Topic 4: Network Layer

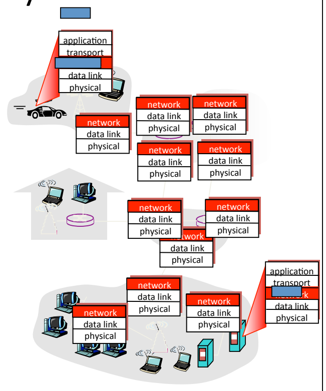
Our goals:

- understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing (versus switching)
  - how a router works
  - routing (path selection)
  - IPv6
- For the most part, the Internet is our example

2

Network layer

- transport segment from sending to receiving host
- on sender side encapsulates segments into datagrams
- on receiver side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



3

Name: a *something*

Address: Where a *something* is

Routing: How do I get to the *something*

4

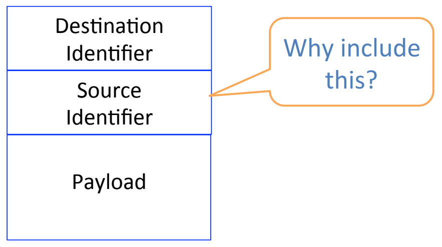
Addressing (at a conceptual level)

- Assume all hosts have unique IDs
- No particular structure to those IDs
- Later in topic I will talk about real IP addressing
- Do I route on location or identifier?
- If a host moves, should its address change?
  - If not, how can you build scalable Internet?
  - If so, then what good is an address for identification?

5

Packets (at a conceptual level)

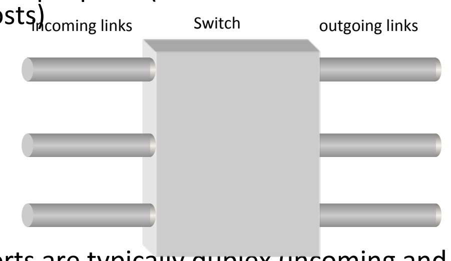
- Assume packet headers contain:
  - Source ID, Destination ID, and perhaps other information



6

Switches/Routers

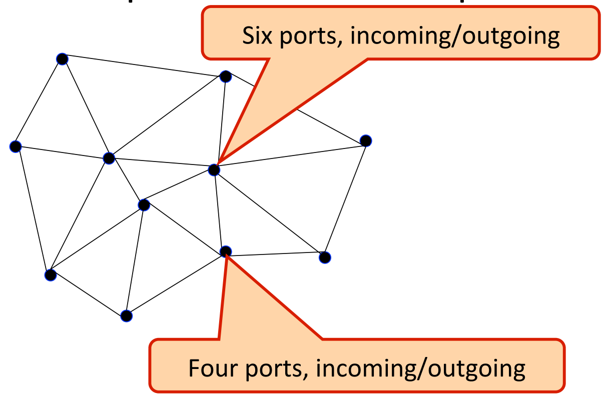
- Multiple ports (attached to other switches or hosts)



- Ports are typically duplex (incoming and outgoing)

7

Example of Network Graph



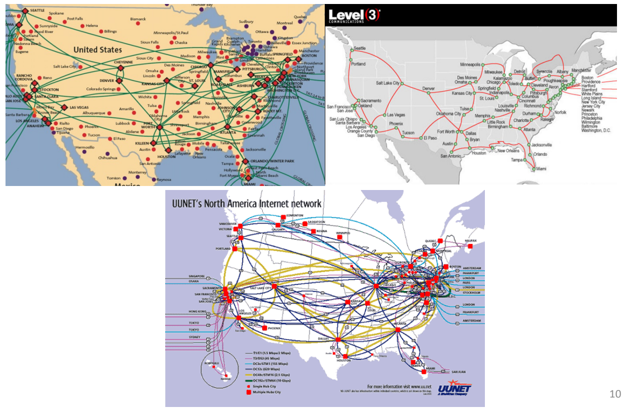
8

A Variety of Networks

- ISPs: carriers
  - Backbone
  - Edge
  - Border (to other ISPs)
- Enterprises: companies, universities
  - Core
  - Edge
  - Border (to outside)
- Datacenters: massive collections of machines
  - Top-of-Rack
  - Aggregation and Core
  - Border (to outside)

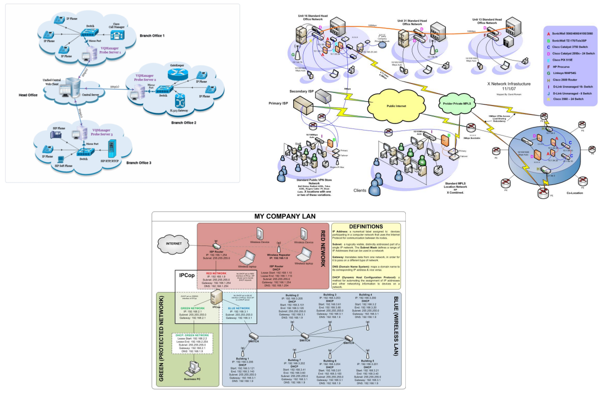
9

ISP networks



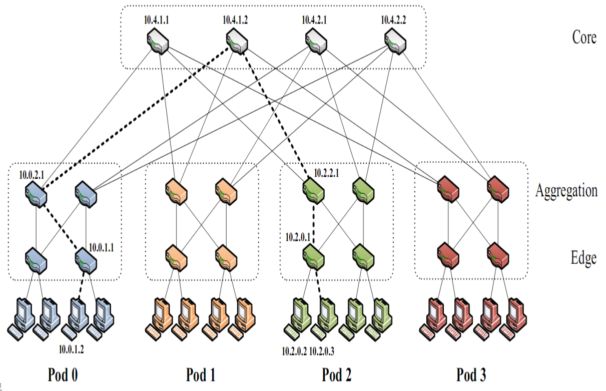
10

Enterprise Network



11

Partial Datacenter Network



12

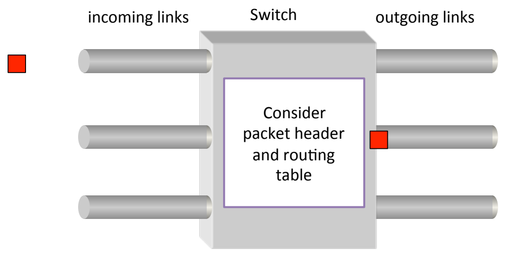
Switches

- Enterprise/Edge: typically 24 to 48 ports
- Aggregation switches: 192 ports or more
- Backbone: typically fewer ports
- Border: typically very few ports

13

Forwarding Decisions

- When packet arrives, must choose outgoing port



- Decision is based on routing state (table) in switch

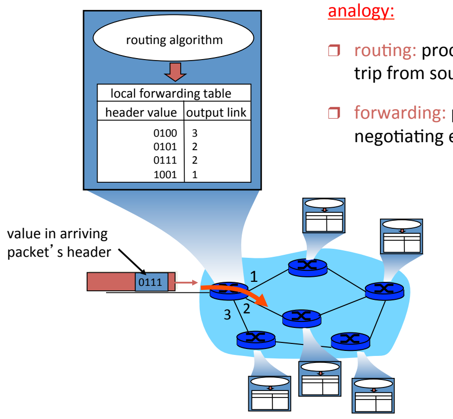
Forwarding Decisions

- When packet arrives..
  - Must decide which outgoing port to use
  - In single transmission time
  - Forwarding decisions must be ***simple***
- Routing state dictates where to forward packets
  - Assume decisions are **deterministic**
- *Global routing state* means collection of routing state in each of the routers
  - Will focus on where this routing state comes from
  - But first, a few preliminaries….

Forwarding vs Routing

- Forwarding: "data plane"
  - Directing a data packet to an outgoing link
  - Individual router using routing state
- Routing: "control plane"
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Jointly creating the routing state
- Two very different timescales….

Interplay between routing and forwarding



local forwarding table	
header value	output link
0100	3
0101	2
0111	2
1001	1

analogy:

- routing: process of planning trip from source to dest
- forwarding: process of negotiating each intersection

Connection setup

- 3rd important function in *some* network architectures:
  - ATM, frame relay, X.25, Software Defined Networks
- before datagrams flow, two end hosts *and* intervening routers establish virtual connection
  - routers get involved
- network vs transport layer connection service:
  - network: between two hosts (may also involve intervening routers in case of VCs)
  - transport: between two processes

  Remember: Ask youself "what is doing the multiplexing?"

18

Network service model

Q: What *service model* for the "channel" transporting datagrams from sender to receiver?

