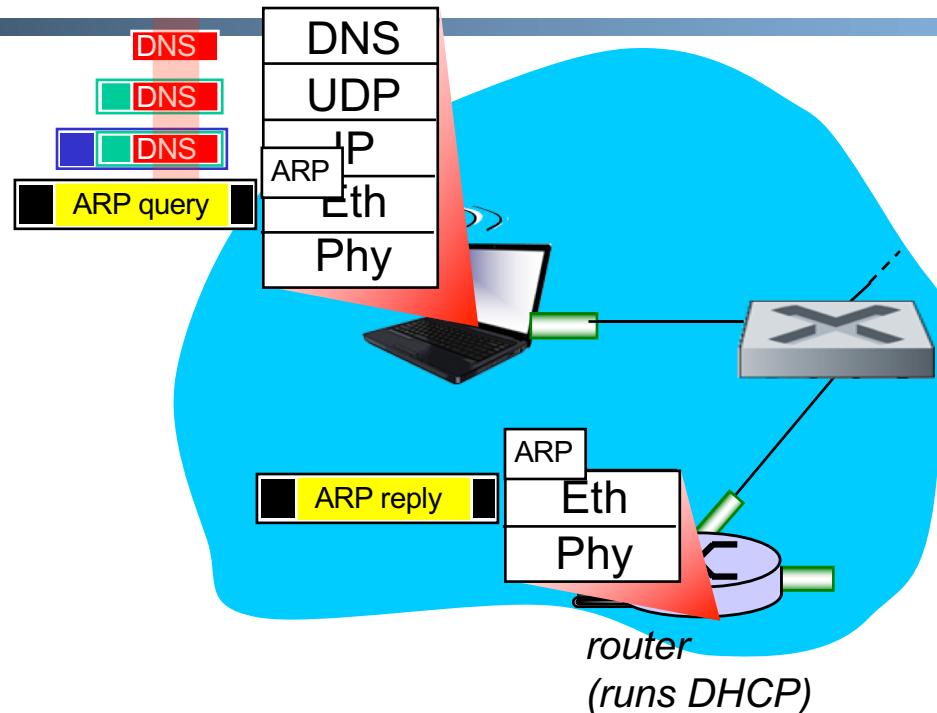
A Day in the Life... Connecting to the Internet



- ❑ DHCP server formulates **DHCP ACK** containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- ❑ encapsulation at DHCP server, frame forwarded (**switch learning**) through LAN, demultiplexing at client
- ❑ DHCP client receives DHCP ACK reply

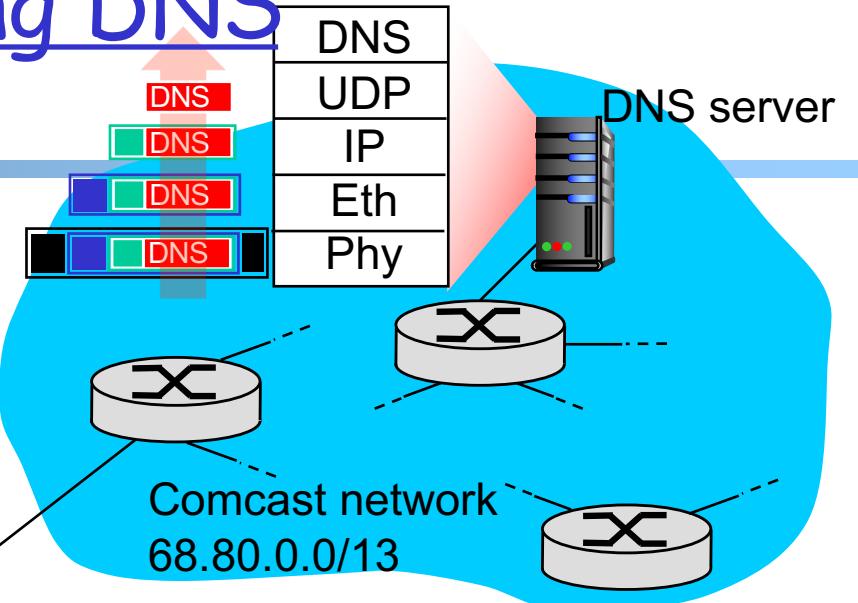
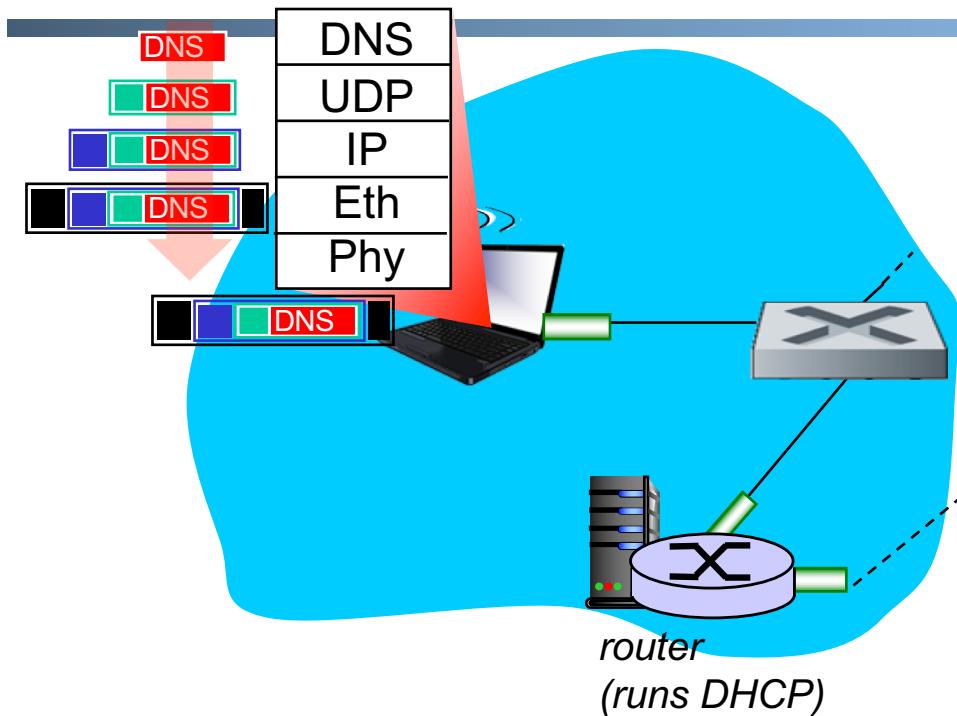
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A Day in the life... ARP (before DNS, before HTTP)



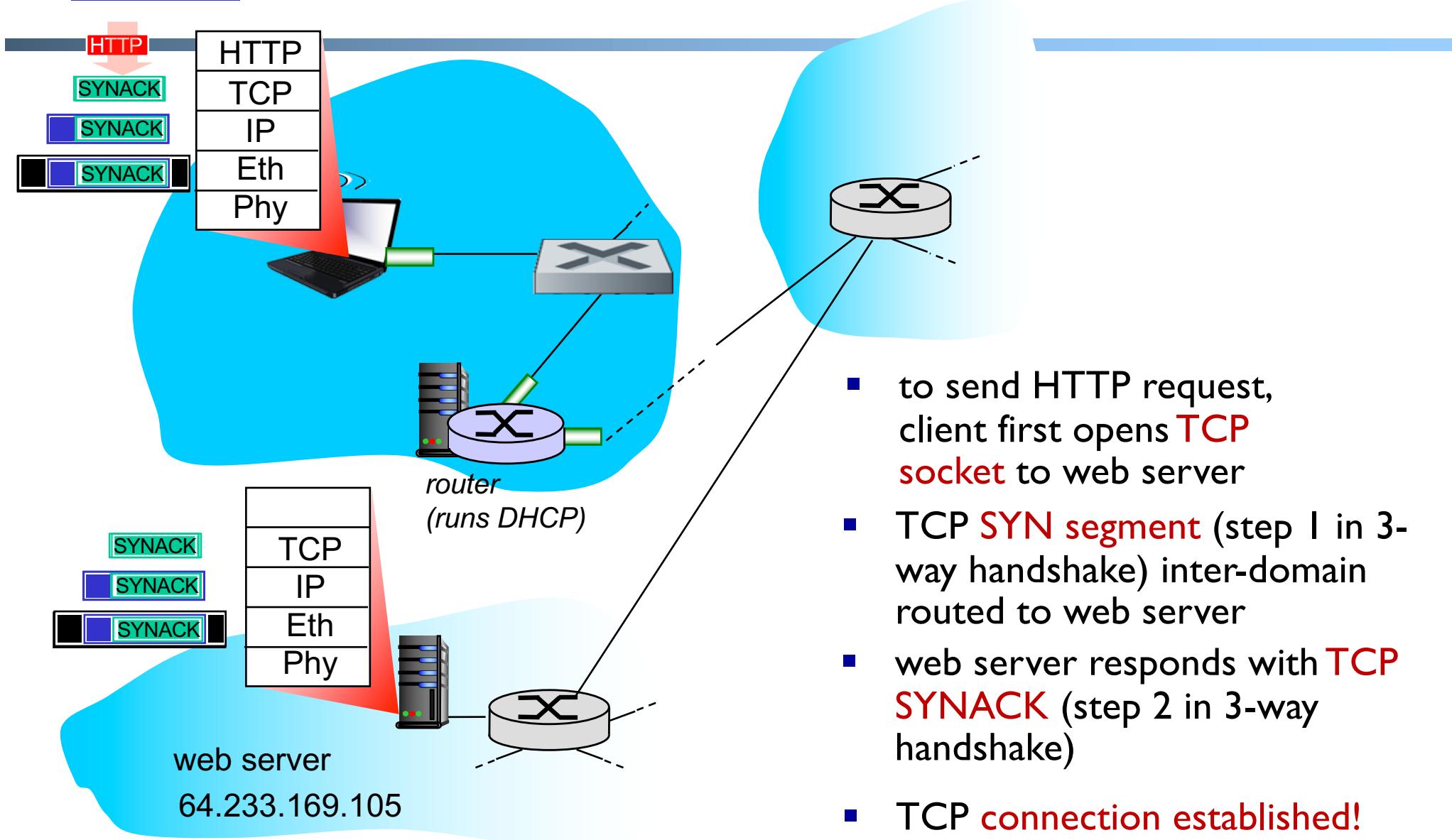
- ❑ before sending *HTTP* request, need IP address of www.tencent.com: **DNS**
- ❑ DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: **ARP**
- ❑ **ARP query** broadcast, received by router, which replies with **ARP reply** giving MAC address of router interface
- ❑ client now knows MAC address of first hop router, so can now send frame containing DNS query

A Day in the Life... Using DNS

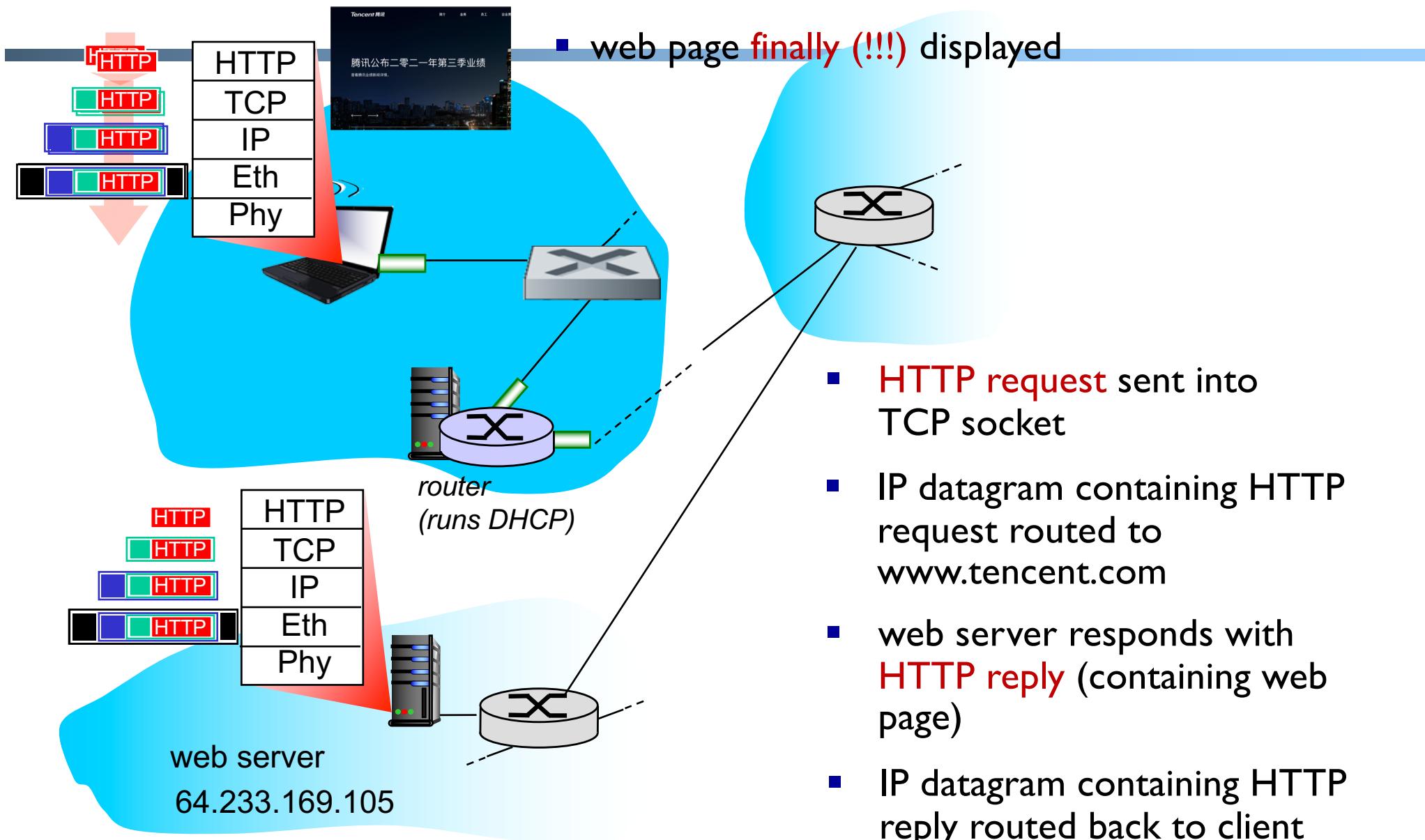


- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by **RIP**, **OSPF**, **IS-IS** and/or **BGP** routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of **www.tencent.com**

