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 – again.

Name: a *something*

Address: Where a *something* is

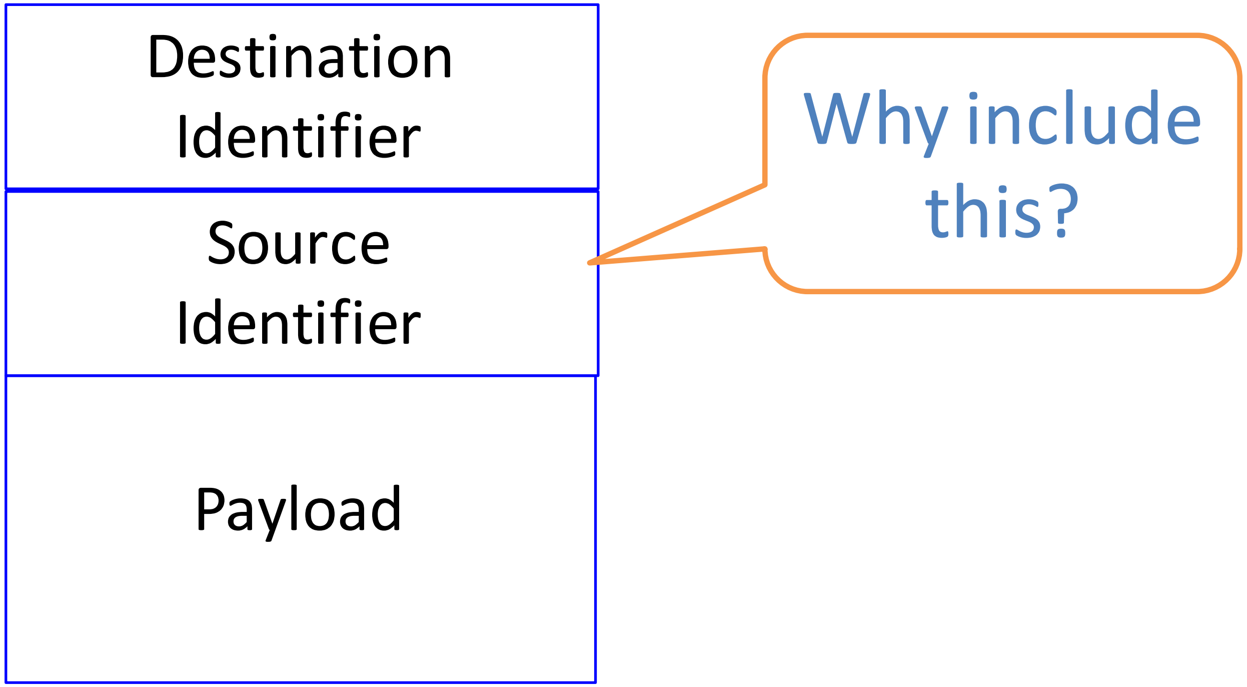
Routing: How do I get to the *something*

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?

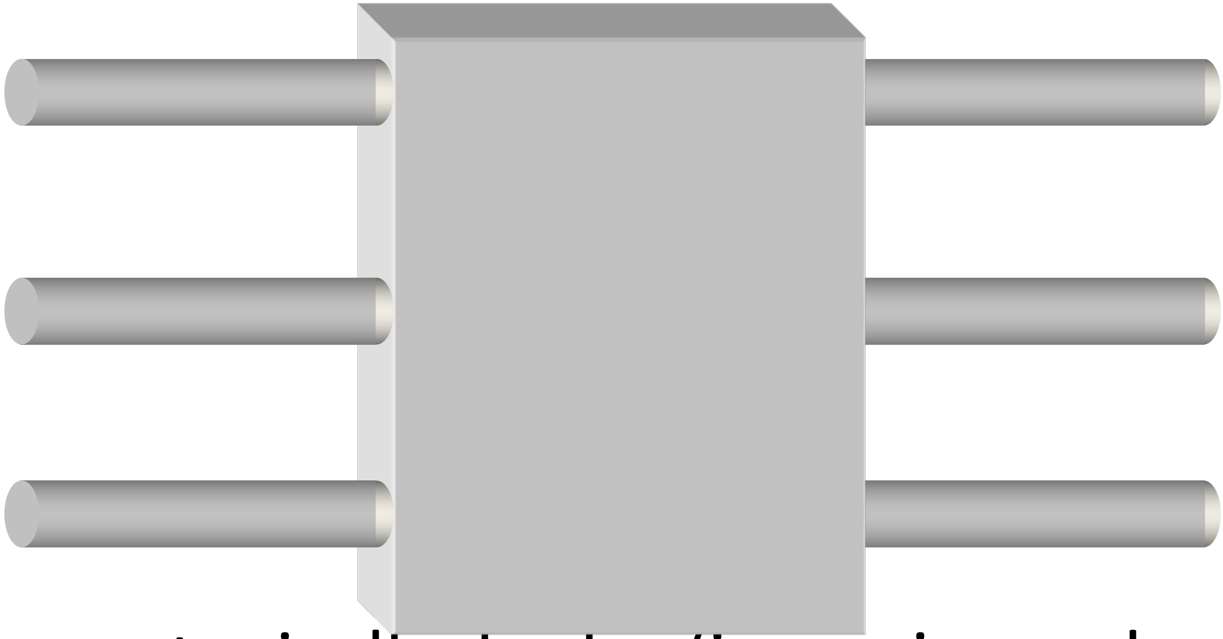
Packets (at a conceptual level)

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



Switches/Routers

- Multiple ports (attached to other switches or hosts)



incoming links

Switch

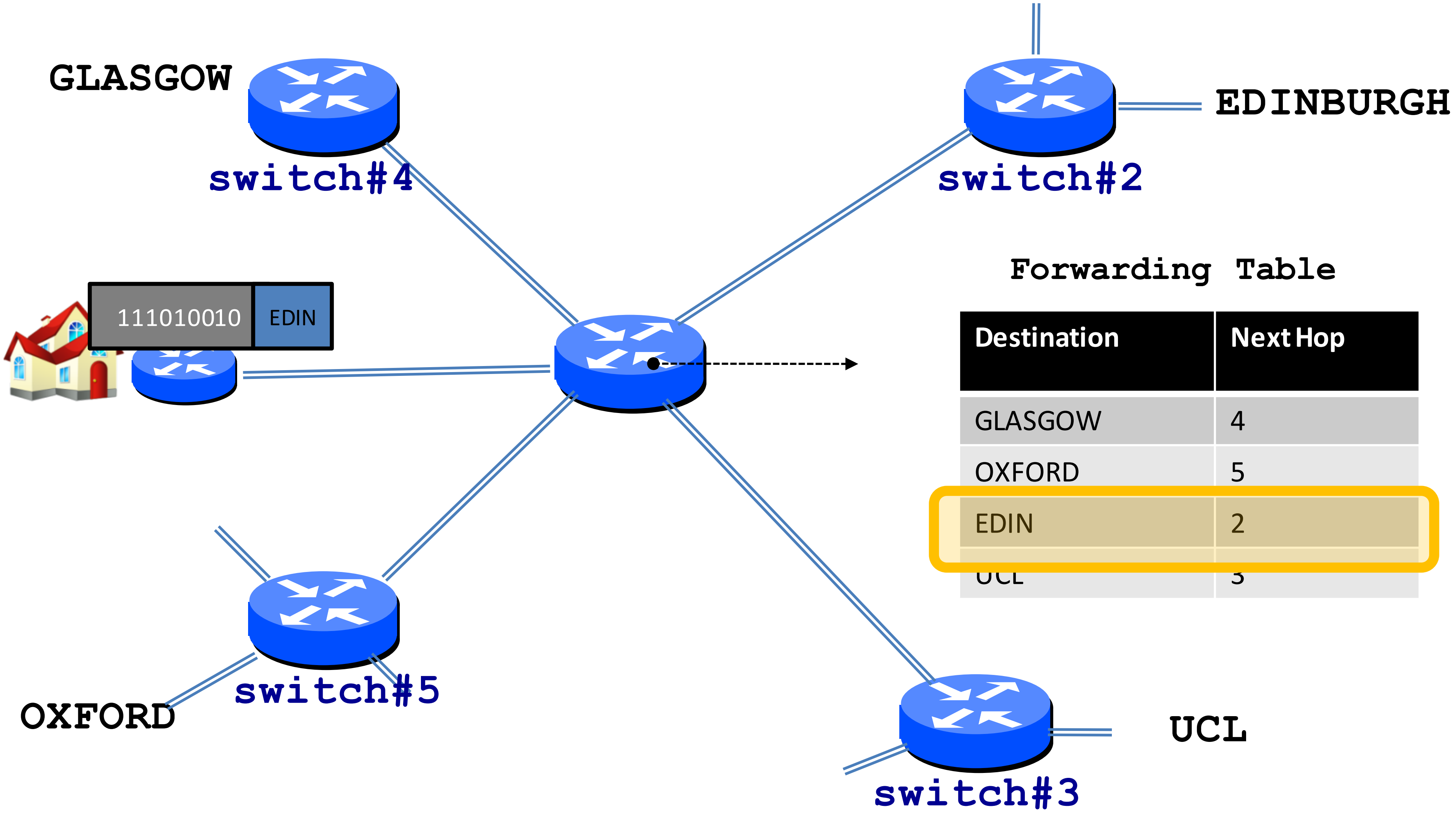
outgoing links

- Ports are typically duplex (incoming and outgoing)

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)

Switches forward packets



Forwarding Table

Destination	Next Hop
GLASGOW	4
OXFORD	5
EDIN	2
UCL	3

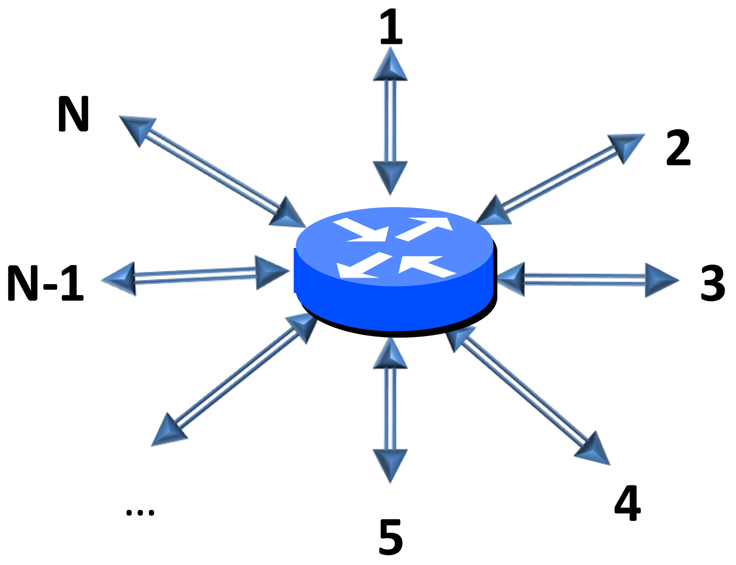
Forwarding Decisions

- 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....

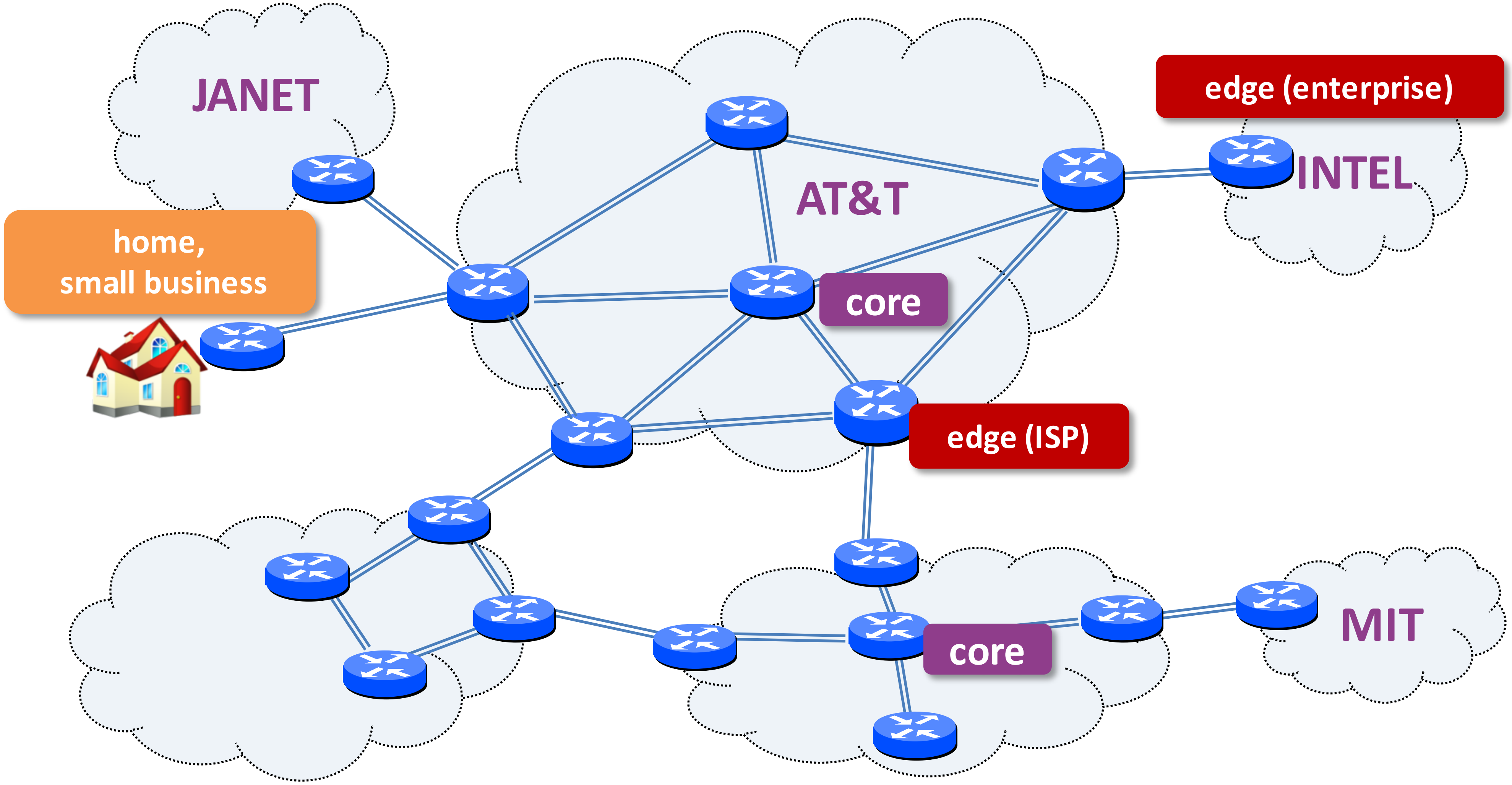
Router definitions



R *bits/sec*

- N = number of external router "ports"
- R = speed ("line rate") of a port
- Router capacity = N x R

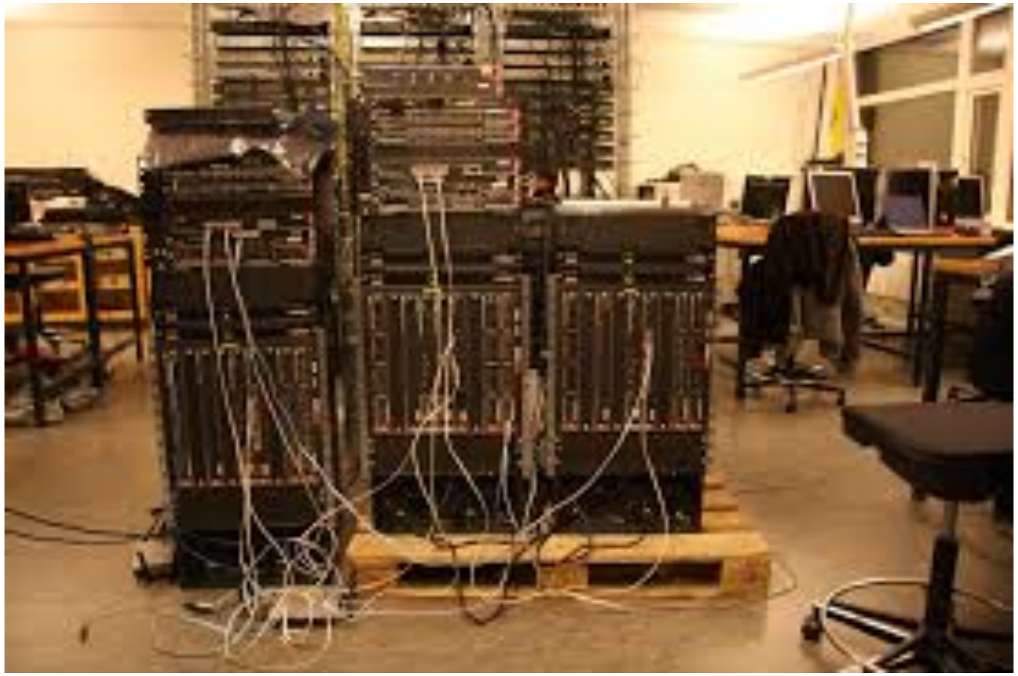
Networks and routers



Examples of routers (core)

Cisco CRS

- R=10/40/100 Gbps
- NR = 922 Tbps
- Netflix: 0.7GB per hour (1.5Mb/s)
- ~600 million concurrent Netflix users



72 racks, >1MW

Examples of routers (edge)

Cisco ASR

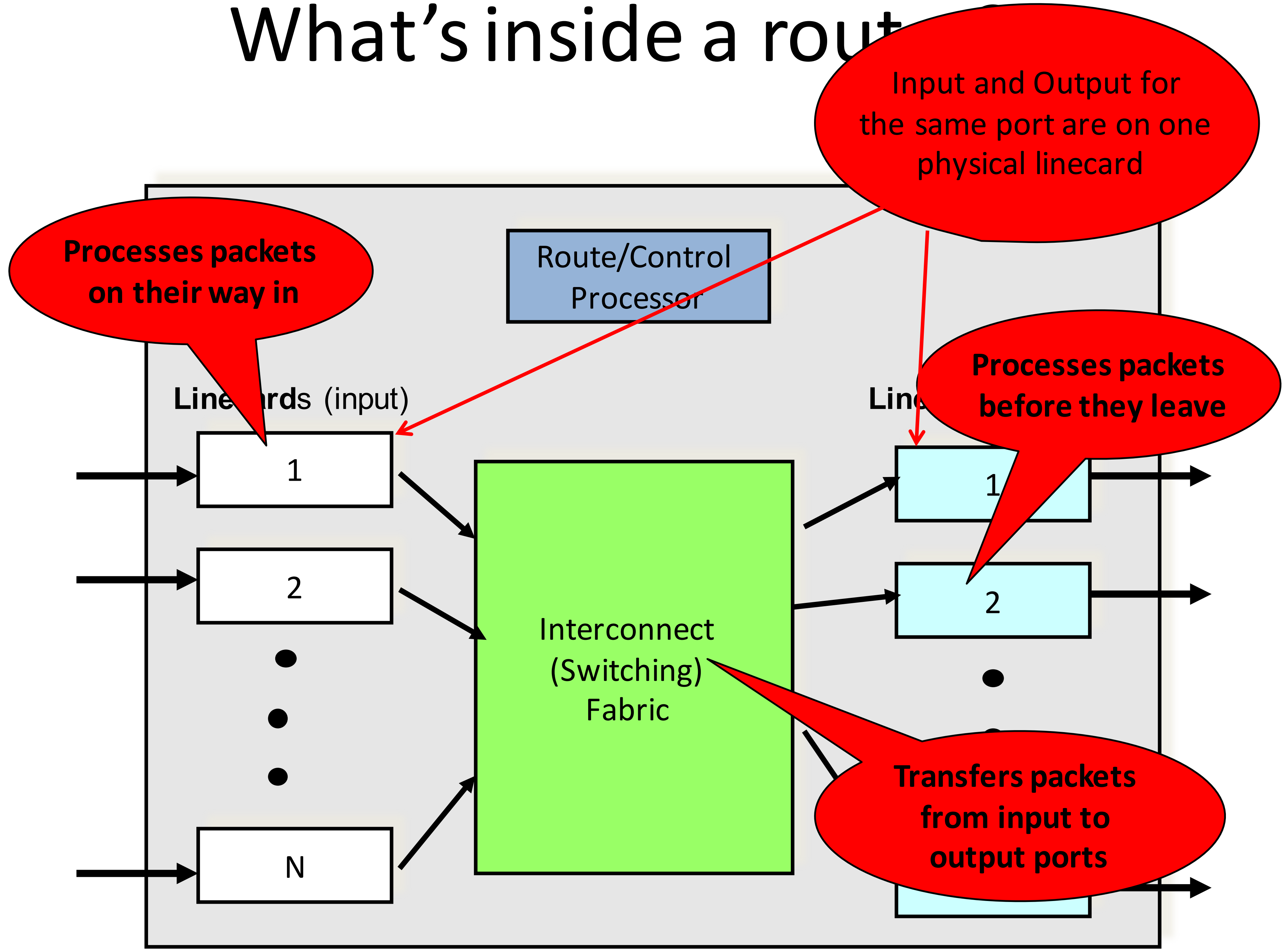
- R=1/10/40 Gbps
- NR = 120 Gbps

Examples of routers (small business)

Cisco 3945E

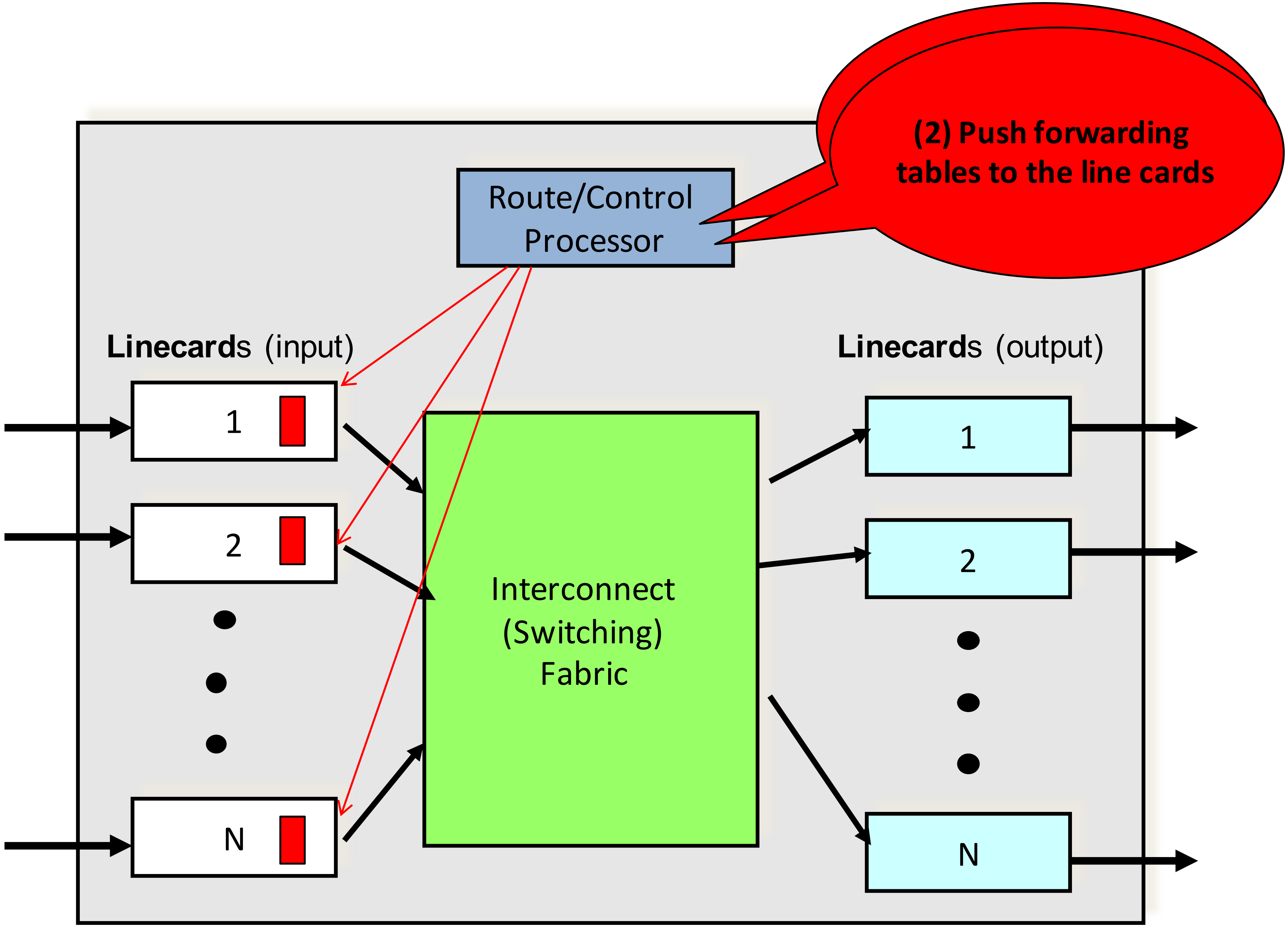
- R = 10/100/1000 Mbps
- NR < 10 Gbps

What's inside a rout



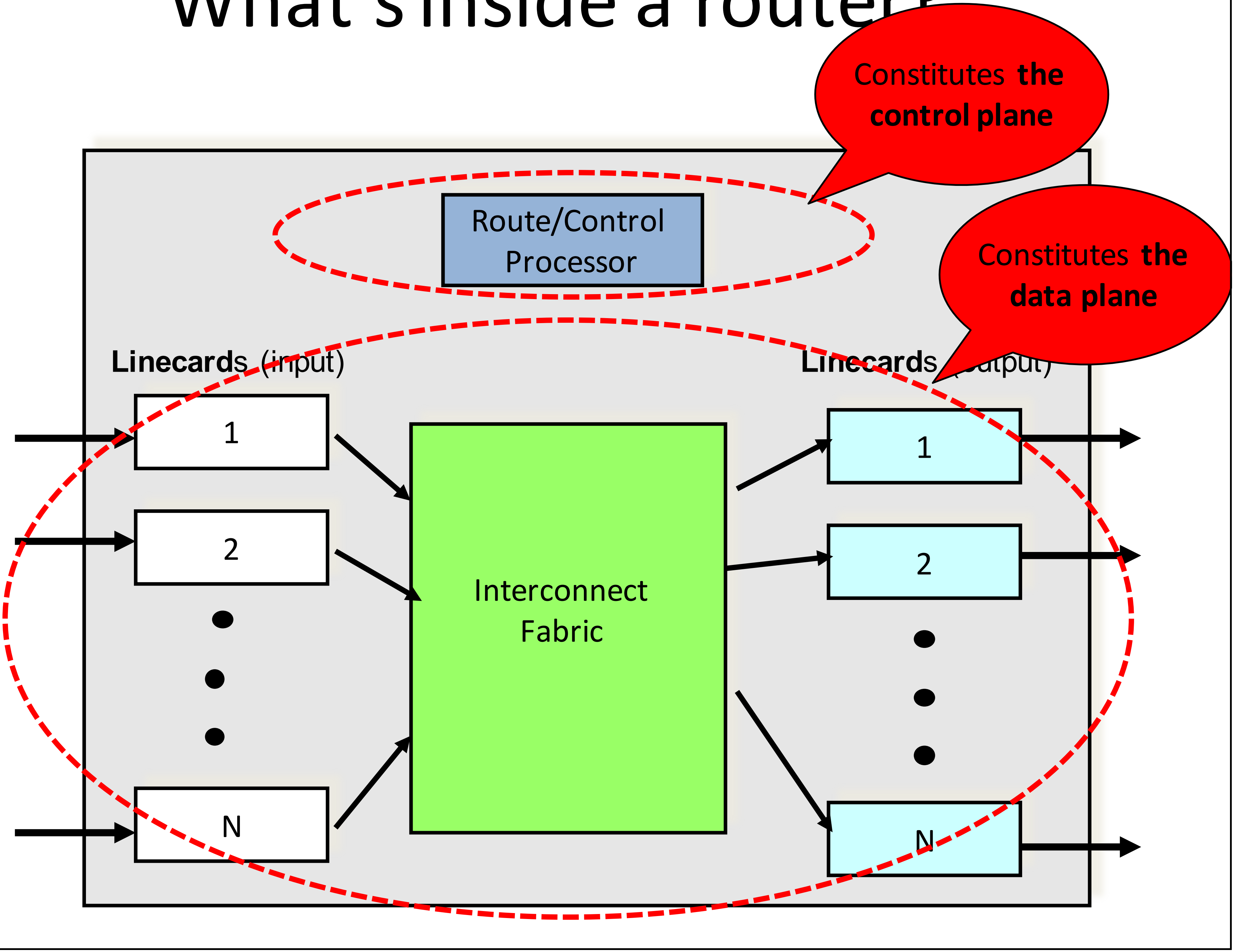
Input and Output for the same port are on one physical linecard
Processes packets on their way in
Route/Control Processor
Line rds (input)
Lin
Processes packets before they leave
1
2
N
1
2
Interconnect (Switching) Fabric
Transfers packets from input to output ports

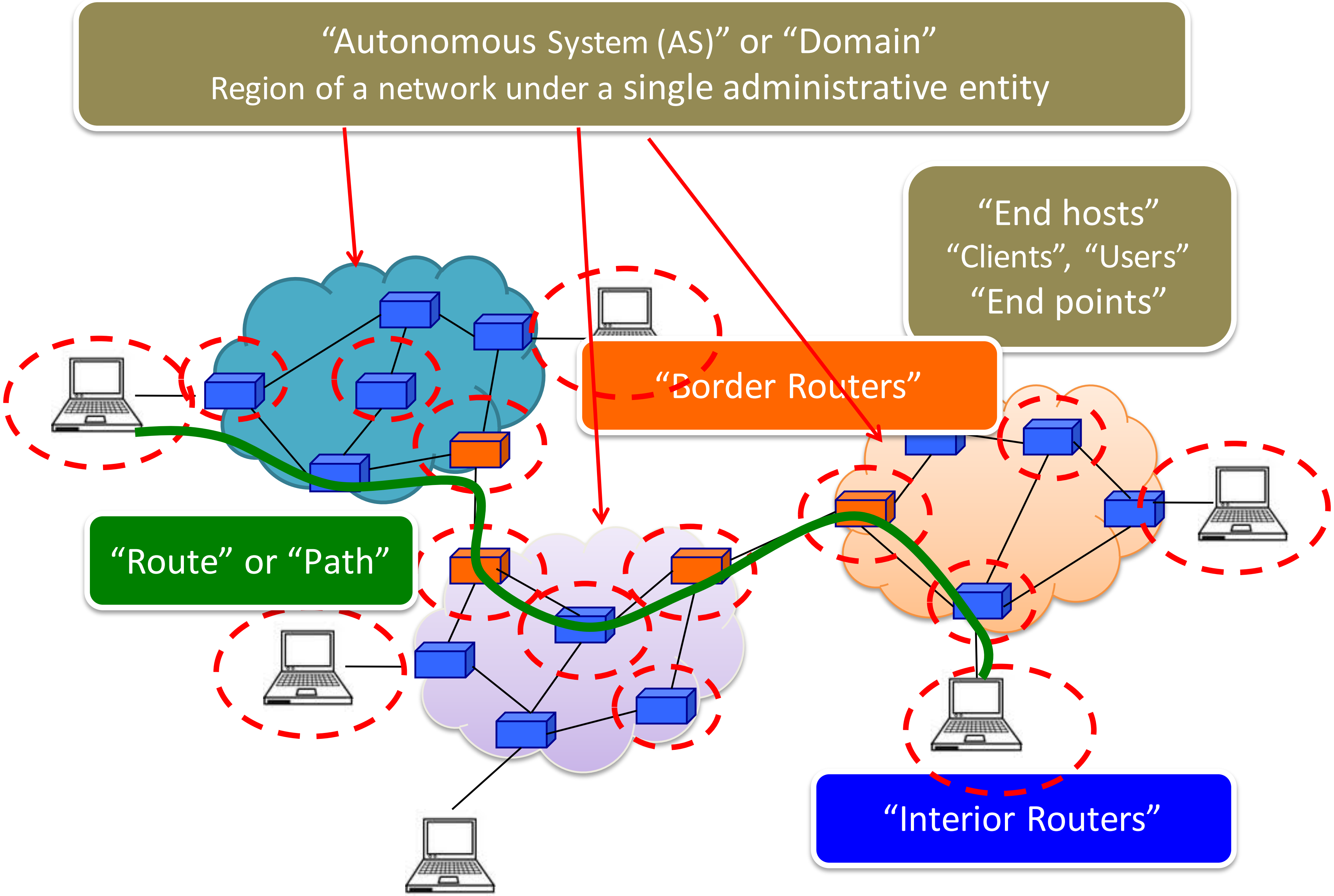
What’s inside a router?



(2) Push forwarding tables to the line cards
Route/Control Processor
Linecards (input)
1
2
N
Interconnect (Switching) Fabric
Linecards (output)
1
2
N

What's inside a router?