Example services for individual datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

Example services for a flow of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

19

Network layer service models:

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

20

Network layer connection and connection-less service

- datagram network provides network-layer connectionless service
- Virtual Circuit (VC) – a connection-orientated network – provides network-layer connection service
- analogous to the transport-layer services, but:
  - service: host-to-host
  - no choice: network provides one or the other
  - implementation: in network core

21

Virtual circuits

"source-to-dest path behaves much like telephone circuit"
- performance-wise
- network actions along source-to-dest path

- call setup, teardown for each call *before* data can flow
- each packet carries VC identifier (not destination host address)
- *every* router on source-dest path maintains "state" for each passing connection
- link, router resources (bandwidth, buffers) may be *allocated* to VC (dedicated resources = predictable service)

22

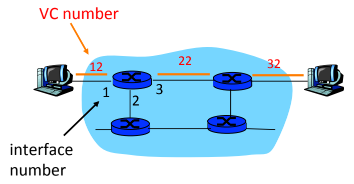
VC implementation

a VC consists of:
1. path from source to destination
2. VC numbers, one number for each link along path
3. entries in forwarding tables in routers along path

- packet belonging to VC carries VC number (rather than dest address)
- VC number can be changed on each link.
  - New VC number comes from forwarding table

23

Forwarding table



Forwarding table in northwest router:

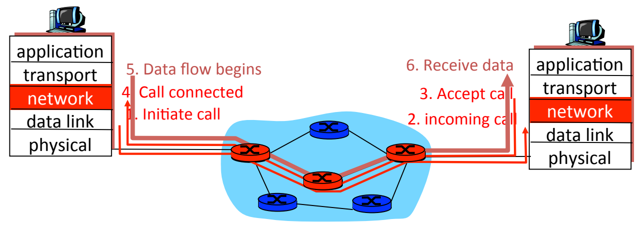
Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
…	…	…	…

Routers maintain connection state information!

24

Virtual circuits: signaling protocols

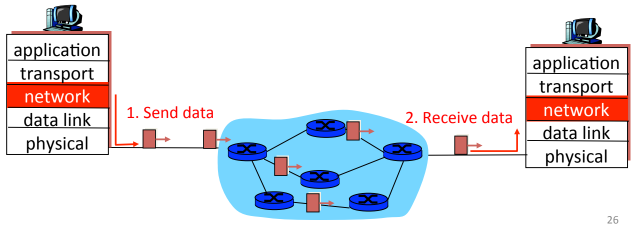
- used to setup, maintain teardown VC
- used in ATM, frame-relay, X.25
- not used in today's Internet



25

Datagram networks

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of "connection"
- packets forwarded using destination host address
  - packets between same source-dest pair may take different paths



26

Forwarding tables

IP address } 32 bits wide → ~ 4 billion unique address

Naïve approach:
One entry per address

Entry	Destination	Port
1	0.0.0.0	1
2	0.0.0.1	2
⋮	⋮	⋮
2^{32}	255.255.255.255	12

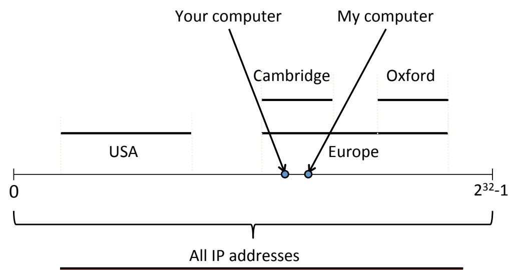
} **~ 4 billion entries**

Improved approach:
Group entries to reduce table size

Entry	Destination	Port
1	0.0.0.0 – 127.255.255.255	1
2	128.0.0.1 – 128.255.255.255	2
⋮	⋮	⋮
50	248.0.0.0 – 255.255.255.255	12

27

IP addresses as a line



Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

28

Longest Prefix Match (LPM)

Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

} Universities
} Continents
Planet

Matching entries:
- Cambridge — Most specific
- Europe
- Everywhere

To: Cambridge	Data

29

Longest Prefix Match (LPM)

Entry	Destination	Port	
1	Cambridge	1	Universities
2	Oxford	2	
3	Europe	3	Continents
4	USA	4	
5	Everywhere (default)	5	Planet

Matching entries:

- Europe Most specific
- Everywhere

To: France	Data

30

Implementing Longest Prefix Match

Entry	Destination	Port	
1	Cambridge	1	Searching
2	Oxford	2	
3	Europe	3	
4	USA	4	FOUND
5	Everywhere (default)	5	

Most specific

$\downarrow$

Least specific

31

Datagram or VC network: why?

Internet (datagram)

- data exchange among computers
  - "elastic" service, no strict timing req.
- "smart" end systems (computers)
  - can adapt, perform control, error recovery
  - simple inside network, complexity at "edge"
- many link types
  - different characteristics
  - uniform service difficult

ETHERNET

ATM (VC)

- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- "dumb" end systems
  - telephones
  - complexity inside network

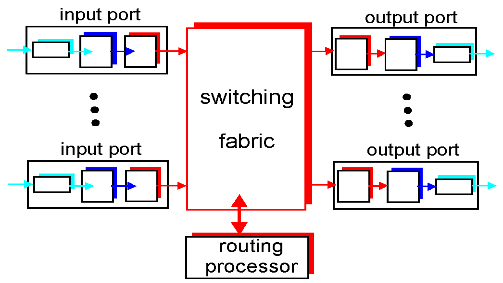
aDSL

32

Router Architecture Overview

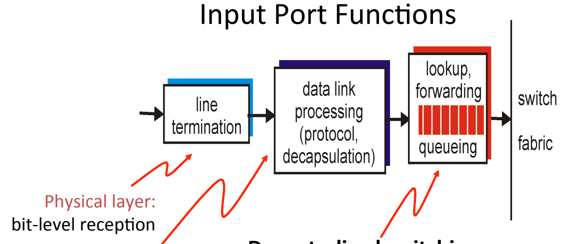
Two key router functions:

- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link



33

Input Port Functions



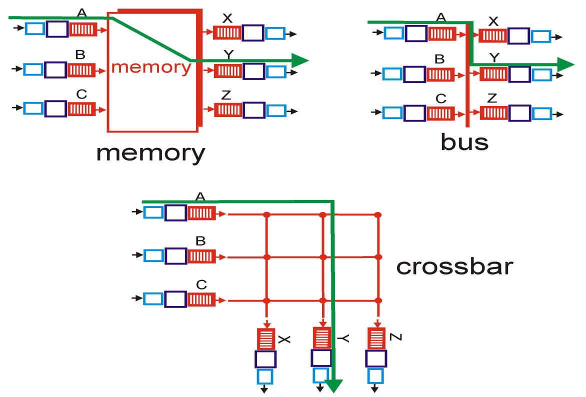
Physical layer:
bit-level reception

Data link layer:
e.g., Ethernet
see chapter 5

Decentralized switching:
- given datagram dest., lookup output port using forwarding table in input port memory
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

34

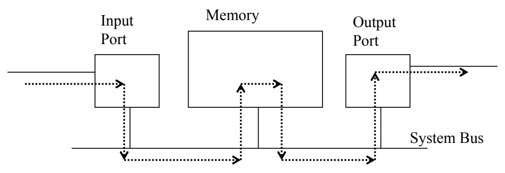
Three types of switching fabrics
(comparison criteria: speed, contention, complexity)



35

Switching Via Memory

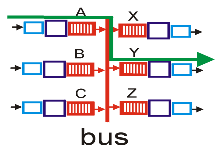
First generation routers:
- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



36

Switching Via a Bus

- datagram from input port memory to output port memory via a shared bus
- bus contention: switching speed limited by bus bandwidth
- Lots of ports?? speed up the bus
  no contention bus speed =
  2 x port speed x port count
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers

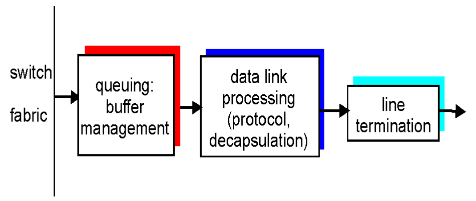


Switching Via An Interconnection Network

- overcome bus bandwidth limitations
- Banyan networks, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network

38

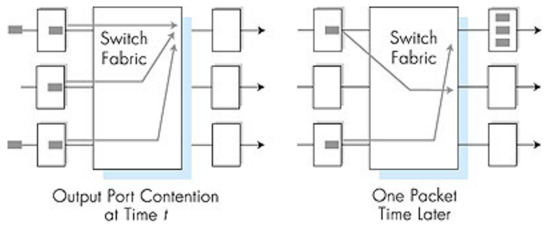
Output Ports


switch
fabric
queuing: buffer management
data link processing (protocol, decapsulation)
line termination

- *Buffering* required when datagrams arrive from fabric faster than the transmission rate
- *Scheduling discipline* chooses among queued datagrams for transmission
  → Who goes next?

39

Output port queueing

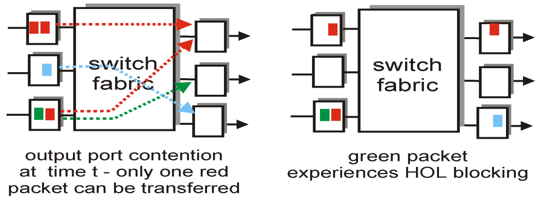


- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

40

Input Port Queuing

- Fabric slower than input ports combined -> queueing may occur at input queues
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward
- *queueing delay and loss due to input buffer overflow!*


switch fabric
output port contention at time t - only one red packet can be transferred
switch fabric
green packet experiences HOL blocking

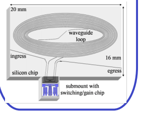
41

Buffers in Routers

- So how large should the buffers be?