A Day in the Life... TCP Connection Carrying HTTP



A Day in the Life... HTTP Request/Reply



Course Topics Summary

- The Internet is a general-purpose, large-scale, distributed computer network
- Major design features/principles
 - packet switching/statistical multiplexing
 - time-reversibility, queueing theory and performance analysis
 - layered architecture, hour-glass architecture
 - end-to-end principle
 - decentralized (social-technocal) architecture
 - e.g., DNS (hierarchy delegation), interdomain routing (peer-to-peer)
 - resource allocation framework
 - axiom-based design (NBS); optimization decomposition through duality
 - adaptive control
 - e.g., sliding window self clocking, AIMD adaptation, Ethernet exp backoff
 - tradeoff between theoretical impossibility and practice

First-Day Class: What is a Network Protocol?



- A **network protocol** defines the **format** and the **order** of messages exchanged between two or more communicating entities, as well as the **actions** taken on the transmission and/or receipt of a message or other **events**.

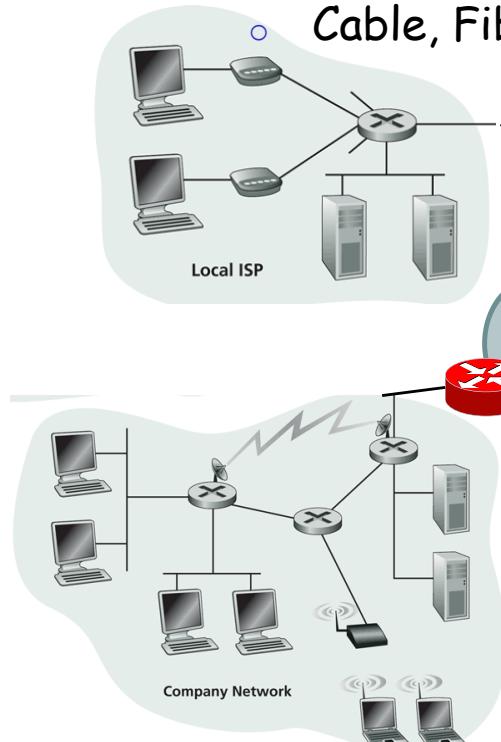
Protocols that we have touched on?

First-Day Class: Internet Physical Infrastructure



Residential access

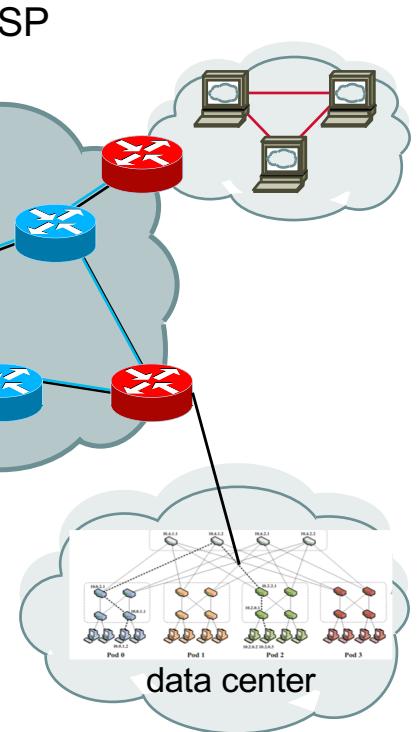
- Cable, Fiber, DSL, Wireless



Campus access,
e.g.,

- Ethernet
- Wireless

- The Internet is a network of networks
- Each individually administrated network is called an Autonomous System (AS)



First-Day Class: General Complexity



- **Complexity** in highly organized systems arises primarily from design strategies intended to create **robustness to uncertainty** in their environments and component parts.
 - **Scalability** is robustness to changes to the size and complexity of a system as a whole.
 - **Evolvability** is robustness of lineages to large changes on various (usually long) time scales.
 - **Reliability** is robustness to component failures.
 - **Efficiency** is robustness to resource scarcity.
 - **Modularity** is robustness to component rearrangements.

First-Day Class: Evolution

- Driven by Technology, Infrastructure, Policy, Applications (usage), and Understanding:
 - technology
 - e.g., wireless/optical communication technologies and device miniaturization (sensors)
 - infrastructure
 - e.g., cloud computing vs local computing
 - applications (usage)
 - e.g., mobile computing, content distribution, game, tele presence, sensing
 - understanding
 - e.g., resource sharing principle, routing principles, mechanism design, optimal stochastic control (randomized access)
- Complexity comes from evolution.
- Don't be afraid to challenge the foundation and redesign!

Have a great winter break!



Optional Read:
Channel Partitioning MAC

Example Channel Partitioning: CDMA

CDMA (Code Division Multiple Access)

- Used mostly in wireless broadcast channels (cellular, satellite, etc)
- A spread-spectrum technique



Hedy Lamarr and George Antheil. Photo of Hedy Lamarr courtesy of the Academy of Motion Picture Arts & Sciences. Photo of George Antheil courtesy of the Estate of George Antheil.

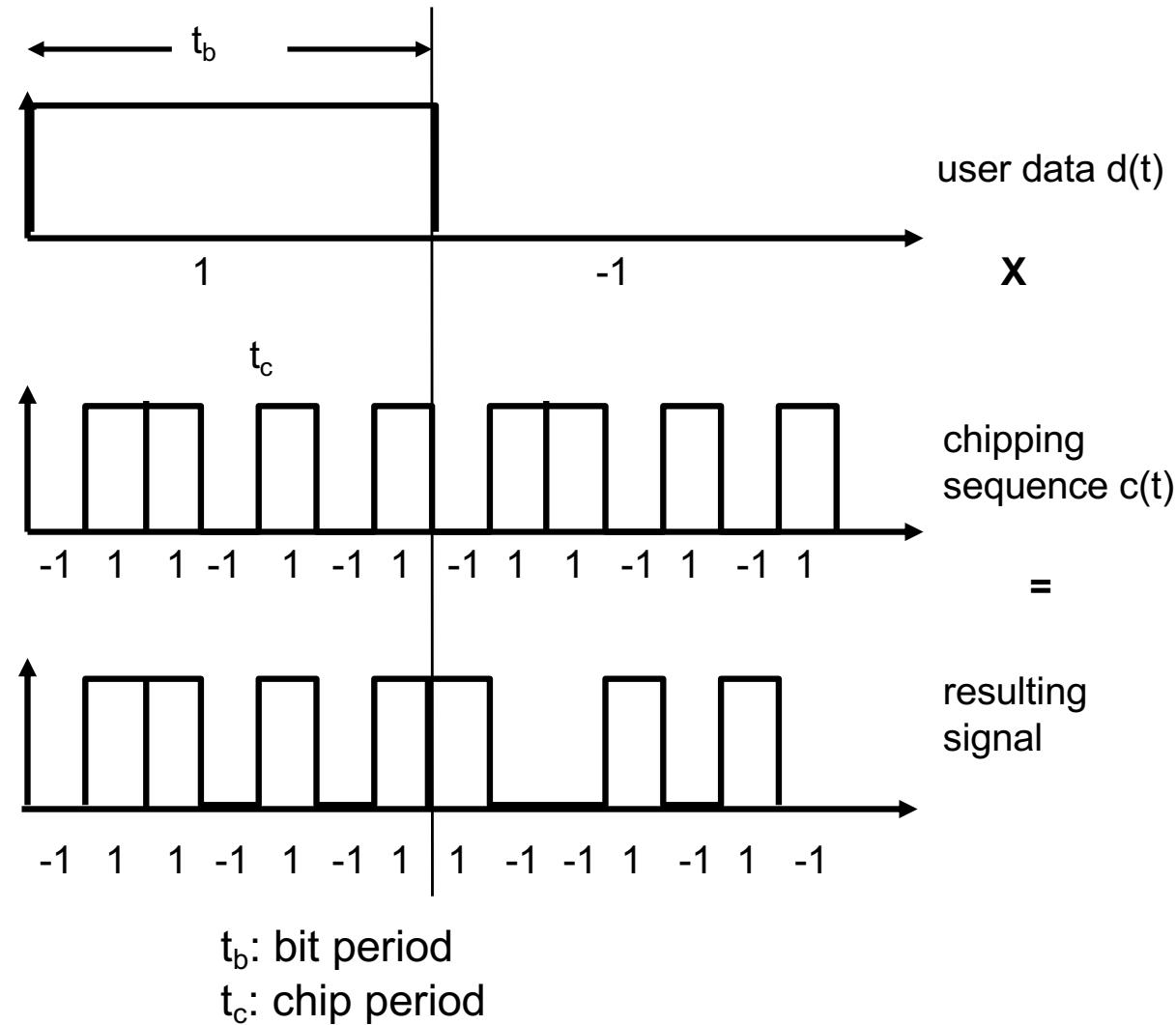
History: http://people.seas.harvard.edu/~jones/cscie129/nu_lectures/lecture7/hedy/lemarr.htm

Examples: Sprint and Verizon, WCDMA

CDMA: Encoding

- All users share same frequency, but each user m has its own unique “chipping” sequence (i.e., code) c_m to encode data, i.e., code set partitioning
 - e.g. $c_m = 1 \ 1 \ 1 \ -1 \ 1 \ -1 \ -1 \ -1$
- Assume original data are represented by 1 and -1
- *Encoded signal* = (original data) modulated by (chipping sequence)
 - assume $c_m = 1 \ 1 \ 1 \ -1 \ 1 \ -1 \ -1 \ -1$
 - if data is d , send $d c_m$,
 - if data d is 1, send c_m
 - if data d is -1 send $-c_m$