"Autonomous System (AS)" or "Domain"
Region of a network under a single administrative entity

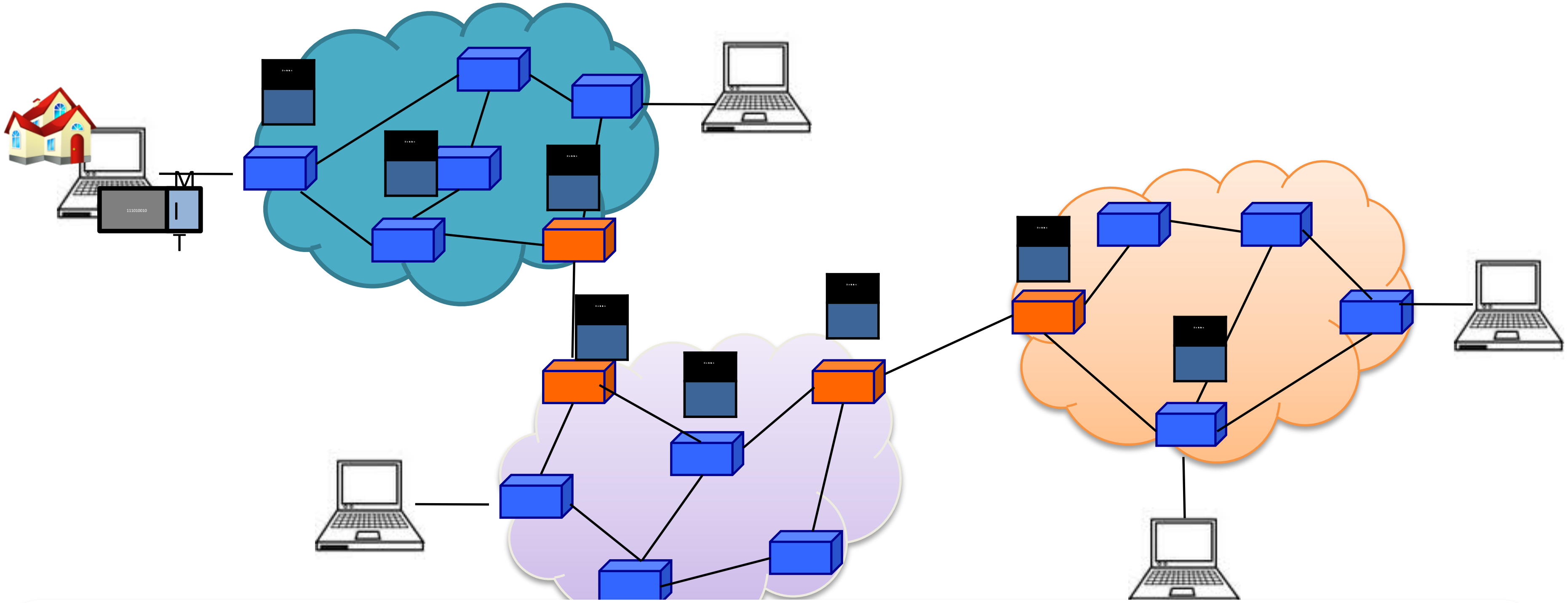
"End hosts"
"Clients", "Users"
"End points"

"Border Routers"

"Route" or "Path"

"Interior Routers"

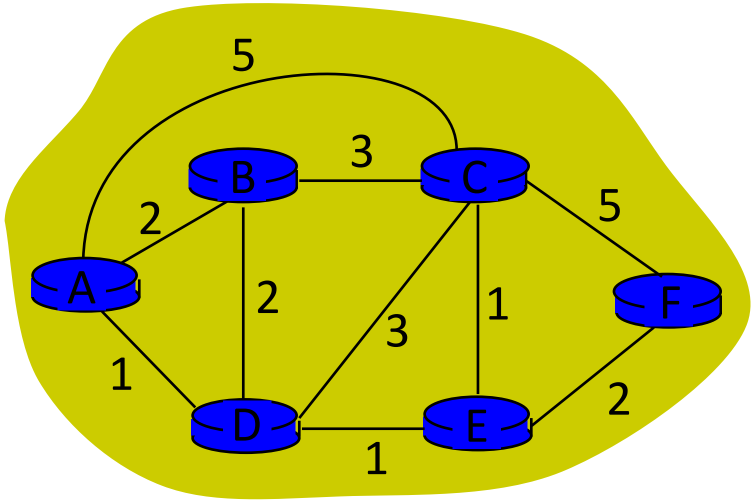
Context and Terminology



Internet routing protocols are responsible for constructing and updating the forwarding tables at routers

Routing Protocols

- Routing protocols implement the core function of a network
  - Establish paths between nodes
  - Part of the network’s “control plane”
- Network modeled as a graph
  - Routers are graph vertices
  - Links are edges
  - Edges have an associated “cost”
    - e.g., distance, loss



- Goal: compute a “good” path from source to destination
  - “good” usually means the shortest (least cost) path

Internet Routing

- Internet Routing works at two levels

- Each AS runs an intra-domain routing protocol that establishes routes within its domain
  - (AS -- region of network under a single administrative entity)
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)

- ASes participate in an inter-domain routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)

Addressing (for now)

- Assume each host has a unique ID (address)
- No particular structure to those IDs
- Later in course will talk about real IP addressing

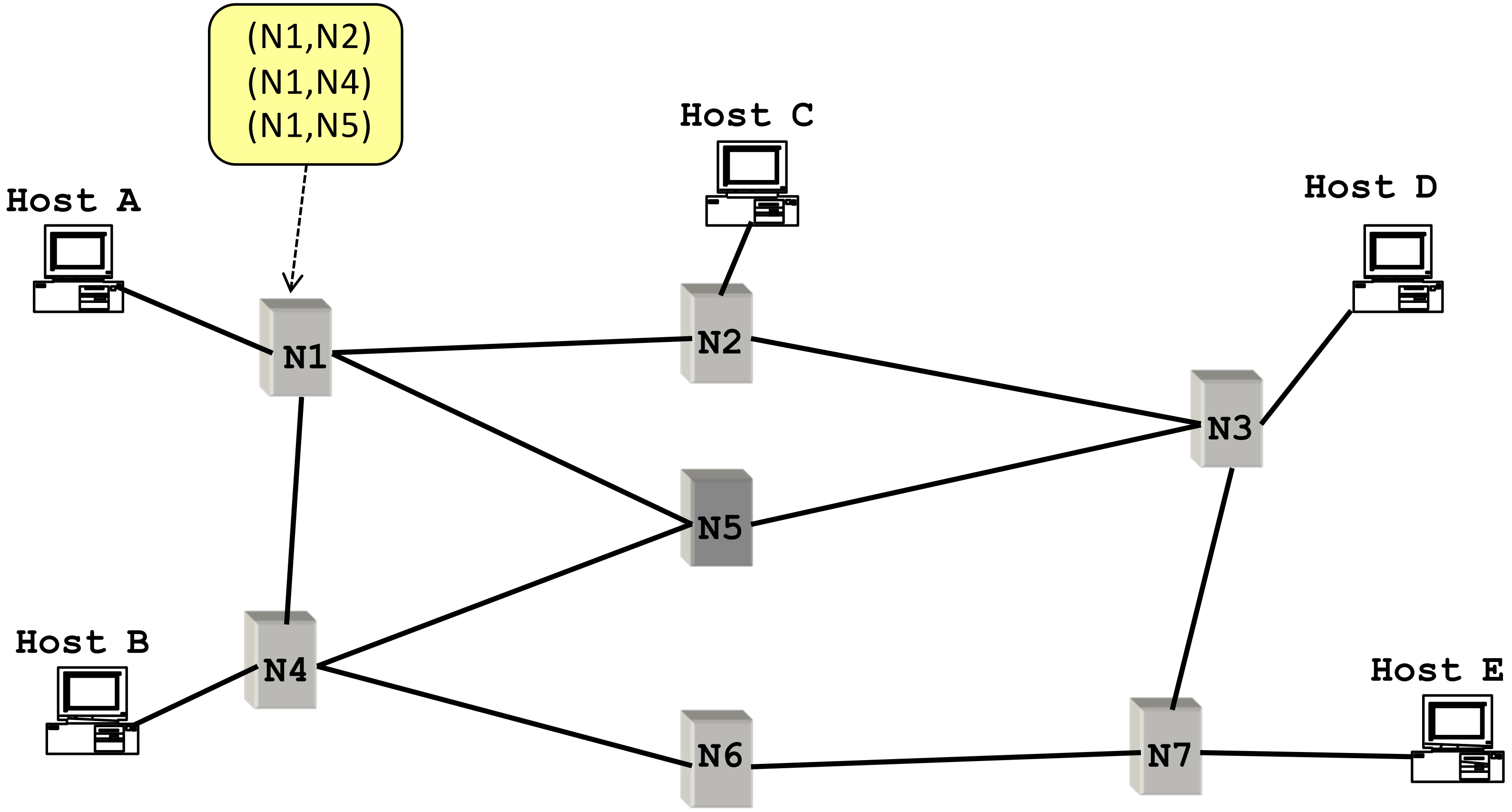
Outline

- Link State
- Distance Vector
- Routing: goals and metrics (if time)

Link-State

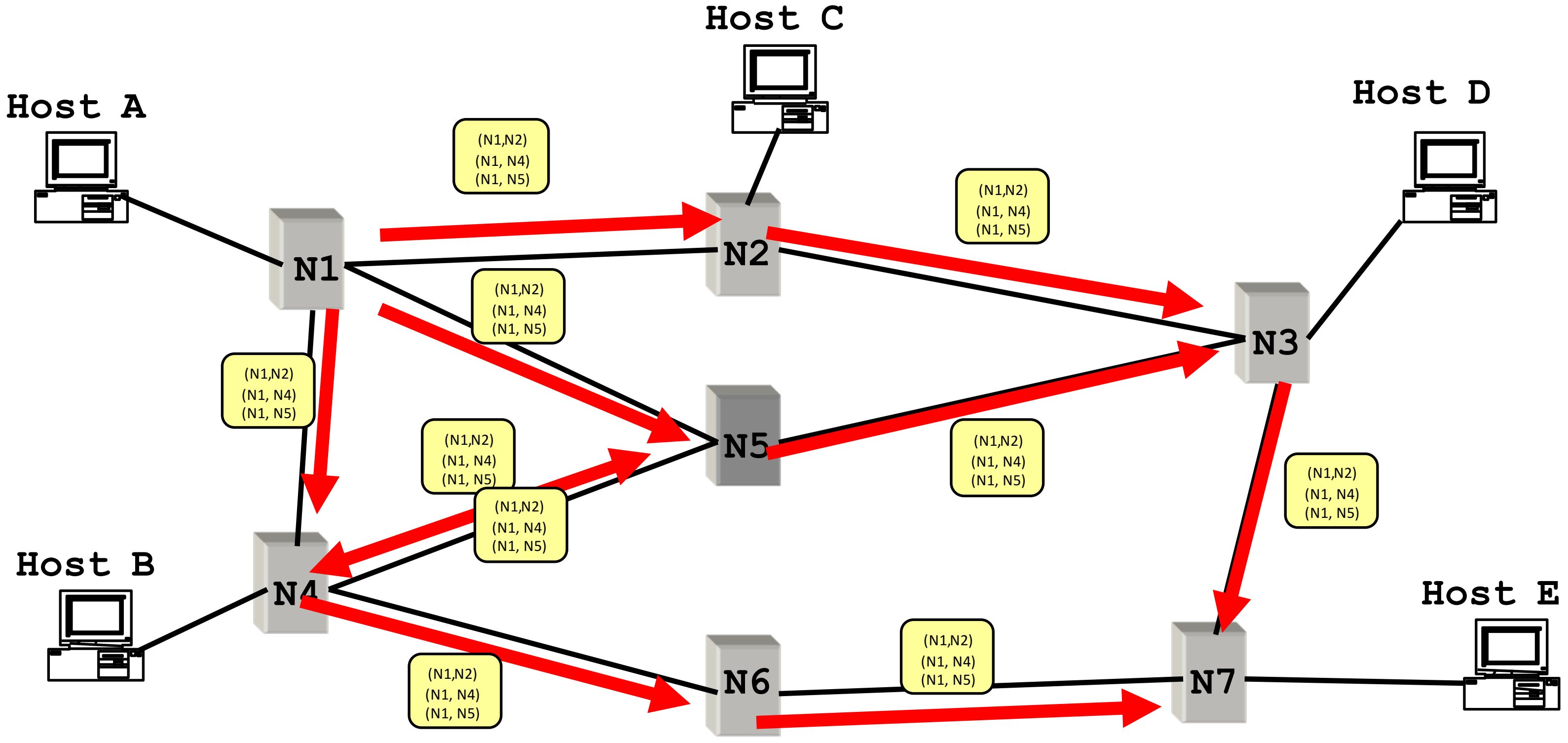
Link State Routing

- Each node maintains its local “link state” (LS)
  - i.e., a list of its directly attached links and their costs



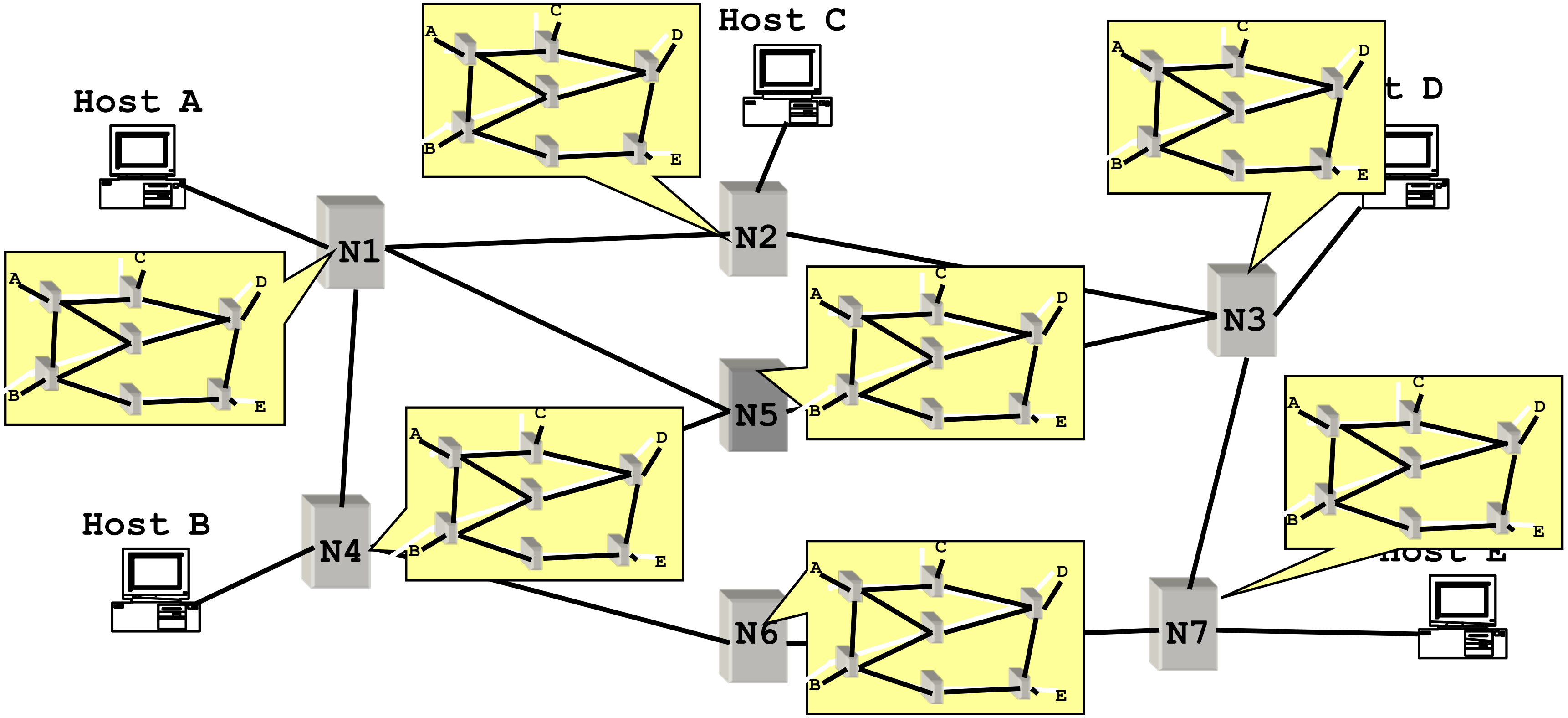
Link State Routing

- Each node maintains its local "link state" (LS)
- Each node floods its local link state
  - on receiving a new LS message, a router forwards the message to all its neighbors other than the one it received the message from



Link State Routing

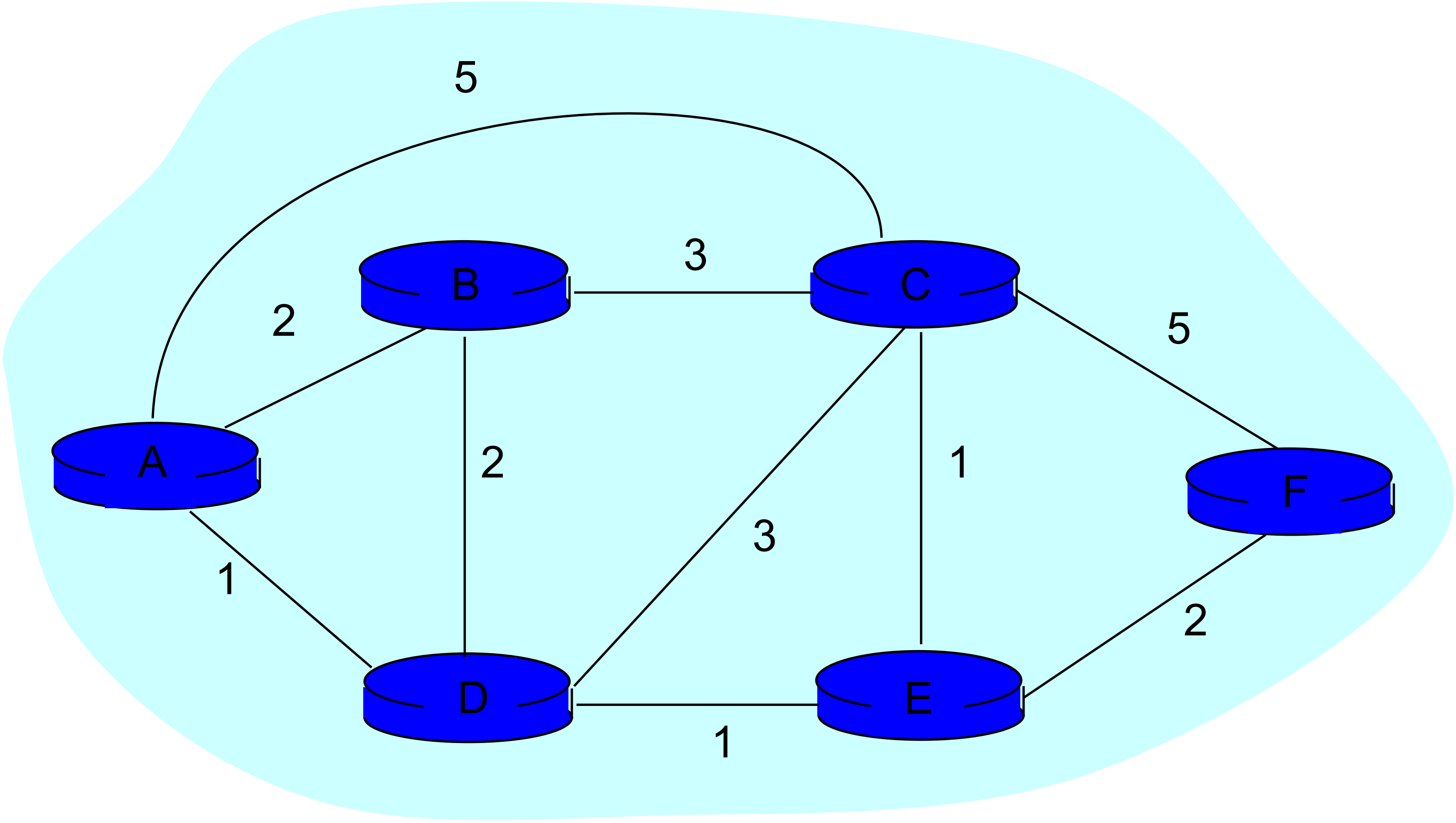
- Each node maintains its local "link state" (LS)
- Each node floods its local link state
- Hence, each node learns the entire network topology
  - Can use Dijkstra's to compute the shortest paths between nodes



Dijkstra’s Shortest Path Algorithm

- INPUT:
  - Network topology (graph), with link costs
- OUTPUT:
  - Least cost paths from one node to all other nodes
- Iterative: after *k* iterations, a node knows the least cost path to its *k* closest neighbors

Example



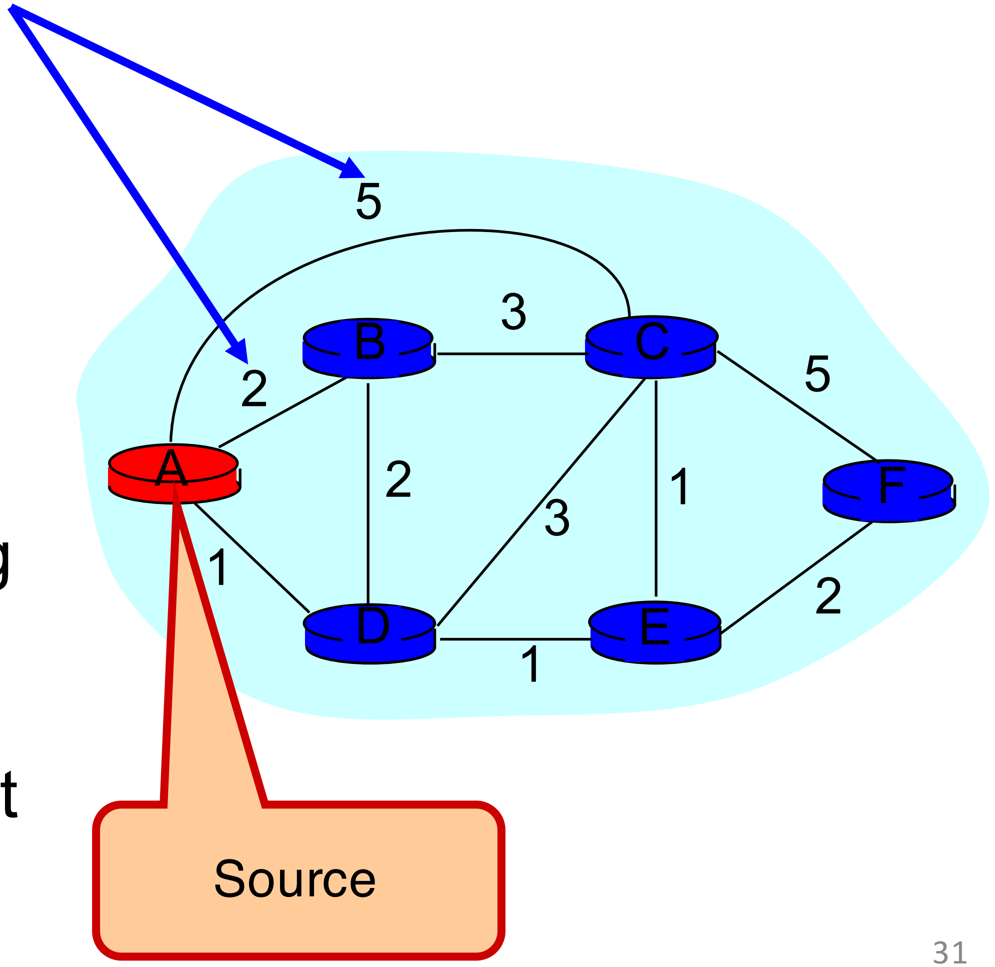
Notation

- c(i,j): link cost from node *i* to *j*; cost is infinite if not direct neighbors; **≥ 0**

- D(v): total cost of the current least cost path from source to destination *v*

- p(v): *v*’s predecessor along path from source to *v*

- S: set of nodes whose least cost path definitively known



Source

Dijkstra’s Algorithm

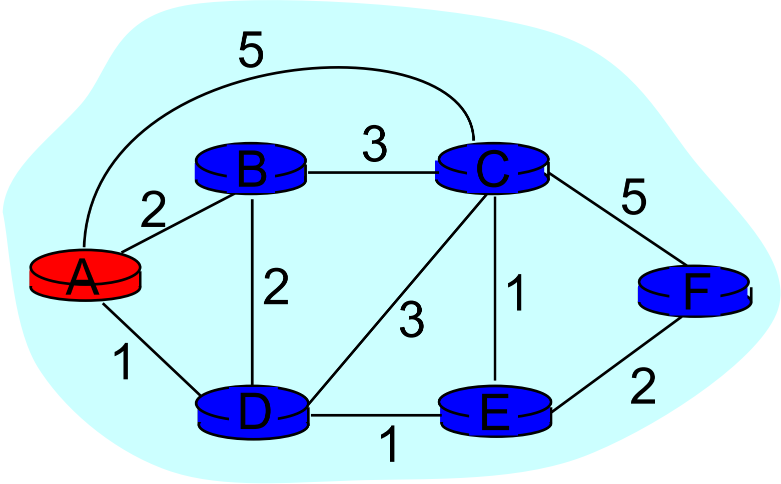
```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then $D(v) = c(A,v)$;
      else $D(v) = \infty$;

Loop
    find w not in S such that $D(w)$ is a minimum;
    add w to S;
    update $D(v)$ for all v adjacent to w and not in S:
      if $D(w) + c(w,v) < D(v)$ then
        // w gives us a shorter path to v than we’ve found so far
        $D(v) = D(w) + c(w,v)$; $p(v) = w$;
until all nodes in S;
```

- c(i,j): link cost from node *i* to *j*
- D(v): current cost source → *v*
- p(v): *v*’s predecessor along path from source to *v*
- S: set of nodes whose least cost path definitively known

Example: Dijkstra’s Algorithm

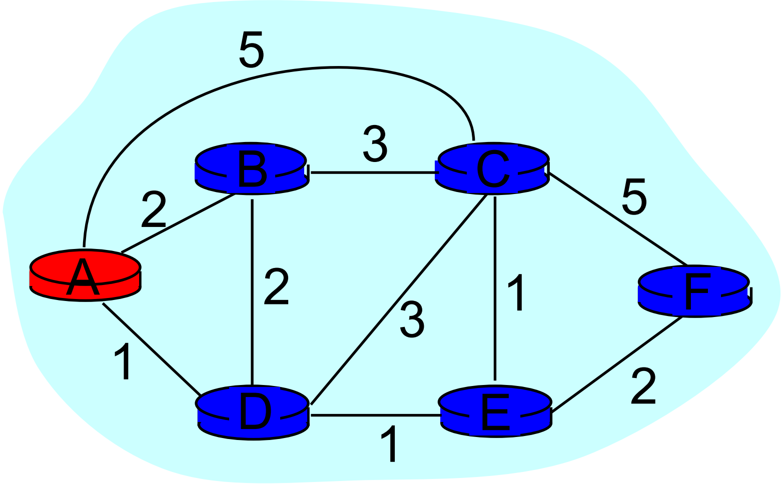
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v);
      else D(v) = ∞;
...
```

Example: Dijkstra’s Algorithm

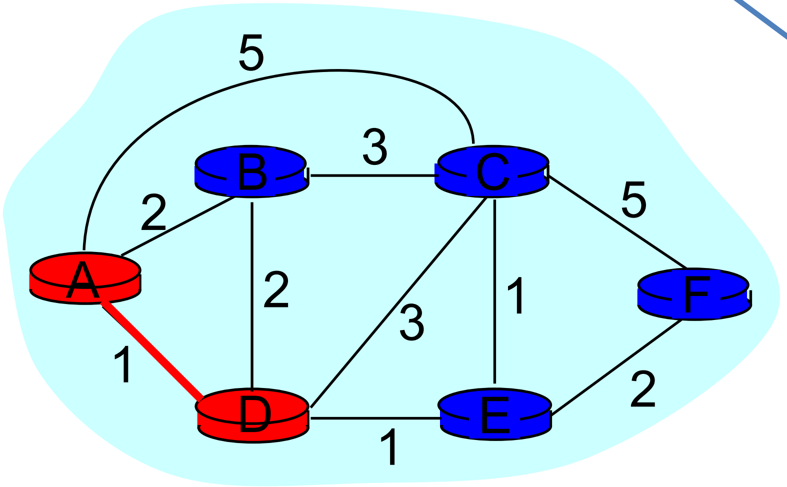
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



```
…
Loop
  find w not in S s.t. D(w) is a minimum;
  add w to S;
  update D(v) for all v adjacent
    to w and not in S:
  If D(w) + c(w,v) < D(v) then
     D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

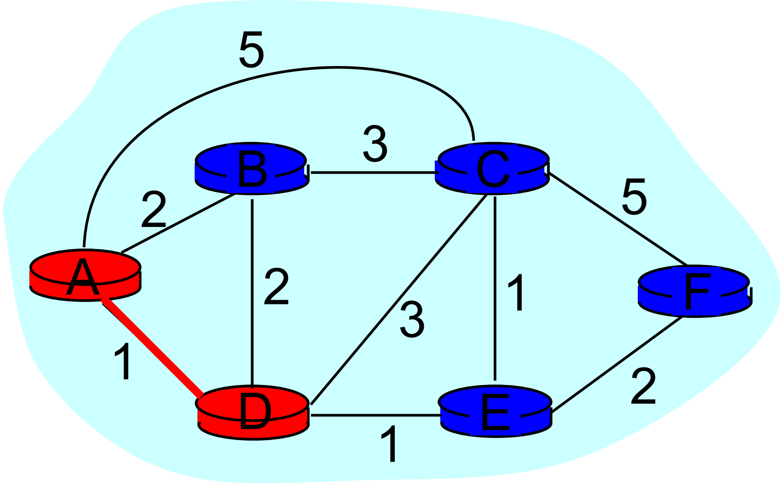
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
    until all nodes in S;
```

Example: Dijkstra’s Algorithm

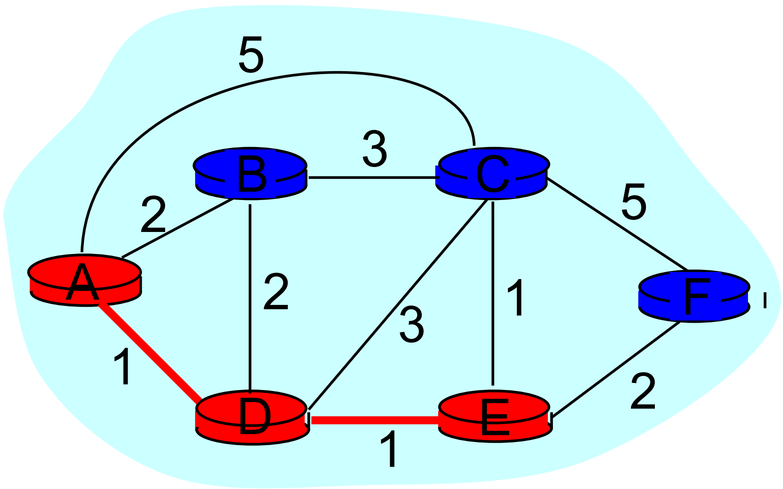
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2						
3						
4						
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
     to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
  until all nodes in S;
```

Example: Dijkstra’s Algorithm

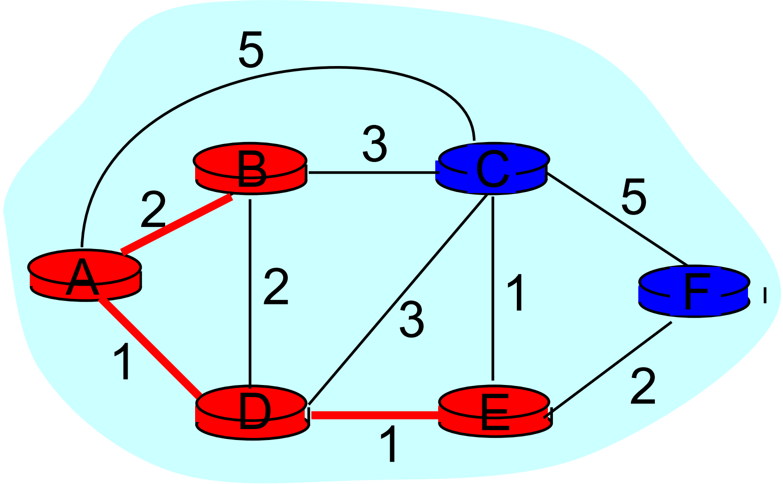
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3						
4						
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
 until all nodes in S;
```

Example: Dijkstra’s Algorithm

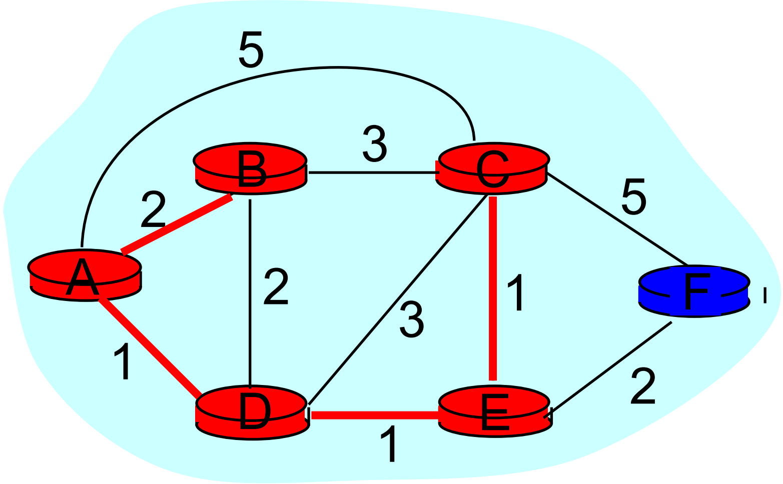
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4						
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent
      to w and not in S:
    If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

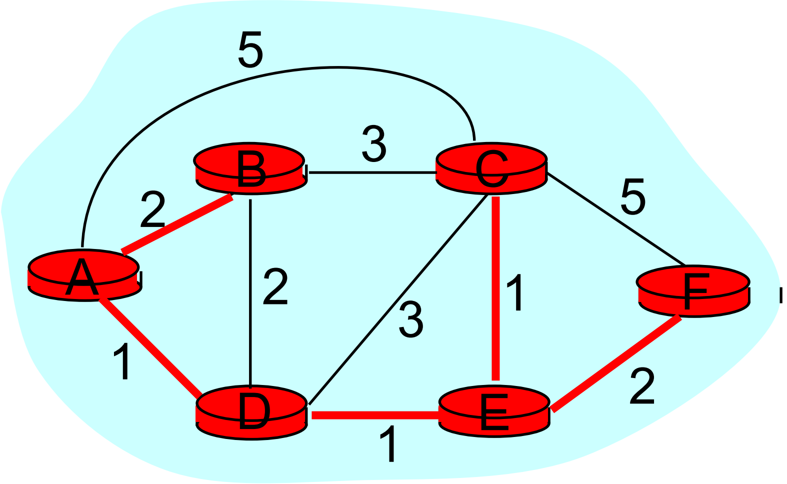
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
      D(v) = D(w) + c(w,v); p(v) = w;
 until all nodes in S;
```

Example: Dijkstra’s Algorithm

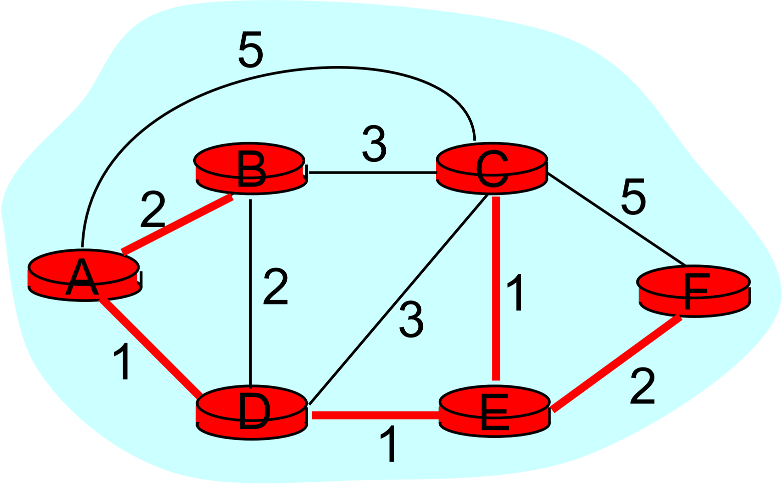
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent
      to w and not in S:
    If D(w) + c(w,v) < D(v) then
        D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

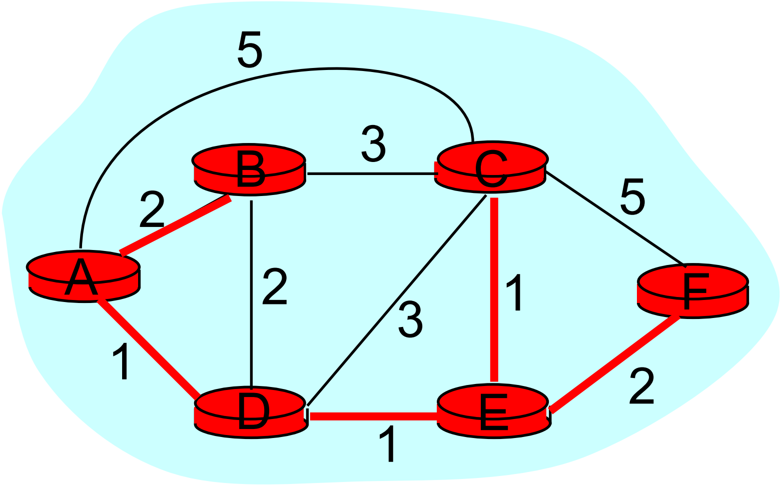
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



To determine path A → C (say), work backward from C via p(v)