Buffer size matters
- End-to-end delay
  - Transmission, propagation, and queueing de
  - The only variable part is queueing delay
- Router architecture
  - Board space, power consumption, and cos
  - On chip buffers: higher density, higher c
  - Optical buffers: all-optical routers

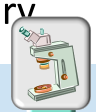
1.4m long spiral waveguide with input from HeNe laser



You are now touching the edge of the *research* zone……

42

Buffer Sizing Story



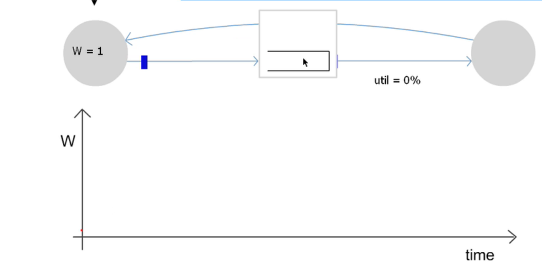
	Rule-of-thumb	Small Buffers	Tiny Buffers
	$2T \times C$	$\frac{2T \times C}{\sqrt{n}}$	$O(\log W)$
# of packets	1,000,000	10,000	20 - 50
Intuition	TCP Sawtooth	Sawtooth Smoothing	Non-bursty Arrivals
Assume	Single TCP Flow, 100% Utilization	Many Flows, 100% Utilization	Paced TCP, 85-90% Utilization
Evidence	Simulation, Emulation	Simulations, Test-bed and Real Network Experiments	Simulations, Test-bed Experiments

43

Continuous ARQ (TCP) adapting to congestion

Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2

Only W packets may be outstanding

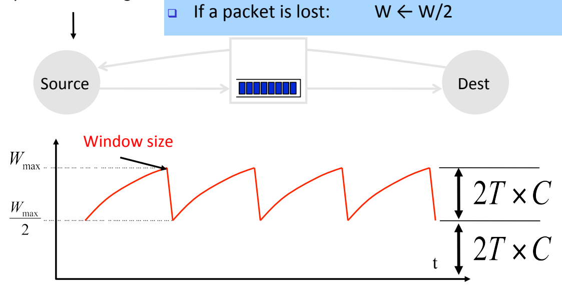


44

Rule-of-thumb – Intuition

Rule for adjusting W
- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2

Only W packets may be outstanding

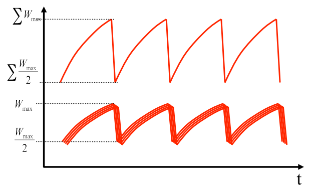


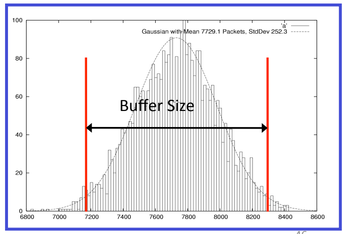
45

Small Buffers – Intuition

Synchronized Flows
- Aggregate window has same dynamics
- Therefore buffer occupancy has same dynamics
- Rule-of-thumb still holds.

Many TCP Flows
- Independent, desynchronized
- Central limit theorem says the aggregate becomes Gaussian
- Variance (buffer size) decreases as N increases

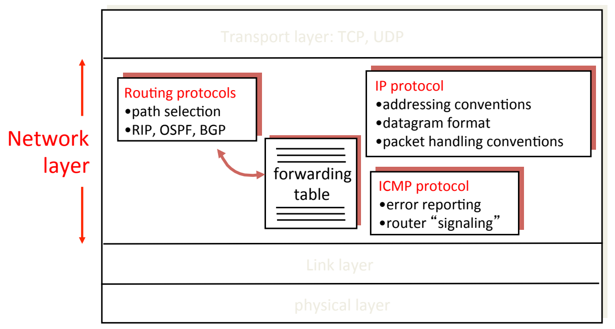




46

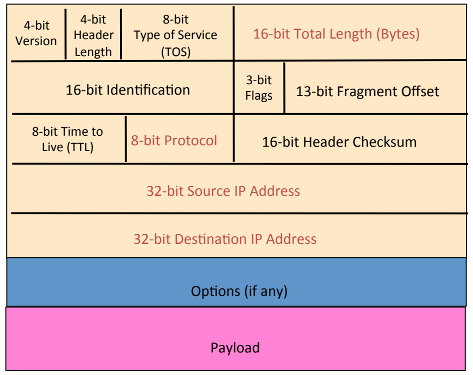
The Internet version of a Network layer

Host, router network layer functions:



47

IPv4 Packet Structure
20 Bytes of Standard Header, then Options



4-bit Version	4-bit Header Length	8-bit Type of Service (TOS)	16-bit Total Length (Bytes)	
16-bit Identification			3-bit Flags	13-bit Fragment Offset
8-bit Time to Live (TTL)	8-bit Protocol	16-bit Header Checksum		
32-bit Source IP Address				
32-bit Destination IP Address				
Options (if any)				
Payload				

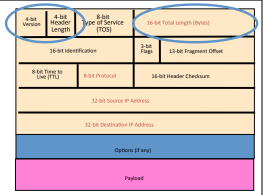
48

(Packet) Network Tasks One-by-One

- Read packet correctly
- Get packet to the destination
- Get responses to the packet back to source
- Carry data
- Tell host what to do with packet once arrived
- Specify any special network handling of the packet
- Deal with problems that arise along the path

49

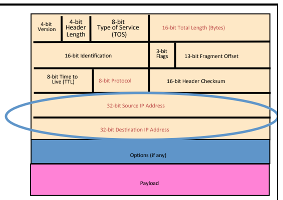
Reading Packet Correctly



- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically "4" (for IPv4), and sometimes "6" (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically "5" (for a 20-byte IPv4 header)
  - Can be more when IP options are used
- Total length (16 bits)
  - Number of bytes in the packet
  - Maximum size is 65,535 bytes (2^{16} -1)
  - … though underlying links may impose smaller limits

50

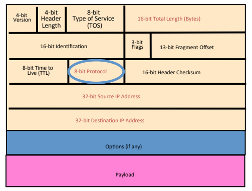
Getting Packet to Destination and Back



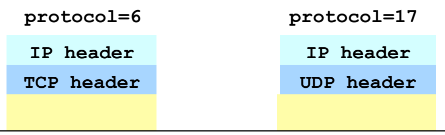
- Two IP addresses
  - Source IP address (32 bits)
  - Destination IP address (32 bits)
- Destination address
  - Unique identifier/locator for the receiving host
  - Allows each node to make forwarding decisions
- Source address
  - Unique identifier/locator for the sending host
  - Recipient can decide whether to accept packet
  - Enables recipient to send a reply back to source

51

Telling Host How to Handle Packet

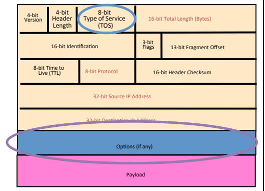


- Protocol (8 bits)
  - Identifies the higher-level protocol
  - Important for demultiplexing at receiving host
- Most common examples
  - E.g., "6" for the Transmission Control Protocol (TCP)
  - E.g., "17" for the User Datagram Protocol (UDP)



52

Special Handling



- Type-of-Service (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer
  - Has been redefined several times
- Options

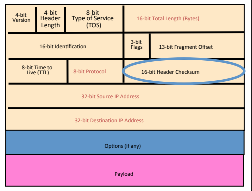
53

Potential Problems

- Header Corrupted: **Checksum**
- Loop: **TTL**
- Packet too large: **Fragmentation**

54

Header Corruption

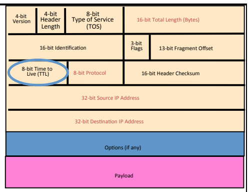


- Checksum (16 bits)
  - Particular form of checksum over packet header
- If not correct, router discards packets
  - So it doesn't act on bogus information
- Checksum recalculated at every router
  - **Why?**
  - **Why include TTL?**
  - **Why only header?**

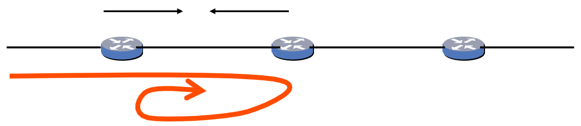
55

Preventing Loops

(aka Internet Zombie plan)



- Forwarding loops cause packets to cycle forever
  - As these accumulate, eventually consume **all** capacity

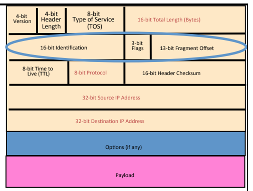


- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0
  - …and "time exceeded" message is sent to the source
    - Using "ICMP" control message; basis for **traceroute**

56

Fragmentation

(some assembly required)

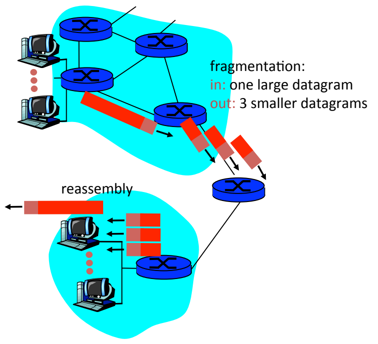


- Fragmentation: when forwarding a packet, an Internet router can split it into multiple pieces ("fragments") if too big for next hop link
- Must reassemble to recover original packet
  - Need fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset

57

IP Fragmentation & Reassembly

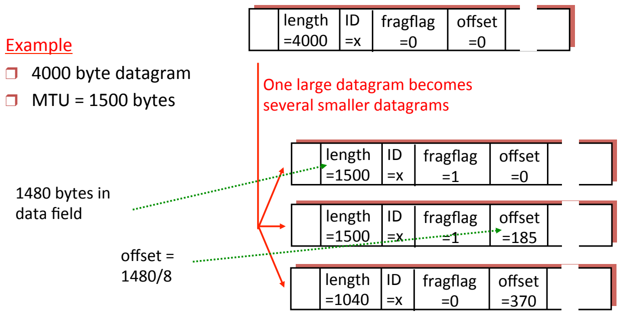
- network links have MTU (max.transfer size) - largest possible link-level frame.
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related fragments
- IPv6 does things differently



4-58

IP Fragmentation and Reassembly

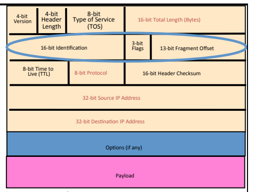
Example
- 4000 byte datagram
- MTU = 1500 bytes



Pop quiz question: What happens when a fragment is lost?