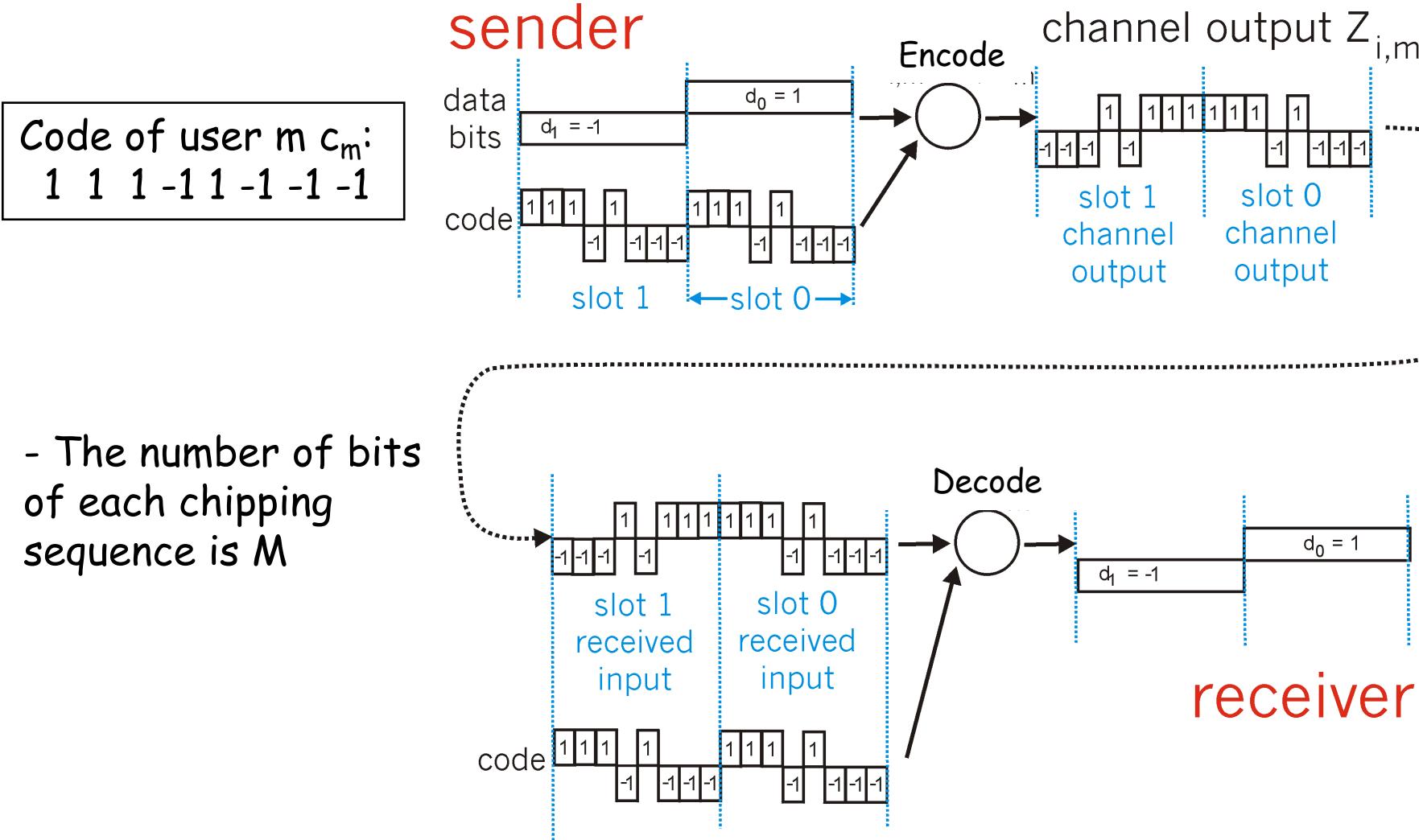
CDMA: Encoding



CDMA: Decoding

- Inner-product (summation of bit-by-bit product) of encoded signal and chipping sequence
 - if inner-product > 0, the data is 1; else -1

CDMA Encode/Decode



CDMA: Deal with Multiple-User Interference

- Two codes c_i and c_j are **orthogonal**, if
 - $c_j \bullet c_i = 0$, where we use “ \bullet ” to denote inner product,
e.g.

$C_1:$	1	1	1	-1	1	-1	-1	-1
$C_2:$	1	-1	1	1	1	-1	1	1

$C_1 \bullet C_2 =$	1 + (-1) + 1 + (-1)	+ 1 + 1 + (-1) + (-1) = 0						

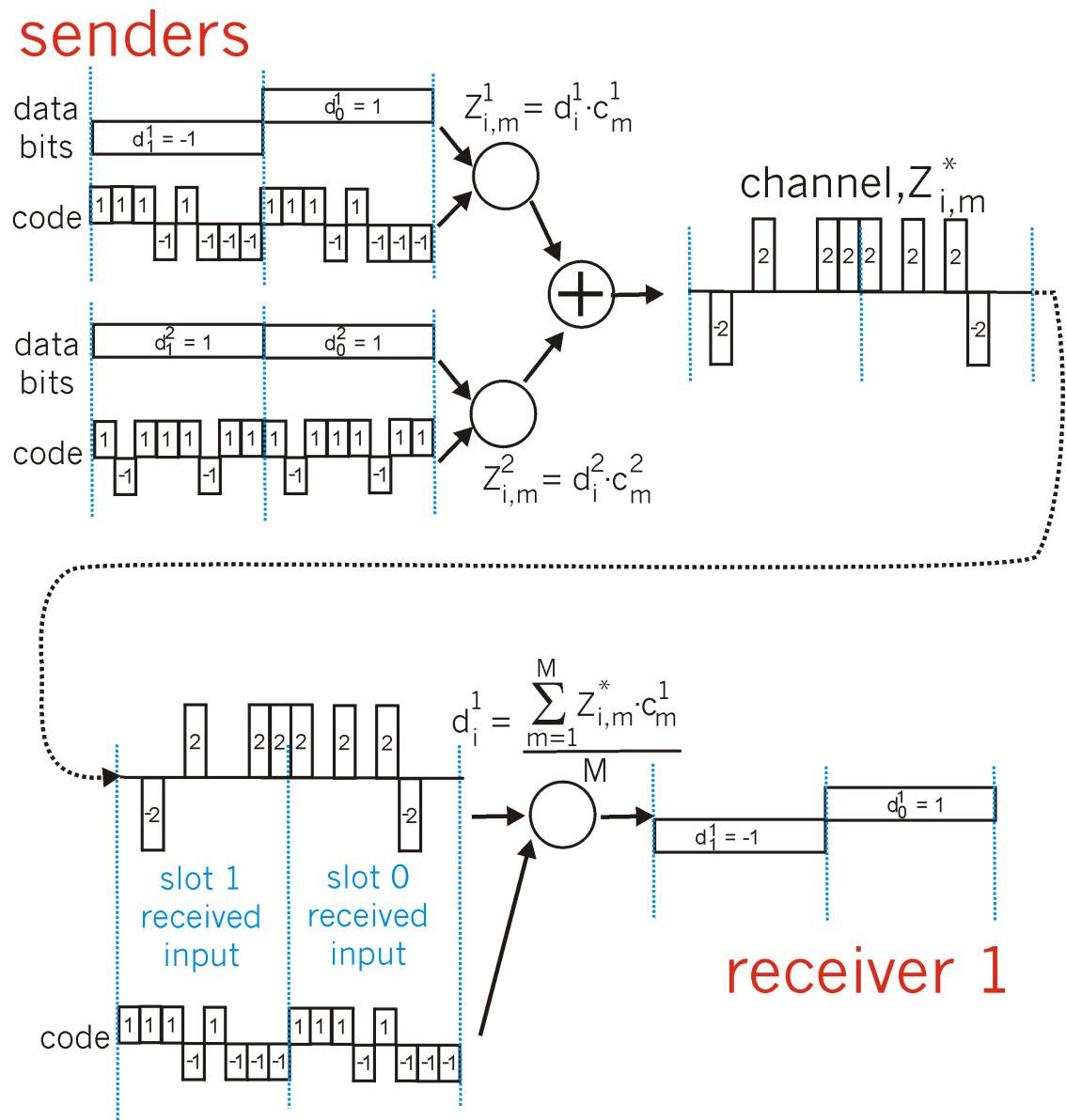
- If codes are orthogonal, multiple users can “coexist” and transmit simultaneously with minimal interference:

$$\left(\sum_j d_j c_j \right) \bullet c_i = d_i \|c_i\|$$

Analogy: Speak in different languages!

CDMA: Two-Sender Interference

Code 1: 1 1 1 -1 1 -1 -1 -1
 Code 2: 1 -1 1 1 1 -1 1 1



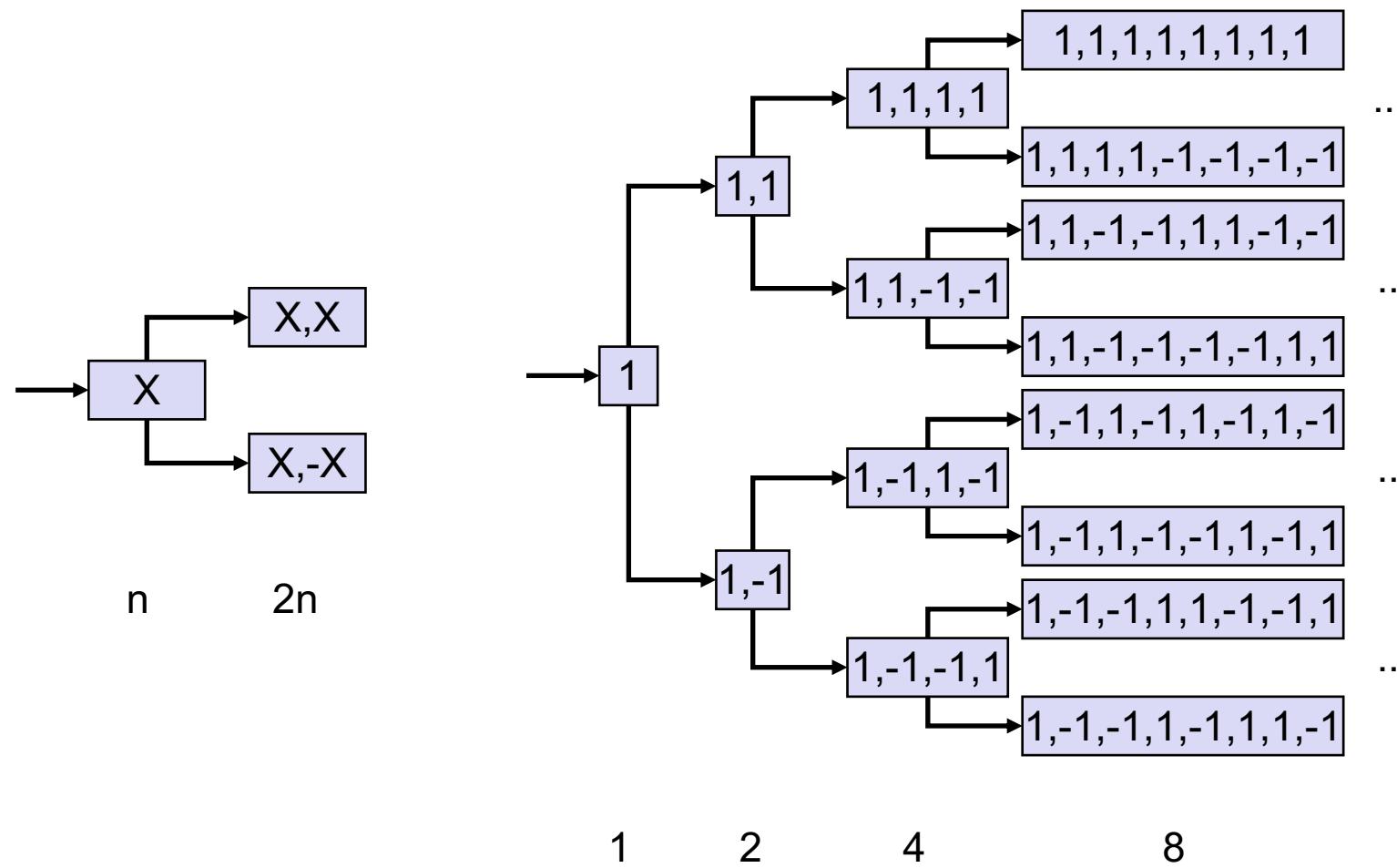
Generating Orthogonal Codes

- The most commonly used orthogonal codes in current CDMA implementation are the Walsh Codes