The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the *forwarding table*



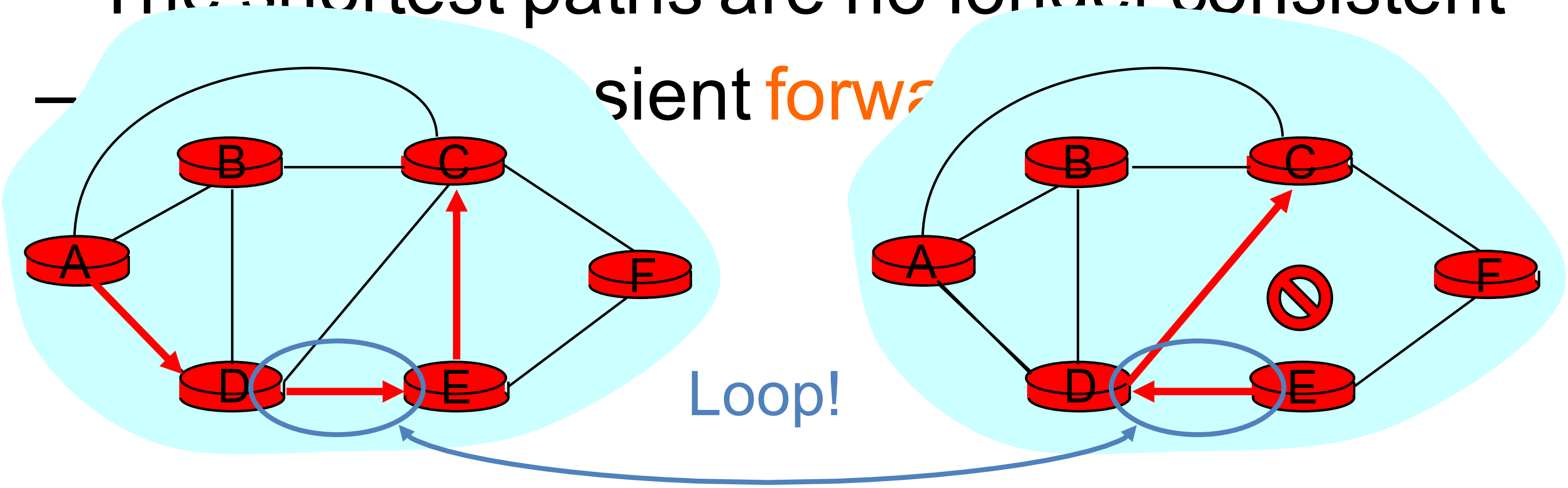
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

Issue #1: Scalability

- How many messages needed to flood link state messages?
  - O(N x E), where N is #nodes; E is #edges in graph
- Processing complexity for Dijkstra's algorithm?
  - O(N^2), because we check all nodes w not in S at each iteration and we have O(N) iterations
  - more efficient implementations: O(N log(N))
- How many entries in the LS topology database? O(E)
- How many entries in the forwarding table? O(N)

Issue#2: Transient Disruptions

- Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - sient forwa



A and D think that this is the path to C

E thinks that this is the path to C

Distance Vector

Learn-By-Doing

Let’s try to collectively develop
distance-vector routing from first principles

Experiment

- Your job: find the (route to) the youngest person in the room
- Ground Rules
  - **You may not** leave your seat, nor shout loudly across the class
  - **You may** talk with your immediate neighbors

(N-S-E-W only)

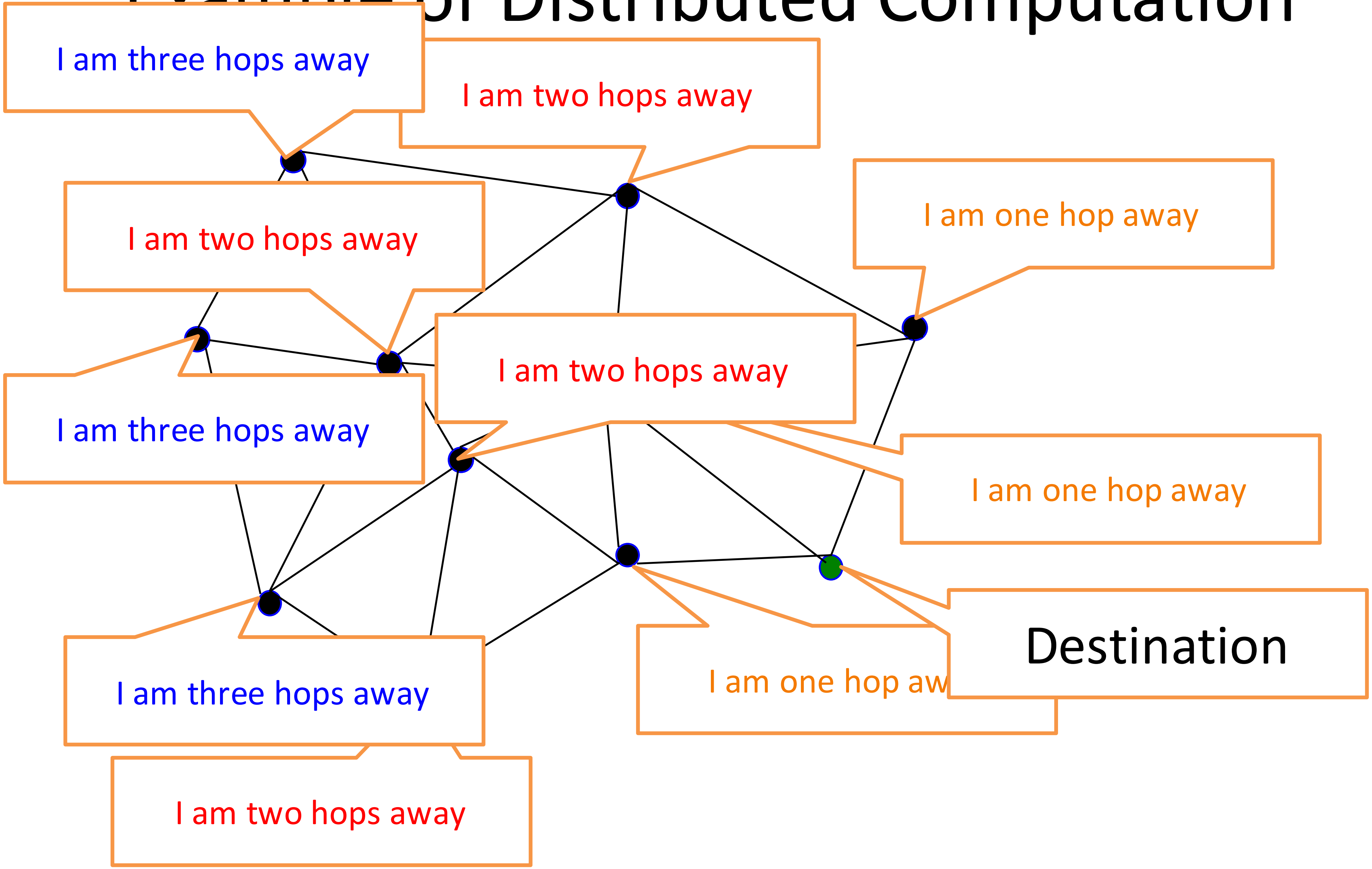
(hint: "exchange updates" with them)

- At the end of 5 minutes, I will pick a victim and ask:
  - who is the youngest person in the room? (date&name)
  - which one of your neighbors first told you this info.?

Go!

Distance-Vector

Example of Distributed Computation



Distance Vector Routing

- Each router knows the links to its neighbors
  - Does *not* flood this information to the whole network
- Each router has provisional "shortest path" to every other router
  - E.g.: Router A: "I can get to router B with cost 11"
- Routers exchange this distance vector information with their neighboring routers
  - Vector because one entry per destination
- Routers look over the set of options offered by their neighbors and select the best one
- Iterative process converges to set of shortest paths

A few other inconvenient truths

- What if we use a non-additive metric?
  - E.g., maximal capacity

- What if routers don’t use the same metric?
  - I want low delay, you want low loss rate?

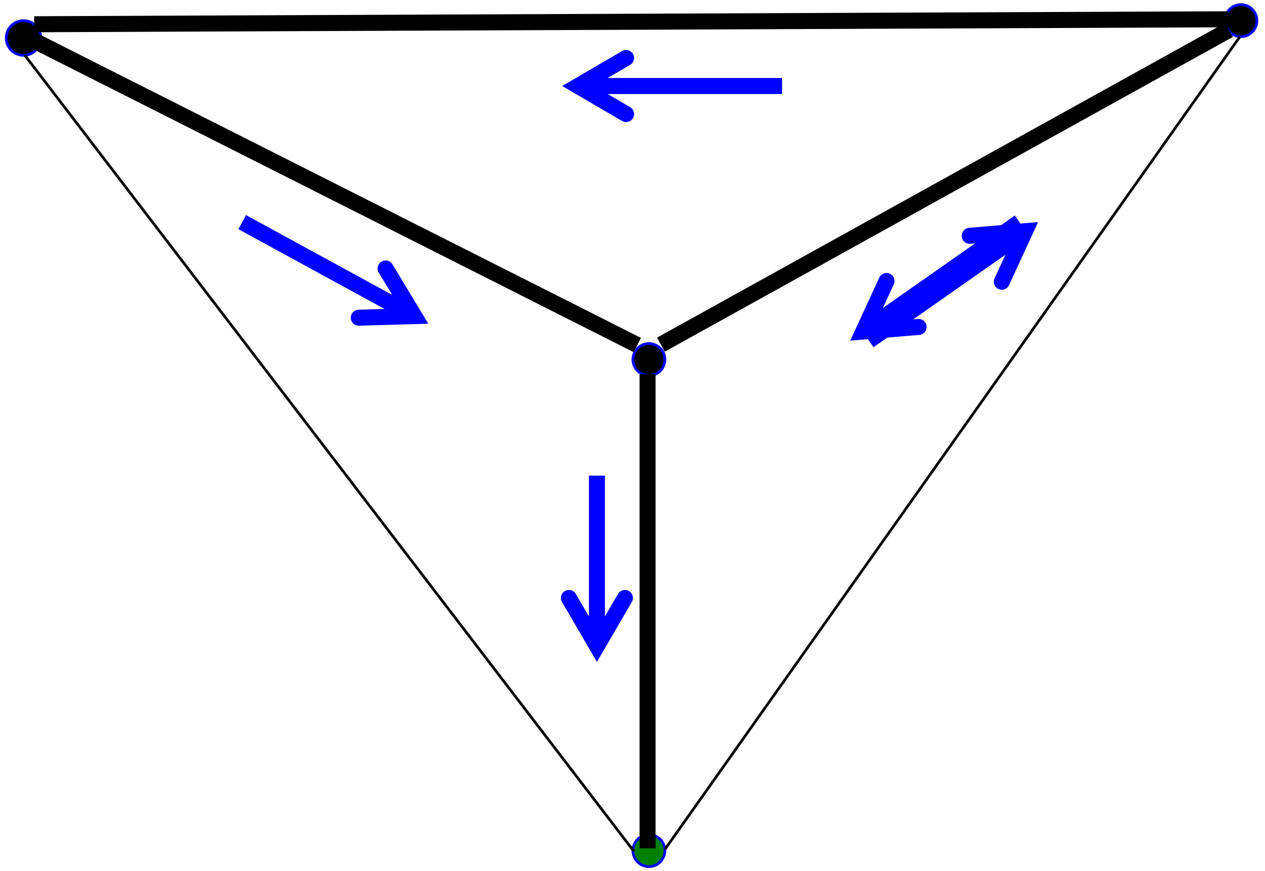
- What happens if nodes lie?

Can You Use Any Metric?

- I said that we can pick any metric. Really?
- What about maximizing capacity?

What Happens Here?

Problem:"cost" does not change around loop



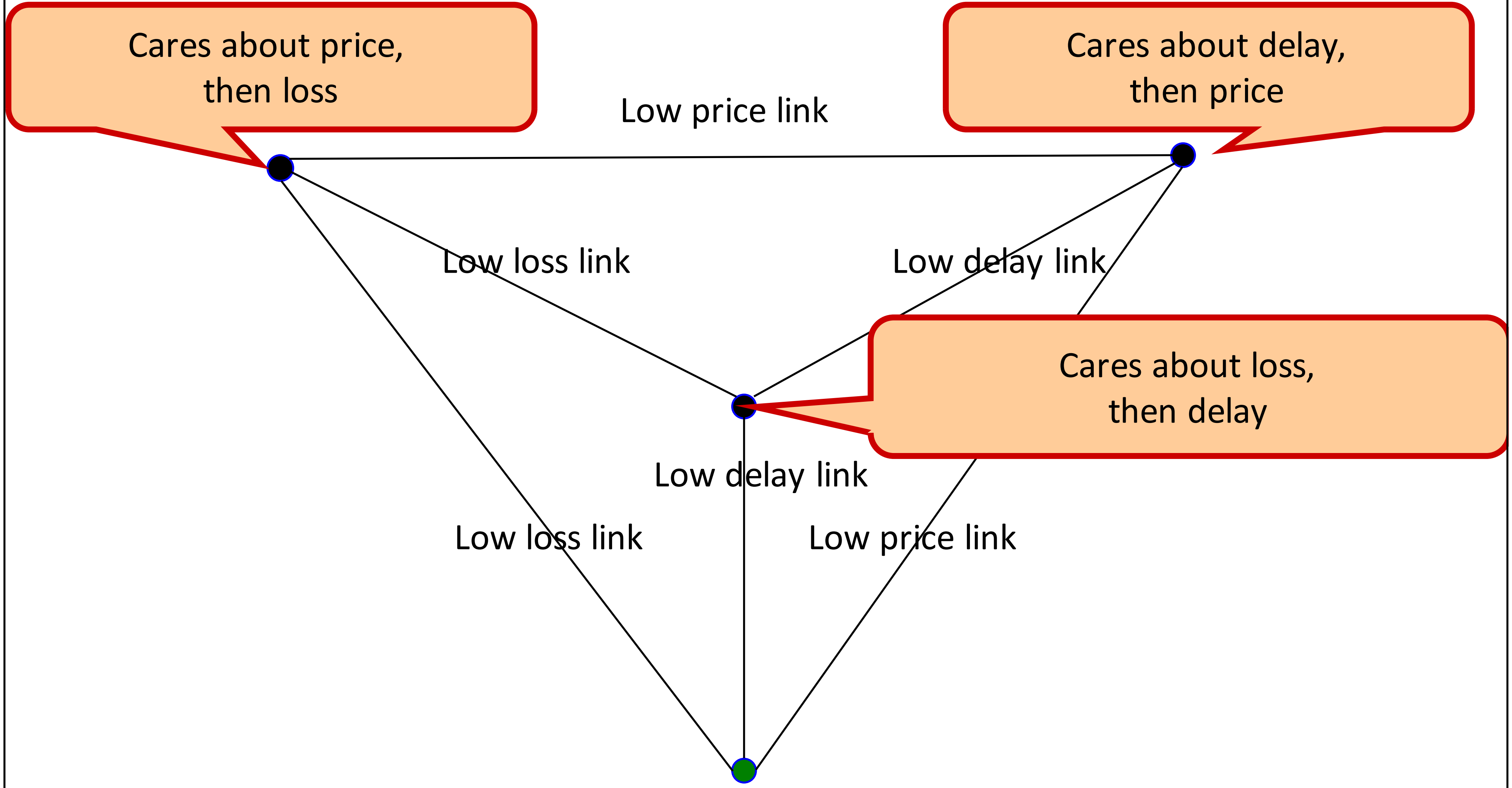
Additive measures avoid this problem!

No agreement on metrics?

- If the nodes choose their paths according to different criteria, then bad things might happen
- Example
  - Node A is minimizing latency
  - Node B is minimizing loss rate
  - Node C is minimizing price
- Any of those goals are fine, if globally adopted
  - Only a problem when nodes use different criteria

- Consider a routing algorithm where paths are described by delay, cost, loss

What Happens Here?



Cares about price, then loss

Cares about delay, then price

Cares about loss, then delay

Must agree on loop-avoiding metric

- When all nodes minimize same metric
- And that metric increases around loops
- Then process is guaranteed to converge

What happens when routers lie?

- What if a router claims a 1-hop path to everywhere?
- All traffic from nearby routers gets sent there
- How can you tell if they are lying?
- Can this happen in real life?
  - It has, several times….

Link State vs. Distance Vector

- Core idea
  - LS: tell all nodes about your immediate neighbors
  - DV: tell your immediate neighbors about (your least cost distance to) all nodes

Link State vs. Distance Vector

- LS: each node learns the complete network map; each node computes shortest paths independently and in parallel
- DV: no node has the complete picture; nodes cooperate to compute shortest paths in a distributed manner

→LS has higher messaging overhead

→LS has higher processing complexity

→LS is less vulnerable to looping

Link State vs. Distance Vector

Message complexity

- LS: O(NxE) messages;
  - N is #nodes; E is #edges
- DV: O(#Iterations x E)
  - where #Iterations is ideally O(network diameter) but varies due to routing loops or the count-to-infinity problem

Processing complexity

- LS: O(N^2)
- DV: O(#Iterations x N)

Robustness: what happens if router malfunctions?

- LS:
  - node can advertise incorrect *link* cost
  - each node computes only its *own* table
- DV:
  - node can advertise incorrect *path* cost
  - each node's table used by others; error propagates through network

Routing: Just the Beginning

- Link state and distance-vector are the deployed routing paradigms for intra-domain routing

- Inter-domain routing (BGP)
  – more Part II (Principles of Communications)
  – A version of DV

What are desirable goals for a routing solution?

- “Good” paths (least cost)
- Fast convergence after change/failures
  - no/rare loops
- Scalable
  - #messages
  - table size
  - processing complexity
- Secure
- Policy
- Rich metrics (more later)

Delivery models

- What if a node wants to send to more than one destination?
  - broadcast: send to all
  - multicast: send to all members of a group
  - anycast: send to any member of a group

- What if a node wants to send along more than one path?

Metrics

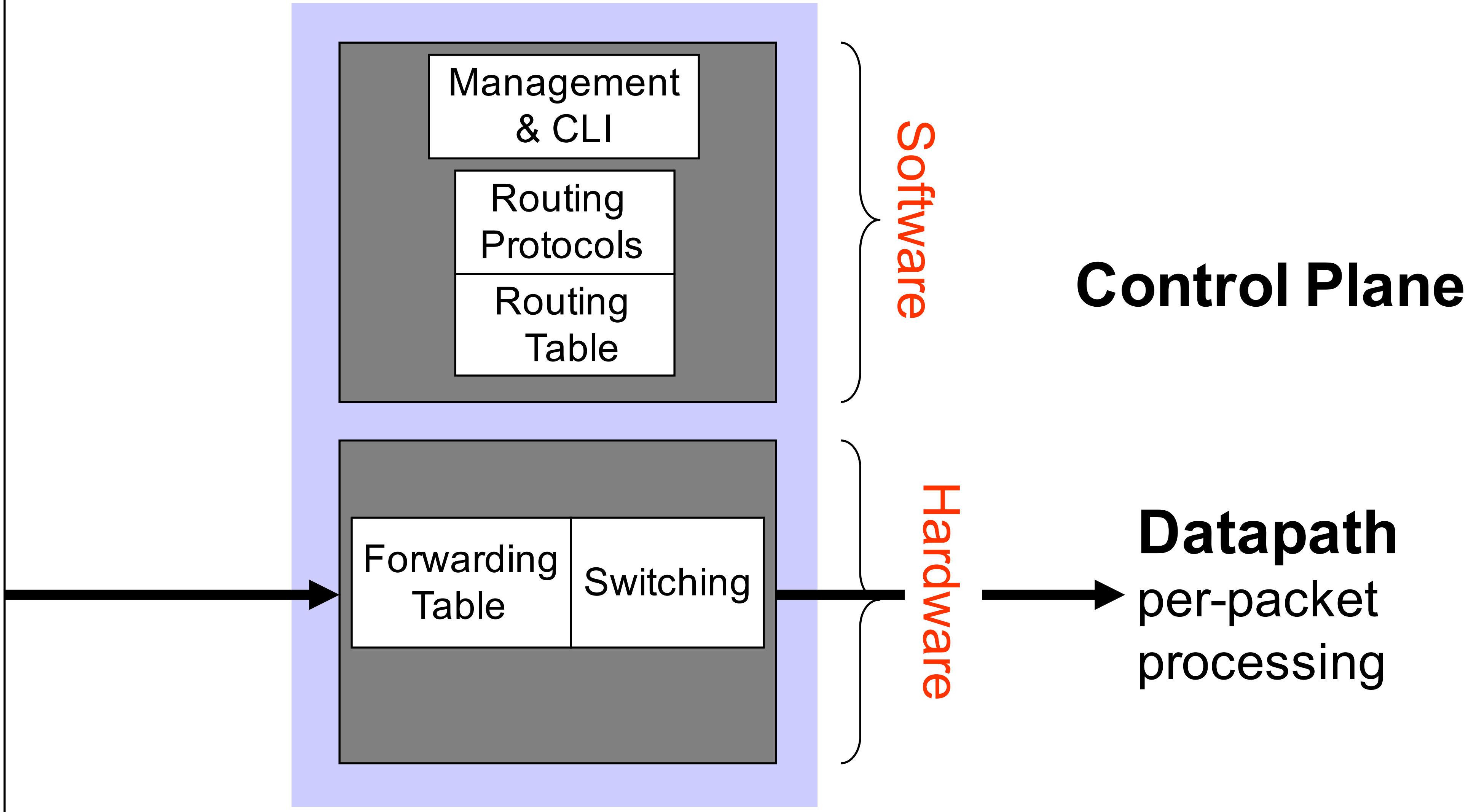
- Propagation delay
- Congestion
- Load balance
- Bandwidth (available, capacity, maximal, bbw)
- Price
- Reliability
- Loss rate
- Combinations of the above

In practice, operators set abstract “weights” (much like our costs); how exactly is a bit of a black art

From Routing back to Forwarding

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

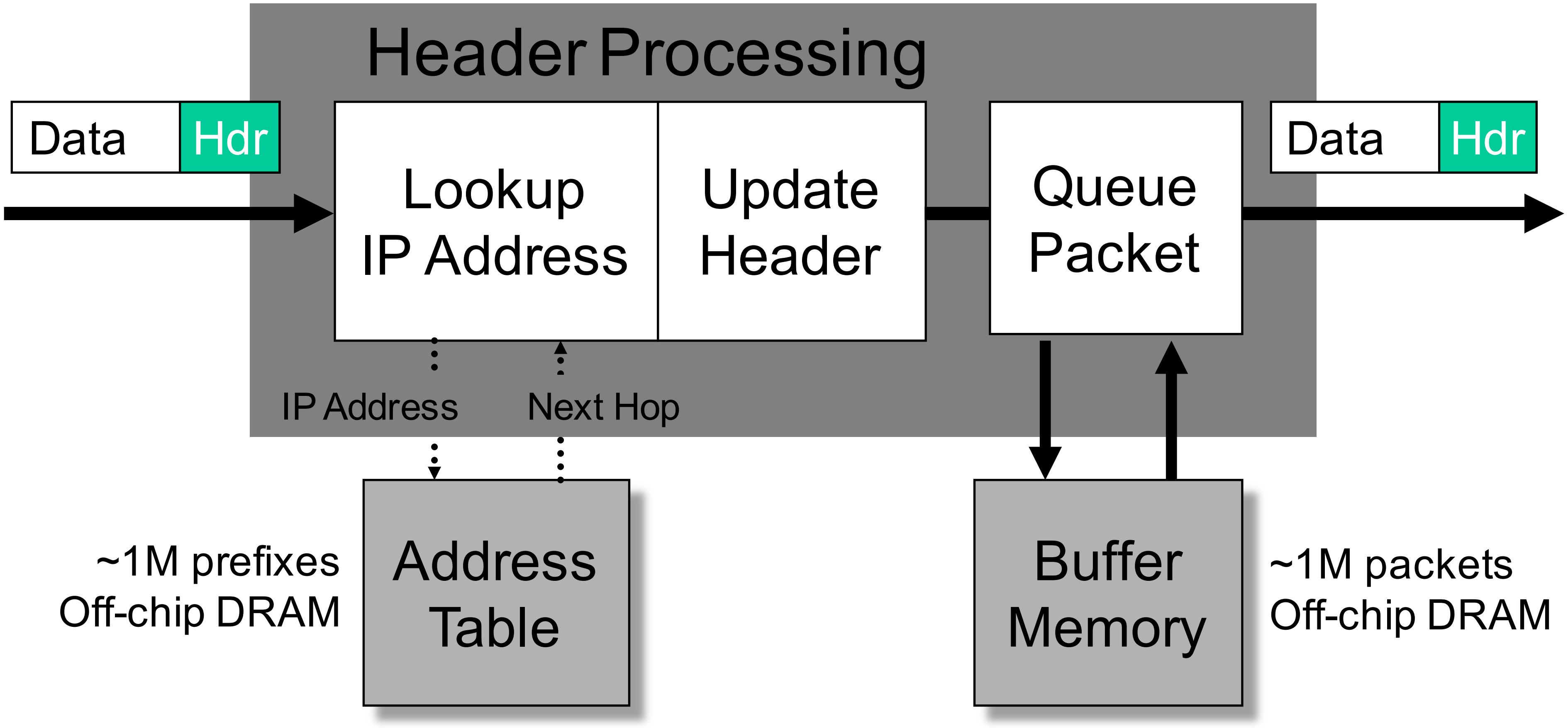
Basic Architectural Components of an IP Router



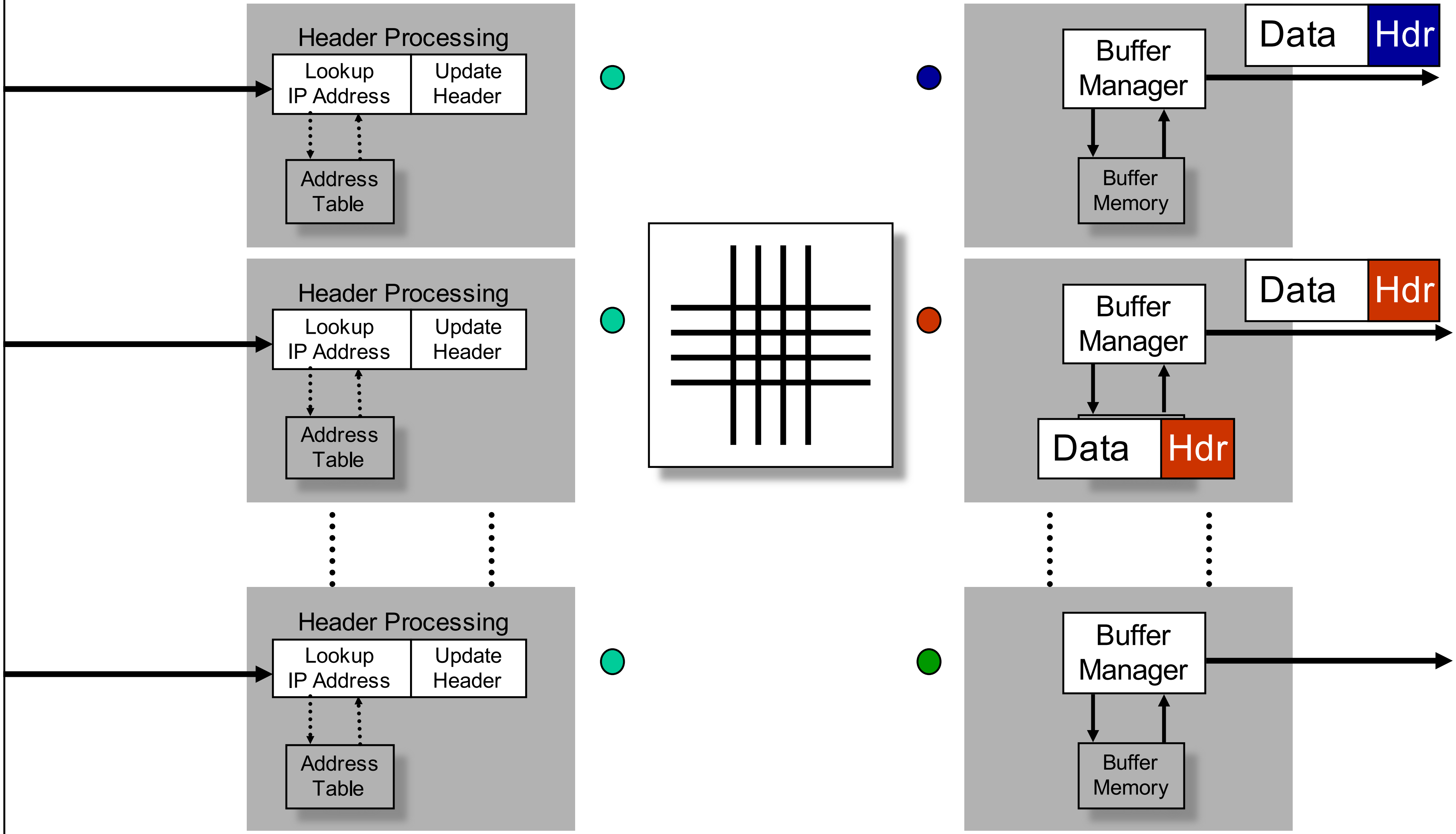
Per-packet processing in an IP Router

1. Accept packet arriving on an incoming link.
2. Lookup packet destination address in the forwarding table, to identify outgoing port(s).
3. Manipulate packet header: e.g., decrement TTL, update header checksum.
4. Send packet to the outgoing port(s).
5. Buffer packet in the queue.
6. Transmit packet onto outgoing link.

Generic Router Architecture



Generic Router Architecture



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

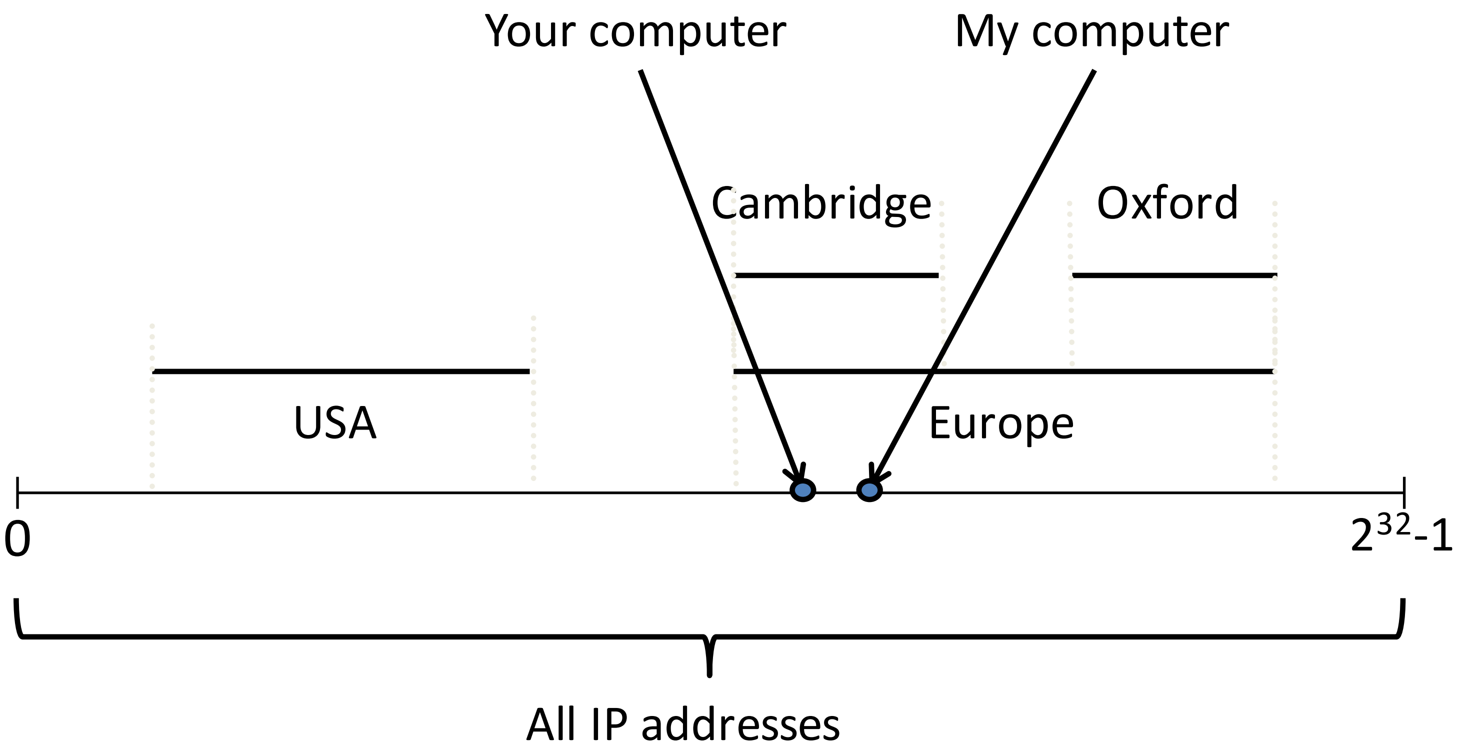
~ 4 billion entries

Improved approach:

Group entries to reduce table size

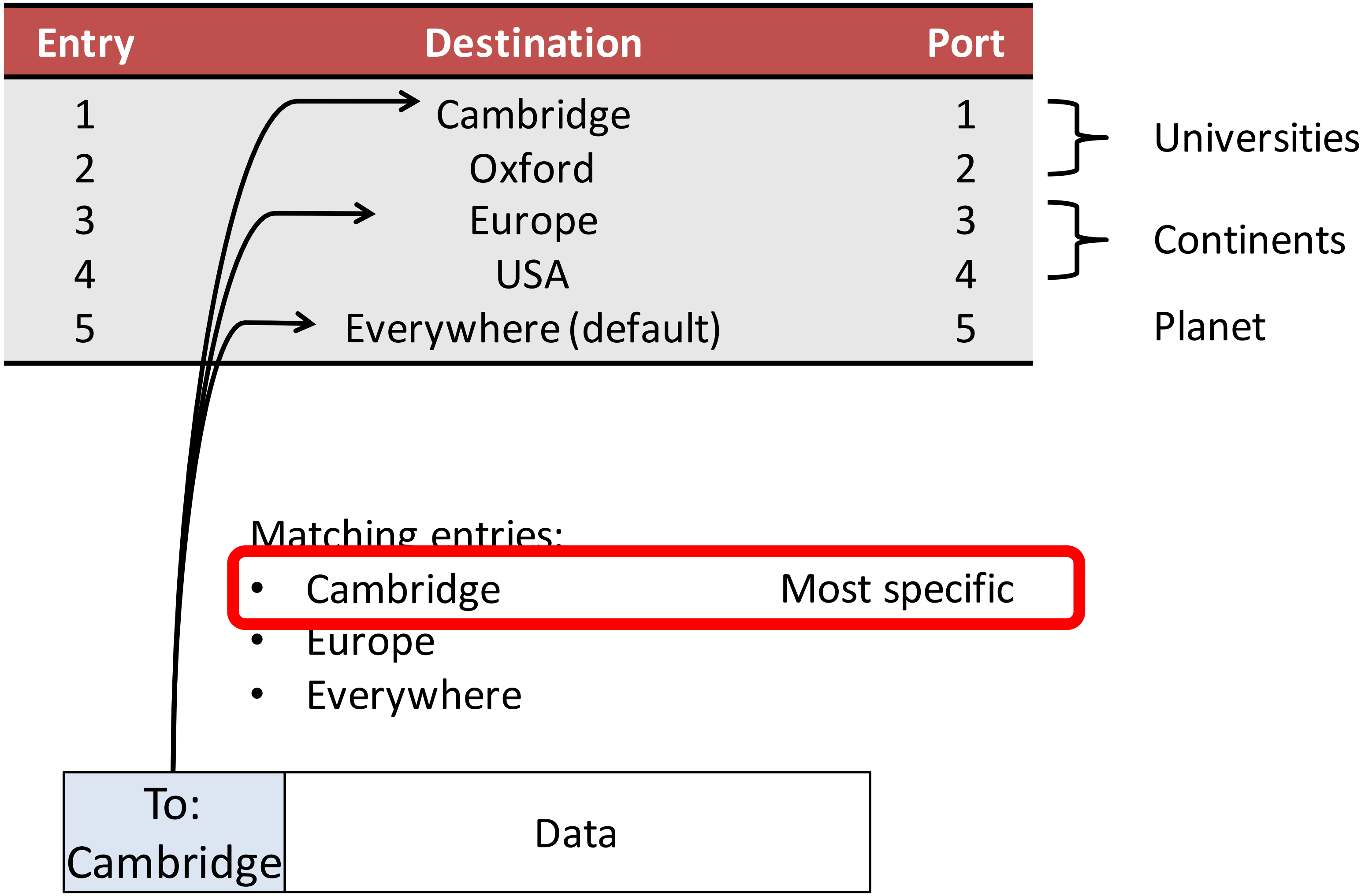
Entry	Destination	Port
1	0.0.0.0 – 127.255.255.255	1
2	128.0.0.1 – 128.255.255.255	2
⋮	⋮	⋮
50	248.0.0.0 – 255.255.255.255	12