4-59

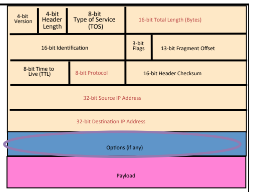
Fragmentation Details



- Identifier (16 bits): used to tell which fragments belong together
- Flags (3 bits):
  - Reserved **(RF):** unused bit
  - Don't Fragment **(DF):** instruct routers to **not** fragment the packet even if it won't fit
    - Instead, they **drop** the packet and send back a "Too Large" ICMP control message
    - Forms the basis for "Path MTU Discovery"
  - More **(MF)**: this fragment is not the last one
- Offset (13 bits): what part of datagram this fragment covers in 8-byte units

60 Pop quiz question: Why do frags use offset and not a frag number?

Options

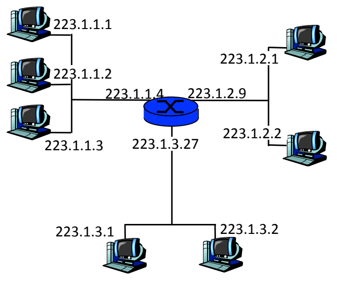


- End of Options List
- No Operation (padding between options)
- Record Route
- Strict Source Route
- Loose Source Route
- Timestamp
- Traceroute
- Router Alert
- .....

61

IP Addressing: introduction

- IP address: 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one interface
  - IP addresses associated with each interface

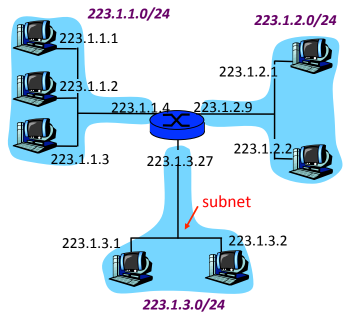


$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

62

Subnets

- IP address:
  - subnet part (high order bits)
  - host part (low order bits)
- *What's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other without intervening router



network consisting of 3 subnets

$$\overleftrightarrow{\text{subnet part}}\ \overleftrightarrow{\text{host part}}$$

11011111 00000001 00000011 00000000

223.1.3.0/24

Subnet mask: /24

CIDR: Classless InterDomain Routing
- subnet portion of address of arbitrary length
- address format: a.b.c.d/x, where x is # bits in subnet portion of address

63

IP addresses: how to get one?

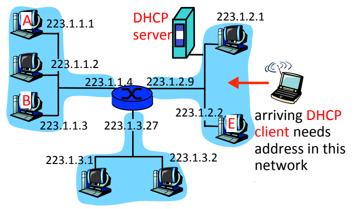
Q: How does a *host* get IP address?

- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

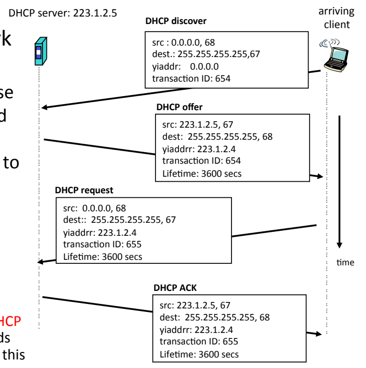
64

DHCP client-server scenario

Goal: allow host to *dynamically* obtain its IP address from network server when it joins network
- Can renew its lease on address in use
- Allows reuse of addresses (only hold address while connected an "on")
- Support for mobile users who want to join network (more shortly)



arriving DHCP client needs address in this network



DHCP discover
src : 0.0.0.0, 68
dest.: 255.255.255.255,67
yiaddr: 0.0.0.0
transaction ID: 654

DHCP offer
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 654
Lifetime: 3600 secs

DHCP request
src: 0.0.0.0, 68
dest:: 255.255.255.255, 67
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

DHCP ACK
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

65

IP addresses: how to get one?

Q: How does *network* get subnet part of IP addr?

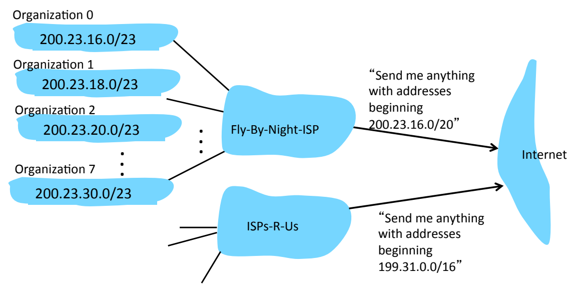
A: gets allocated portion of its provider ISP's address space

ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100 00000000	200.23.20.0/23
...	....	....
Organization 7	11001000 00010111 00011110 00000000	200.23.30.0/23

66

Hierarchical addressing: route aggregation

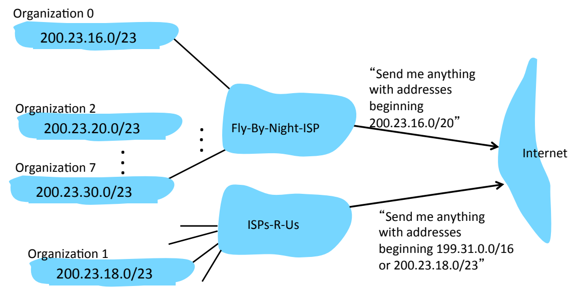
Hierarchical addressing allows efficient advertisement of routing information:



67

Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1



68

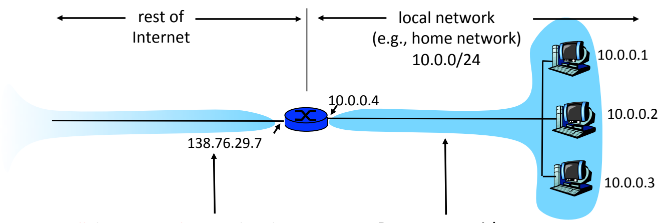
IP addressing: the last word...

Q: How does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers
- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

69

NAT: Network Address Translation



All datagrams *leaving* local network have same single source NAT IP address: 138.76.29.7, different source port numbers

Datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

70

NAT: Network Address Translation

- Motivation: local network uses just one IP address as far as outside world is concerned:
  - range of addresses not needed from ISP: just one IP address for all devices
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - devices inside local net not explicitly addressable, visible by outside world (a security plus).

71

NAT: Network Address Translation

Implementation: NAT router must:

- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

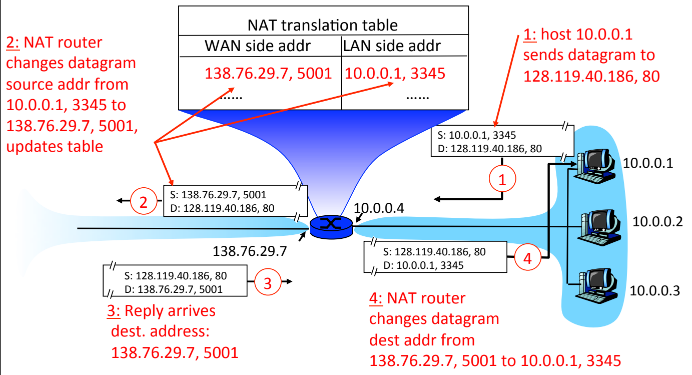
  . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.

- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair

- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

72

NAT: Network Address Translation



73

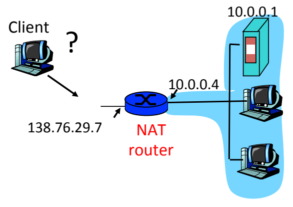
NAT: Network Address Translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, eg, P2P applications
  - address shortage should instead be solved by IPv6

74

NAT traversal problem

- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
- solution 1: statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000

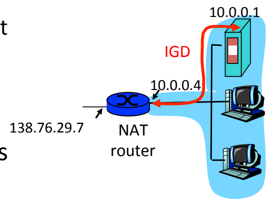


75

NAT traversal problem

- solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to:
  - ❖learn public IP address (138.76.29.7)
  - ❖add/remove port mappings (with lease times)

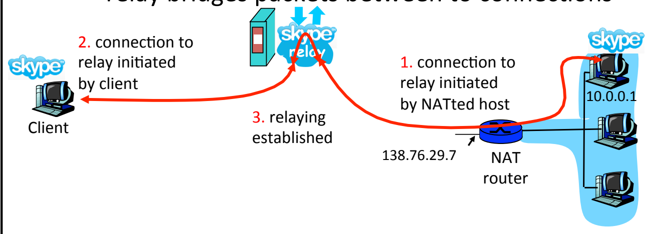
i.e., automate static NAT port map configuration



76

NAT traversal problem

- solution 3: relaying (used in Skype)
  - NATed client establishes connection to relay
  - External client connects to relay
  - relay bridges packets between to connections


2. connection to relay initiated by client
skype relay
1. connection to relay initiated by NATted host
skype
Client
3. relaying established
10.0.0.1
138.76.29.7
NAT router

77

ICMP: Internet Control Message Protocol

- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

78

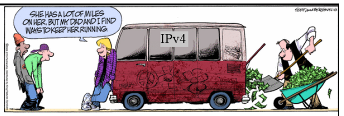
Traceroute and ICMP

- Source sends series of UDP segments to dest
  - First has TTL =1
  - Second has TTL=2, etc.
  - Unlikely port number
- When nth datagram arrives to nth router:
  - Router discards datagram
  - And sends to source an ICMP message (type 11, code 0)
  - Message includes name of router& IP address
- When ICMP message arrives, source calculates RTT
- Traceroute does this 3 times

Stopping criterion
- UDP segment eventually arrives at destination host
- Destination returns ICMP "host unreachable" packet (type 3, code 3)
- When source gets this ICMP, stops.