$$W_0 = (1)$$

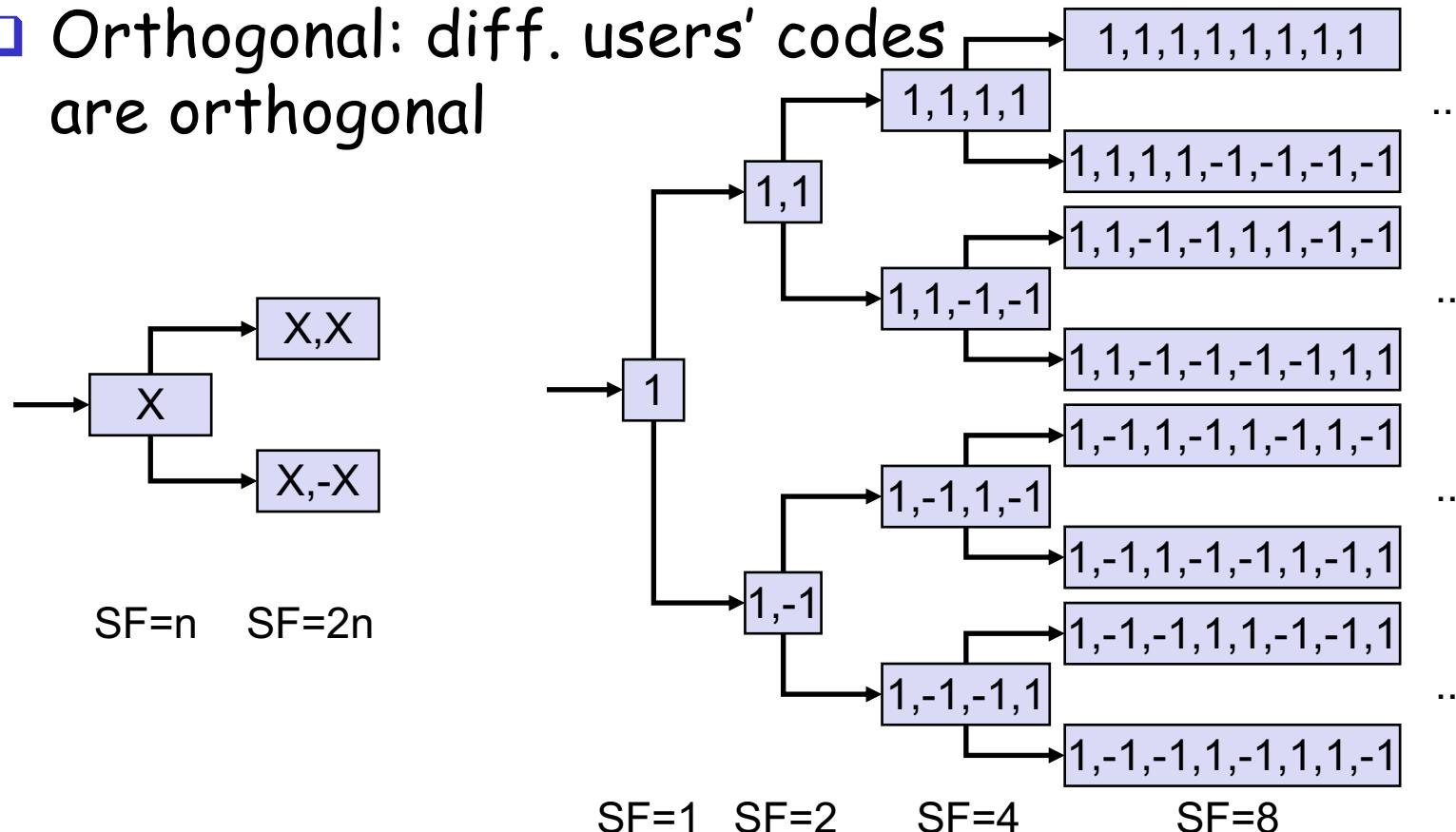
$$W_{2n} = \begin{pmatrix} W_n & W_n \\ W_n & \bar{W}_n \end{pmatrix}$$

Walsh Codes



Orthogonal Variable Spreading Factor (OSVF)

- Variable codes: Different users use different lengths spreading codes
- Orthogonal: diff. users' codes are orthogonal



If user 1 is given code $[1,1]$, what orthogonal codes can we give to other users?

WCDMA Orthogonal Variable Spreading Factor (OSVF)

□ Flexible code (spreading factor) allocation

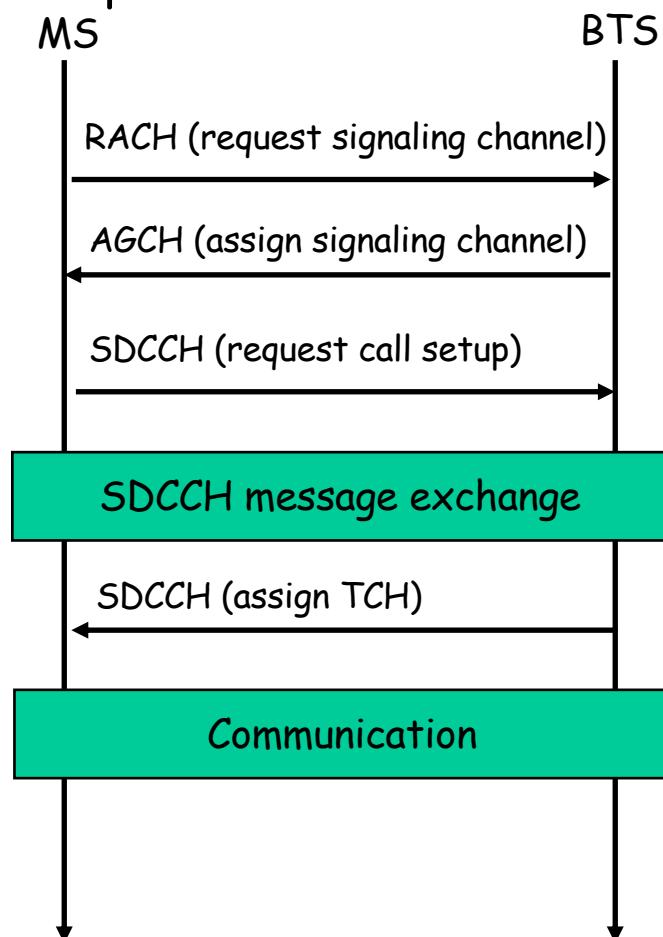
- up link SF: 4 - 256
- down link SF: 4 - 512

WCDMA downlink

Spreading factor	Channel symbol rate (kbps)	Channel bit rate (kbps)	DPDCH channel bit rate range (kbps)	Max. user data rate with $\frac{1}{2}$ rate coding (approx.)
512	7.5	15	3-6	1-3 kbps
256	15	30	12-24	6-12 kbps
128	30	60	42-51	20-24 kbps
64	60	120	90	45 kbps
32	120	240	210	105 kbps
16	240	480	432	215 kbps
8	480	960	912	456 kbps
4	960	1920	1872	936 kbps
4, with 3 parallel codes	2880	5760	5616	2.3 Mbps

Combined Random Access + Channel Partition: GSM Logical Channels and Request

- call setup from an MS



- Control channels

- Broadcast control channel (BCCH)
 - from base station, announces cell identifier, synchronization
- Common control channels (CCCH)
 - paging channel (PCH): base transceiver station (BTS) pages a mobile host (MS)
 - random access channel (RACH): MSs for initial access, **slotted Aloha**
 - access grant channel (AGCH): BTS informs an MS its allocation
- Dedicated control channels
 - standalone dedicated control channel (SDCCH): signaling and short message between MS and an MS

- Traffic channels (TCH)

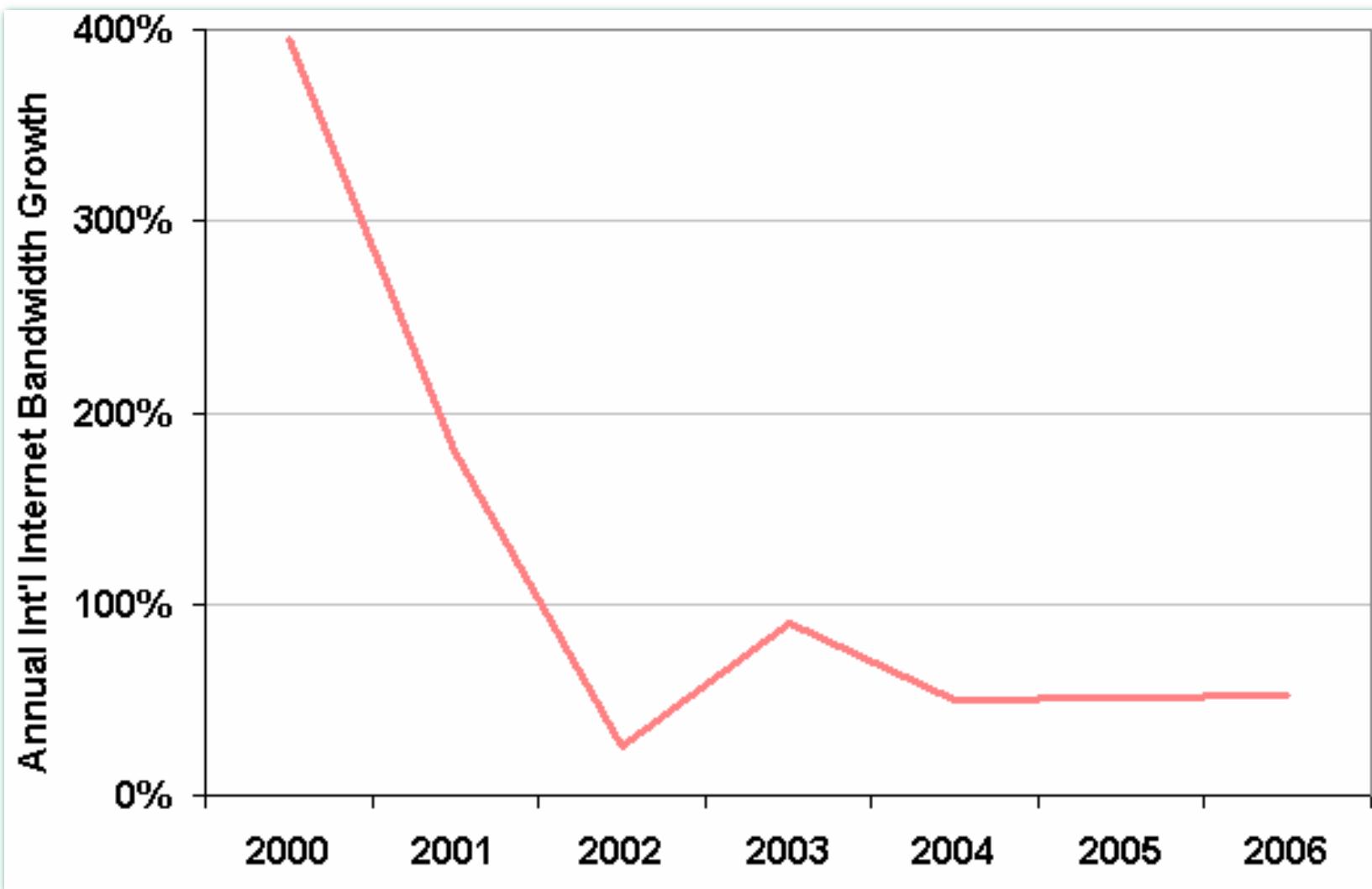
Discussions

- Advantages of channel partitioning over random access

- Advantages of random access over channel partitioning

Backup Slides: Physical Layer

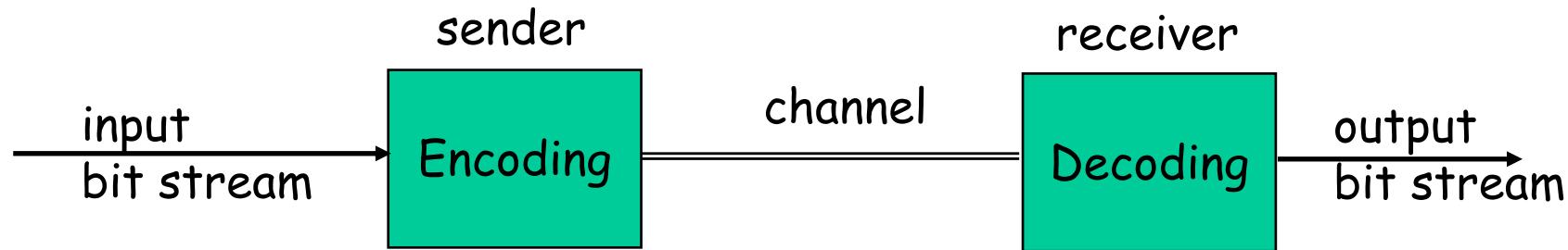
Internet Bandwidth Growth



Source: TeleGeograph Research

What Determines Transmission Rate?

- Service: transmit a bit stream from a sender to a receiver



Question to be addressed: how much can we send through the channel ?

Basic Theory: Channel Capacity

- The maximum number of bits that can be transmitted per second (bps) by a physical media is:

$$W \log_2 \left(1 + \frac{S}{N} \right)$$

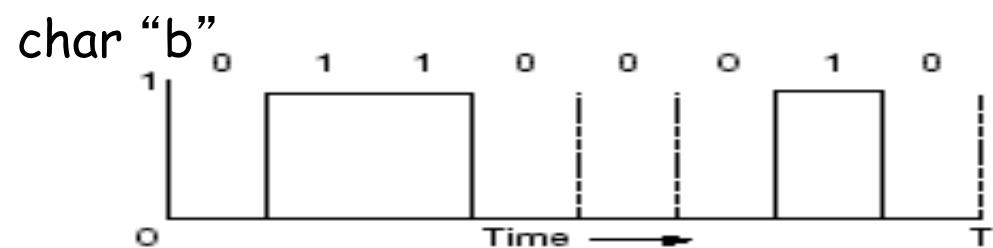
where W is the frequency range, S/N is the signal noise ratio. We assume Gaussian noise.