IP addresses as a line



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

Longest Prefix Match (LPM)

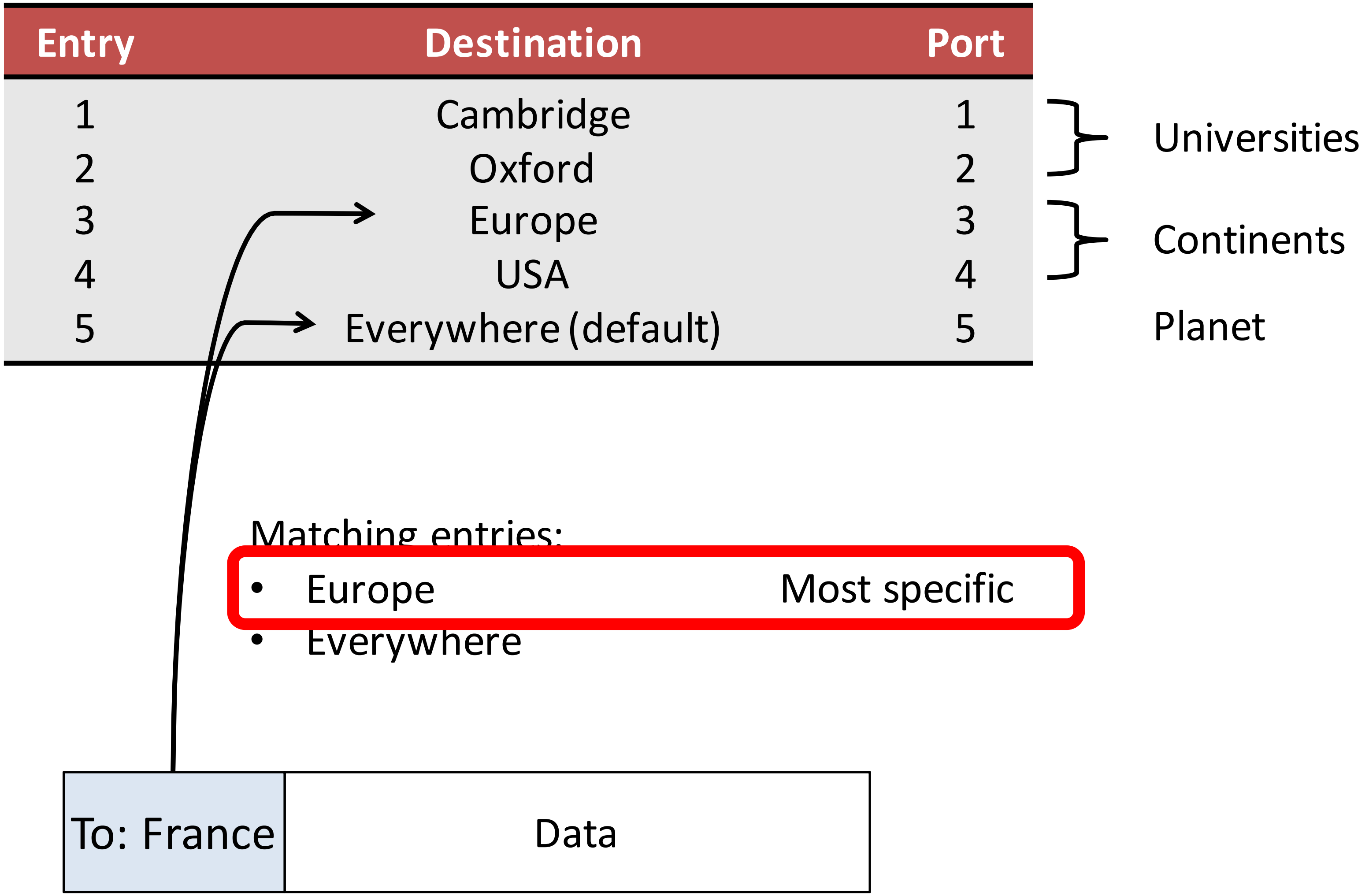


Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

Matching entries:

- Cambridge Most specific
- Europe
- Everywhere

Longest Prefix Match (LPM)



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

Matching entries:

- Europe Most specific
- Everywhere

Implementing Longest Prefix Match

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

Searching

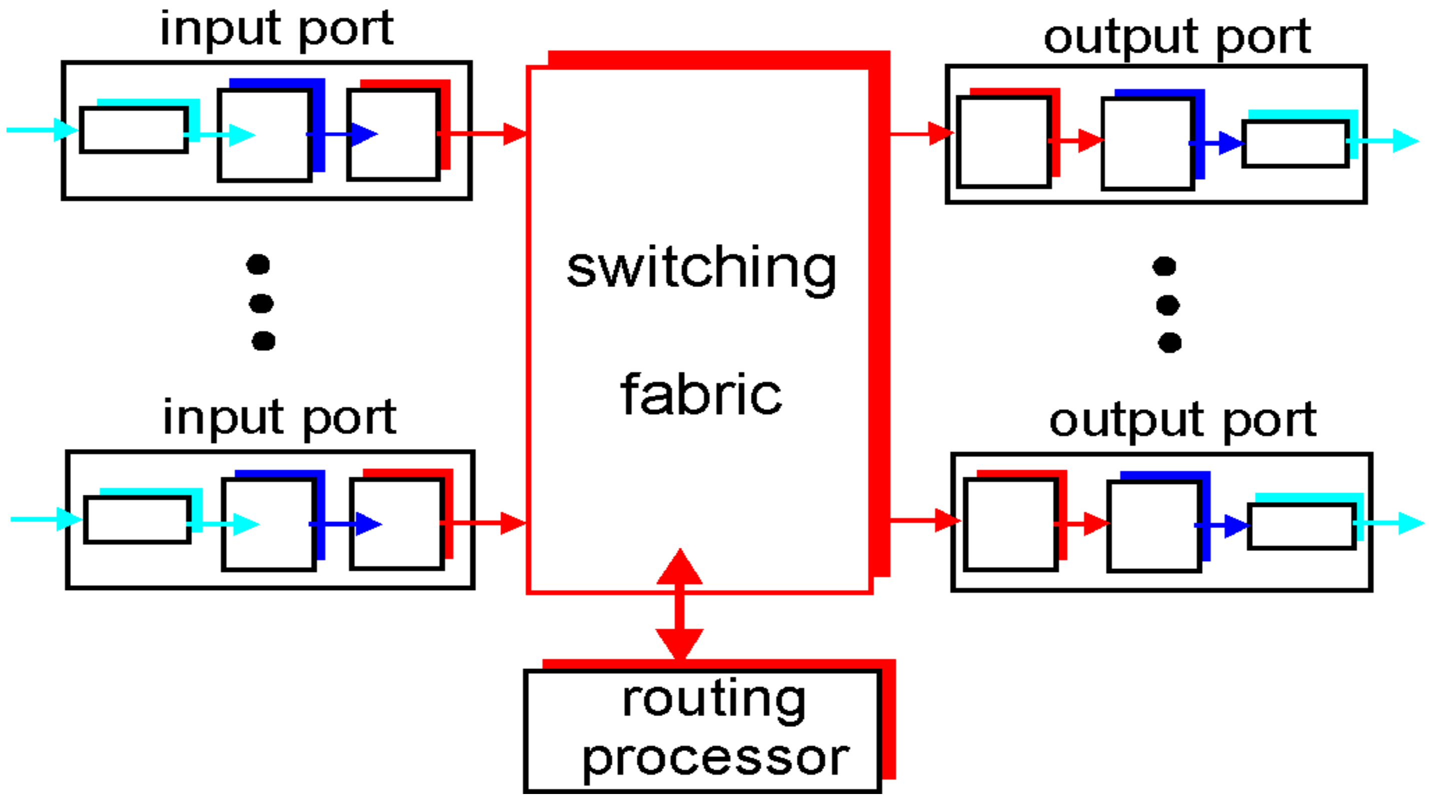
FOUND

Most specific
↓
Least specific

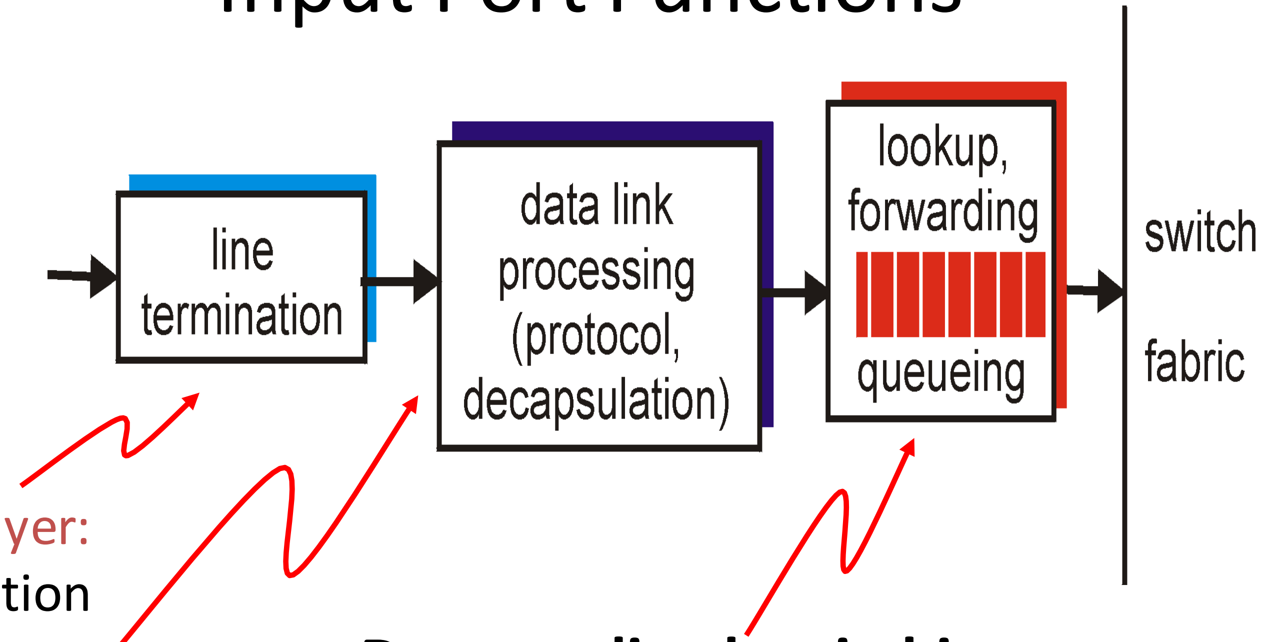
Router Architecture Overview

Two key router functions:

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



Input Port Functions



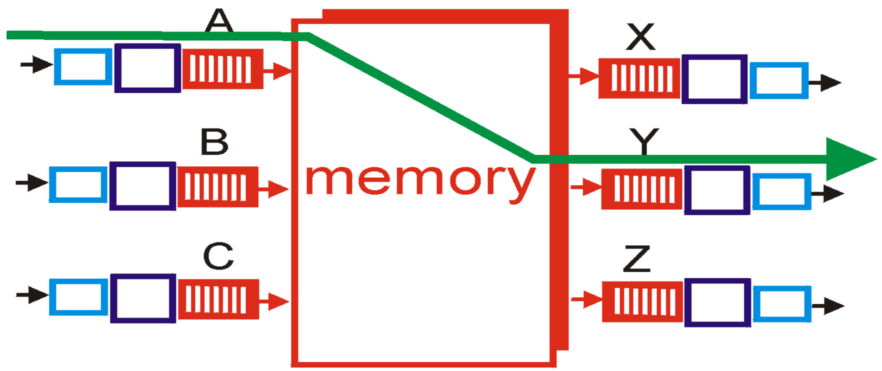
Physical layer:
bit-level reception

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

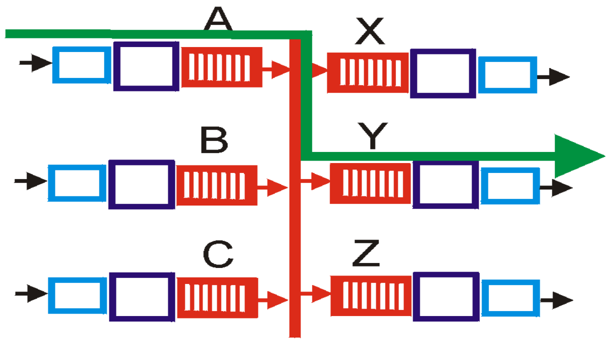
Decentralized switching*:*

- 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

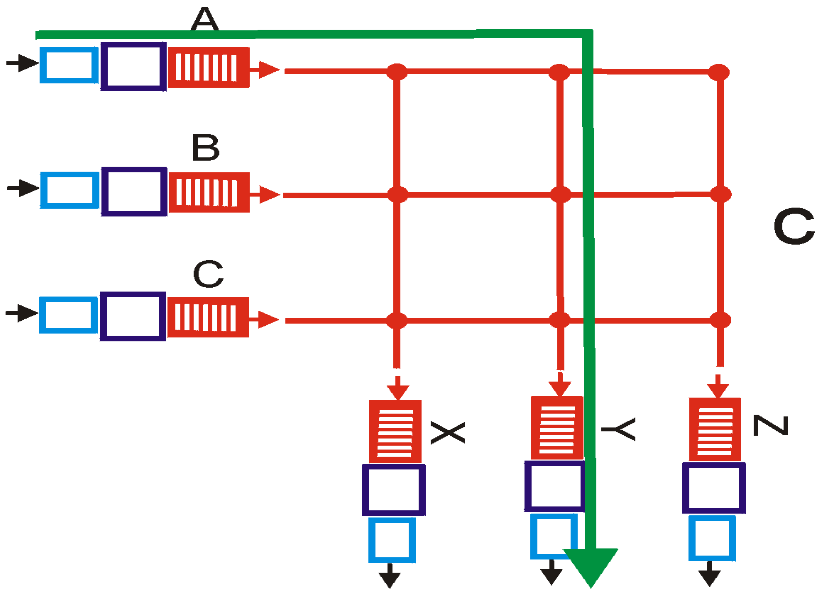
Three examples of switching fabrics (comparison criteria: speed, contention, complexity)



memory



bus

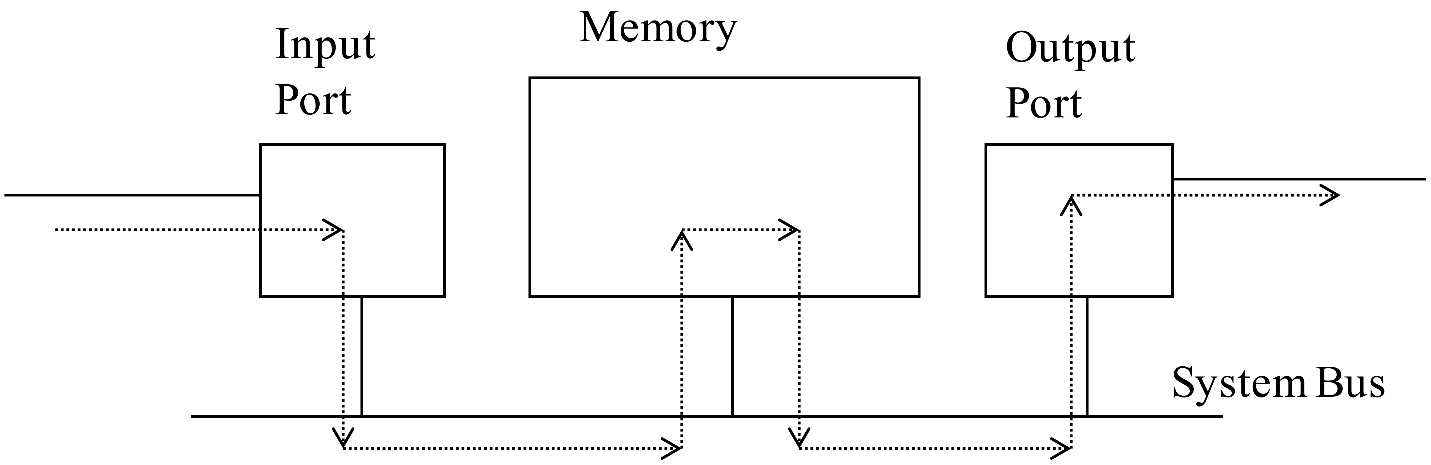


crossbar

Switching Via Memory

First generation routers:

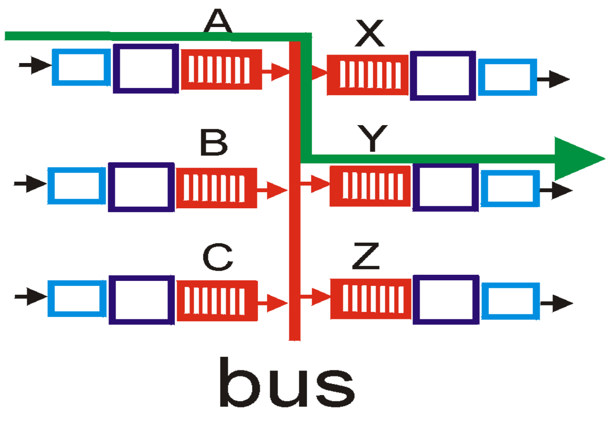
- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



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 =

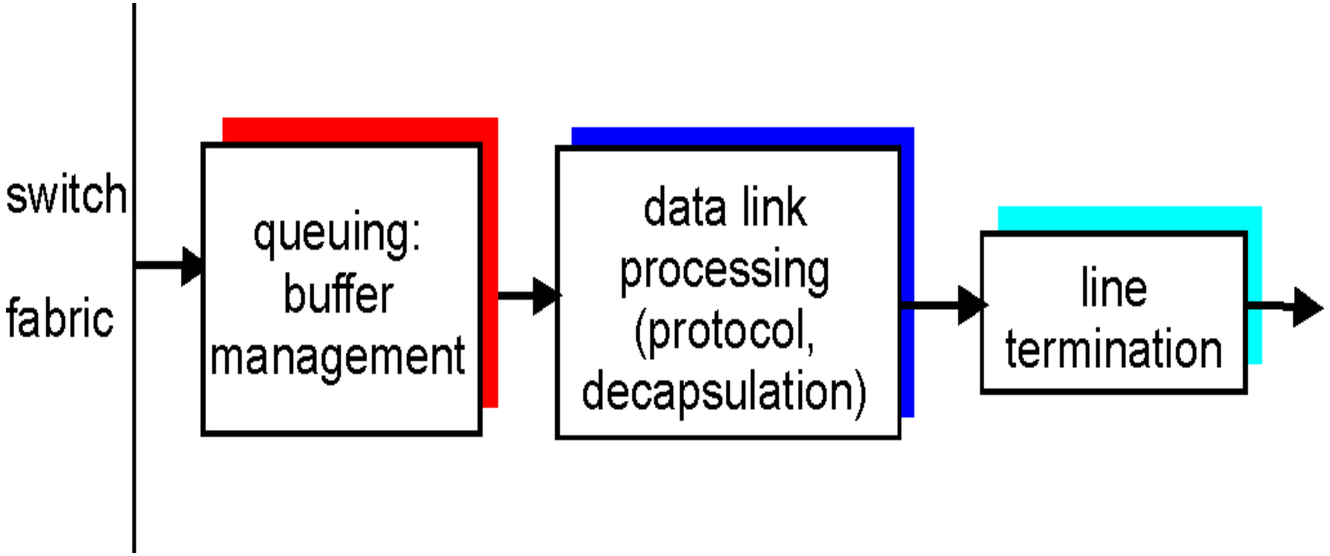
  2 x port speed x port count

- 32 Gbps bus, Cisco 5600: sufficient speed for access routers

Switching Via An Interconnection Network

- overcome bus bandwidth limitations
- Banyan networks, other interconnection nets initially developed to connect processors in multiprocessor stages
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco CRS-1: switches 1.2 Tbps through the interconnection network

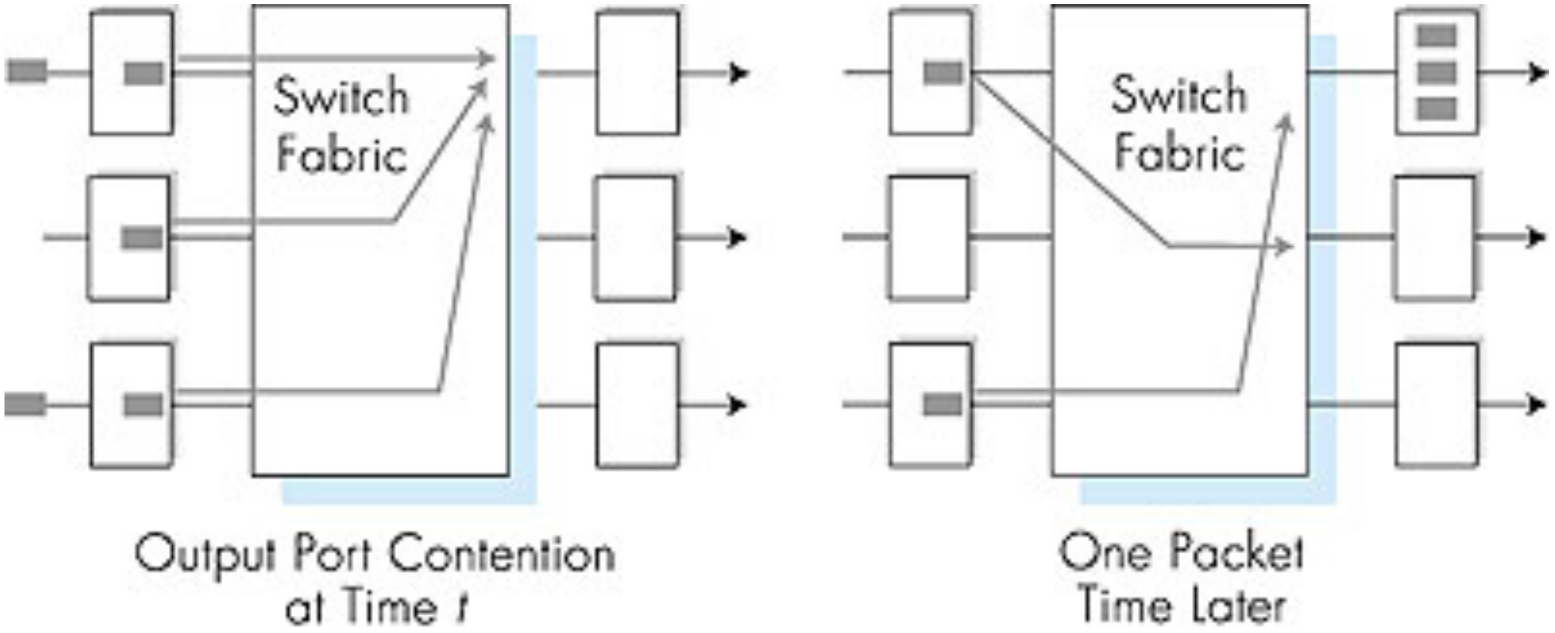
Output Ports



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

  ➔ Who goes next?

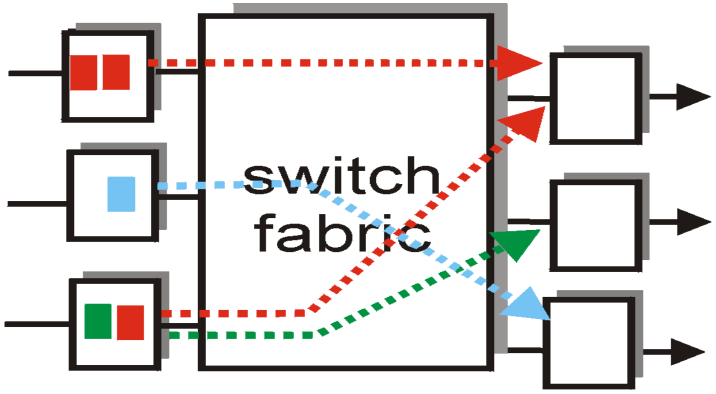
Output port queueing



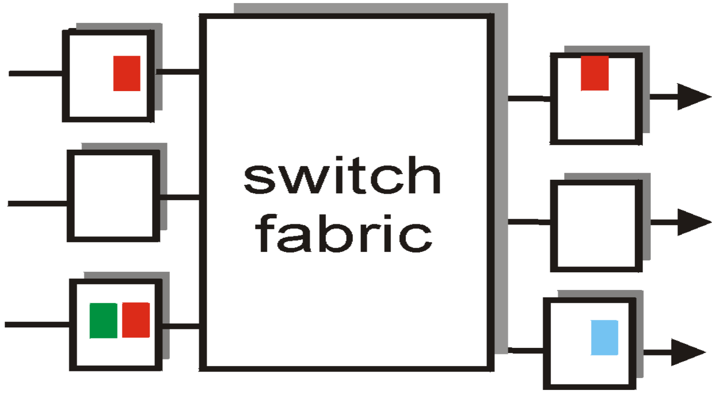
- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

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!*



output port contention
at time t - only one red
packet can be transferred



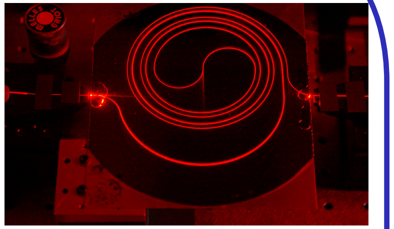
green packet
experiences HOL blocking

Buffers in Routers

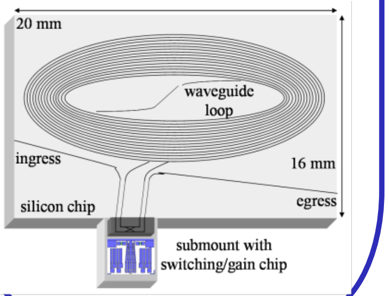
- So how large should the buffers be?

Buffer size matters

- End-to-end delay
  - Transmission, propagation, and queueing del
  - The only variable part is queueing delay
- Router architecture
  - Board space, power consumption, and cost
  - 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……

Buffer Sizing Story

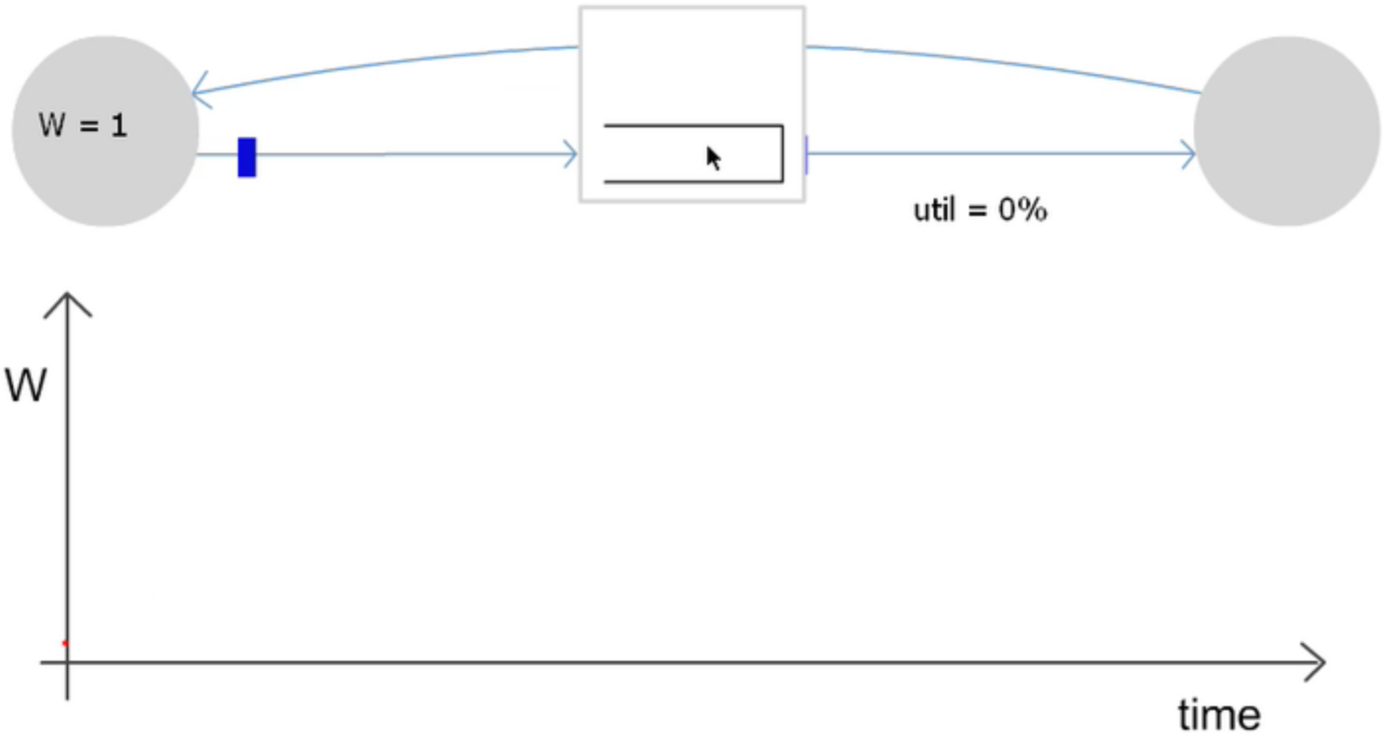
	Rule-of-thumb	Small Buffers	Tiny Buffers
	$2T \times C$	$\frac{2T \times C}{\sqrt{n}}$	$O(\log W)$
# of packets	1,000,000	10,000	20 - 50
Intuition	TCP Sawtooth	Sawtooth Smoothing	Non-bursty Arrivals
Assume	Single TCP Flow, 100% Utilization	Many Flows, 100% Utilization	Paced TCP, 85-90% Utilization
Evidence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

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

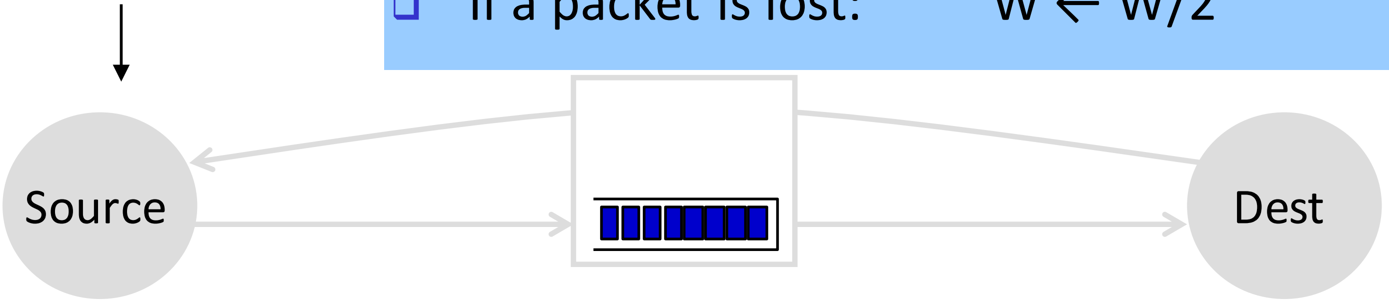


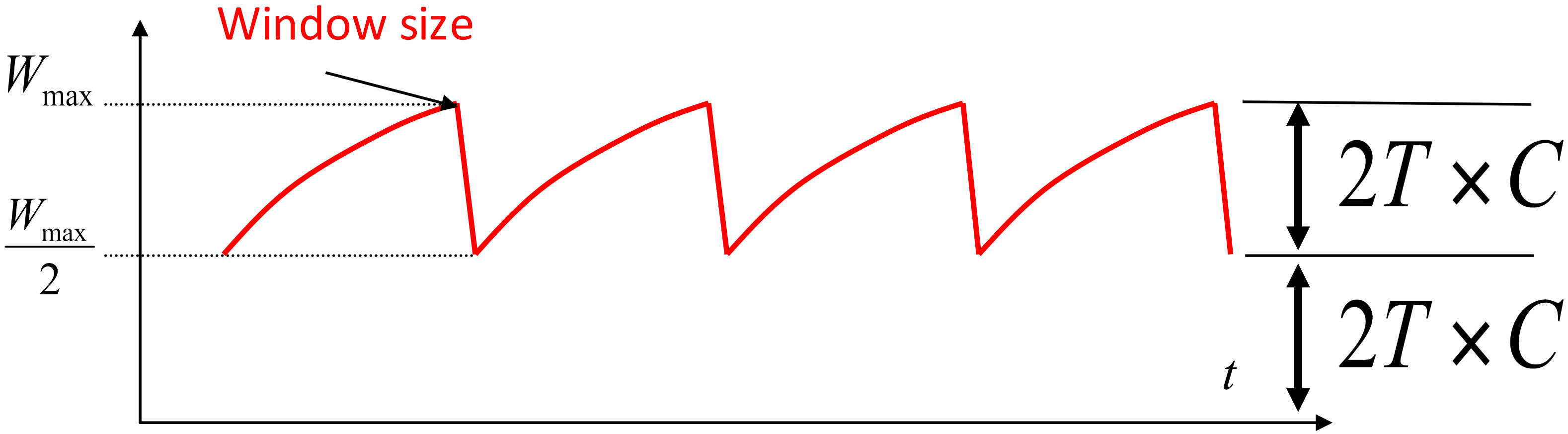
Rule-of-thumb – Intuition

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





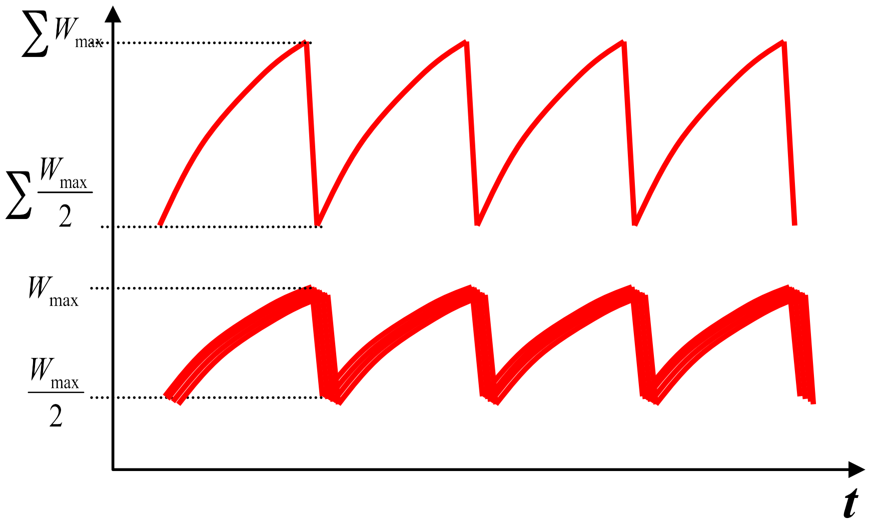
Small Buffers – Intuition

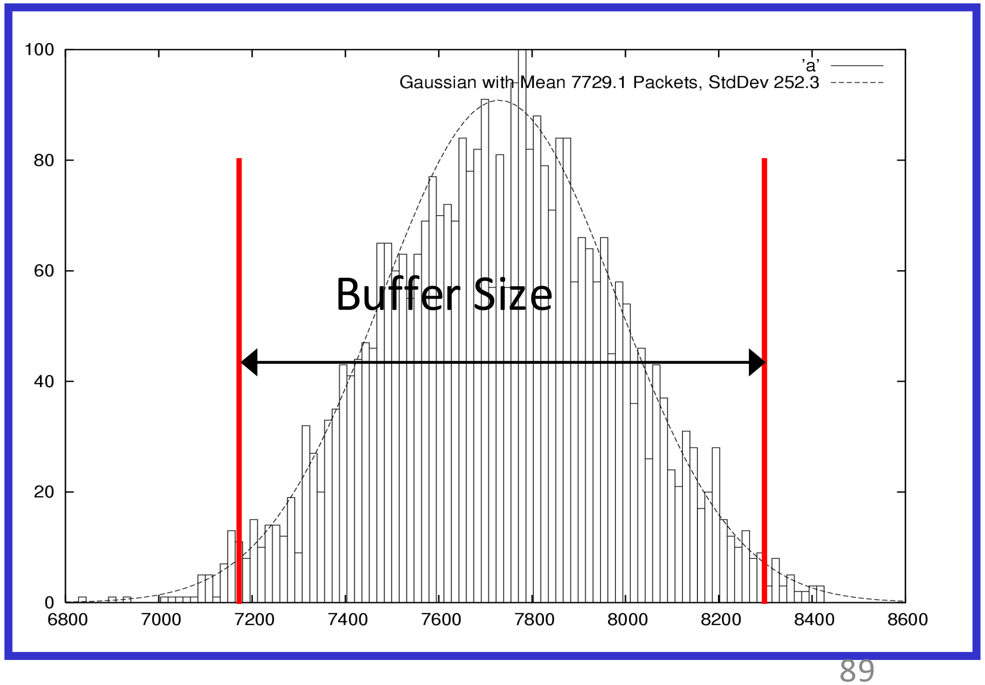
Synchronized Flows

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

Many TCP Flows

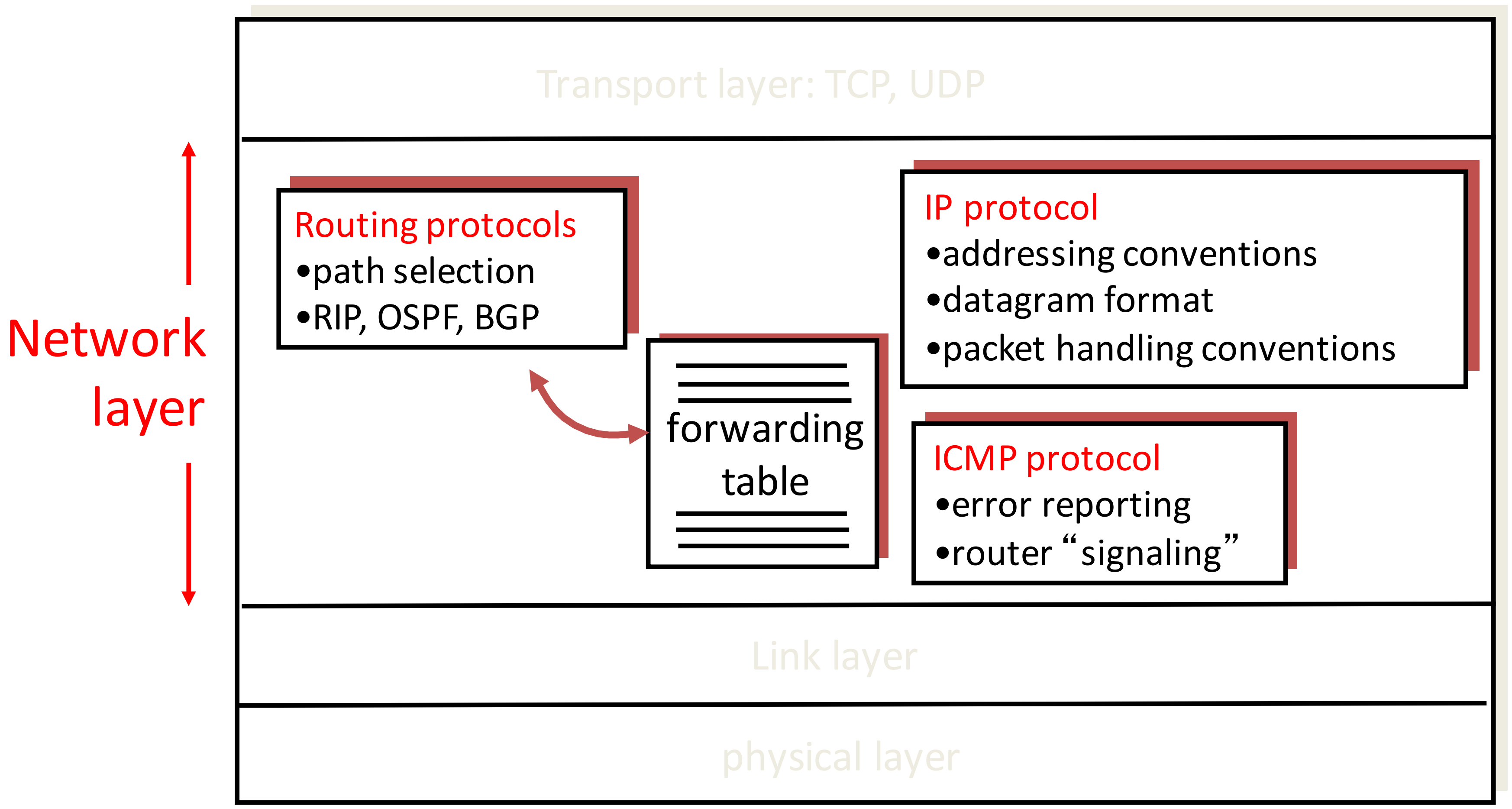
- **Independent, desynchronized**
- **Central limit theorem says the aggregate becomes Gaussian**
- **Variance (buffer size) decreases as *N* increases**