79

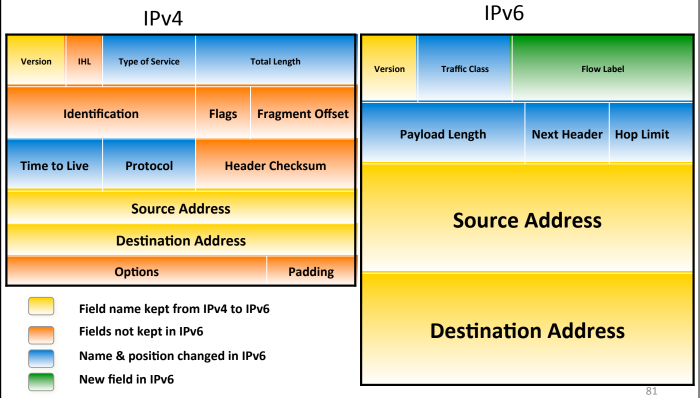
IPv6


SHE HAS A LOT OF MILES ON HER, BUT MY DAD AND I FIND WAYS TO KEEP HER RUNNING.
IPv4

- Motivated (prematurely) by address exhaustion
  - Addresses *four* times as big
- Steve Deering focused on simplifying IP
  - Got rid of all fields that were not absolutely necessary
  - "Spring Cleaning" for IP
- Result is an elegant, if unambitious, protocol

80

IPv4 and IPv6 Header Comparison



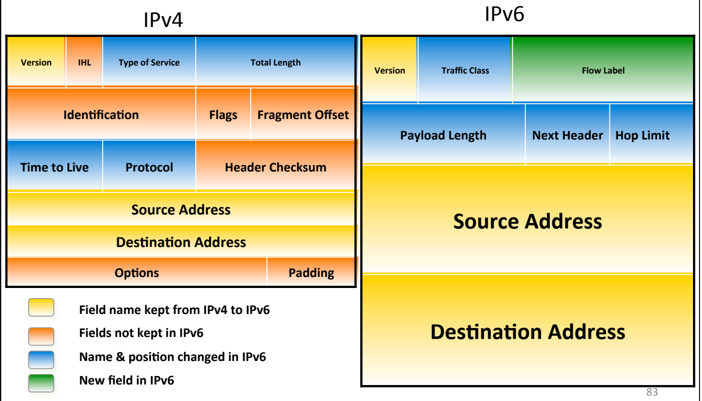
81

Summary of Changes

- Eliminated fragmentation ***(why?)***
- Eliminated header length ***(why?)***
- Eliminated checksum ***(why?)***
- New options mechanism (next header) ***(why?)***
- Expanded addresses ***(why?)***
- Added Flow Label ***(why?)***

82

IPv4 and IPv6 Header Comparison



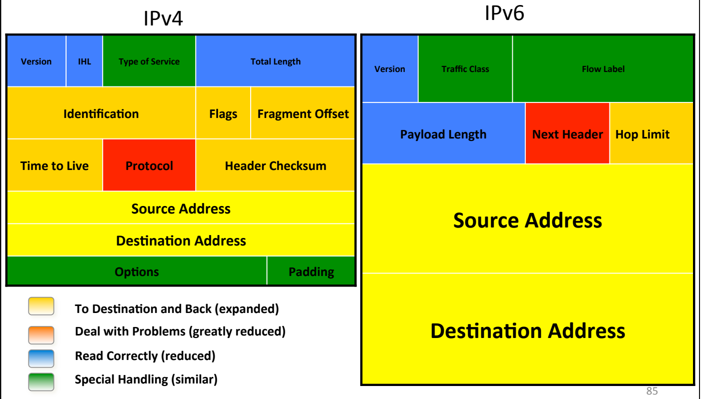
83

Philosophy of Changes

- Don't deal with problems: leave to ends
  - Eliminated fragmentation
  - Eliminated checksum
  - ***Why retain TTL?***
- Simplify handling:
  - New options mechanism (uses next header approach)
  - Eliminated header length
    - ***Why couldn't IPv4 do this?***
- Provide general flow label for packet
  - Not tied to semantics
  - Provides great flexibility

84

Comparison of Design Philosophy



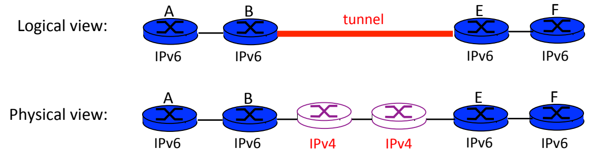
85

Transition From IPv4 To IPv6

- Not all routers can be upgraded simultaneous
  - no "flag days"
  - How will the network operate with mixed IPv4 and IPv6 routers?
- *Tunneling:* IPv6 carried as payload in IPv4 datagram among IPv4 routers

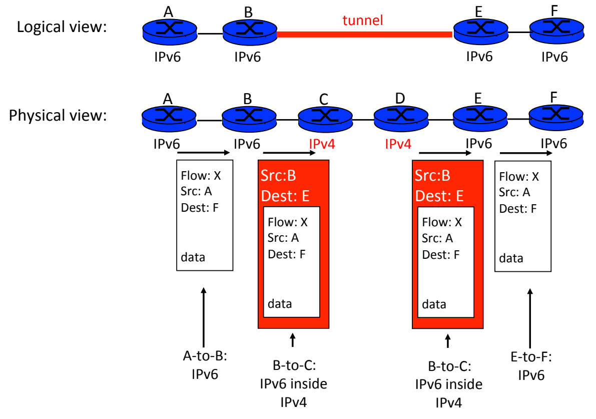
86

Tunneling



87

Tunneling



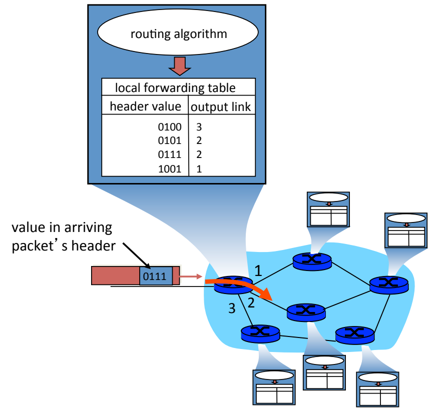
88

Improving on IPv4 and IPv6?

- Why include unverifiable source address?
  - Would like accountability ***and*** anonymity (now neither)
  - Return address can be communicated at higher layer
- Why packet header used at edge same as core?
  - Edge: host tells network what service it wants
  - Core: packet tells switch how to handle it
    - One is local to host, one is global to network
- Some kind of payment/responsibility field?
  - Who is responsible for paying for packet delivery?
  - Source, destination, other?
- Other ideas?

89

Interplay between routing and forwarding



"Valid" Routing State

- Global routing state is "valid" if it produces forwarding decisions that always deliver packets to their destinations
  - ***Valid is not standard terminology***
- Goal of routing protocols: compute valid state
  - But how can you tell if routing state if valid?

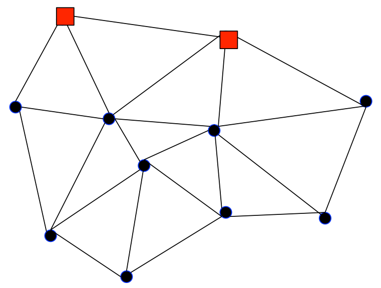
Necessary and Sufficient Condition

- Global routing state is valid ***if and only if***:
  - There are no dead ends (other than destination)
  - There are no loops
- A dead end is when there is no outgoing port
  - A packet arrives, but the forwarding decision does not yield any outgoing port
- A loop is when a packet cycles around the same set of nodes forever

Necessary: Obvious

- If you run into a deadend before hitting destination, you'll never reach the destination
- If you run into a loop, you'll never reach destination
  - With deterministic forwarding, once you loop, you'll loop forever (assuming routing state is static)

Wandering Packets



Packet reaches deadend and stops

Packet falls into loop and never reaches destination

Sufficient: Easy

- Assume no deadends, no loops
- Packet must keep wandering, without repeating
  - If ever enter same switch from same port, will loop
  - Because forwarding decisions are deterministic
- Only a finite number of possible ports for it to visit
  - It cannot keep wandering forever without looping
  - Must eventually hit destination