Fourier Transform

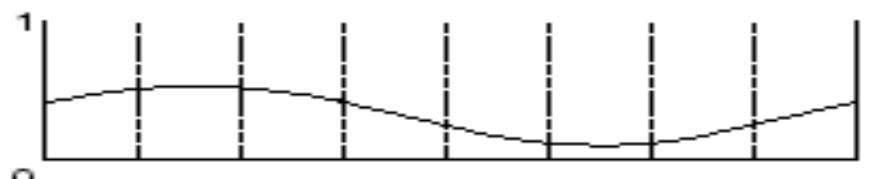
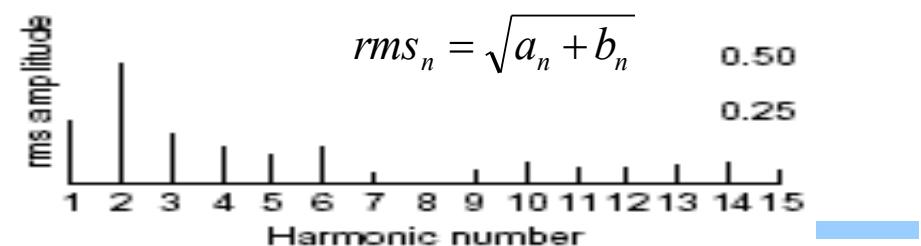
- Suppose the period of a data unit is $f (=1/T)$, then the data unit can be represented as the sum of many harmonics ($\sin()$, $\cos()$) with frequencies $f, 2f, 3f, 4f, \dots$
- A reasonably behaved periodic function $g(t)$, with minimal period T , can be constructed as the sum of a series of sines and cosines:

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft)$$

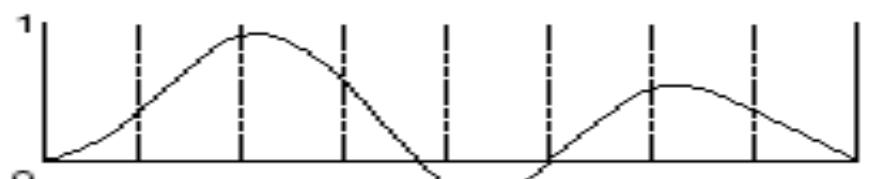
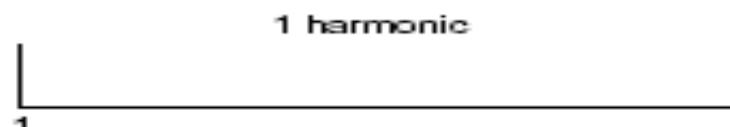
$$\begin{cases} f = 1/T \\ c = \frac{2}{T} \int_0^T g(t)dt \\ a_n = \frac{2}{T} \int_0^T g(t) \sin(2\pi nft) dt \\ b_n = \frac{2}{T} \int_0^T g(t) \cos(2\pi nft) dt \end{cases}$$



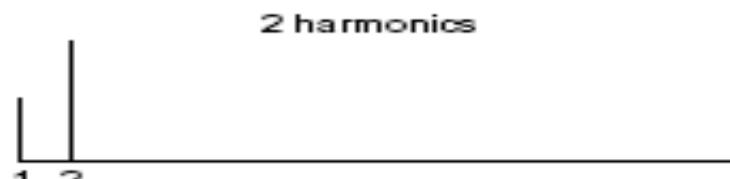
(a)



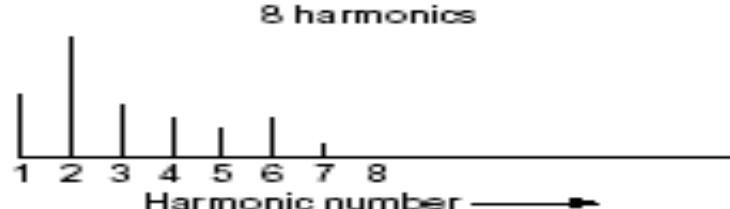
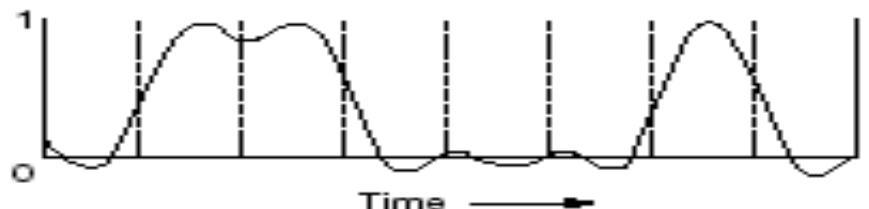
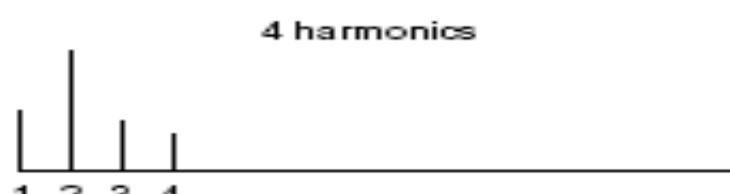
(b)



(c)



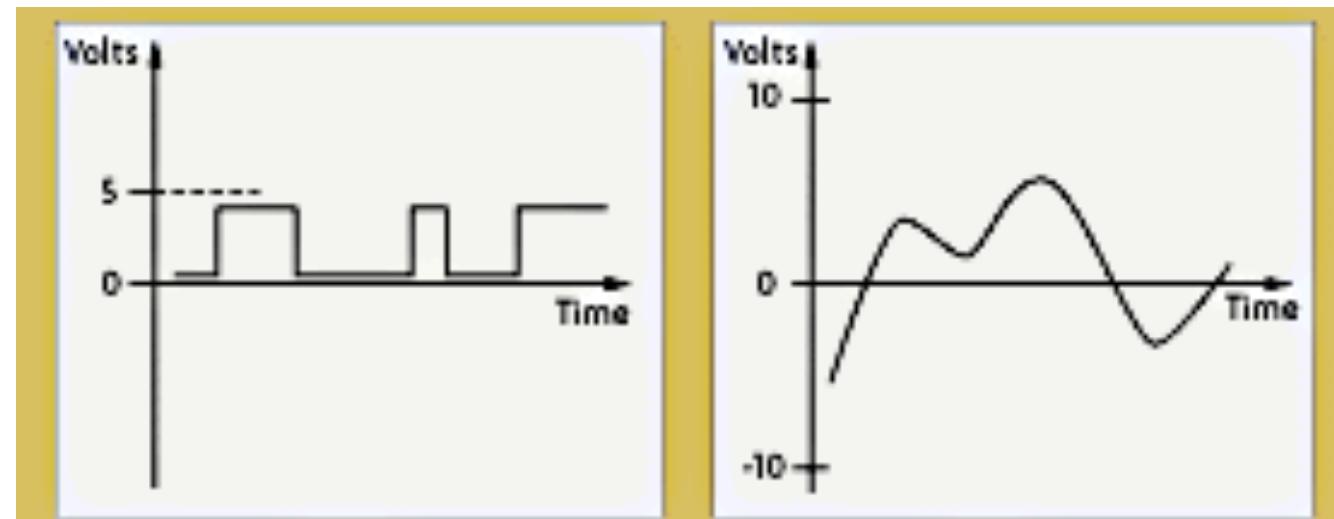
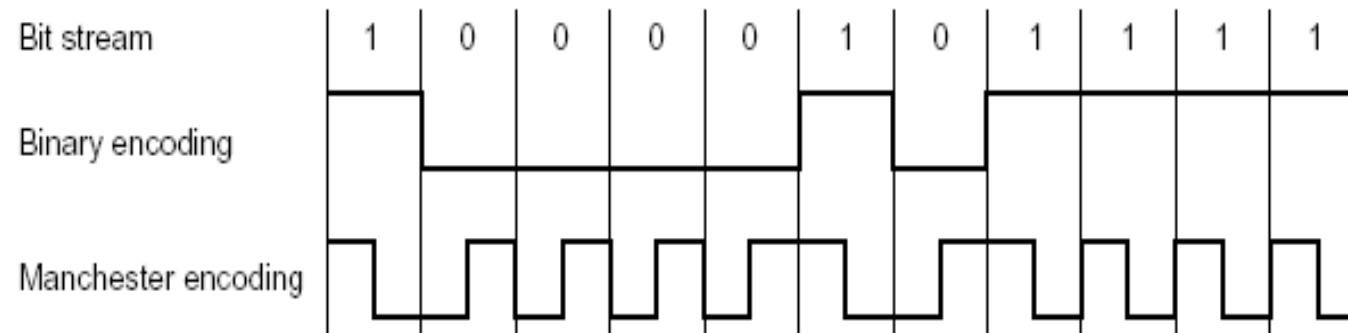
(d)



Signal Attenuation

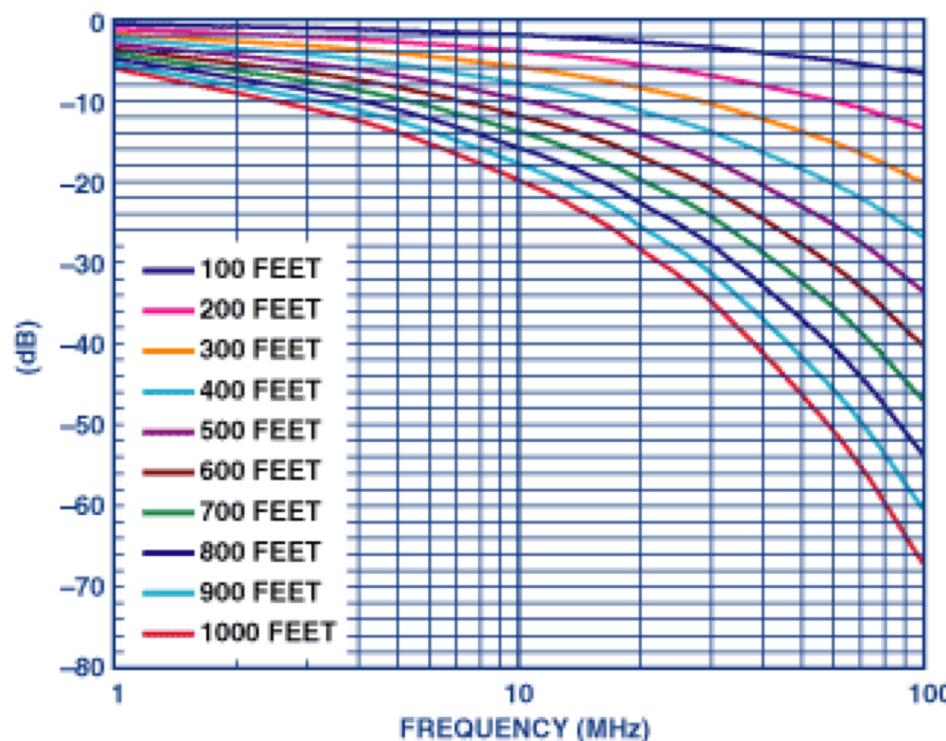
$$W \log_2 \left(1 + \frac{S}{N}\right)$$

- The quality of signal will degrade when it travels
 - loss, frequency passing



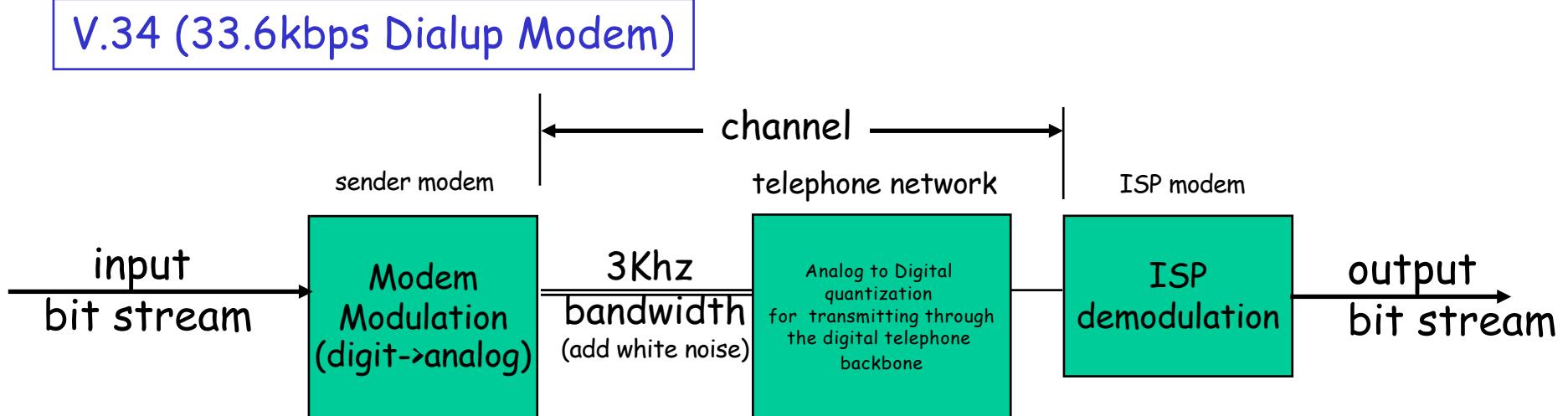
Frequency Dependent Attenuation

- The received signal will be distorted even when there is no interference and the transmitted signal is “perfect” square waveform