The Internet version of a Network layer

Host, router network layer functions:



Transport layer: TCP, UDP

Network layer

Routing protocols
- path selection
- RIP, OSPF, BGP

forwarding table

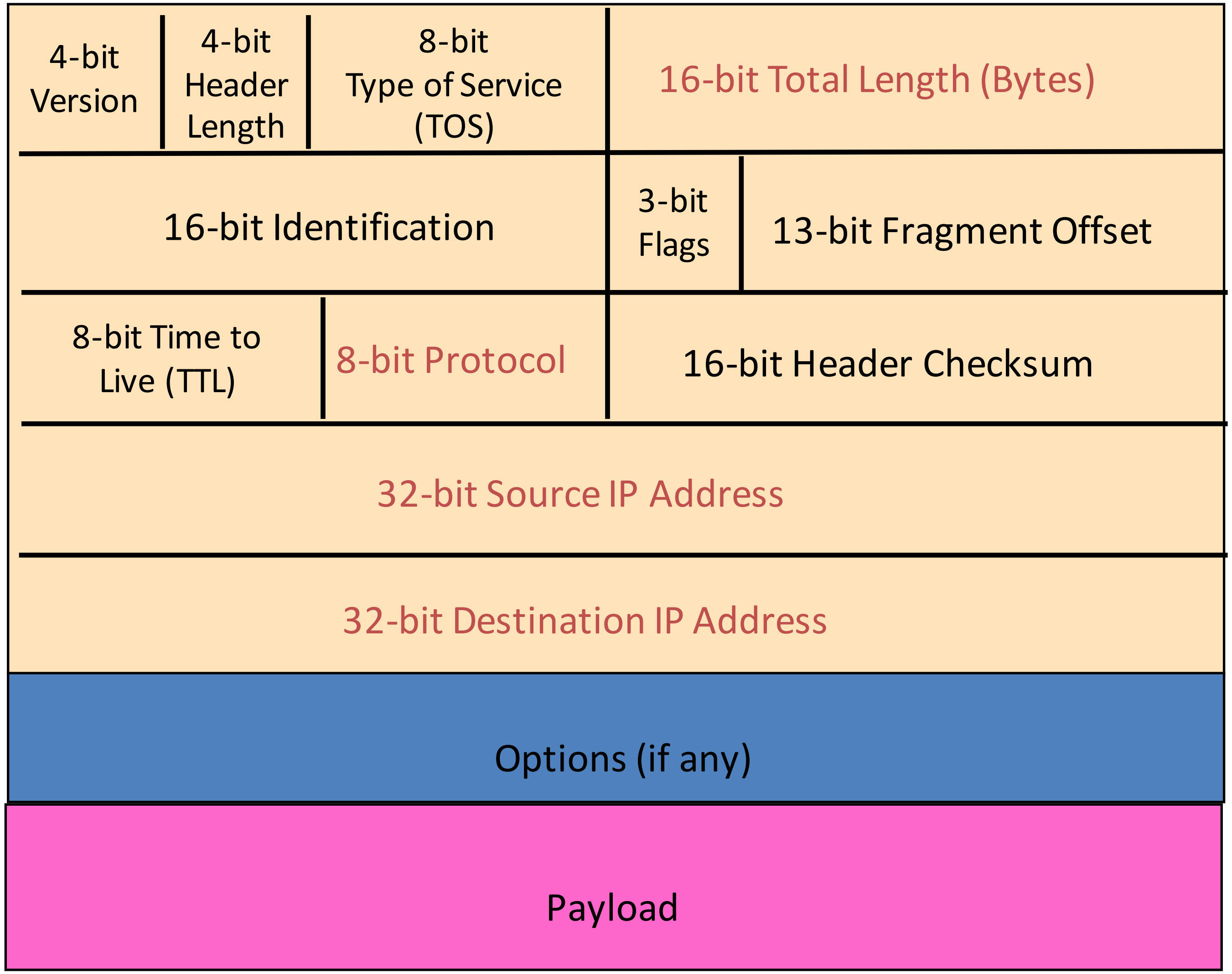
IP protocol
- addressing conventions
- datagram format
- packet handling conventions

ICMP protocol
- error reporting
- router “signaling”

Link layer

physical layer

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

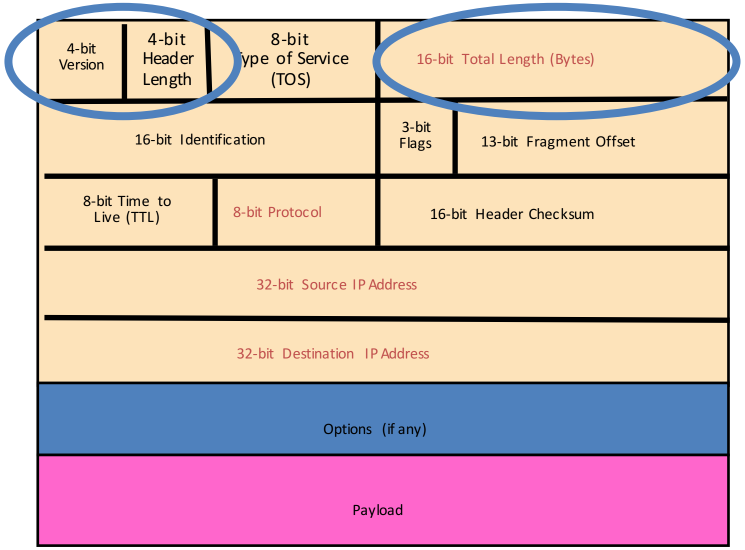
Options (if any)

Payload

(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

Reading Packet Correctly



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

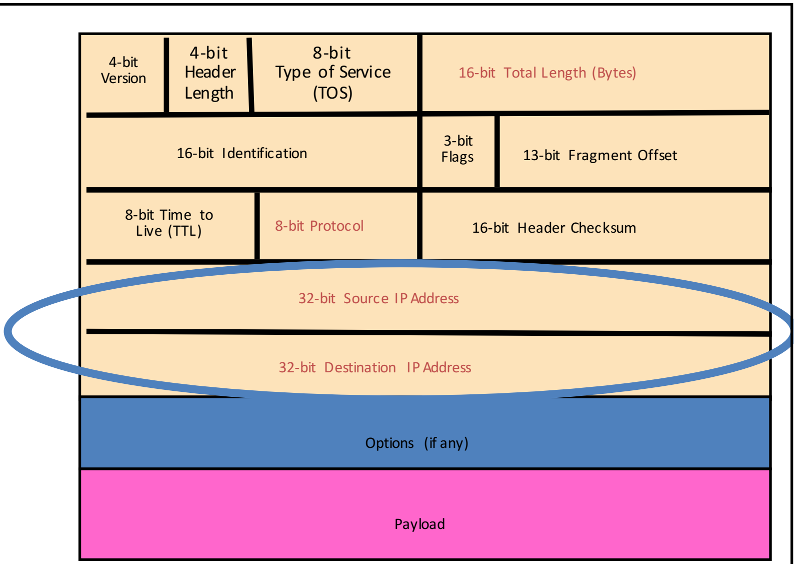
32-bit Destination IP Address

Options (if any)

Payload

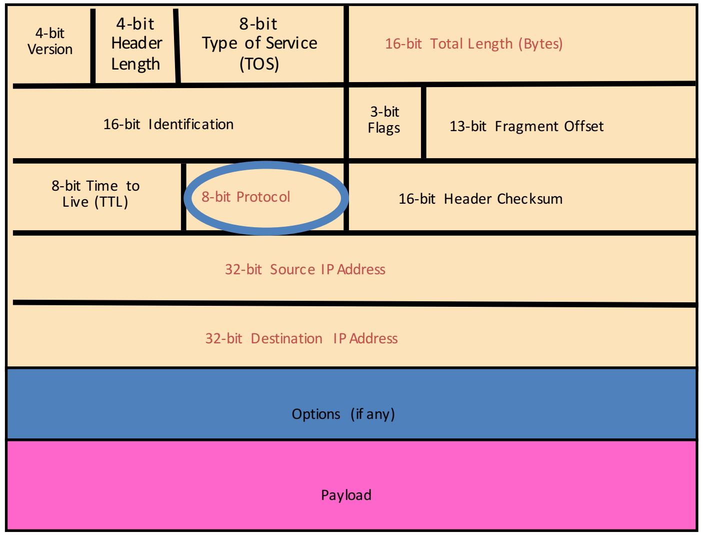
- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically “4” (for IPv4), and sometimes “6” (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically “5” (for a 20-byte IPv4 header)
  - Can be 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

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

Telling Host How to Handle Packet


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

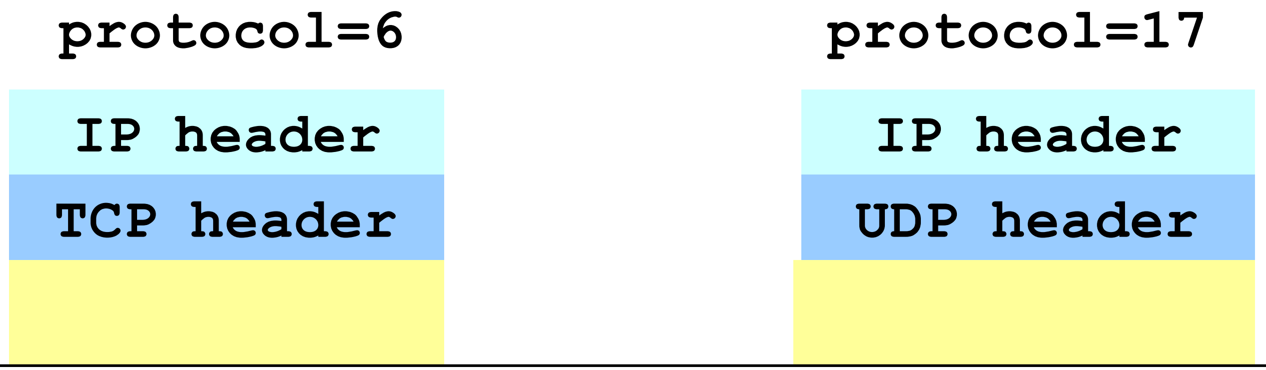
32-bit Source IP Address

32-bit Destination IP Address

Options (if any)

Payload

- 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)


protocol=6

IP header

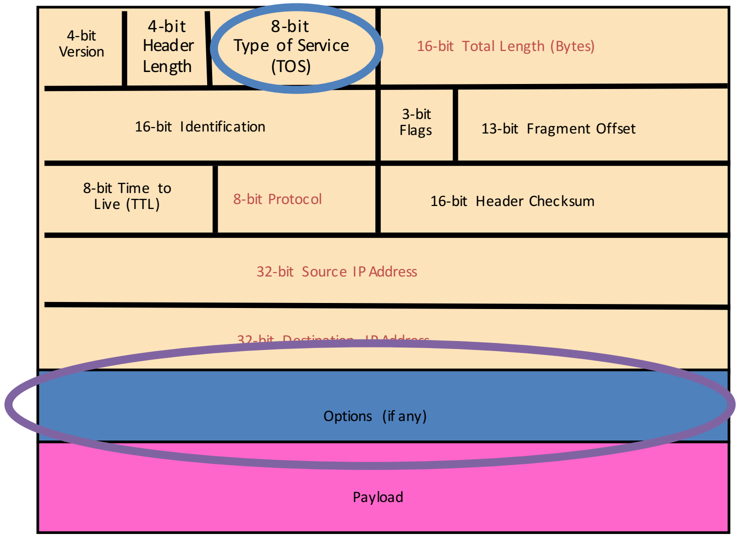
TCP header

protocol=17

IP header

UDP header

Special Handling



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)

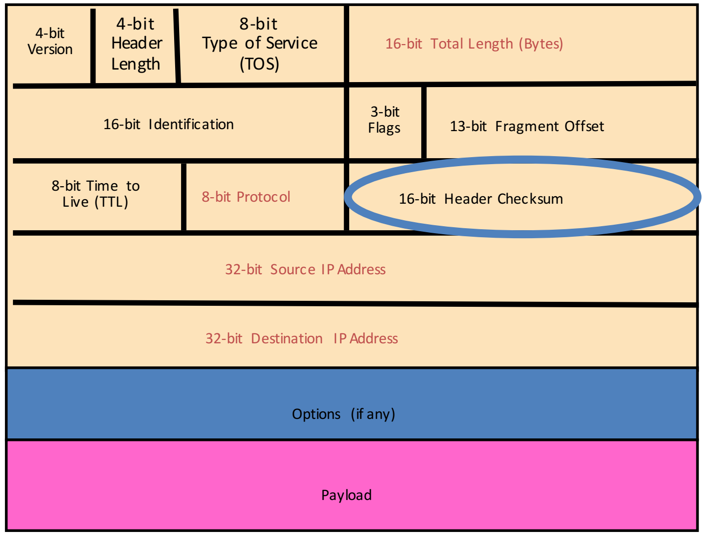
Payload

- 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

Potential Problems

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

Header Corruption



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

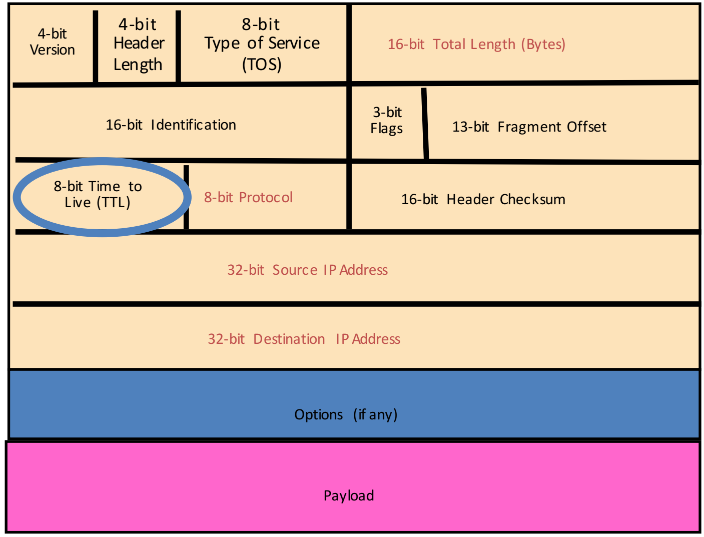
Options (if any)

Payload

- 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?**

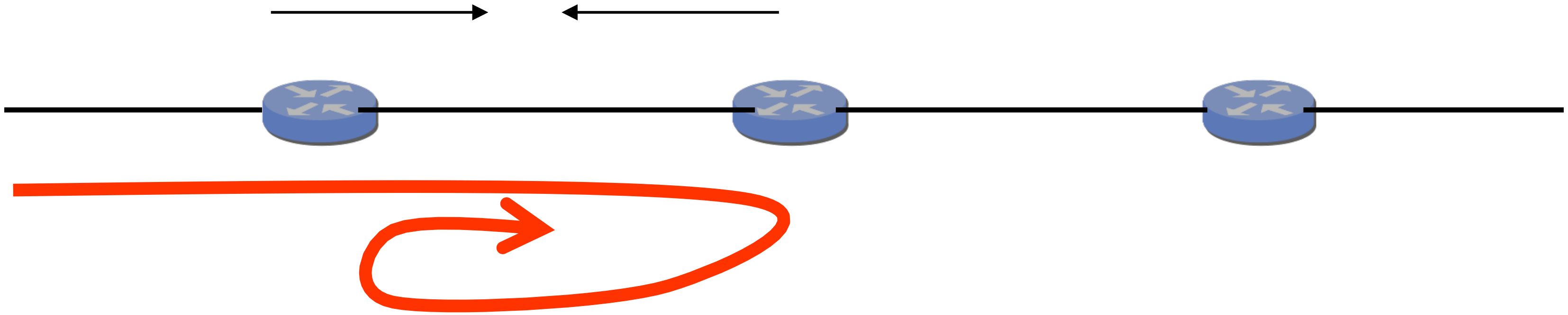
Preventing Loops

(aka Internet Zombie plan)



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			

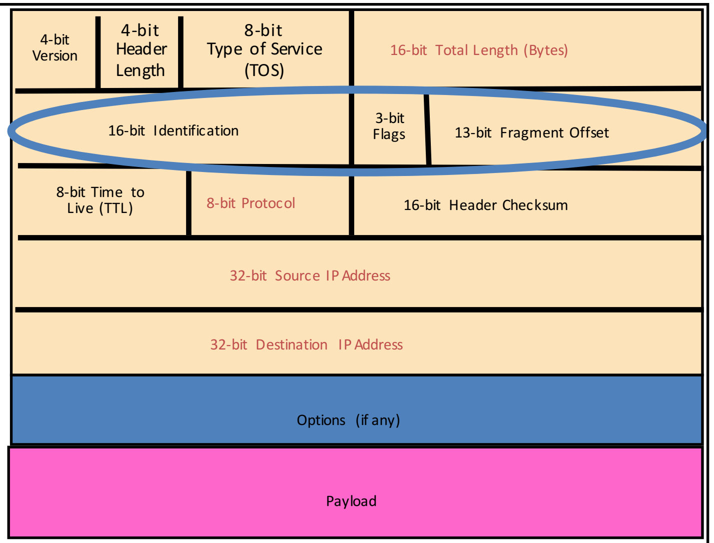
- Forwarding loops cause packets to cycle forever
  - As these accumulate, eventually consume **all** capacity



- 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**

Fragmentation

(some assembly required)



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)

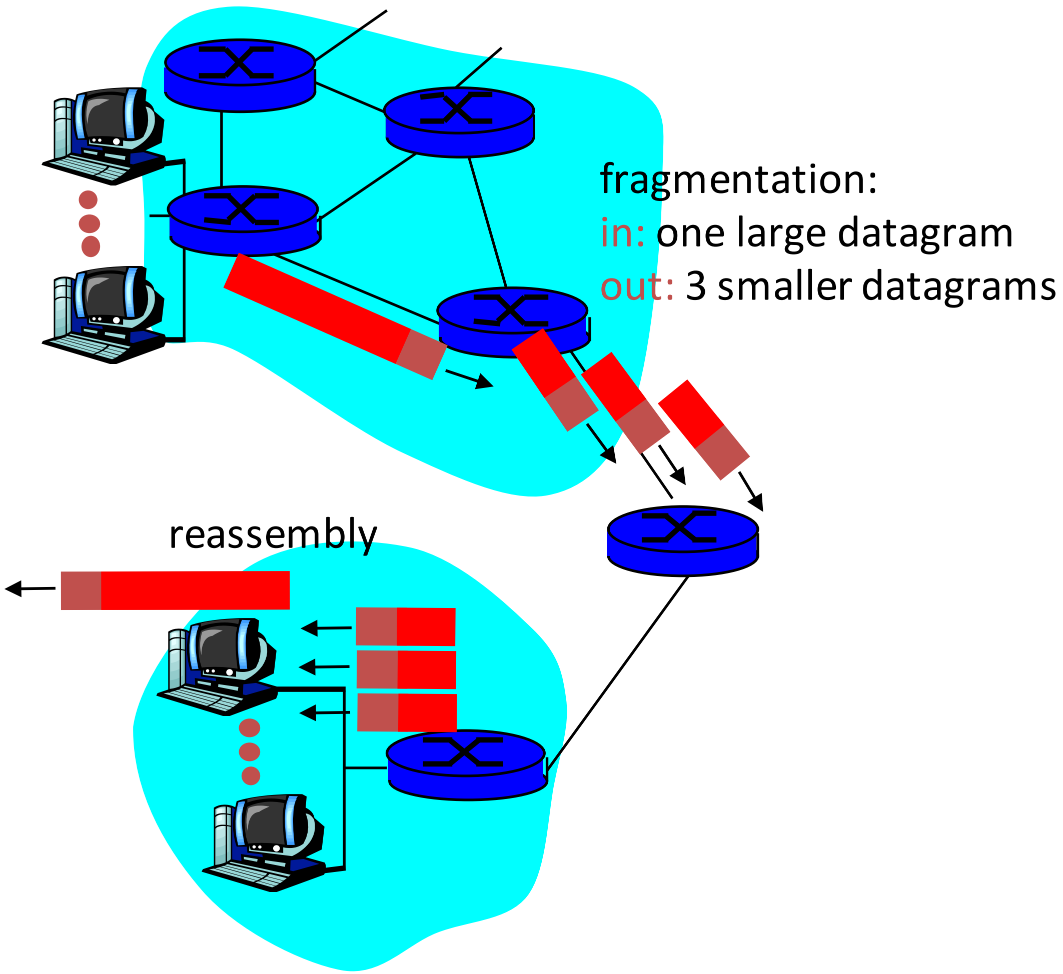
Payload

- 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

IP Fragmentation & Reassembly

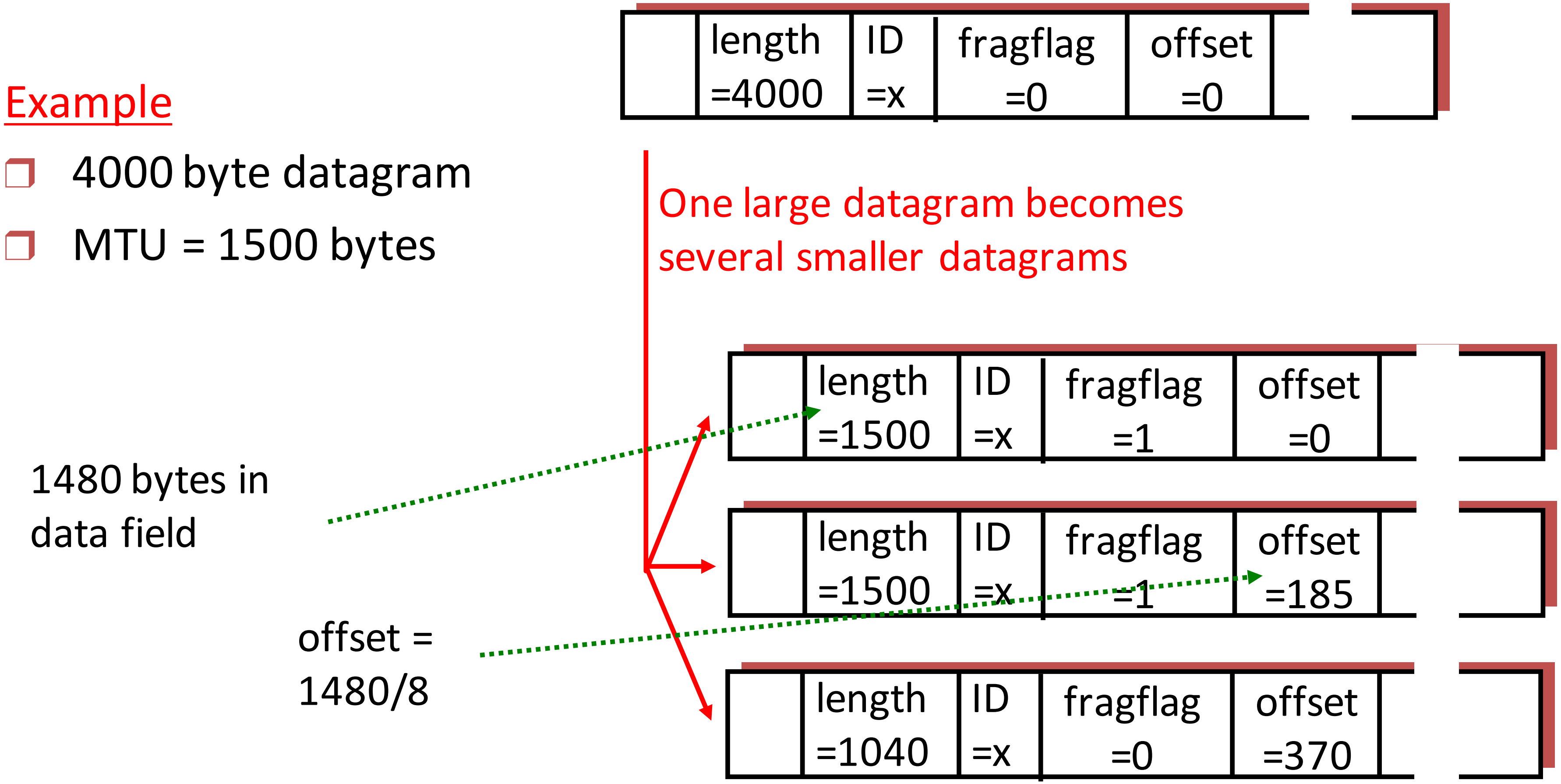
- network links have MTU (max.transfer size) - largest possible link-level frame.
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify,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



IP Fragmentation and Reassembly

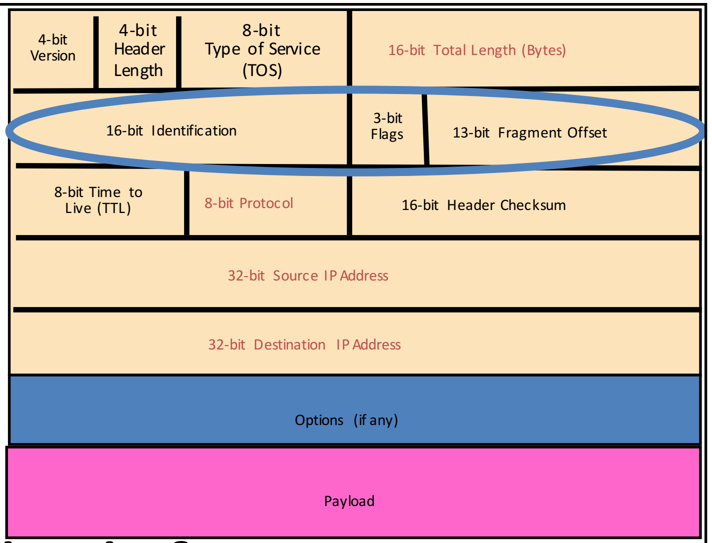
Example

- 4000 byte datagram
- MTU = 1500 bytes



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

Fragmentation Details

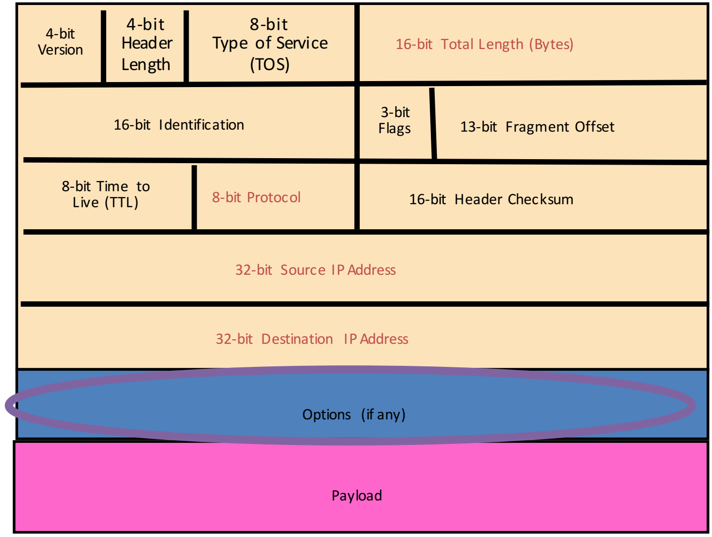


4-bit Version
4-bit Header Length
8-bit Type of Service (TOS)
16-bit Total Length (Bytes)
16-bit Identification
3-bit Flags
13-bit Fragment Offset
8-bit Time to Live (TTL)
8-bit Protocol
16-bit Header Checksum
32-bit Source IP Address
32-bit Destination IP Address
Options (if any)
Payload

- 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

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

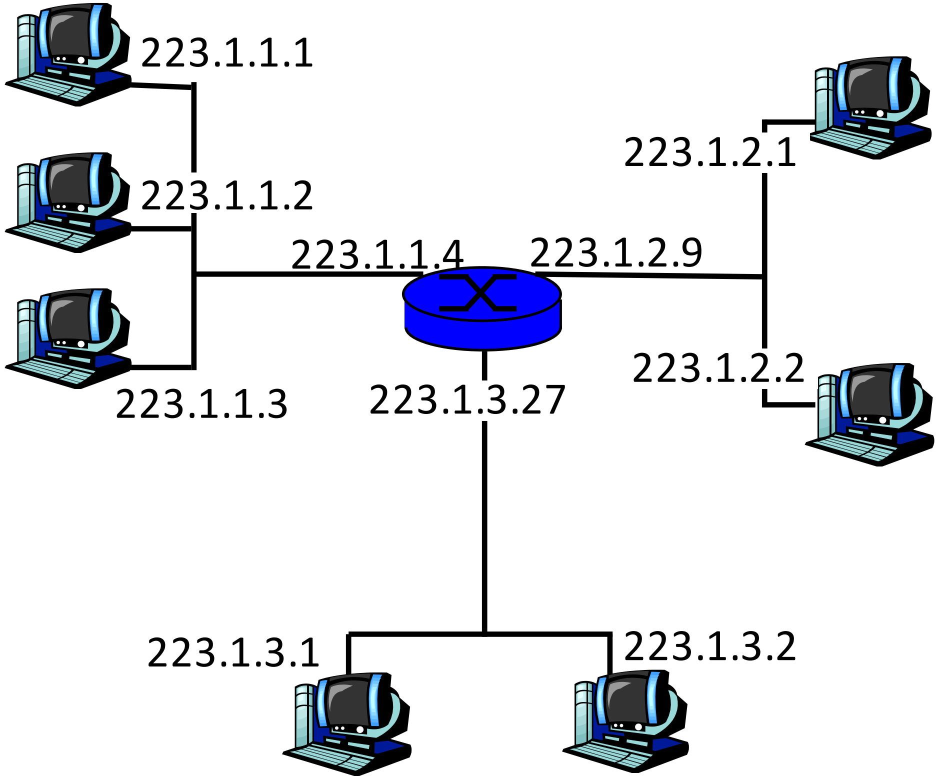
Options



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

IP Addressing: introduction

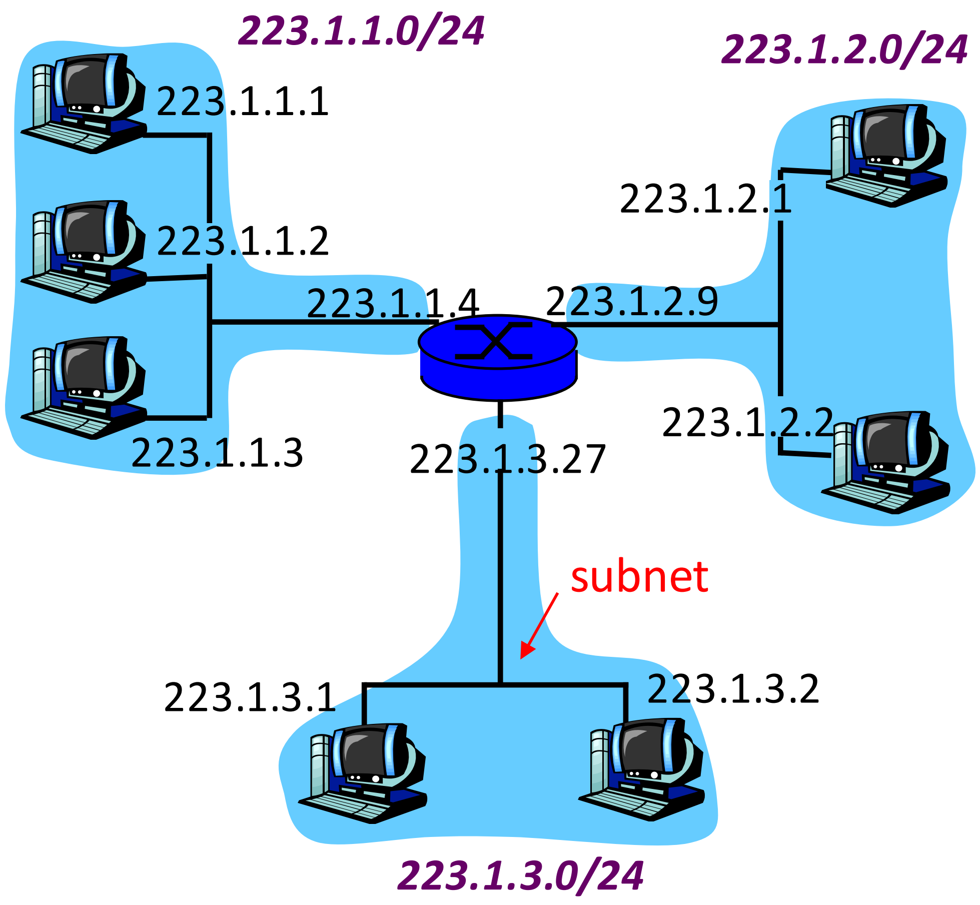
- IP address: 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - router’s typically have multiple interfaces
  - host typically has one interface
  - IP addresses associated with each interface