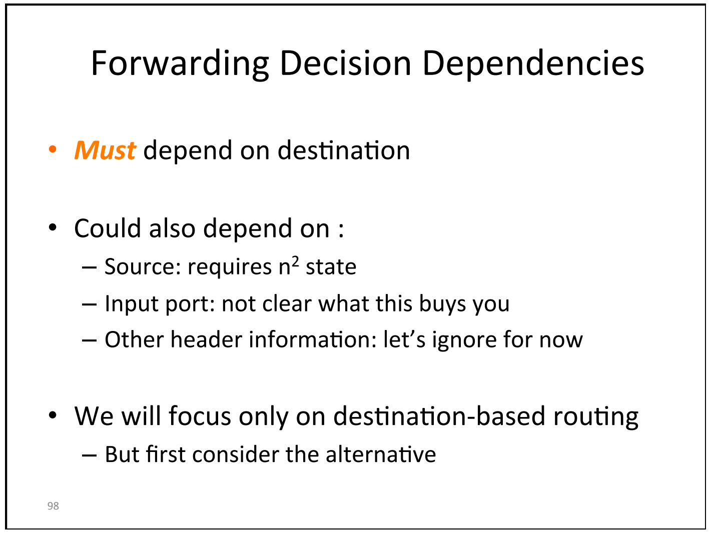
The "Secret" of Routing

- Avoiding deadends is easy
- Avoiding loops is hard
- The key difference between routing protocols is how they avoid loops!
  - Don't focus on details of mechanisms
  - Just ask "how are loops avoided?"
- Will return to this later…. a little this term

  a lot more in Part II *Principles of Communications*

Making Forwarding Decisions

- Map PacketState+RoutingState into OutgoingPort
  - At line rates…..
- Packet State:
  - Destination ID
  - Source ID
  - Incoming Port *(from switch, not packet)*
  - Other packet header information?
- Routing State:
  - Stored in router

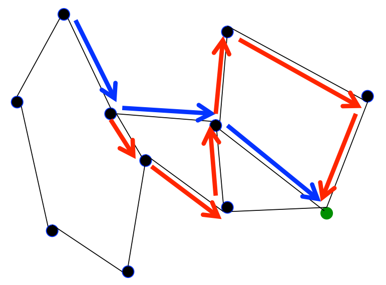


Forwarding Decision Dependencies

- *Must* depend on destination
- Could also depend on :
  - Source: requires n^2 state
  - Input port: not clear what this buys you
  - Other header information: let's ignore for now
- We will focus only on destination-based routing
  - But first consider the alternative

98

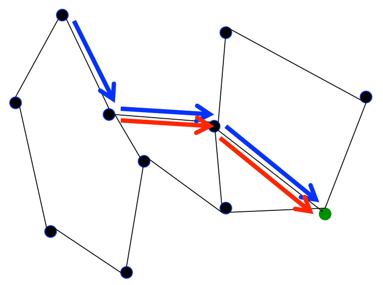
Source/Destination-Based Routing



Paths from two different sources (to same destination) can be very different

99

Destination-Based Routing



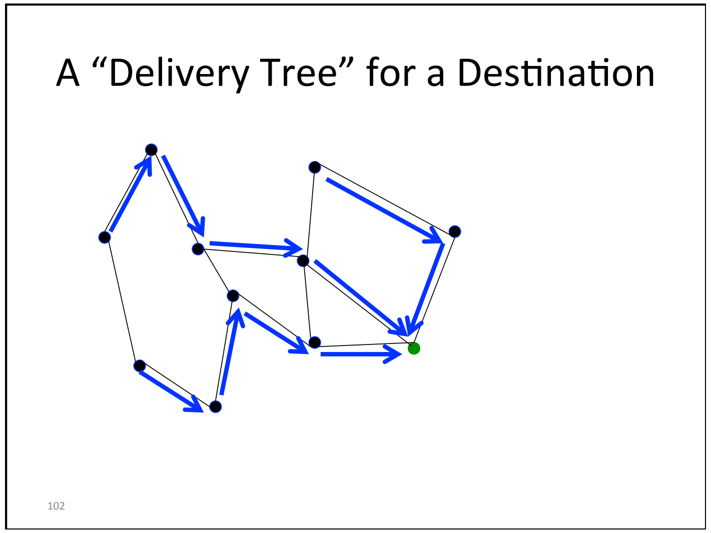
Paths from two different sources (to same destination) must coincide once they overlap

100

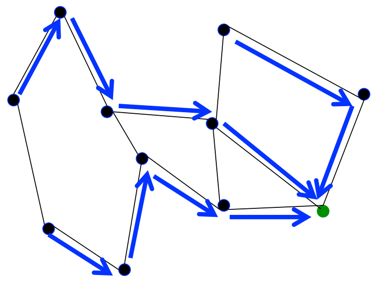
Destination-Based Routing

- Paths to same destination never cross
- Once paths to destination meet, they never split
- Set of paths to destination create a "delivery tree"
  - Must cover every node exactly once
  - **Spanning Tree rooted at destination**

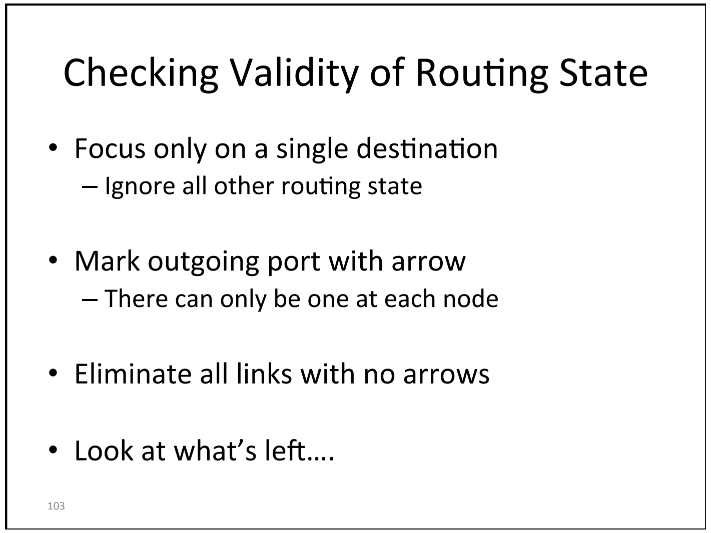
101



A “Delivery Tree” for a Destination



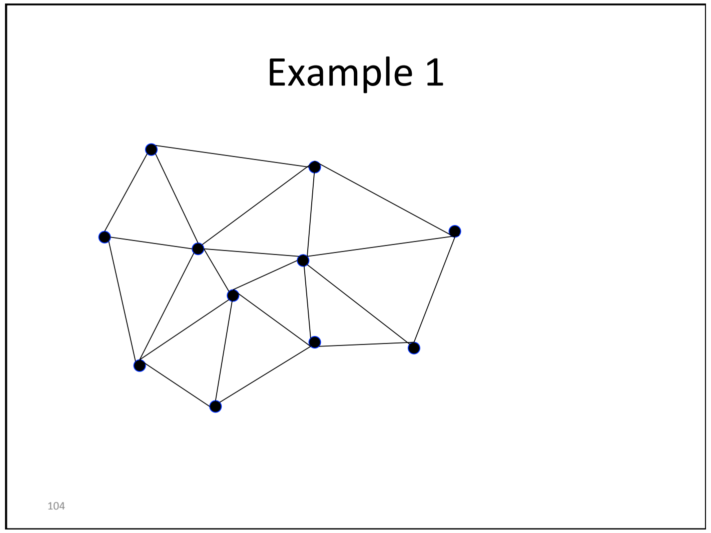
102



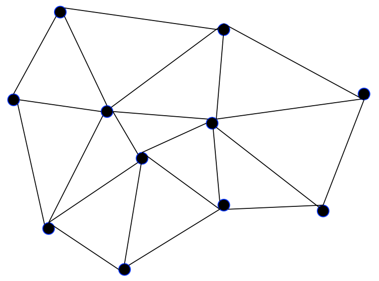
Checking Validity of Routing State

- Focus only on a single destination
  - Ignore all other routing state
- Mark outgoing port with arrow
  - There can only be one at each node
- Eliminate all links with no arrows
- Look at what’s left….