Example: Voltage-attenuation magnitude ratios of Category 5 cable. For example, 500 feet of cable attenuates a 10-MHz, 1-V signal to 0.32 V, which corresponds to about -9.90 dB (= $20 \log 1/0.32$)

Example



Example: $W=3000\text{Hz}$, $S/N \approx 4000$

$$\text{max bandwidth} = 3000 \log_2(1 + 4000) \approx 36\text{kbps}$$

Example: ADSL

- Spectrum allocation: divided into a total of 256 downstream and 32 upstream tones, where each tone is a standard 4kHz voice channel
- During initial negotiation, a tone is used only if the S/N is above 6 db (≈ 4)

$$down = 256 * 4000 \log_2(1 + 4) \approx 2.4 Mbps$$

$$up = 32 * 4000 \log_2(1 + 4) \approx 297 kbps$$

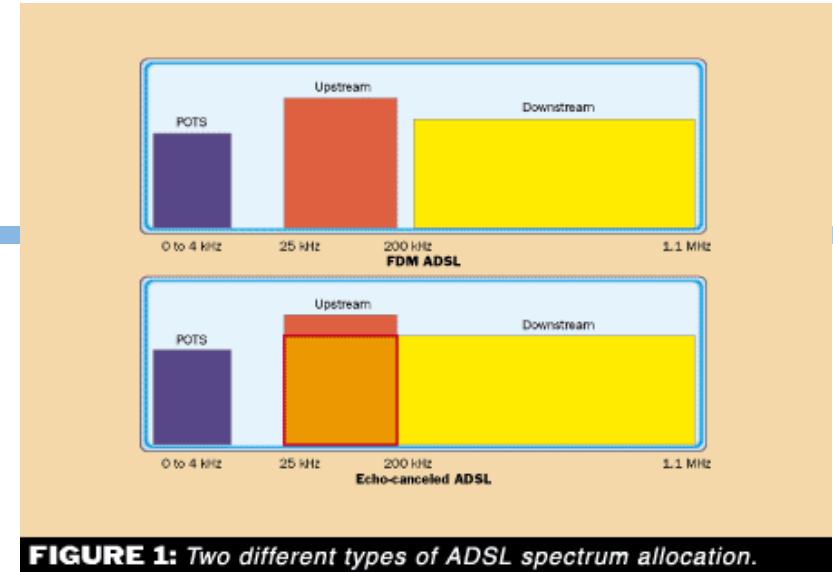
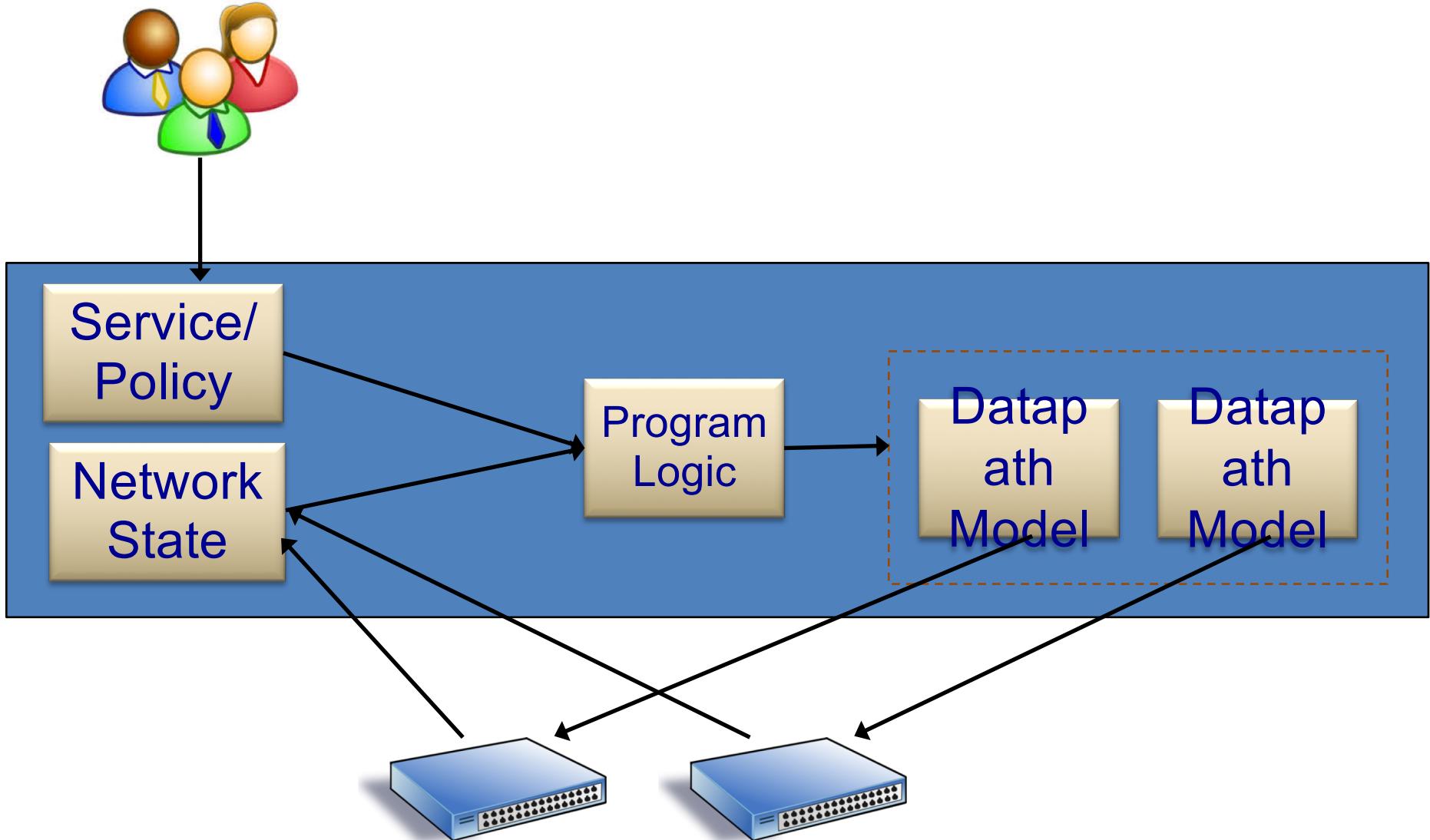


FIGURE 1: Two different types of ADSL spectrum allocation.

Backup Slides: SDN Control Programming

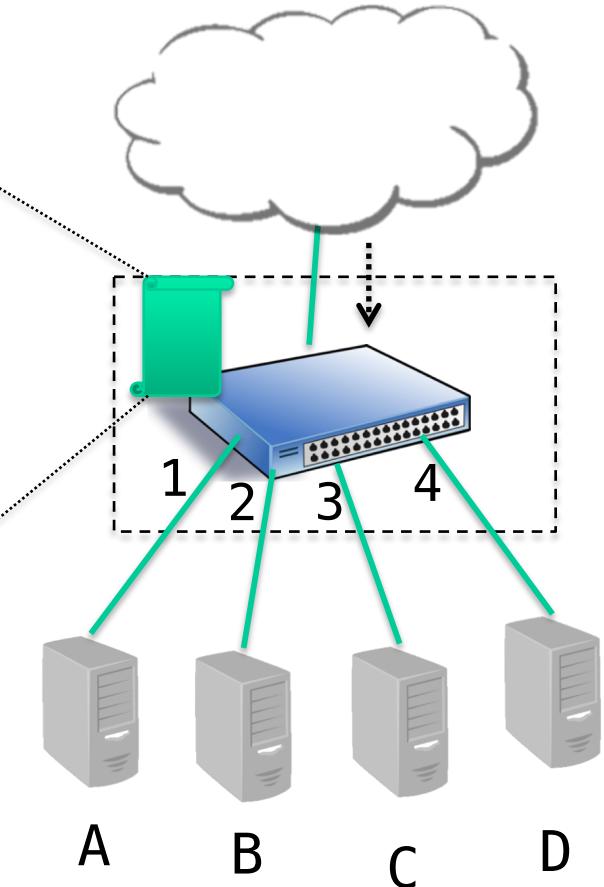
SDN Programming Model



An Example: A Simple SDN Controller

```
badPort = 22          // policy
hostTbl = {A:1,B:2,
           C:3,D:4} // net view

def onPacketIn(p):
    if badPort == p.tcp_dst:
        drop
    else:
        forward([hostTbl(p.eth_dst)])
```



Assume only packets to A,B,C,D can appear.

Controller Program with Flow Table Management

```
hostTbl = {A:1,B:2,C:3,D:4}

def onPacketIn(p):
    if 22 == p.tcp_dst:
        drop

    installRule({'match':{'tcp_dst':22},
                 'action':[]})

    else:
        forward( [hostTbl(p.eth_dst)] )

    installRule({'match': {'eth_dst':p.eth_dst,
                          'tcp_dst':!=22},
                 'action':[hostTbl(p.eth_dst)]})
```

match does not support logic negation

Controller Program with Flow Table Management

```
hostTbl = {A:1,B:2,C:3,D:4}

def onPacketIn(p):
    if 22 == p.tcp_dst:
        drop

    installRule({'priority':1,
                'match':{'tcp_dst':22},
                'action':[]})

    else:
        forward( [hostTbl(p.eth_dst)] )

    installRule({'priority':0,
                'match': {'eth_dst':p.eth_dst},
                'action':[hostTbl(p.eth_dst)]})
```

Does the Program Work?

Switch

```
{`priority`:0,'match':{'eth_dst':A),'action':[1]}
```

EthDst:A,
TcpDst:80

A security bug!

EthDst:A,
TcpDst:22

Controller

```
def onPacketIn(p):
    if 22 == p.tcp_dst:
        drop
        installRule({`priority`:1,'match':{'tcp_dst':22}, 'action':[]})
    else:
        installRule({`priority`:0,'match':{'eth_dst':p.eth_dst},
                    'action': [hostTbl(p.eth_dst)]})
        forward( [hostTbl(p.eth_dst)])
```

OVS Approach: Using Exact Match

```
hostTbl = {A:1,B:2,C:3,D:4}

def onPacketIn(p):
    if 22 == p.tcp_dst:
        drop

    installRule({'priority':1,
                'match':exactMatch(p),
                'action':[]})

    else:
        forward( [hostTbl(p.eth_dst)] )

    installRule({'priority':0,
                'match':exactMatch(p),
                'action':[hostTbl(p.eth_dst)]})
```

Problems of OVS Approach

Match											
Priority	inport	MAC src	MAC dst	MAC proto	IP src	IP dst	TCP src	TCP dst	...	Action	
0	1	B	A	0x0806	12...	78...	...	22		drop	
0	1	B	A	0x0806	12...	78...	9881	80		port 1	
0	1	B	A	0x0806	12...	78...	...	7231		port 1	
0	2	C	A	0x0806	32...	78...	...	7232		port 1	
0	1	A	B	0x0806	78...	12...	80	9881		port 2	

- Each new TCP flow delayed for flow setup (10-100 ms)
- Number of flow table rules may exceed capacity (typically a few thousands)

Outline

- Admin and recap
- Link layer
- Software defined networking
 - Motivation: from simple forwarding to network functions
 - Design overview
 - Data path (OpenFlow)
 - Control path (programming model)
 - Problems
 - Algorithmic network programming

Goal of SDN Programming

Let programmers write the **most obvious** (\Rightarrow **data path independent**) code:

```
def onPacketIn(p)
    if 22 == p.tcp_dst:
        drop
    else:
        forward( [hostTbl(p.eth_dst)] )
```

Design a system that can **automatically** generate **correct, highly-optimized** data path.

Algorithmic SDN Programming Abstraction:

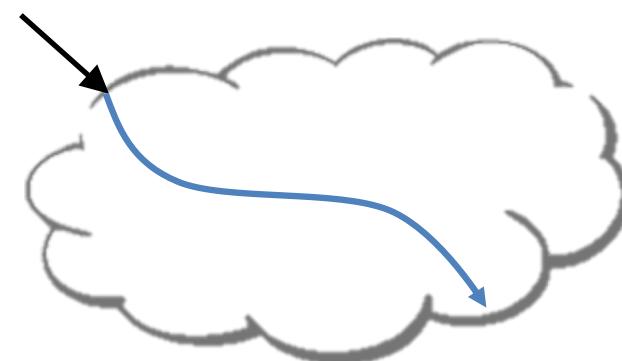
Programmer's View

r **Conceptually** programmer's network control

function f is **invoked on every packet**
entering the network.

r f expressed in an **existing**, general purpose
language (e.g., Java, Python), describing how
a packet should be routed, **not** how data path
flow tables are configured

f : packet \rightarrow route



Example Algorithmic Policy in Java

```
Route f(Packet p) {
    if (p.tcpDstIs(22)) return null();
    else {
        Location sloc = hostTable(p.ethSrc());
        Location dloc = hostTable(p.ethDst());
        Route path      = myRoutingAlg(topology(),
                                         sloc,dloc);
        return path;
    }
}
Route myRoutingAlg(Topology topo,
                   Location sLoc, Location dLoc) {
    if ( isSensitive(sLoc) || isSensitive(dLoc) )
        return secureRoutingAlg(topo, sloc, dloc);
    else
        return standardRoutingAlg(topo, sloc, dloc);
}
```

Does not specify anything on flow tables!