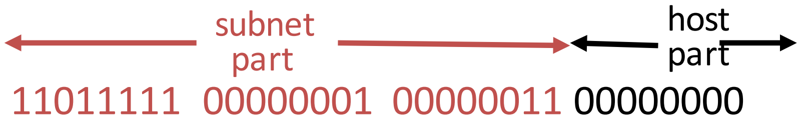


$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

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





223.1.3.0/24

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

Subnet mask: /24

network consisting of 3 subnets

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 (circa 1980's your mileage will vary)
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

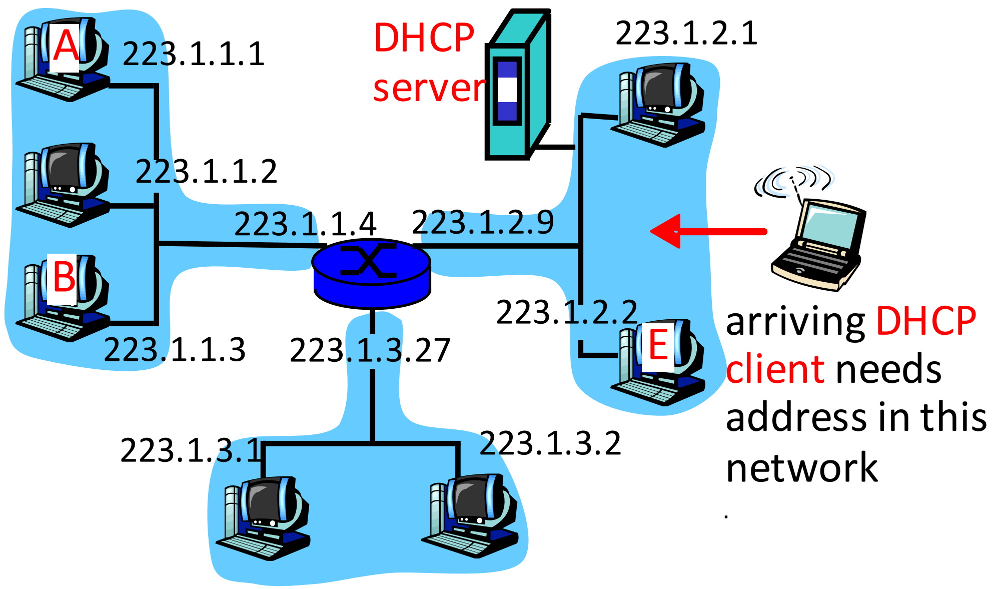
DHCP client-server scenario

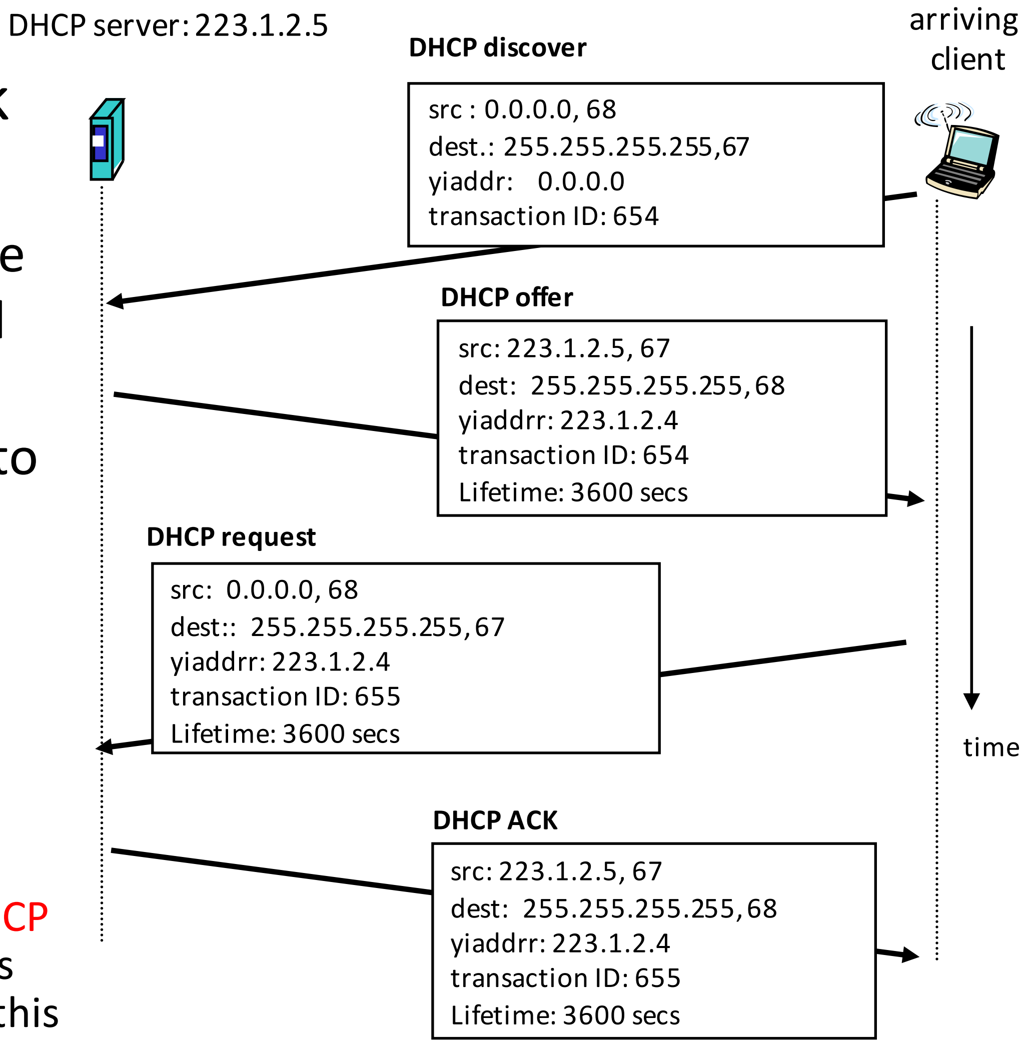
Goal: allow host to *dynamically* obtain its IP address from network server when it joins network

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)





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

IP addresses: how to get one?

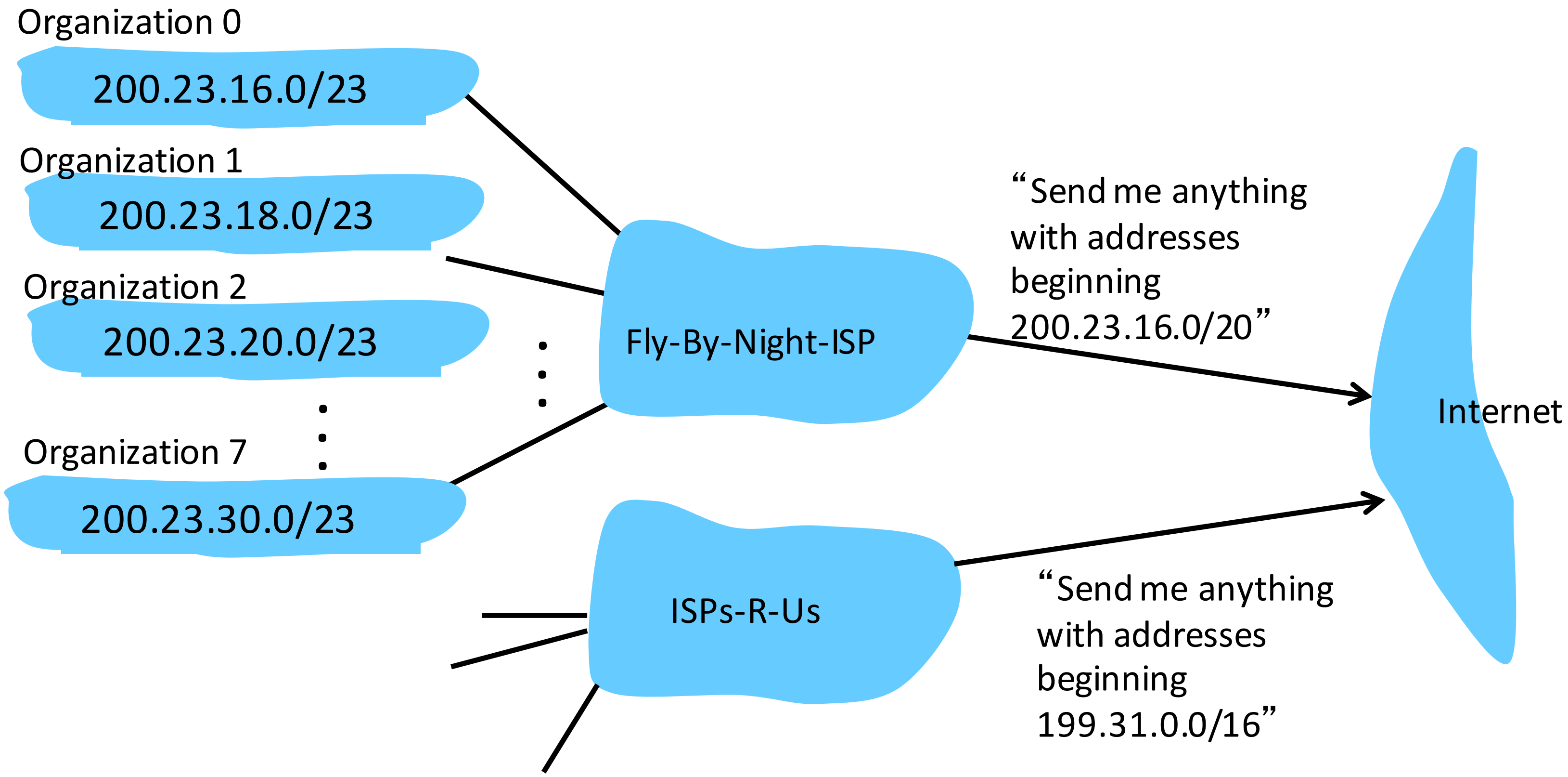
Q: How does *network* get subnet part of IP addr?

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

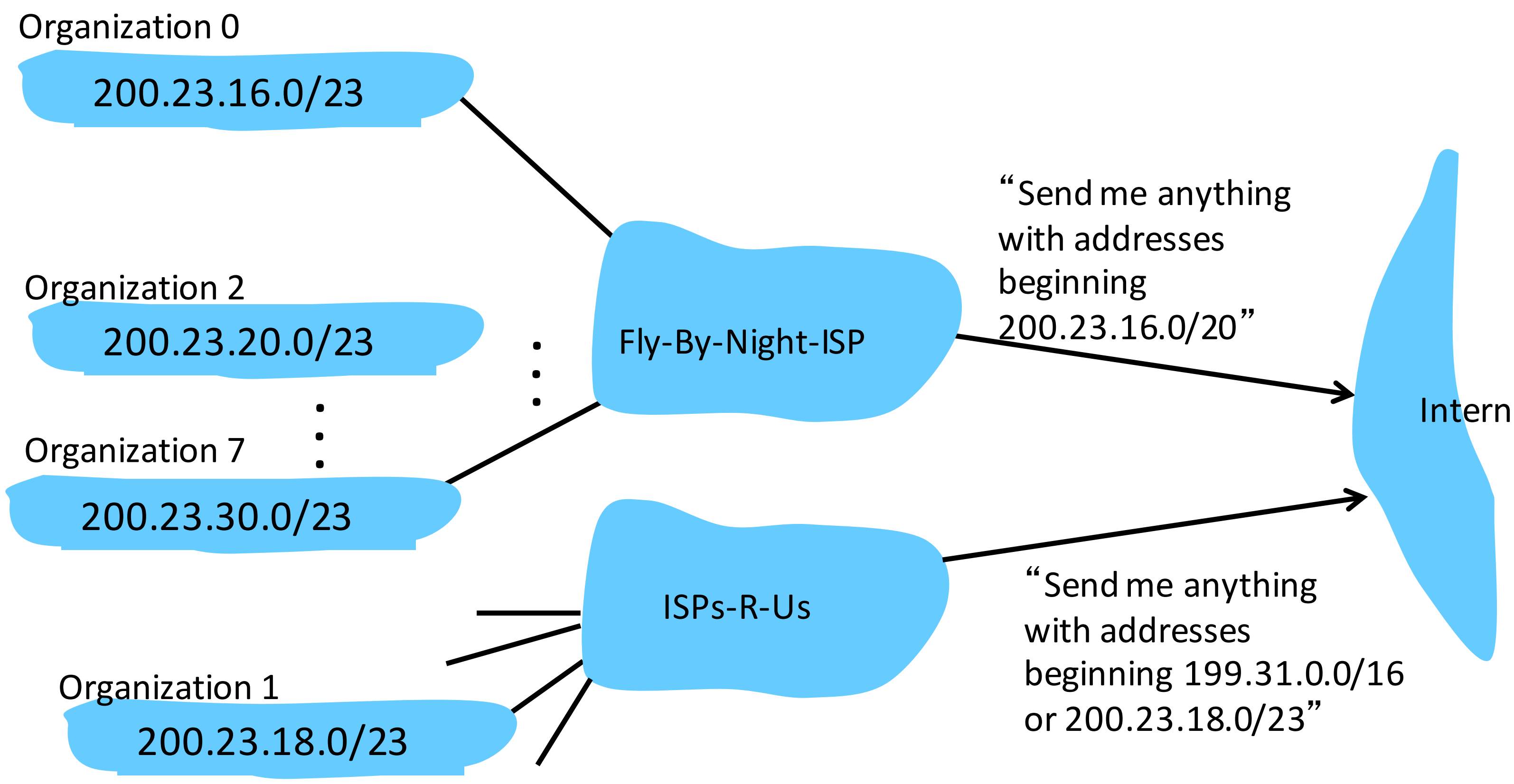
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

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



IP addressing: the last word...

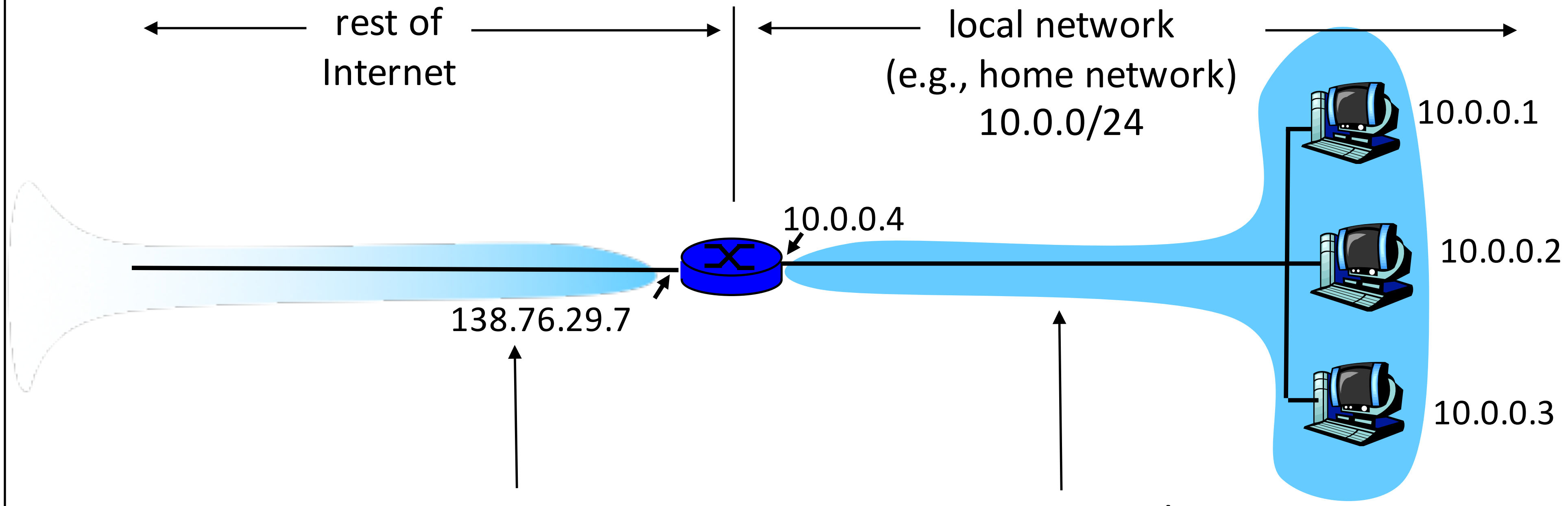
Q: How does an ISP get a block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

Cant get more IP addresses? well there is always…..

NAT: Network Address Translation



rest of Internet

local network (e.g., home network) 10.0.0/24

10.0.0.1

10.0.0.4

10.0.0.2

138.76.29.7

10.0.0.3

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)

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).

NAT: Network Address Translation

Implementation: NAT router must:

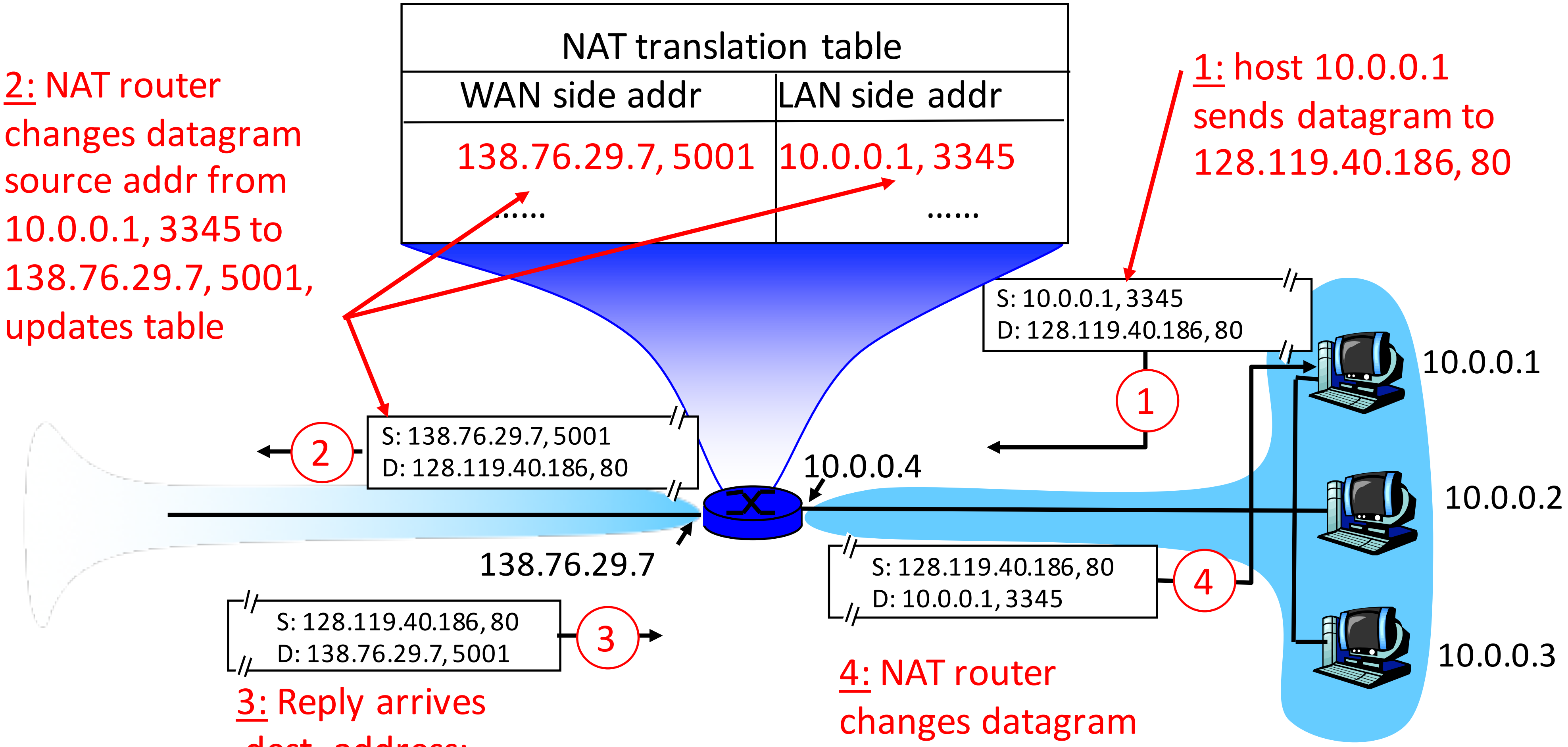
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  . . . 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

NAT: Network Address Translation



NAT translation table

WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

3: Reply arrives dest. address: 138.76.29.7, 5001

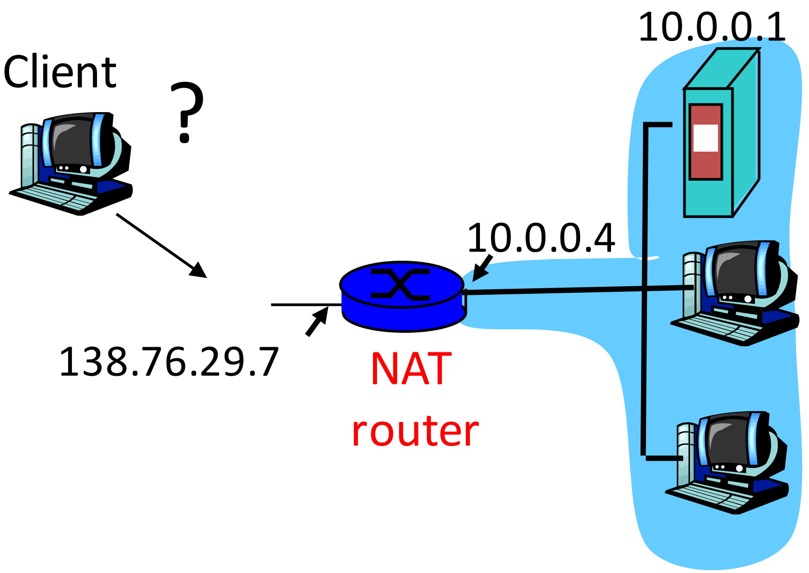
4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

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

NAT traversal problem

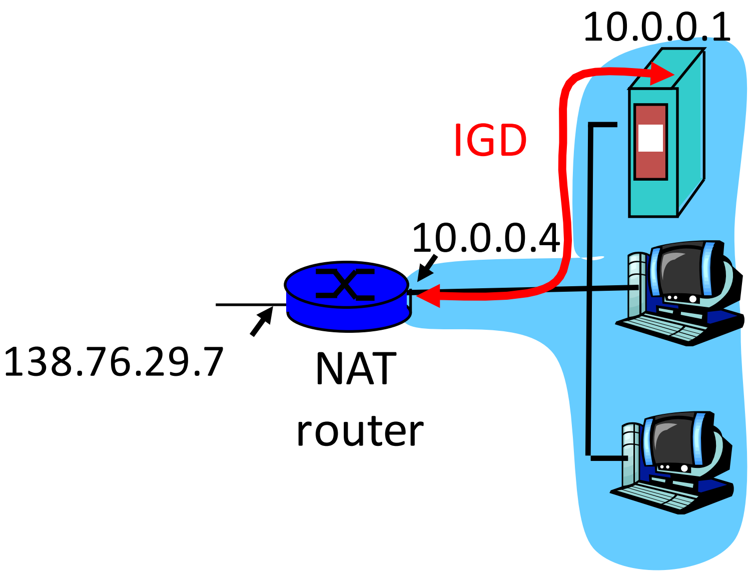
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
- solution 1: statically configure NAT to forward incoming connection requests at given port to server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



NAT traversal problem

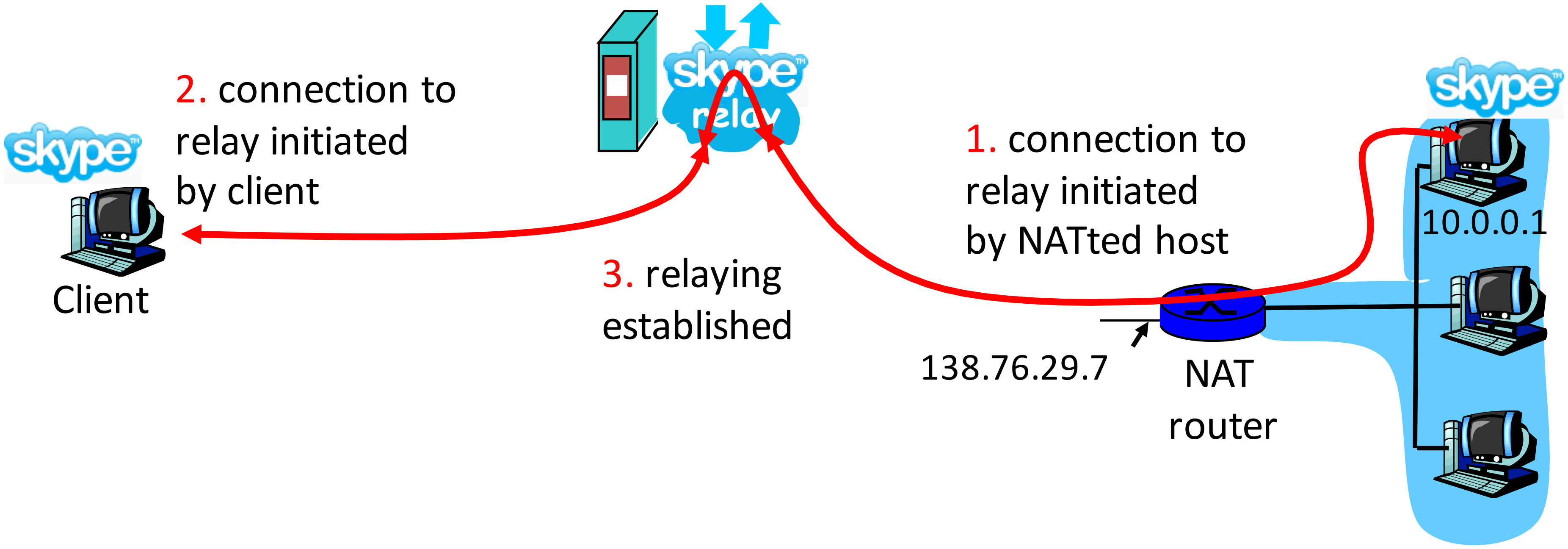
- solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to:
  - ❖learn public IP address (138.76.29.7)
  - ❖add/remove port mappings (with lease times)

i.e., automate static NAT port map configuration



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



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

Remember this? Traceroute at work…

traceroute: rio.cl.cam.ac.uk to munnari.oz.au

(tracepath on pwf is similar)

Three delay measurements from rio.cl.cam.ac.uk to gatwick.net.cl.cam.ac.uk

```
traceroute munnari.oz.au
traceroute to munnari.oz.au (202.29.151.3), 30 hops max, 60 byte packets
 1  gatwick.net.cl.cam.ac.uk (128.232.32.2)  0.416 ms  0.384 ms  0.427 ms
 2  cl-sby.route-nwest.net.cam.ac.uk (193.60.89.9)  0.393 ms  0.440 ms  0.494 ms
 3  route-nwest.route-mill.net.cam.ac.uk (192.84.5.137)  0.407 ms  0.448 ms  0.501 ms
 4  route-mill.route-enet.net.cam.ac.uk (192.84.5.94)  1.006 ms  1.091 ms  1.163 ms
 5  xe-11-3-0.camb-rbr1.eastern.ja.net (146.97.130.1)  0.300 ms  0.313 ms  0.350 ms
 6  ae24.lowdss-sbr1.ja.net (146.97.37.185)  2.679 ms  2.664 ms  2.712 ms
 7  ae28.londhx-sbr1.ja.net (146.97.33.17)  5.955 ms  5.953 ms  5.901 ms
 8  janet.mx1.lon.uk.geant.net (62.40.124.197)  6.059 ms  6.066 ms  6.052 ms
 9  ae0.mx1.par.fr.geant.net (62.40.98.77)  11.742 ms  11.779 ms  11.724 ms
10  ae1.mx1.mad.es.geant.net (62.40.98.64)  27.751 ms  27.734 ms  27.704 ms
11  mb-so-02-v4.bb.tein3.net (202.179.249.117)  138.296 ms  138.314 ms  138.282 ms
12  sg-so-04-v4.bb.tein3.net (202.179.249.53)  196.303 ms  196.293 ms  196.264 ms
13  th-pr-v4.bb.tein3.net (202.179.249.66)  225.153 ms  225.178 ms  225.196 ms
14  pyt-thairen-to-02-bdr-pyt.uni.net.th (202.29.12.10)  225.163 ms  223.343 ms  223.363 ms
15  202.28.227.126 (202.28.227.126)  241.038 ms  240.941 ms  240.834 ms
16  202.28.221.46 (202.28.221.46)  287.252 ms  287.306 ms  287.282 ms
17  * * *
18  * * *
19  * * *
20  coe-gw.psu.ac.th (202.29.149.70)  241.681 ms  241.715 ms  241.680 ms
21  munnari.OZ.AU (202.29.151.3)  241.610 ms  241.636 ms  241.537 ms
```

trans-continent link

* means no response (probe lost, router not replying)

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.

ICMP: Internet Control Message Protocol

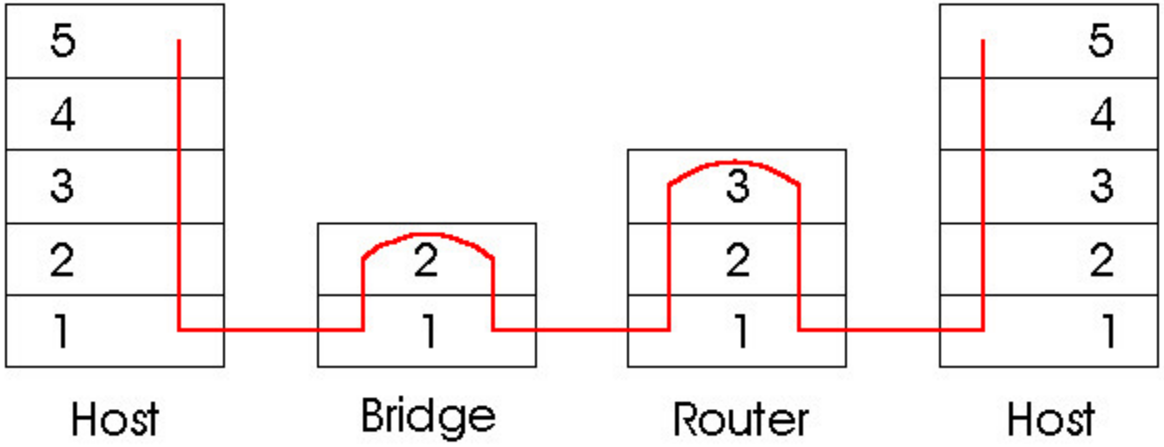
- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing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

Gluing it together:
How does my Network (address) interact with my Data-Link (address) ?

Switches vs. Routers Summary

- both store-and-forward devices
  - routers: network layer devices (examine network layer headers)
  - switches are link layer devices
- routers maintain routing tables, implement routing algorithms
- switches maintain switch tables, implement filtering, learning algorithms



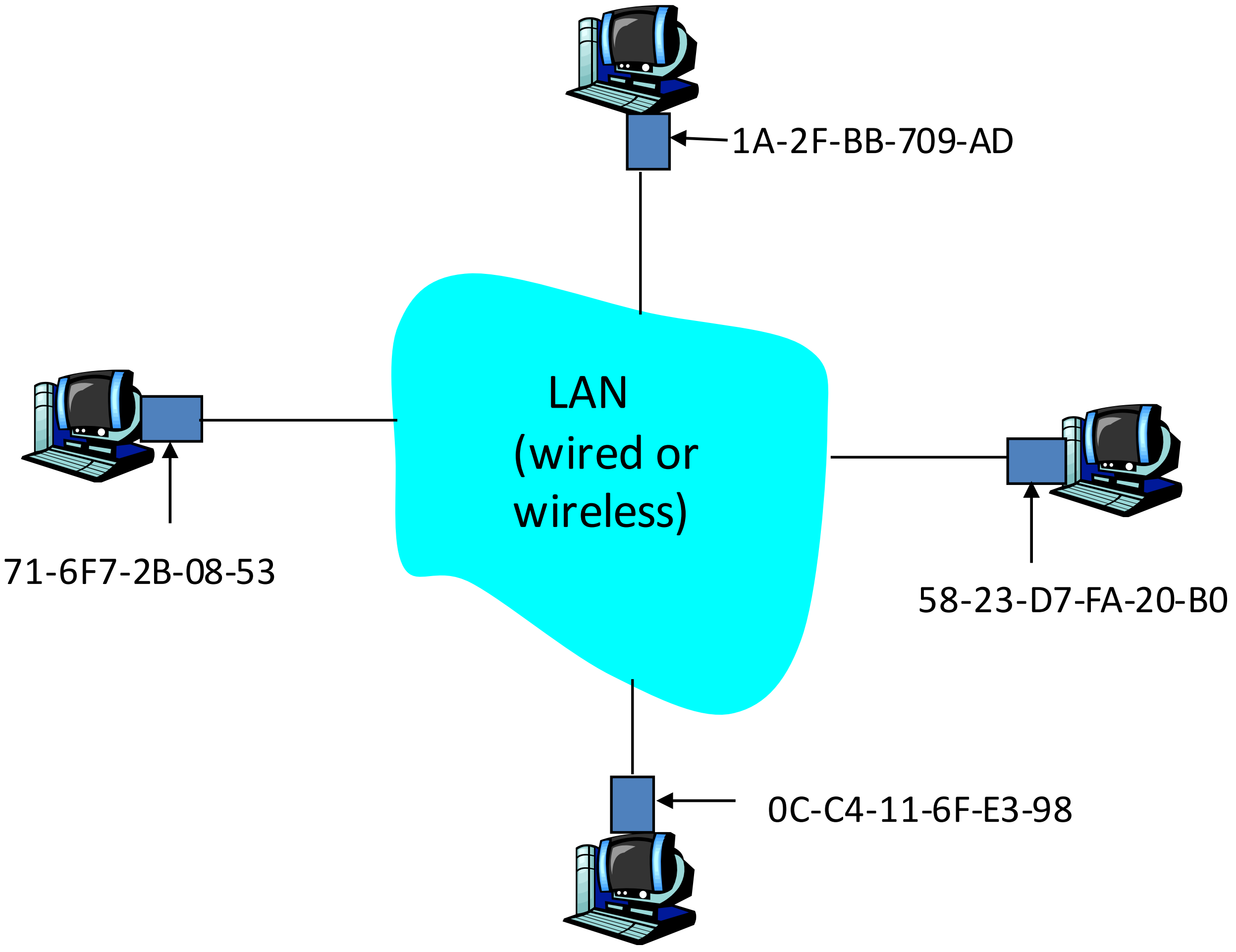
MAC Addresses (and IPv4 ARP)

or How do I glue my network to my data-link?