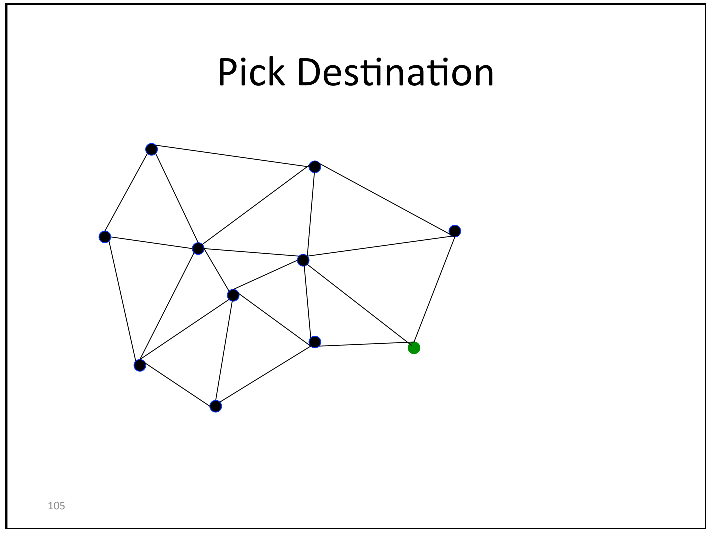
103



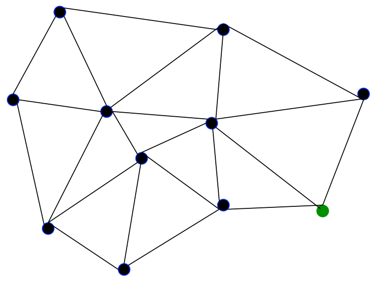
Example 1



104

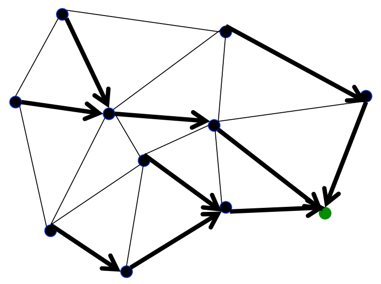


Pick Destination

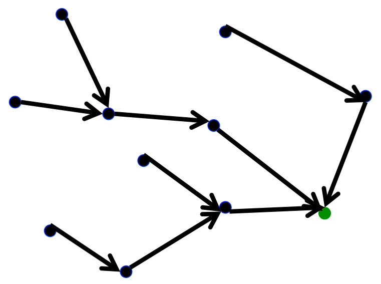


105

Put Arrows on Outgoing Ports

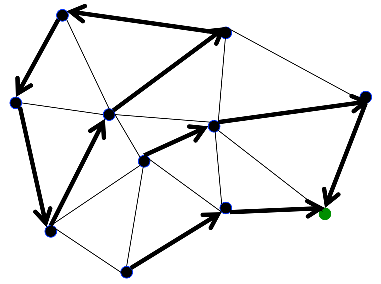


Remove Unused Links

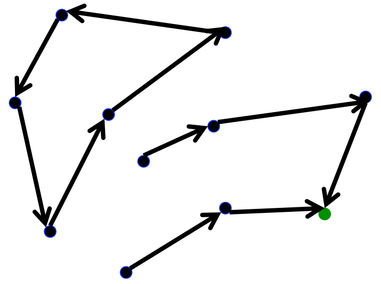


Leaves Spanning Tree: Valid

Second Example



Second Example



Is this valid?

Lesson….

- Very easy to check validity of routing state for a particular destination
- Deadends are obvious
  – Node without outgoing arrow
- Loops are obvious
  – Disconnected from rest of graph

110

Computing Routing State

111

Forms of Route Computation

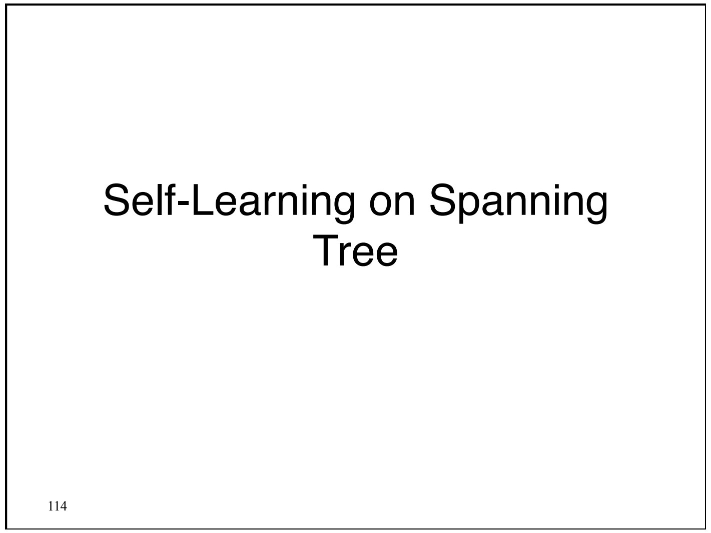
- Learn from observing….
  – Not covered in your reading
- Centralized computation
  – One node has the entire network map
- Pseudo-centralized computation
  – All nodes have the entire network map
- Distributed computation
  – No one has the entire network map

112

How Can You Avoid Loops?

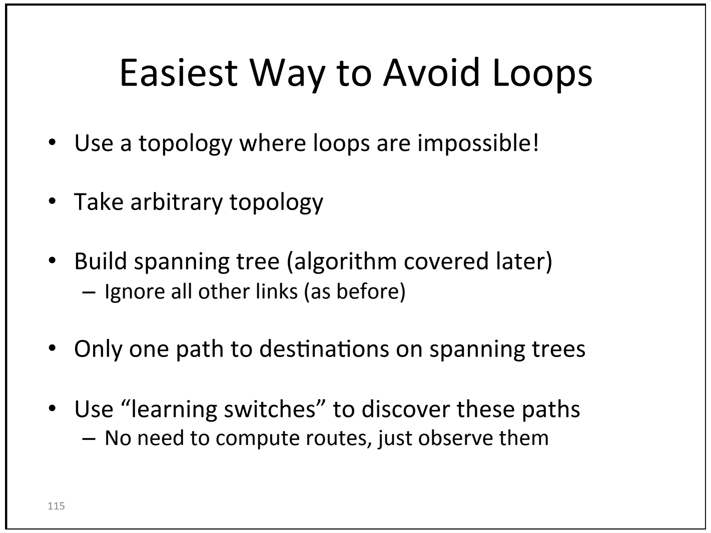
- Restrict topology to spanning tree
  – If the topology has no loops, packets can't loop!
- Central computation
  – Can make sure no loops
- Minimizing metric in distributed computation
  – Loops are never the solution to a minimization problem

113



Self-Learning on Spanning Tree

114

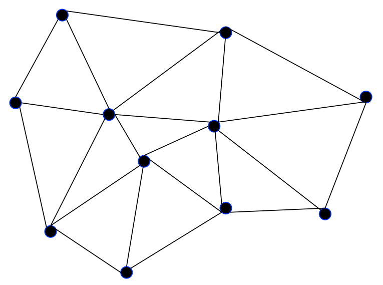


Easiest Way to Avoid Loops

- Use a topology where loops are impossible!
- Take arbitrary topology
- Build spanning tree (algorithm covered later)
  - Ignore all other links (as before)
- Only one path to destinations on spanning trees
- Use “learning switches” to discover these paths
  - No need to compute routes, just observe them

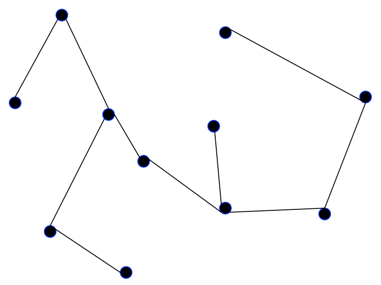
115

Consider previous graph



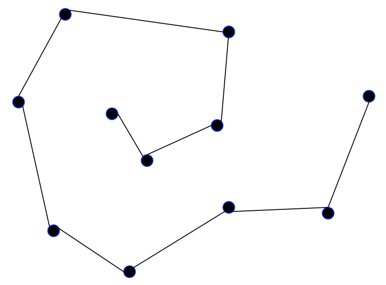
116

A Spanning Tree

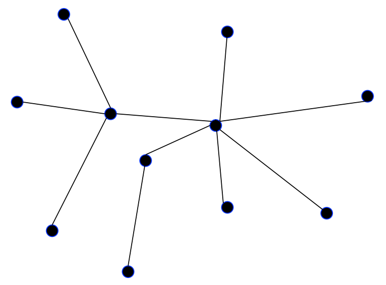


117

Another Spanning Tree



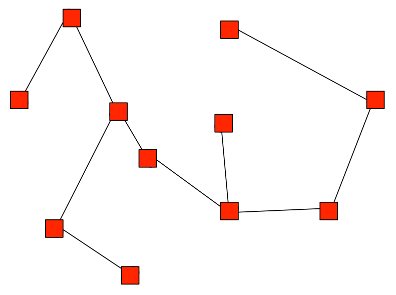
Yet Another Spanning Tree



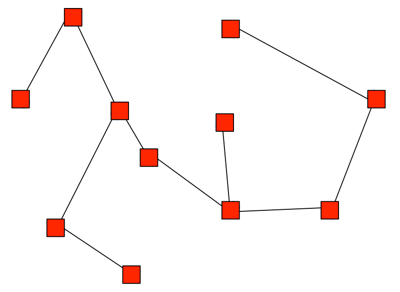
Flooding on a Spanning Tree

- If you want to send a packet that will reach all nodes, then switches can use the following rule:
  – Ignoring all ports not on spanning tree!
- Originating switch sends "flood" packet out all ports
- When a "flood" packet arrives on one incoming port, send it out all other ports

Flooding on Spanning Tree



Flooding on Spanning Tree (Again)



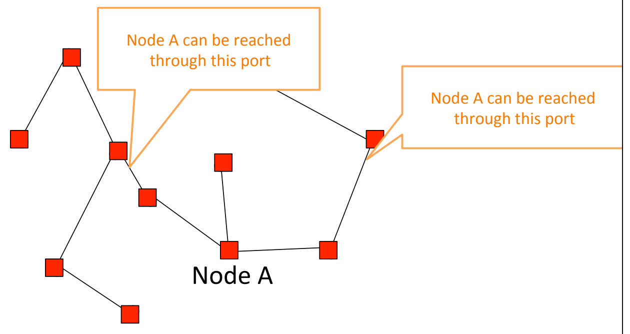
Flooding on a Spanning Tree

- This works because the lack of loops prevents the flooding from cycling back on itself
- Eventually all nodes will be covered, exactly once

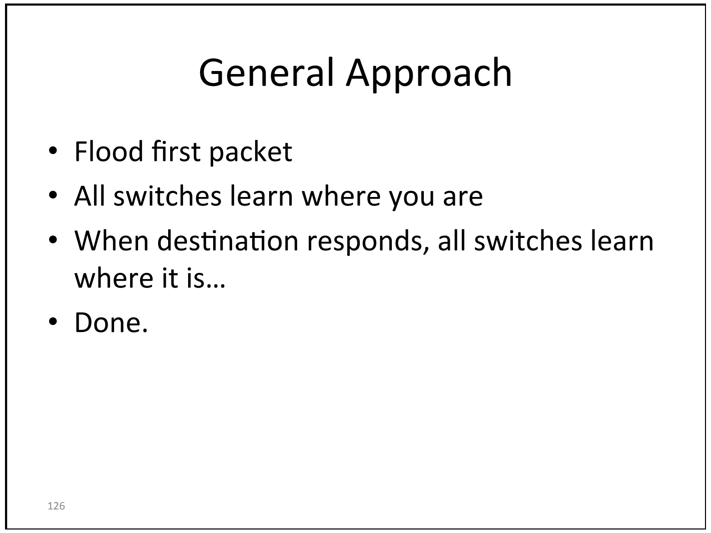
This Enables Learning!