More Complex Example:

ACL+Routing

```
void insert(ACLItem acl, int pos,  
           List<ACLItem> acls) {  
    acls.add(pos, acl);  
}  
  
void delete(int pos,  
            List<ACLItem> acls) {  
    acls.remove(pos);  
}
```

```
boolean permit(Packet p,  
               List<ACLItem> acls) {  
    for (ACLItem item : acls) {  
        if (match(p, item)) {  
            return item.action == PERMIT;  
        }  
    }  
    return false;  
}
```

```
List<ACLItem> acls;  
Route f(Packet p) {  
    if (!permit(p, acls)) {  
        return null();  
    }  
    Location sloc = hostTable(p.ethSrc());  
    Location dloc = hostTable(p.ethDst());  
    Route path = myRoutingAlg(topology(), sloc, dloc);  
    return path;  
}
```

Challenge and Basic Idea

- r Challenge: How to derive datapath (flow-tables) from a datapath-oblivious SDN controller function f ?
- r Basic idea of handling the challenge w/o a compiler:
 - m There are two representations of computation
 - A sequence of instructions (whitebox)
 - Memorization tables (blackbox)
 - m Although the decision function f does **not** specify how flow tables are configured, if for a given decision (e.g., drop), one know the dependency of the decision, one can construct the flow tables (aka, memorization tables).

Basic Idea

- r Only requirement: Program f uses a simple library to access pkt

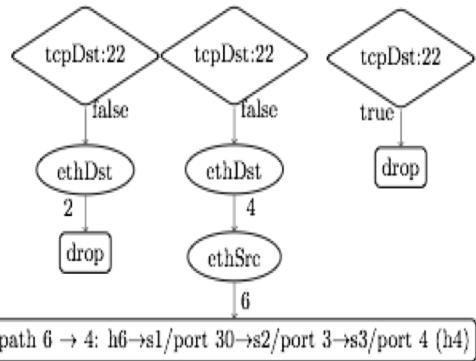
attr ..

```
readPacketField :: Field -> Value
testEqual      :: (Field,Value) -> Bool
ipSrcInPrefix  :: IPPrefix -> Bool
ipDstInPrefix  :: IPPrefix -> Bool
```

- r Library provides both **convenience** and more importantly, **decision dependency!**

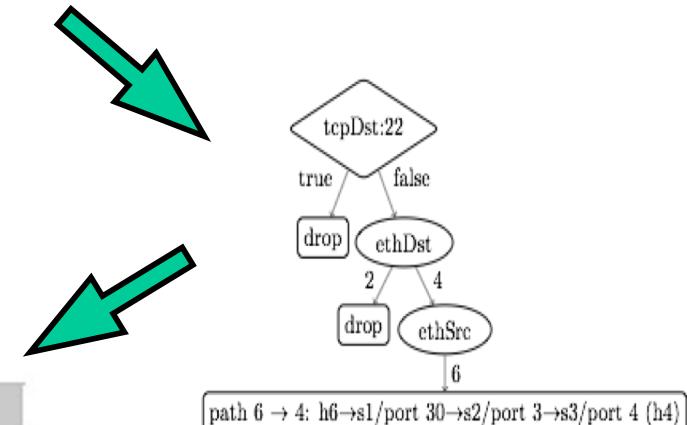
Dynamic Tracing: Abstraction to Flow Tables

1. Observes decision dependency of f on pkt attributes.



Prio	Match	Action
1	tcpDst:22	ToControl
Prio	Match	Action
0	ethDst:2	discard
1	tcpDst:22	ToControl
0	ethDst:2	discard
0	ethDst:4,	port 30
	0	ethDst:4,

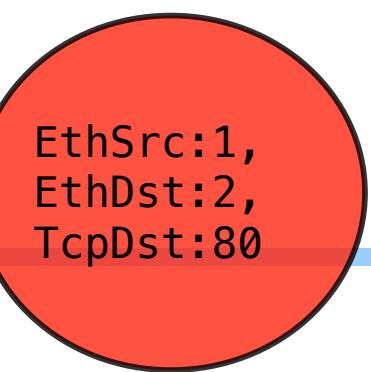
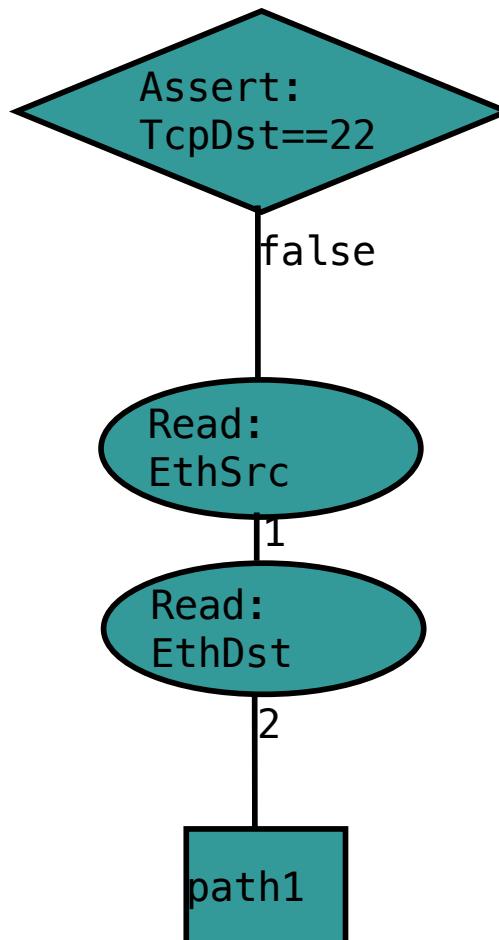
3. Compile trace tree to generate flow tables (FTs).



2. Builds a **trace tree (TT)**, a **universal** (general), partial decision tree representation of any f.

Policy

```
Route f(Packet p) {  
    if (p.tcpDstIs(22))  
        return null();  
  
    else {  
  
        Location sloc =  
            hostTable(p.ethSrc());  
  
        Location dloc =  
            hostTable(p.ethDst());  
  
        Route path      =  
            myRoutingAlg(  
                topology(),sloc,dloc);  
        return path;  
    }  
}
```

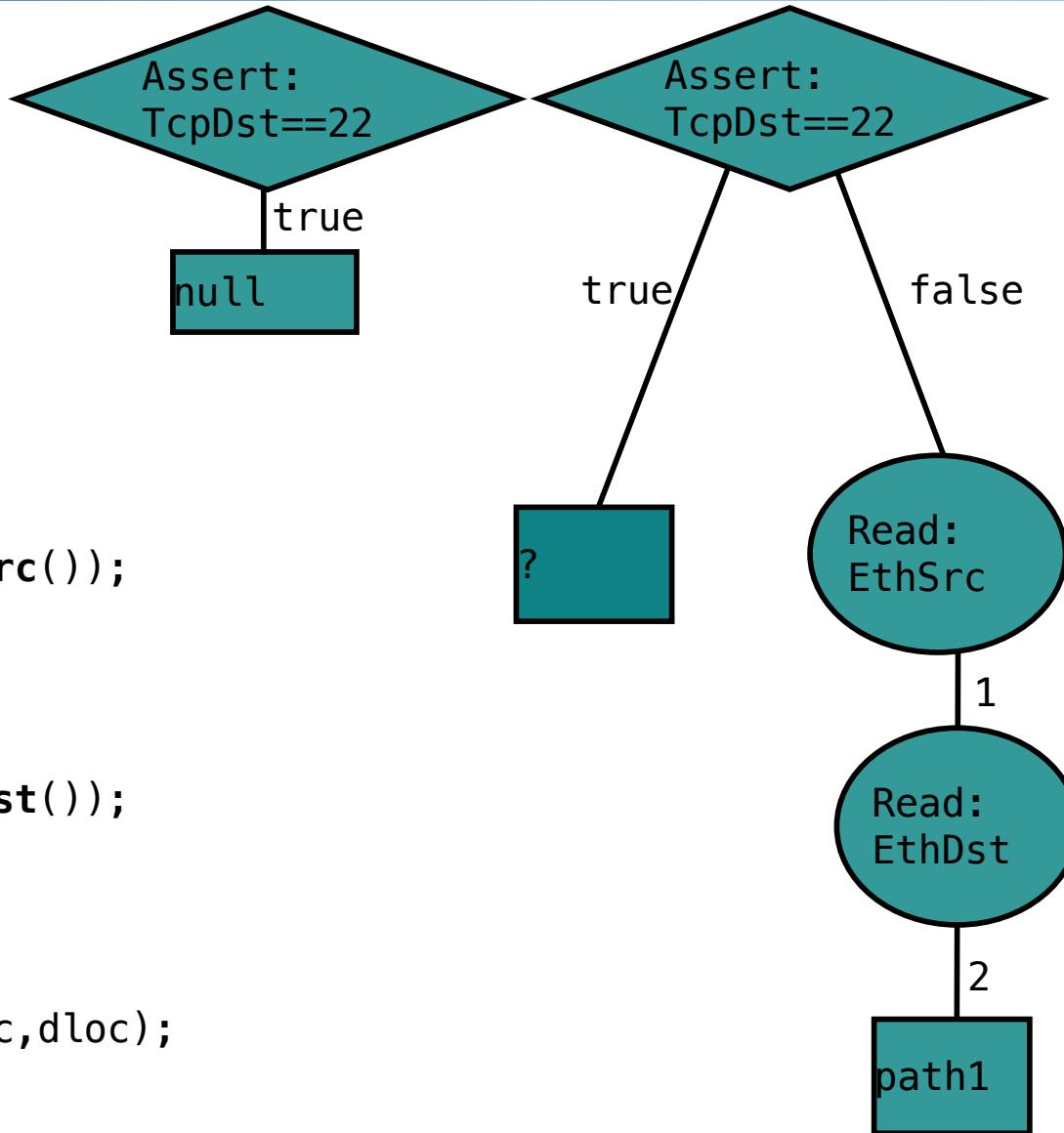


Policy

Trace Tree

EthDst:1,
TcpDst:22

```
Route f(Packet p) {  
  
    if (p.tcpDstIs(22))  
        return null();  
  
    else {  
  
        Location sloc =  
            hostTable(p.ethSrc());  
  
        Location dloc =  
            hostTable(p.ethDst());  
  
        Route path      =  
            myRoutingAlg(  
                topology(),sloc,dloc);  
        return path;  
    }  
}
```