- 32-bit IP address:
  - *network-layer* address
  - used to get datagram to destination IP subnet
- MAC (or LAN or physical or Ethernet) address:
  - function: *get frame from one interface to another physically-connected interface (same network)*
  - 48 bit MAC address (for most LANs)
    - burned in NIC ROM, also (commonly) software settable

LAN Addresses and ARP

Each adapter on LAN has unique LAN address



Ethernet
Broadcast address =
FF-FF-FF-FF-FF-FF

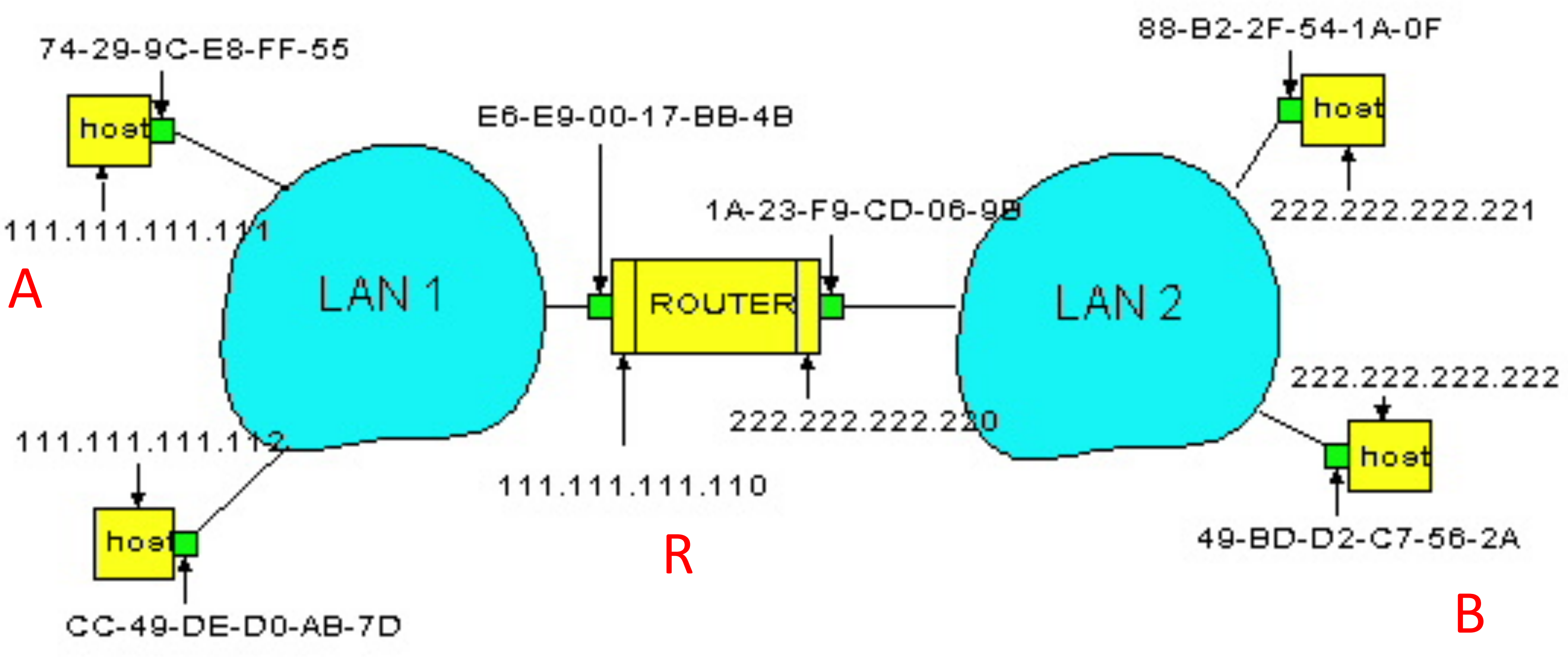
🟦 = adapter

Address Resolution Protocol

- Every node maintains an ARP table
  - <IP address, MAC address> pair
- Consult the table when sending a packet
  - Map destination IP address to destination MAC address
  - Encapsulate and transmit the data packet
- But: what if IP address not in the table?
  - Sender broadcasts: "**Who has IP address 1.2.3.156**?"
  - Receiver responds: "**MAC address 58-23-D7-FA-20-B0**"
  - Sender caches result in its ARP table

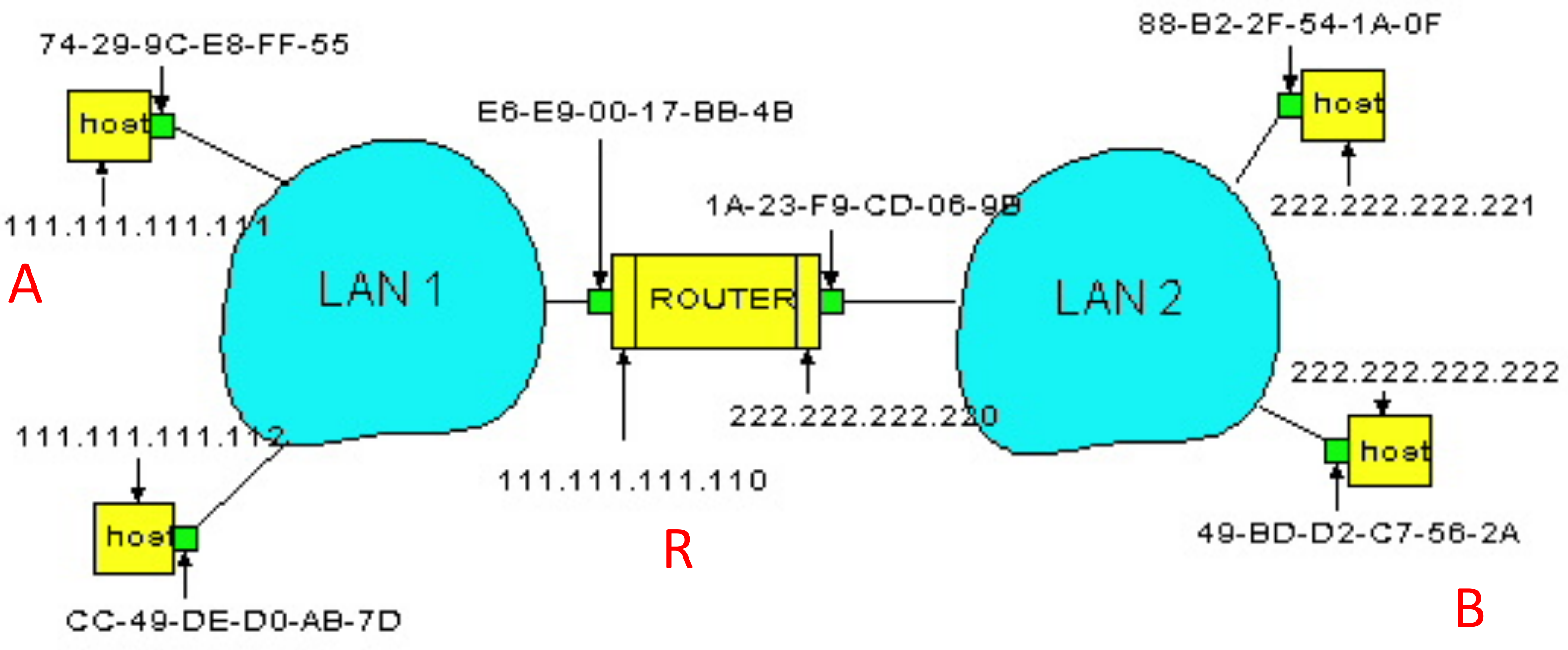
Example: A Sending a Packet to B

How does host A send an IP packet to host B?



Example: A Sending a Packet to B

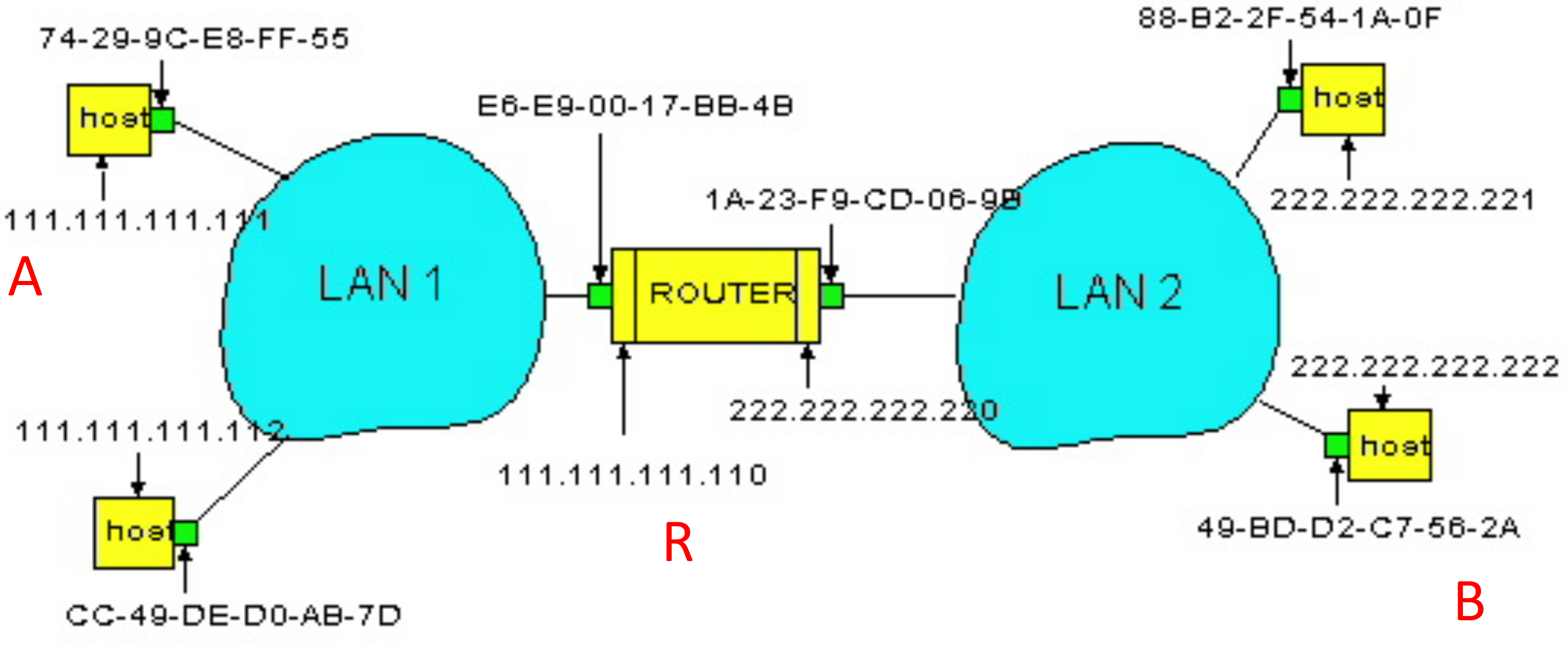
How does host A send an IP packet to host B?



1. **A sends packet to R.**
2. **R sends packet to B.**

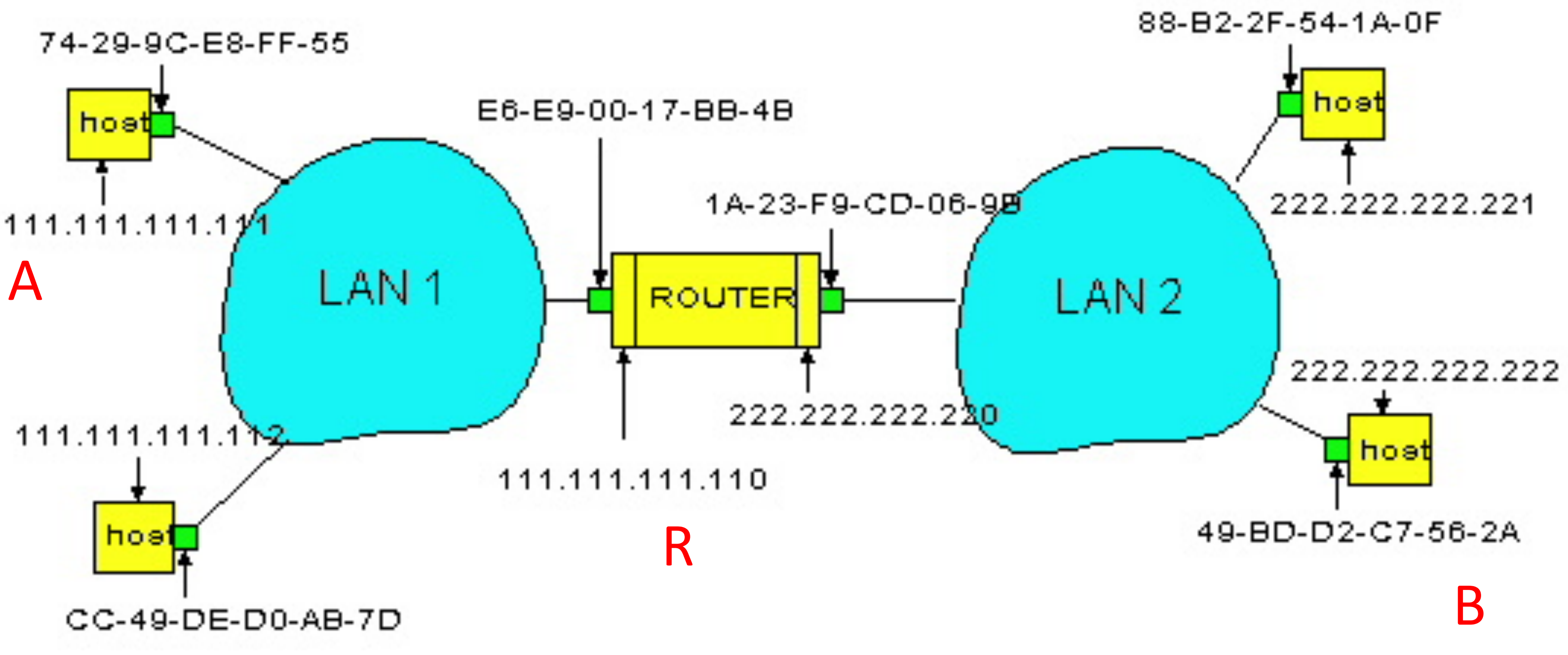
Host A Decides to Send Through R

- Host A constructs an IP packet to send to B
  - Source 111.111.111.111, destination 222.222.222.222
- Host A has a gateway router R
  - Used to reach destinations outside of 111.111.111.0/24
  - Address 111.111.111.110 for R learned via DHCP/config



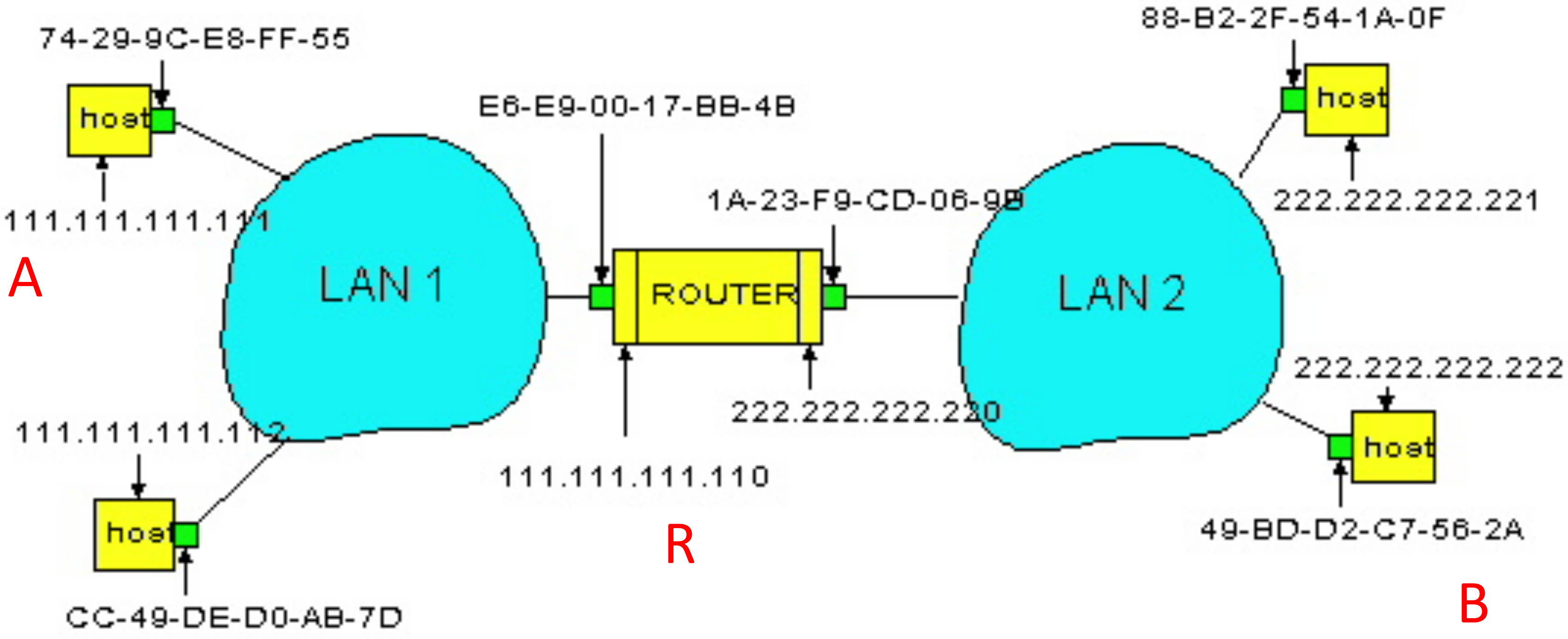
Host A Sends Packet Through R

- Host A learns the MAC address of R's interface
  - ARP request: broadcast request for 111.111.111.110
  - ARP response: R responds with E6-E9-00-17-BB-4B
- Host A encapsulates the packet and sends to R



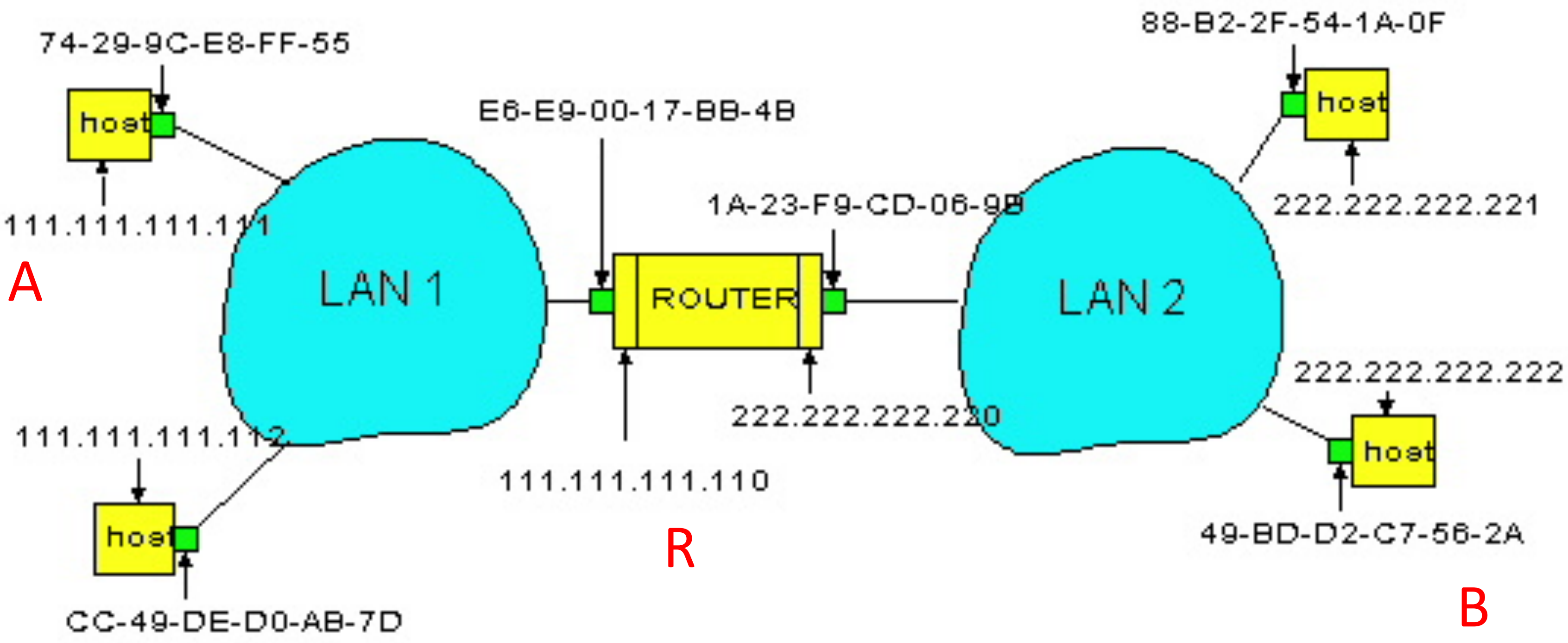
R Decides how to Forward Packet

- Router R's adaptor receives the packet
  - R extracts the IP packet from the Ethernet frame
  - R sees the IP packet is destined to 222.222.222.222
- Router R consults its forwarding table
  - Packet matches 222.222.222.0/24 via other adaptor



R Sends Packet to B

- Router R’s learns the MAC address of host B
  - ARP request: broadcast request for 222.222.222.222
  - ARP response: B responds with 49-BD-D2-C7-52A
- Router R encapsulates the packet and sends to B



Security Analysis of ARP

- Impersonation
  - Any node that hears request can answer …
  - … and can say whatever they want

- Actual legit receiver never sees a problem
  - Because even though later packets carry its IP address, its NIC doesn’t capture them since not its MAC address

Key Ideas in Both ARP and DHCP

- Broadcasting: Can use broadcast to make contact
  - Scalable because of limited size

- Caching: remember the past for a while
  - Store the information you learn to reduce overhead
  - Remember your own address & other host's addresses

- Soft state: eventually forget the past
  - Associate a time-to-live field with the information
  - … and either refresh or discard the information
  - Key for robustness in the face of unpredictable change

Why Not Use DNS-Like Tables?

- When host arrives:
  - Assign it an IP address that will last as long it is present
  - Add an entry into a table in DNS-server that maps MAC to IP addresses

- Answer:
  - Names: explicit creation, and are plentiful
  - Hosts: come and go without informing network
    - Must do mapping on demand
  - Addresses: not plentiful, need to reuse and remap
    - Soft-state enables dynamic reuse

No More IPv4 Addresses

- IPv4 address space in terms of /8's

Class A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	[illegible]	55	56	[illegible]	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Class B

128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	[illegible]	151	152	[illegible]	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

Class C

192	193	194	195	196	197	[illegible]	199	200	[illegible]	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223

Class D & E

224	225	226	227	[illegible]	229	230	231	[illegible]	[illegible]	[illegible]	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 24 /8’s on January 12, 2010

0	1	2	3	4	**5**	6	7	8	9	10	11	12	13	**14**	15
16	17	18	19	20	21	22	**23**	24	25	26	27	28	29	30	**31**
32	33	34	35	**36**	**37**	38	**39**	40	41	**42**	43	44	45	46	47
48	**49**	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	**177**	178	**179**	180	**181**	182	183	184	**185**	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	**223**
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 20 /8’s on April 10, 2010

0	1	2	3	4	**5**	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	**23**	24	25	26	27	28	29	30	**31**
32	33	34	35	**36**	**37**	38	**39**	40	41	**42**	43	44	45	46	47
48	**49**	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	**100**	**101**	**102**	**103**	**104**	**105**	**106**	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	**177**	178	**179**	180	**181**	182	183	184	**185**	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 13 /8's on May 8, 2010

0	1	2	3	4	🟩 5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	🟩 23	24	25	26	27	28	29	30	🟩 31
32	33	34	35	36	🟩 37	38	🟩 39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	🟩 100	101	🟩 102	🟩 103	🟩 104	105	🟩 106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
🟩 176	177	178	🟩 179	180	181	182	183	184	🟩 185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 7 /8’s on November 30th, 2010

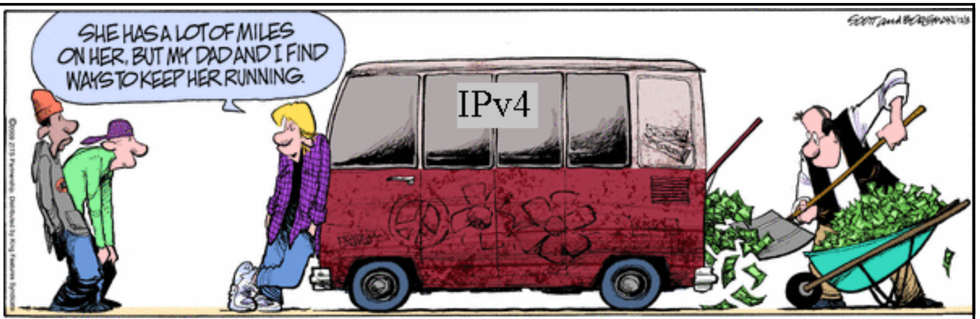
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	**39**	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	**102**	**103**	**104**	105	**106**	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	**179**	180	181	182	183	184	**185**	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- **0 /8's** on January 31st, 2011!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

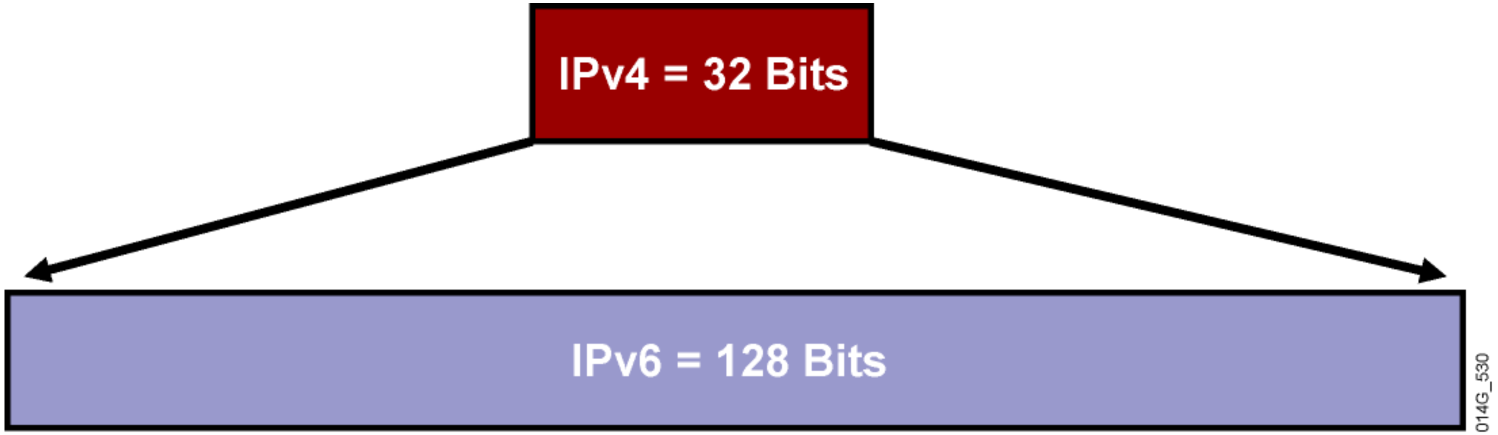
IPv6


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
  - Address field ***four*** times as long
- 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

Larger Address Space

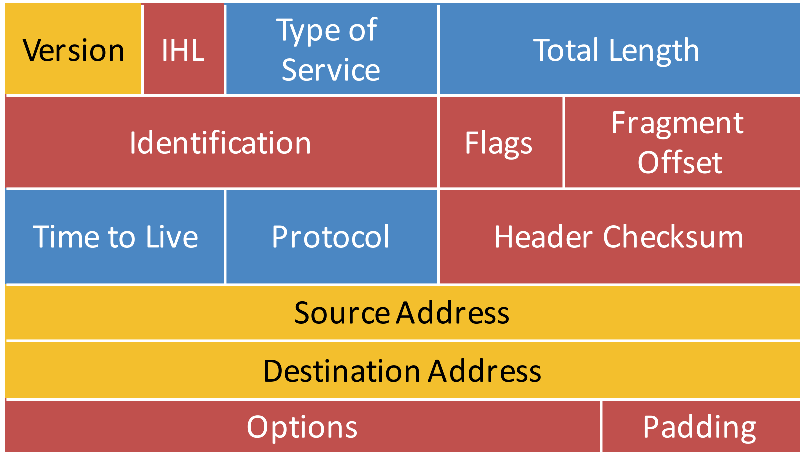
- IPv4 = 4,294,967,295 addresses
- IPv6 = 340,282,366,920,938,463,374,607,432,768,211,456 addresses
- 4x in number of bits translates to **<u>huge</u>** increase in address space!



Other Significant Protocol Changes

- Increased minimum MTU from 576 to 1280
- No enroute fragmentation… fragmentation only at source
- Header changes
- Replace broadcast with multicast

IPv4



Version | IHL | Type of Service | Total Length

Identification | Flags | Fragment Offset

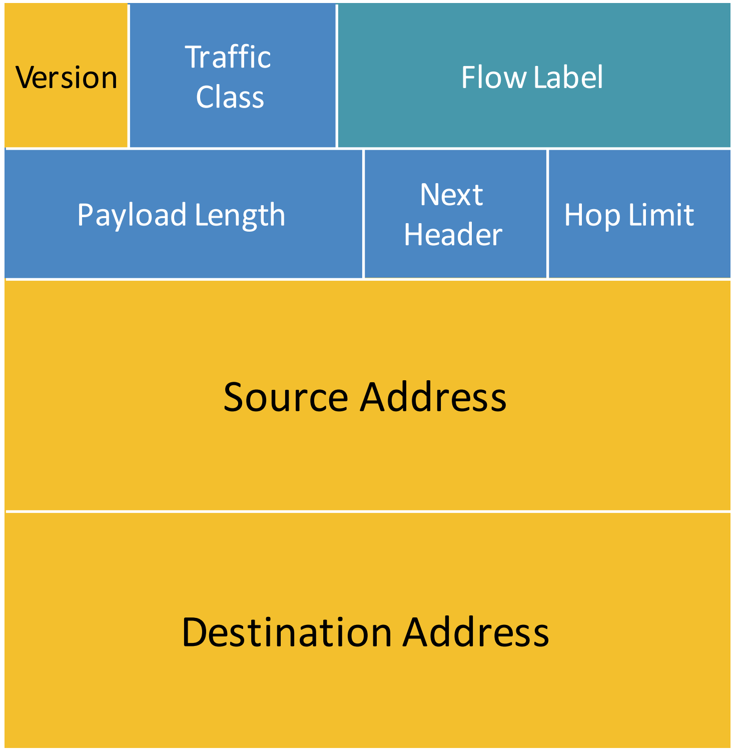
Time to Live | Protocol | Header Checksum

Source Address

Destination Address

Options | Padding

IPv6



Version | Traffic Class | Flow Label

Payload Length | Next Header | Hop Limit

Source Address

Destination Address

Legend

- 🟨 Field’s Name Kept from IPv4 to IPv6
- 🟥 Fields Not Kept in IPv6
- 🟦 Name and Position Changed in IPv6
- New Field in IPv6

IPv4	IPv6
Addresses are 32 bits (4 bytes) in length.	Addresses are 128 bits (16 bytes) in length
Address (A) resource records in DNS to map host names to IPv4 addresses.	Address (AAAA) resource records in DNS to map host names to IPv6 addresses.
Pointer (PTR) resource records in the IN-ADDR.ARPA DNS domain to map IPv4 addresses to host names.	Pointer (PTR) resource records in the IP6.ARPA DNS domain to map IPv6 addresses to host names.
IPSec is optional and should be supported externally	IPSec support is not optional
Header does not identify packet flow for QoS handling by routers	Header contains Flow Label field, which Identifies packet flow for QoS handling by router.
Both routers and the sending host fragment packets.	Routers do not support packet fragmentation. Sending host fragments packets
Header includes a checksum.	Header does not include a checksum.
Header includes options.	Optional data is supported as extension headers.
ARP uses broadcast ARP request to resolve IP to MAC/Hardware address.	Multicast Neighbor Solicitation messages resolve IP addresses to MAC addresses.
Internet Group Management Protocol (IGMP) manages membership in local subnet groups.	Multicast Listener Discovery (MLD) messages manage membership in local subnet groups.
Broadcast addresses are used to send traffic to all nodes on a subnet.	IPv6 uses a link-local scope all-nodes multicast address.
Configured either manually or through DHCP.	Does not require manual configuration or DHCP.
Must support a 576-byte packet size (possibly fragmented).	Must support a 1280-byte packet size (without fragmentation).

Roundup: Why IPv6?

- Larger address space

- Auto-configuration

- Cleanup
- Eliminate fragmentation
- Eliminate checksum
- Pseudo-header (w/o Hop Limit) covered by transport layer
- Flow label
- Increase minimum MTU from 576 to 1280
- Replace broadcasts with multicast

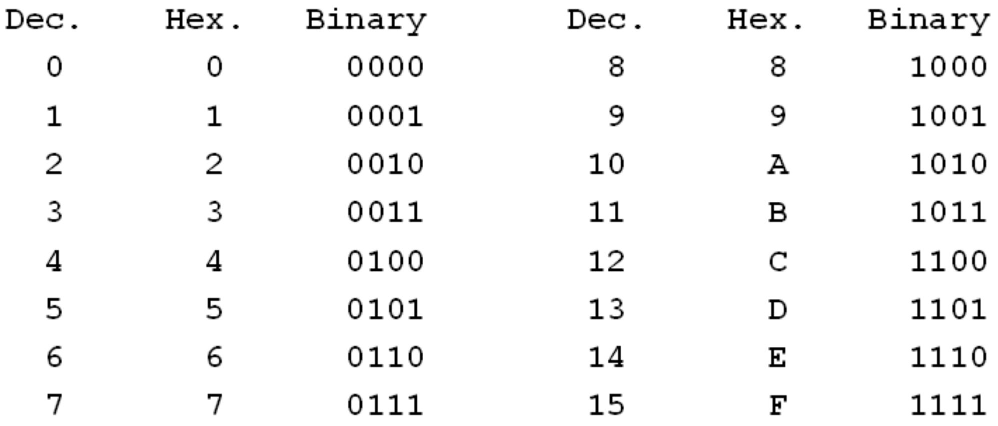
No Checksum!

- Provided by transport layer, if needed
- Ala TCP, includes pseudo-header
- Pseudo-header doesn't include Hop Limit
  - No per-hop re-computation!
  - Allows end-to-end implementation (transport layer)
- UDP checksum required (wasn't in IPv4) rfc6936: **No more zero**
- Pseudo-header added to ICMPv6 checksum

IPv6 Address Notation

- RFC 5952
- 128-bit IPv6 addresses are represented in:
  - Eight 16-bit segments
  - Hexadecimal (non-case sensitive) between 0000 and FFFF
  - Separated by colons
- Example:
  - `3ffe:1944:0100:000a:0000:00bc:2500:0d0b`
- Two rules for dealing with 0's

One Hex digit = 4 bits



Dec.	Hex.	Binary	Dec.	Hex.	Binary
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

0’s Rule 1 – Leading 0’s

- The leading zeroes in any 16-bit segment do not have to be written.