- There is only one path from source to destination
- Each switch can learn how to reach a another node by remembering where its flooding packets came from!
- If flood packet from Node A entered switch from port 4, then to reach Node A, switch sends packets out port 4

Learning from Flood Packets


Node A can be reached through this port
Node A can be reached through this port
Node A

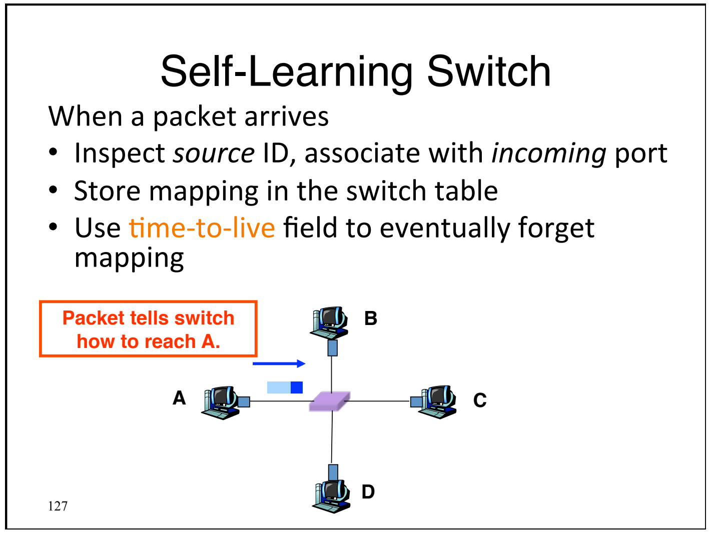
Once a node has sent a flood message, all other switches know how to reach it….



General Approach

- Flood first packet
- All switches learn where you are
- When destination responds, all switches learn where it is…
- Done.

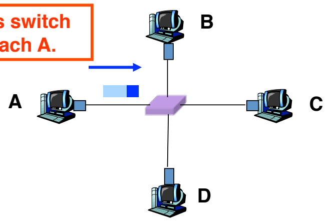
126



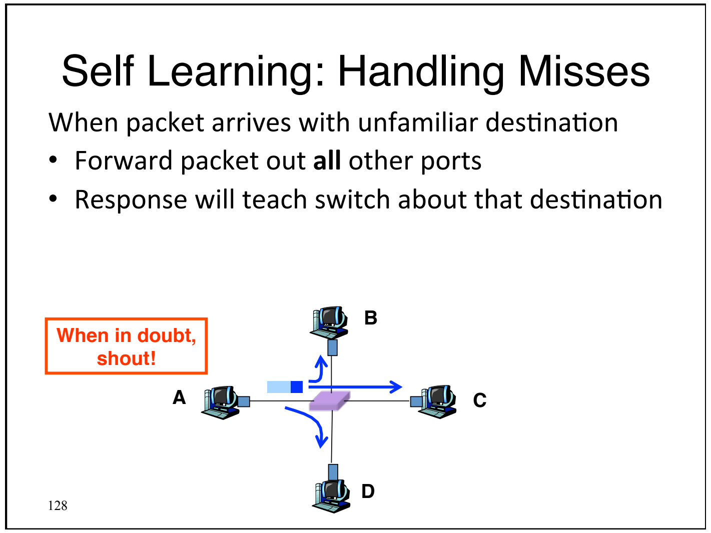
Self-Learning Switch

When a packet arrives

- Inspect *source* ID, associate with *incoming* port
- Store mapping in the switch table
- Use time-to-live field to eventually forget mapping



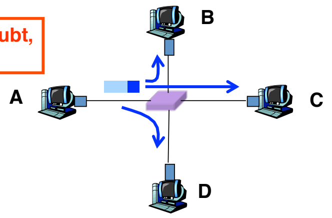
127



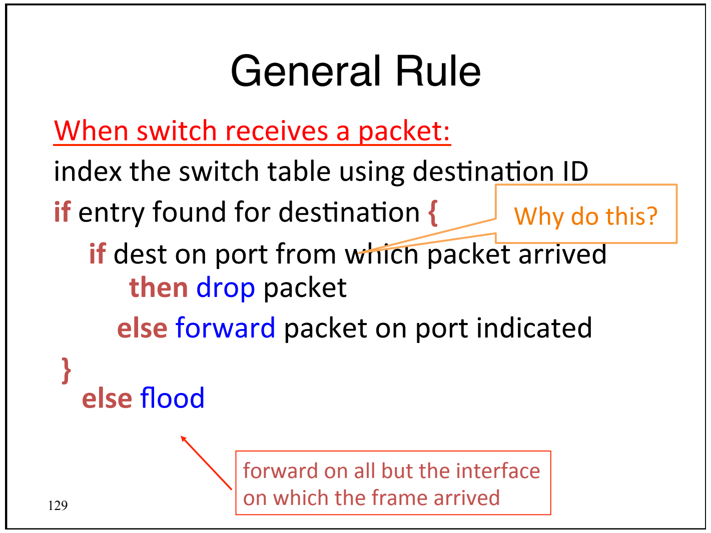
Self Learning: Handling Misses

When packet arrives with unfamiliar destination

- Forward packet out **all** other ports
- Response will teach switch about that destination



128



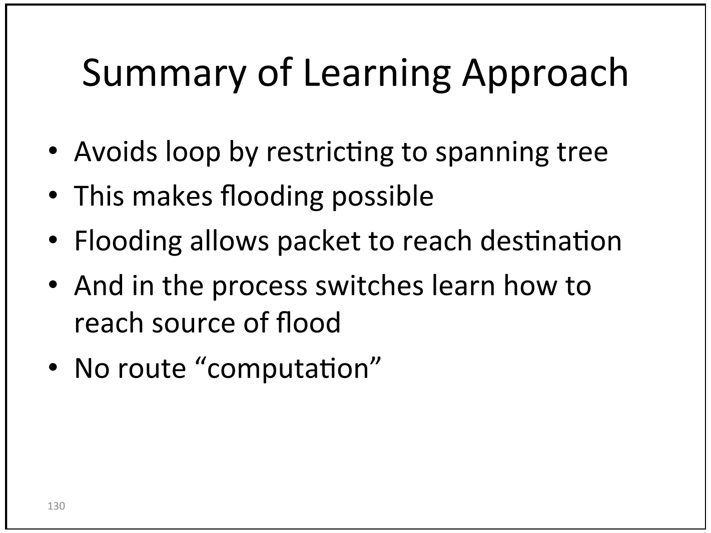
General Rule

```
When switch receives a packet:
index the switch table using destination ID
if entry found for destination {
   if dest on port from which packet arrived
       then drop packet
      else forward packet on port indicated
 }
   else flood
```

Why do this?

forward on all but the interface on which the frame arrived

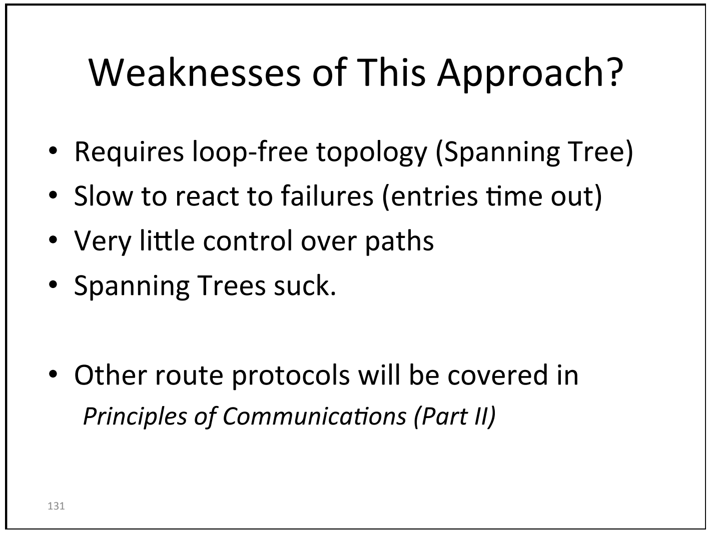
129



Summary of Learning Approach

- Avoids loop by restricting to spanning tree
- This makes flooding possible
- Flooding allows packet to reach destination
- And in the process switches learn how to reach source of flood
- No route "computation"

130



Weaknesses of This Approach?

- Requires loop-free topology (Spanning Tree)
- Slow to react to failures (entries time out)
- Very little control over paths
- Spanning Trees suck.

- Other route protocols will be covered in
  Principles of Communications (Part II)

131