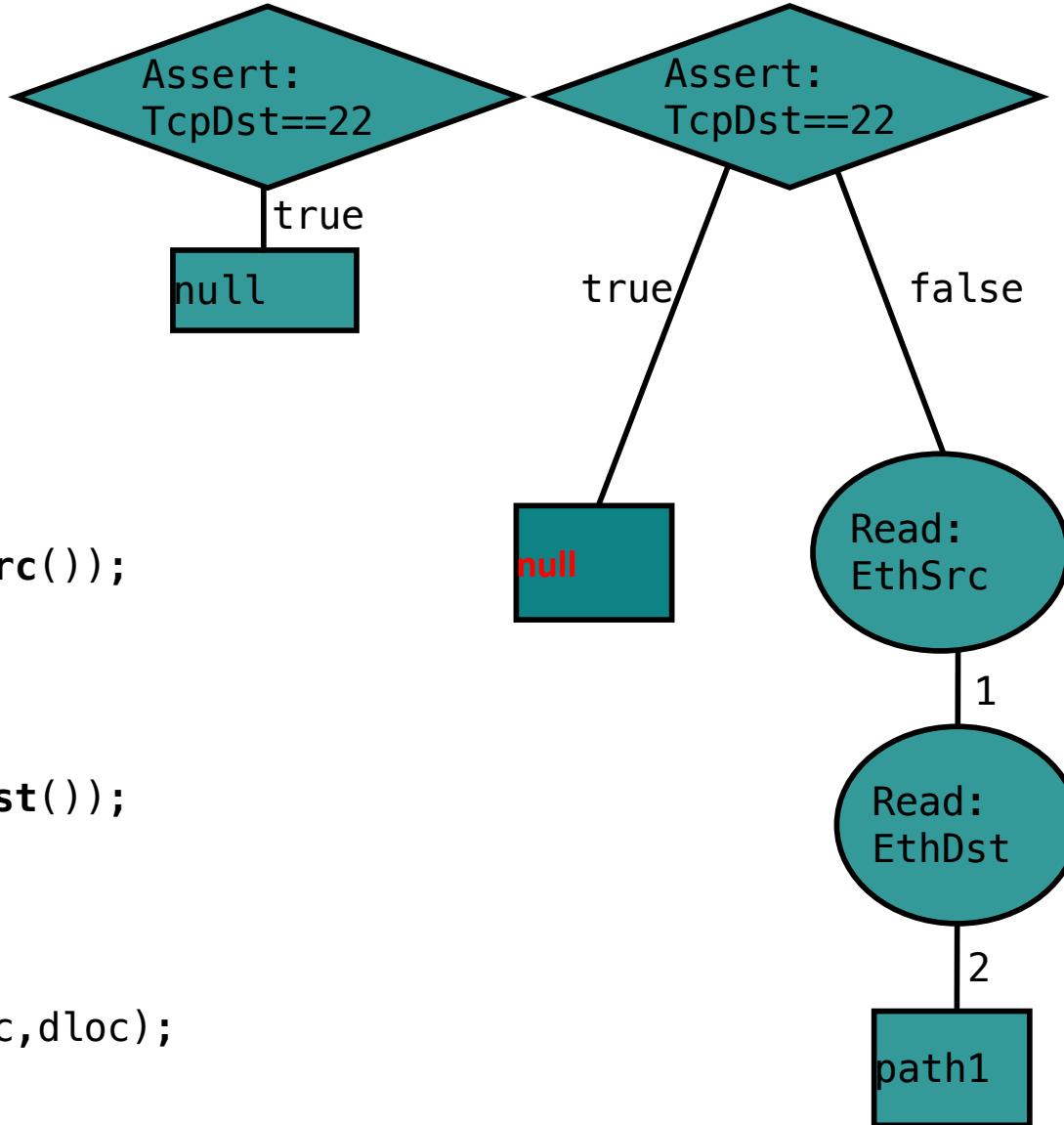


Policy

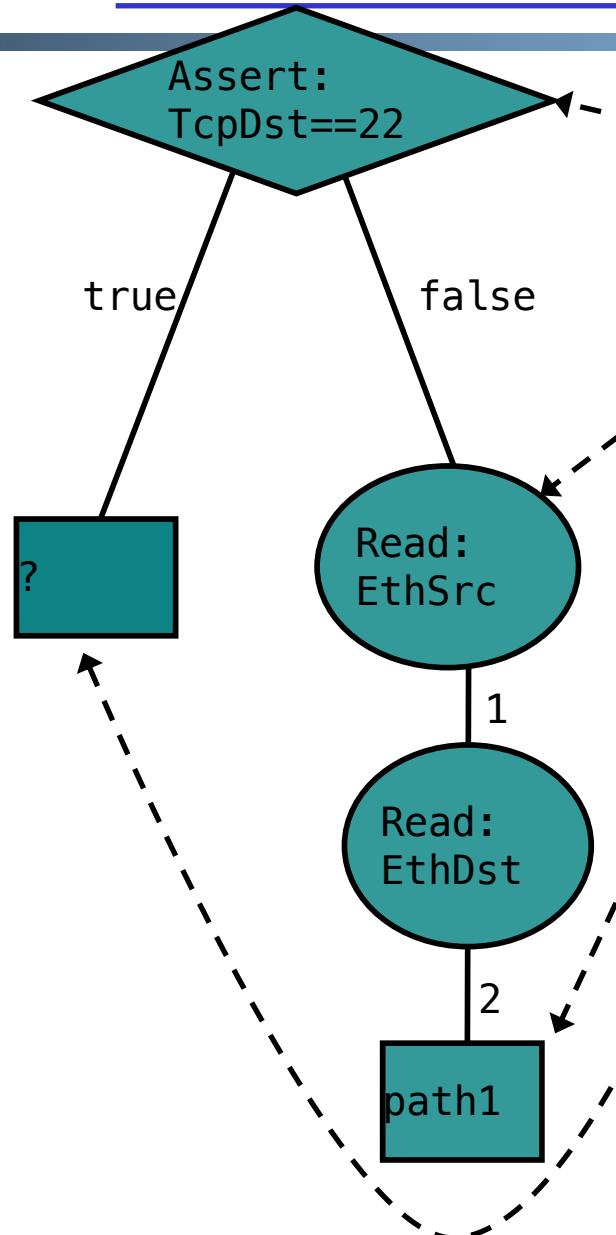
Trace Tree

EthDst:1,
TcpDst:22

```
Route f(Packet p) {  
  
    if (p.tcpDstIs(22))  
        return null();  
  
    else {  
  
        Location sloc =  
            hostTable(p.ethSrc());  
  
        Location dloc =  
            hostTable(p.ethDst());  
  
        Route path      =  
            myRoutingAlg(  
                topology(),sloc,dloc);  
        return path;  
    }  
}
```



Trace Tree Formal Spec



A tree w/ 4 types of nodes

T node: assertion on packet attributes

V node: multi-way branching on a packet attribute

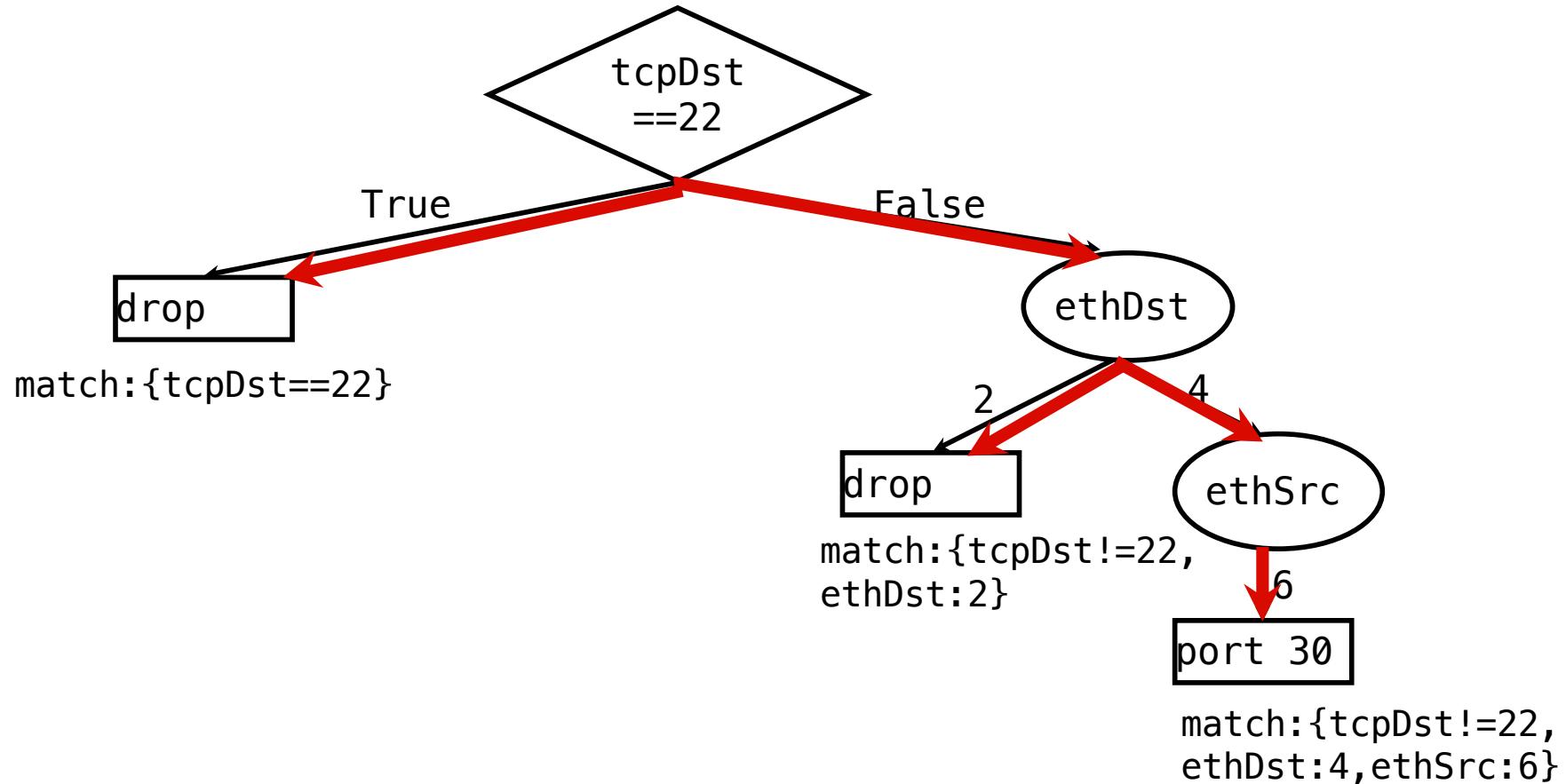
L node: leaf node labeled w/ action

? node: unknown

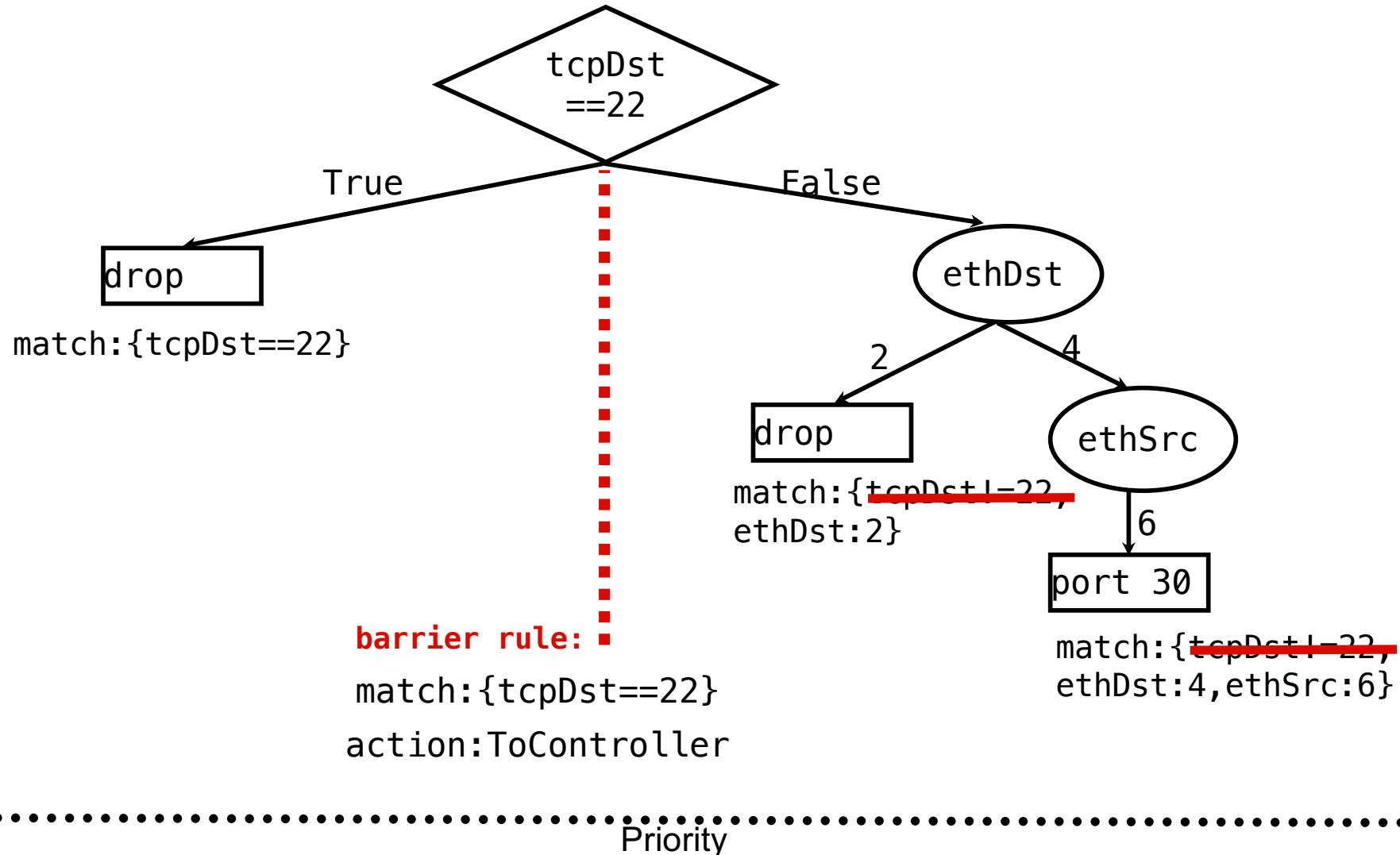
TT search: traverse tree for a given packet to determine action or no answer

TT correctness: if TT records an answer for a given packet

Trace Tree => Flow Table



Trace Tree => Flow Table



Trace Tree => Flow Table

Simple, classical in-order tree traversal generates flow table rules!