- Example
  - `3ffe : 1944 : 0100 : 000a : 0000 : 00bc : 2500 : 0d0b`
  - `3ffe : 1944 :  100 :    a :    0 :   bc : 2500 :  d0b`

`3ffe:1944:100:a:0:bc:2500:d0b`

0’s Rule 1 – Leading 0’s

- Can only apply to **leading zeros**… otherwise ambiguous results
- Example
  - 3ffe : 1944 : 100 : a : 0 : bc : 2500 : d0b
- Could be either
  - 3ffe : 1944 : **0**100 : **000**a : **000**0 : **00**bc : 2500 : **0**d0b
  - 3ffe : 1944 : 100**0** : a**000** : 0**000** : bc**00** : 2500 : d0b**0**
  - *Which is correct?*

0’s Rule 1 – Leading 0’s

- Can only apply to **leading zeros**… otherwise ambiguous results

- Example
  - `3ffe : 1944 :  100 :    a :    0 :   bc : 2500 :  d0b`

- Could be either
  - **`3ffe : 1944 : 0100 : 000a : 0000 : 00bc : 2500 : 0d0b`**
  - `3ffe : 1944 : 1000 : a000 : 0000 : bc00 : 2500 : d0b0`
  - *Which is correct?*

0’s Rule 2 – Double Colon

- Any **single**, **contiguous** string of **16-bit segments** consisting **of all zeroes** can be represented with a **double colon**.

```
ff02 : 0000 : 0000 : 0000 : 0000 : 0000 : 0000 : 0005
ff02 :    0 :    0 :    0 :    0 :    0 :    0 :    5
ff02 :                                         :    5
```

ff02::5

0’s Rule 2 – Double Colon

- Only a **single** contiguous string of all-zero segments can be represented with a double colon.

- Example:

```
2001 : 0d02 : 0000 : 0000 : 0014 : 0000 : 0000 : 0095
```

- Both of these are correct

```
2001 :  d02 ::                  14 :    0 :    0 :   95
```

OR

```
2001 :  d02 :    0 :    0 :  14 ::                   95
```

0’s Rule 2 – Double Colon

- However, using double colon more than once creates ambiguity

- Example

```
2001:d02::14::95

2001:0d02:0000:0000:0000:0014:0000:0095
2001:0d02:0000:0000:0014:0000:0000:0095
2001:0d02:0000:0014:0000:0000:0000:0095
```

Network Prefixes

- In IPv4, network portion of address can by identified by either
  - **Netmask**: `255.255.255.0`
  - **Bitcount**: `/24`

- Only use bitcount with IPv6

`3ffe:1944:100:a::/64`

Special IPv6 Addresses

- Default route: **`::/0`**
- Unspecified Address: **`::/128`**
  - Used in SLAAC (coming later)
- Loopback/Local Host: **`::1/128`**
  - No longer a /8 of addresses but a single address

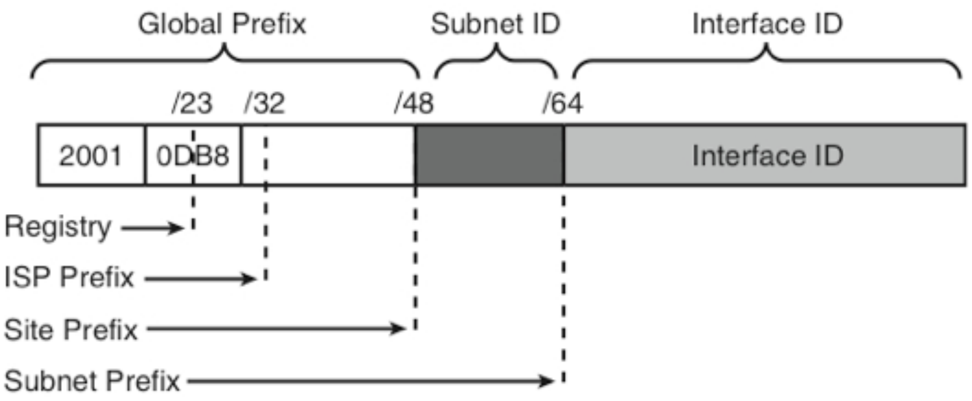
Types of IPv6 Addresses

- RFC 4291– “IPv6 Addressing Architecture”
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Types of IPv6 Addresses

- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Global Unicast Addresses

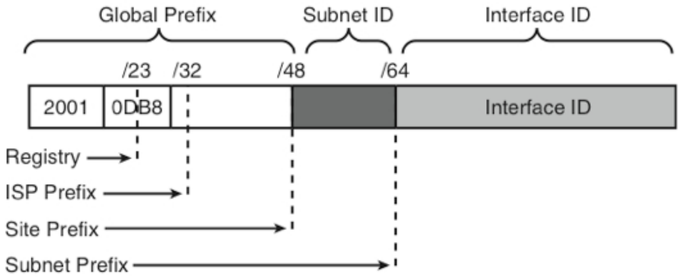


- Globally routable addresses
  - RFC 3587

- 3 parts
  - 48 bit **global routing prefix**
    - Hierarchically-structured value assigned to a site
    - Further broken down into Registry, ISP Prefix, and Site Prefix fields
  - 16 bit **Subnet ID**
    - Identifier of a subnet within a site
  - 64(!) bit **Interface ID**
    - Identify an interface on a subnet
    - Motivated by expected use of MAC addresses (IEEE EUI-64 identifiers) in SLAAC…
  - Except GUAs that start with '000…' binary
    - Used for, e.g., "IPv4-Mapped IPv6 Addresses" (RFC 4308)

Global Unicast Addresses

- Current **ARIN** policy is to assign no longer than /32 to an ISP
  - **A**merican **R**egistry for **I**nternet **N**umbers
  - https://www.arin.net/policy/nrpm.html
  - UCSC allocation is `2607:F5F0::/32`



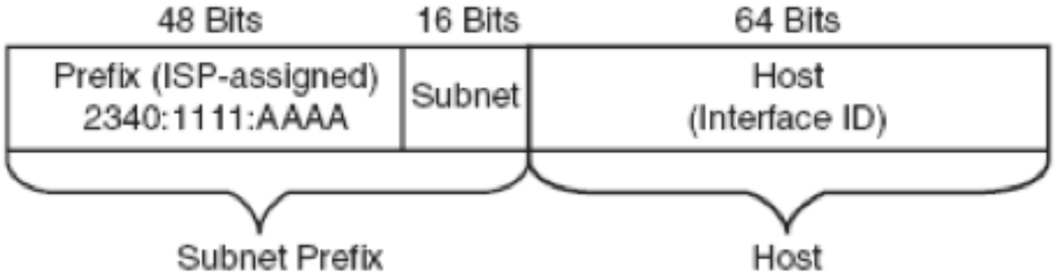
- IANA currently assigning addresses that start with ‘`001…`’ binary
  - `2000::/3`
    - `(2000:: – 3FFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)`
  - Supports
    - Maximum 2^{29} (536,870,912… 1/8 of an **Internet address space** of) ISPs
    - 2^{45} sites (equivalent to 8,192 **IAS**s of sites!)

- ISP can delegate a minimum of 2^{16}, or 65,535 site prefixes
  - Difference between Global Prefix (48 bits) and ISP Prefix (32 bits)

Subnetting GUAs

- Each site can identify 2^{16} (65,535) subnets

```
2340:1111:AAAA:1::/64
2340:1111:AAAA:2::/64
2340:1111:AAAA:3::/64
2340:1111:AAAA:4::/64
...
```



- Subnet has address space of 2^{64}… an IAS of IASs!

- Can extend the subnet ID into the interface ID portion of the address…
  - Sacrifice ability to use EUI-64 style of SLAAC…
  - Maybe not a bad thing… more later

These are huge numbers!!

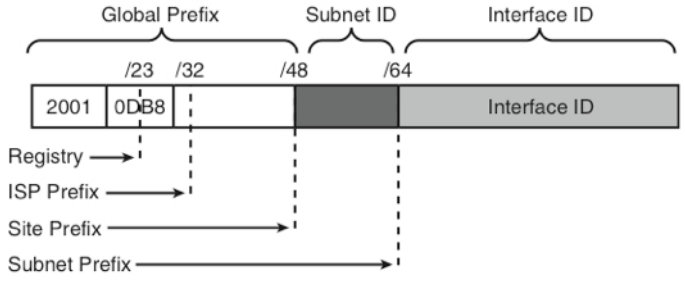
- Assume average /16’s allocated to ISPs and /22’s allocated to sites in IPv4

IPv6 2000::/3 block

Description	Range	Count	Scale vs IPv4
Total # ISPs	/3 – /32	2^{29} = 512M	**9,362**
Total # Sites	/3 – /48	2^{42} = 4T	**1.2M**
Sites/ISP	/48 – /64	2^{16} = 64K	**1,024**

IPv4 class A, B, and C blocks

Total # ISPs	/16 * 7/8	57K
Total # Sites	/22 * 7/8	3.6M
Sites/ISP	/16 - /22	2^{6} = 64



- ***And this keeps assumption of /64 subnets!***

IPv6 Address Space

- **Allocated**
  - 2000::/3Global Unicast
  - FC00::/7Unique Local Unicast
  - FE80::/10 Link Local Unicast
  - FF00::/8Multicast
- ***Accounts for a bit more than 2^{125} of the address space.***

- **Unallocated** (“Reserved by IETF”)
  - /3’s – 4000::, 6000::, 8000::, A000::, C000::
  - /4’s – 1000::, E000::
  - /5’s – 0800::, F000::
  - /6’s – 0400::, F800::
  - /7’s – 0200::
  - /8’s – 0000::, 0100::
  - /9’s – FE00::
  - /10’s – FEC0::
- ***Accounts for a little more than 2^{127}, or more than half, of the address space!!***

http://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xml

Problem with /64 Subnets

- Scanning a subnet becomes a DoS attack!
  - Creates IPv6 version of 2^{64} ARP entries in routers
  - Exhaust address-translation table space

- So now we have:

```
ping6 ff02::1 All nodes in broadcast domain
ping6 ff02::2 All routers in broadcast domain
```

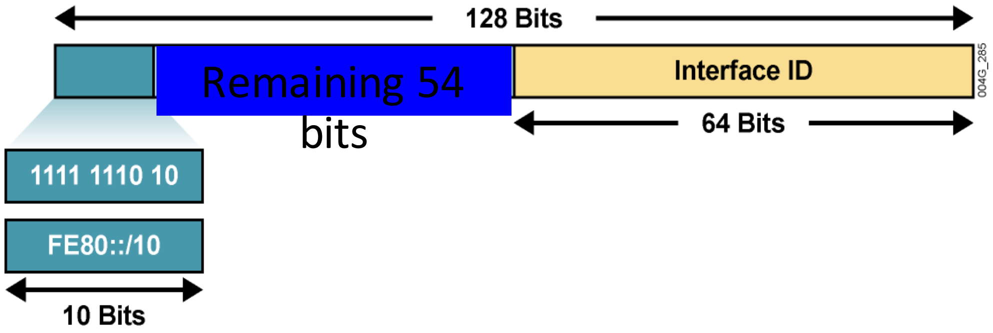
- Solutions
  - RFC 6164 recommends use of /127 to protect router-router links
  - RFC 3756 suggest “clever cache management” to address more generally

Types of IPv6 Addresses

- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Link-Local Addresses

- ‘11111110 10…’ binary (FE80::/10)
  - According to RFC 4291 bits 11-64 should be 0’s… so really FE80::/64?
- For use on a single link.
  - Automatic address configuration
  - Neighbor discovery (IPv6 ARP)
  - When no routers are present
  - Routers must not forward



- Addresses “chicken-or-egg” problem… need an address to get an address.
- Address assignment done unilaterally by node (later)
- IPv4 has link-local address (169.254/16, RFC 3927)
  - Only used if no globally routable addresses available

Types of IPv6 Addresses

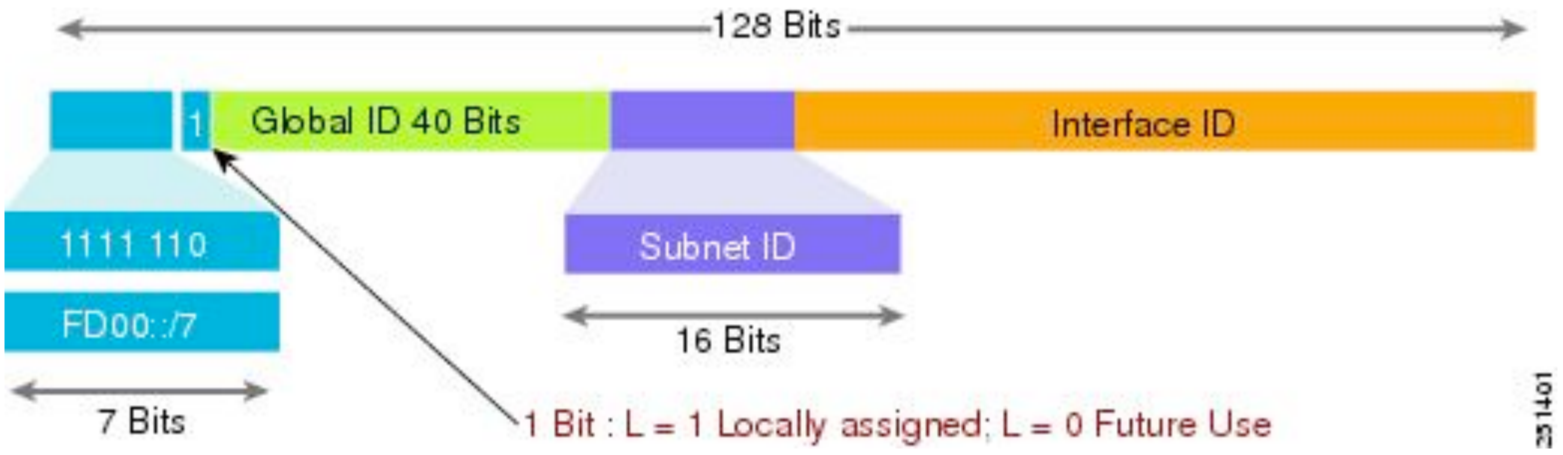
- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Unique Local Addresses

- ‘1111110…’ binary (`FC00::/7`)
- Globally unique addresses intended for local communication
  - IPv6 equivalent of IPv4 RFC 1918 addresses
- Defined in RFC 4193
  - Replace “site local” addresses defined in RFC 1884, deprecated in RFC 3879
- Should not be installed in global DNS
  - Can be installed in “local DNS”

Unique Local Addresses

- 4 parts
  - "**L**" bit always 1
  - **Global ID** (40 bits) randomly generated to enforce the idea that these addresses are not to be globally routed or aggregated
  - **Subnet ID** (16 bits)… same as Globally Unique Subnet ID
  - **Interface ID** (64 bits)… same as Globally Unique Interface ID

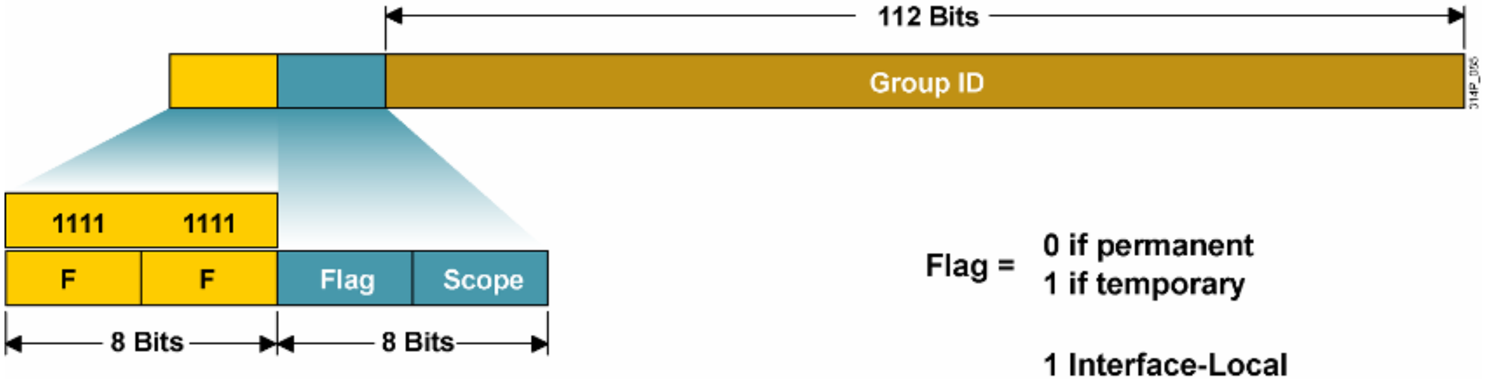


Types of IPv6 Addresses

- RFC 4291– “IPv6 Addressing Architecture”
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Multicast Addresses

- ‘11111111…’ binary (FF00::/8)
- Equivalent to IPv4 multicast (224.0.0.0/8)
- 3 parts
  - **Flag** (4 bits)
  - **Scope** (4 bits)



Flag = 0 if permanent
1 if temporary

Scope =
1 Interface-Local
2 Link-Local
3 ~~Subnet-Local~~
4 Admin-Local
5 Site-Local
8 Organization
E Global

Reserved Multicast Addresses

- All nodes
  - **FF01::1** – interface-local; used for loopback multicast transmissions
  - **FF02::1** – link-local; replaces IPv4 broadcast address (all 1's host)
- All routers
  - **FF01::2** (interface-local), **FF02::2** (link-local)
- Solicited-Node multicast
  - Used in Neighbor Discovery Protocol (later)
  - **FF02::FF00:0/104** (FF02::FFXX:XXXX)
  - Construct by replacing 'XX:XXXX' above with low-order 24 bits of a nodes unicast or anycast address
  - Example
    - For unicast address 4037::01:800:200E:8C6C
    - Solicited-Node multicast is FF02::1:FF0E:8C6C

Types of IPv6 Addresses

- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Anycast Addresses

- Allocated from unicast address space
  - Syntactically indistinguishable from unicast addresses
- An address assigned to more than one node
- Anycast traffic routed to the “nearest” host with the anycast address
- Typically used for a service (e.g. local DNS servers)
- Nodes must be configured to know an address is anycast
  - Don’t do Duplicate Address Detection
  - Advertise a route?

A Node’s Required Addresses

Red = new for IPv6

- Link-local address for each interface
- Configured unicast or anycast addresses
- Loopback address
- All-Nodes multicast interface and link addresses
- Solicited-Node multicast for each configured unicast and anycast address
- Multicast addresses for all groups the node is a member of
- Routers must add
  - Subnet-Router anycast address for each interface
    - Subnet prefix with all 0’s host part
  - All-Routers multicast address

Roundup: IPv6 Addresses

- “Interface ID” (host part) is 64 bits
- New addresses required by all nodes (host or router)
  - Link-local address
  - All-nodes interface-local and link-local multicast
  - Solicited-node multicast for each unicast/anycast address
- New addresses required by routers
  - All-routers interface-local, link-local and site-local multicast
  - Subnet-Router anycast for each interface?

Host Configuration

Assigning Address to Interfaces

- Static (manual) assignment
  - Needed for network equipment
- DHCPv6
  - Needed to track who uses an IP address
- **S**tate**L**ess **A**ddress **A**uto**C**onfiguration (**SLAAC**)
  - New to IPv6
- Describe SLAAC in the following…

SLAAC

- RFC 4862 – IPv6 Stateful Address Autoconfiguration
- Used to assign unicast addresses to interfaces
  - Link-Local Unicast
  - Global Unicast
  - Unique-Local Unicast?
- Goal is to minimize manual configuration
  - No manual configuration of hosts
  - Limited router configuration
  - No additional servers
- Use when "not particularly concerned with the exact addresses hosts use"
  - Otherwise use DHCPv6 (RFC 3315)

SLAAC Building Blocks

- Interface IDs

- Neighbor Discovery Protocol

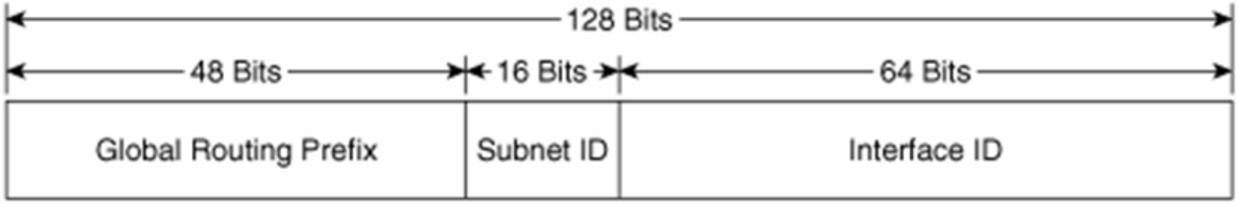
- SLAAC Process

SLAAC Building Blocks

- Interface IDs
- Neighbor Discovery Protocol
- SLAAC Process

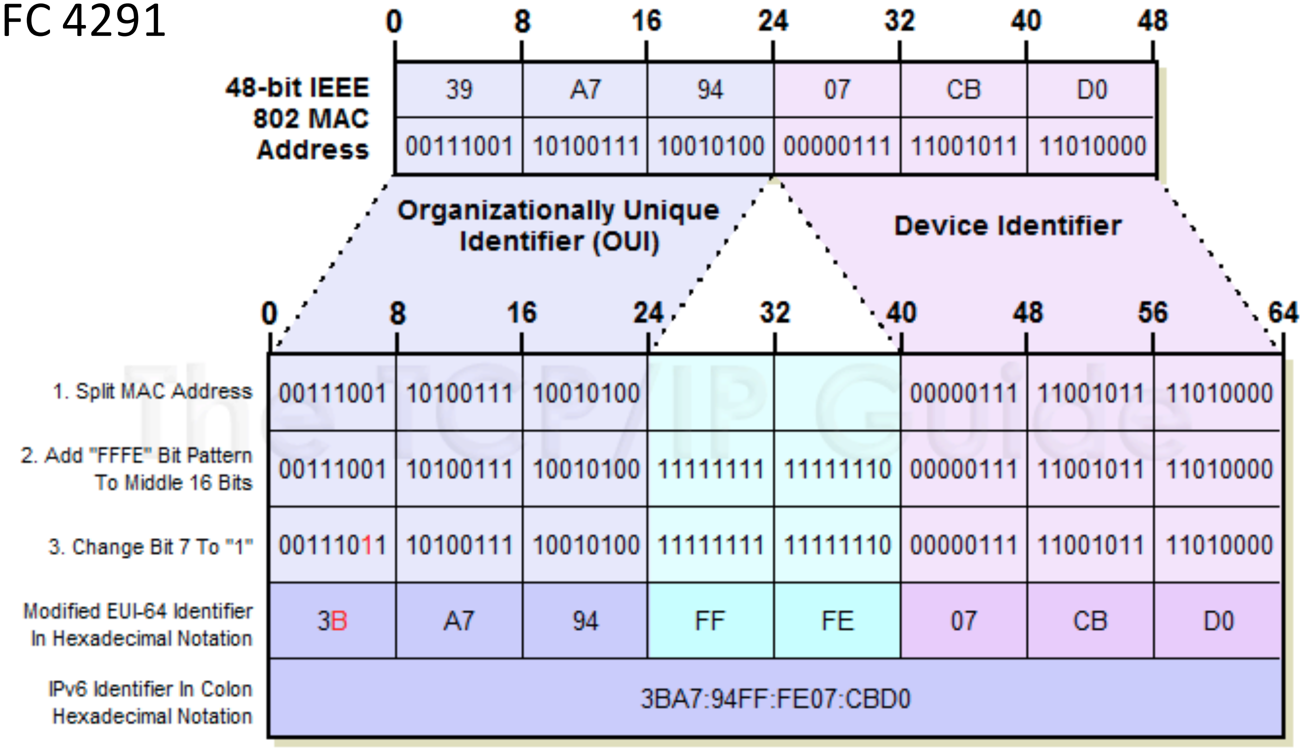
Interface IDs

- Used to identify a unique interface on a link
- Thought of as the “host portion” of an IPv6 address.
- 64 bits: To support both 48 bit and 64 bit IEEE MAC addresses
- Required to be unique on a link
- Subnets using auto addressing must be /64s.
- EUI-64 vs Privacy interface IDs



IEEE EUI-64 Option for Interface ID

- Use interface MAC address
- Insert FFFE to convert EUI-48 to EUI-64
- FlipUniversal/Local bit to “1”
  - Section 2.5.1 RFC 4291



48-bit IEEE 802 MAC Address						
	39	A7	94	07	CB	D0
	00111001	10100111	10010100	00000111	11001011	11010000

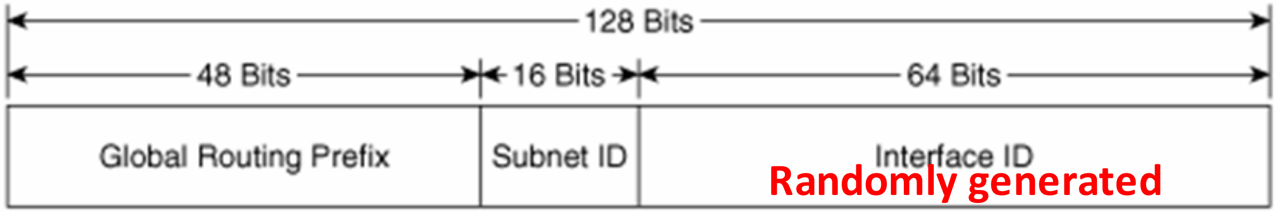
1. Split MAC Address	00111001	10100111	10010100			00000111	11001011	11010000
2. Add "FFFE" Bit Pattern To Middle 16 Bits	00111001	10100111	10010100	11111111	11111110	00000111	11001011	11010000
3. Change Bit 7 To "1"	00111011	10100111	10010100	11111111	11111110	00000111	11001011	11010000
Modified EUI-64 Identifier In Hexadecimal Notation	3B	A7	94	FF	FE	07	CB	D0
IPv6 Identifier In Colon Hexadecimal Notation	3BA7:94FF:FE07:CBD0							

The TCP/IP Guide

64-Bit IPv6 Modified EUI-64 Interface Identifier

Privacy Option for Interface ID

- Using MAC uniquely identifies a host… security/privacy concerns!
- Microsoft(!) defined an alternative solution for Interface IDs (RFC 4941)
- Hosts generates a random 64 bit Interface ID



SLAAC Building Blocks

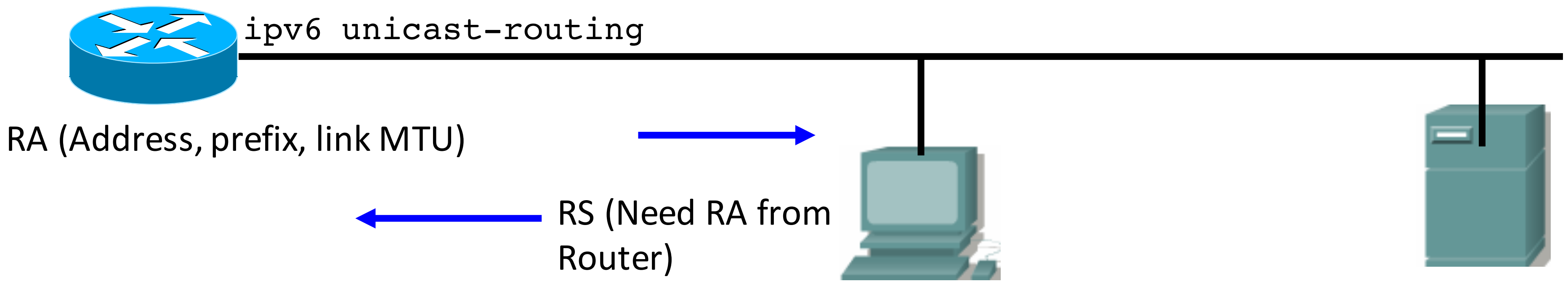
- Interface IDs
- Neighbor Discovery Protocol
- SLAAC Process

NDP

- RFC 4861 – Neighbor Discovery for IPv6
- Used to
  - Determine MAC address for nodes on same subnet (ARP)
  - Find routers on same subnet
  - Determine subnet prefix and MTU
  - Determine address of local DNS server (RFC 6106)
- Uses 5 ICMPv6 messages
  - **Router Solicitation** (**RS**) – request routers to send RA
  - **Router Advertisement** (**RA**) – router's address and subnet parameters
  - **Neighbor Solicitation** (**NS**) – request neighbor's MAC address (ARP Request)
  - **Neighbor Advertisement** (**NA**) – MAC address for an IPv6 address (ARP Reply)
  - **Redirect** – inform host of a better next hop for a destination

NDP RS & RA

- **Router Solicitation** (**RS**)
  - Originated by hosts to request that a router send an RA
  - Source = unspecified (::) or link-local address,
  - Destination = All-routers multicast (FF02::2)
- **Router Advertisement** (**RA**)
  - Originated by routers to advertise their address and link-specific parameters
  - Sent periodically and in response to Router Solicitation messages
  - Source = link-local address,
  - Destination = All-nodes multicast (FF02::1)



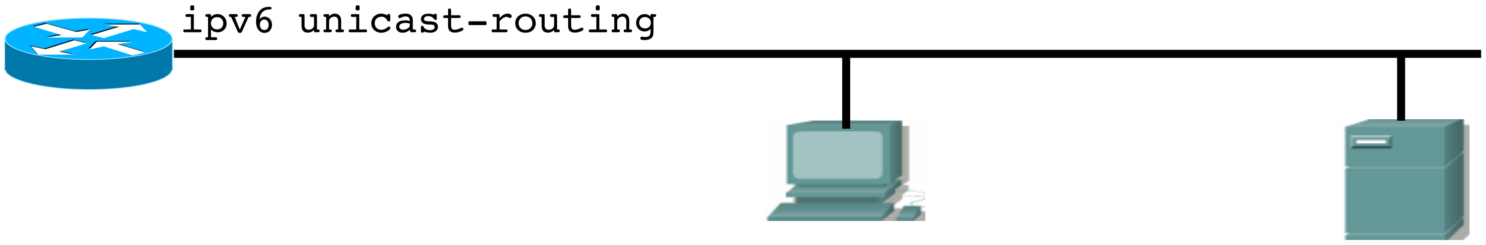
```
ipv6 unicast-routing
```

RA (Address, prefix, link MTU)

RS (Need RA from Router)

NDP NS & NA

- **Neighbor Solicitation** (**NS**)
  - Request ***target*** MAC address while providing target of source (IPv4 ARP Request)
  - Used to resolve address or verify reachability of neighbor
  - Source = unicast or "::" (Duplicate Address Detection… next slide)
  - Destination = solicited-node multicast
- **Neighbor Advertisement** (**NA**)
  - Advertise MAC address for given IPv6 address (IPv4 ARP Reply)
  - Respond to NS or communicate MAC address change
  - Source = unicast, destination = NS's source or all-nodes multicast (if source "::")



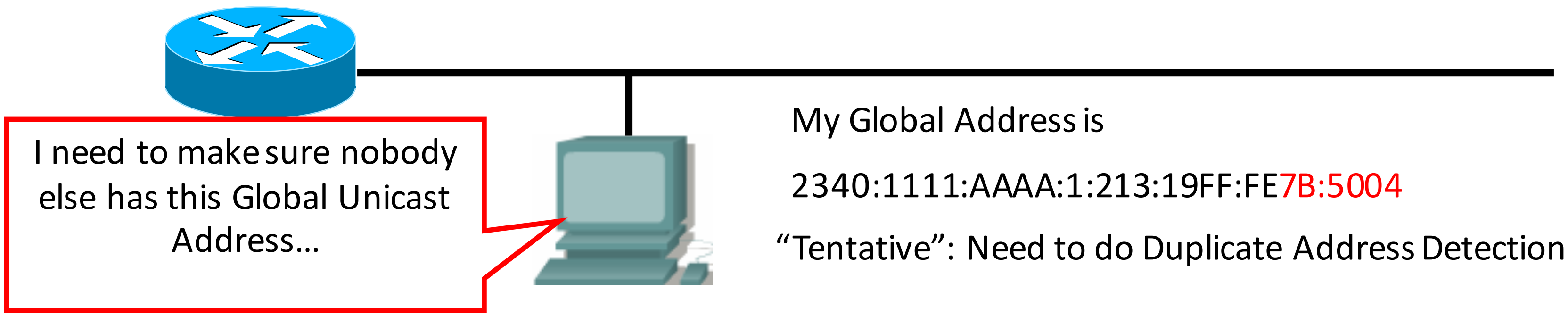
```
ipv6 unicast-routing
```

NS (Request for another node's Link Layer Address) → ← NA (Sent in response to NS)

Duplicate Address Detection

- **Duplicate Address Detection** (**DAD**) used to verify address is unique in subnet prior to assigning it to an interface
- **MUST** take place on all unicast addresses, regardless of whether they are obtained through stateful, stateless or manual configuration
- **MUST NOT** be performed on anycast addresses
- Uses Neighbor Solicitation and Neighbor Advertisement messages
- NS sent to solicited-node multicast; if no NA received address is unique
- **Solicited-node multicast**: `FF02::1:FF:0/104` w/ last 24 bits of target

Duplicate Address Detection



I need to make sure nobody else has this Global Unicast Address…

My Global Address is

2340:1111:AAAA:1:213:19FF:FE7B:5004

“Tentative”: Need to do Duplicate Address Detection

← NS (Neighbor Solicitation) →

- **Target Address** = 2340:1111:AAAA:1:213:19FF:FE7B:5004

Destination: Solicited-Node Multicast Address = `FF02::1::FF7B:5004`

SLAAC Building Blocks

- Interface IDs
- Neighbor Discovery Protocol
- SLAAC Process

SLAAC Steps

- Select link-local address
- Verify “tentative” address not in use by another host with DAD
- Send RS to solicit RAs from routers
- Receive RA with
  - router address,
  - subnet MTU,
  - subnet prefix,
  - local DNS server (RFC 6106)
- Generate global unicast address
- Verify address is not in use by another host with DAD

Create Link-local address

Link-local Address =
Link-local Prefix + Interface Identifier (EUI-64 format)
FE80 [64 bits]   + [48 bit MAC u/l flipped + 16 bit FFFE]

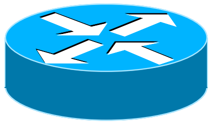
Make sure Link-local address is unique

NS (Neighbor Solicitation)

Make sure Link-local address is unique
DAD: Okay if no NA returned

Destination: Solicited-Node Multicast Address
Target address = Link-local address

Get Network Prefix to create Global unicast address



RS (Router Solicitation)

Get Prefix and other information

RA (Router Advertisement)

Source = Link-local address
Destin = FF02::1 All nodes multicast address
Query   = Prefix, Default Router, MTU, options

IPv6 Address =
Prefix + Interface ID (EUI-64 format)
[64 bits] + [48 bit MAC u/l flipped + 16 bit FFFE]

DAD

NS (Neighbor Solicitation)

Make sure IPv6 Address is unique
Target Address = IPv6 Address
DAD: Okay if no NA returned

Prefix Leases

- Prefix information contained in RA includes lifetime information
  - **Preferred lifetime**: when an address's preferred lifetime expires SHOULD only be used for existing communications
  - **Valid lifetime**: when an address's valid lifetime expires it MUST NOT be used as a source address or accepted as a destination address.

- Unsolicited RAs can reduce prefix lifetime values
  - Can be used to force re-addressing

Roundup: ICMPv6

- Implements router discovery and ARP functions

- ICMPv6 messages
  - Router Solicitation/Router Advertisement
  - Neighbor Solicitation/Neighbor Advertisement
  - (Next hop) Redirect

- Duplicate Address Detection (DAD)
  - verify unique link-local and global-unicast addresses
  - Uses:
    - NS/NA (i.e. gratuitous ARP)
    - Solicited node multicast address

Review - SLAAC

- Assigns link-local and global-unicast addresses
- Goals
  - Eliminate manual configuration
  - Require minimal router configuration
  - Require no additional servers
- Host part options
  - EUI-64
  - Random (“privacy” addresses)
- Steps
  - Generate link-local address and verify with DAD
  - Find router - RS/RA
  - Generate global unicast address and verify with DAD

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?

Summary Network Layer

- understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing (versus switching)
  - how a router works
  - routing (path selection)
  - IPv6
- Algorthims
  - Two routing approaches (LS vs DV)
  - One of these in detail (LS)
  - ARP