Overview of last class

The Internet 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

IP datagram format

IP protocol version number

header length (bytes)

"type" of data

max number remaining hops (decremented at each router)

upper layer protocol to deliver payload to

32 bits

ver	head. len	type of service	length
16-bit identifier		flgs	fragment offset
time to live	upper layer	header checksum
32 bit source IP address
32 bit destination IP address
options (if any)
data (variable length, typically a TCP or UDP segment)

total datagram length (bytes)

for fragmentation/ reassembly

e.g. timestamp, record route taken, specify list of routers to visit.

how much overhead?
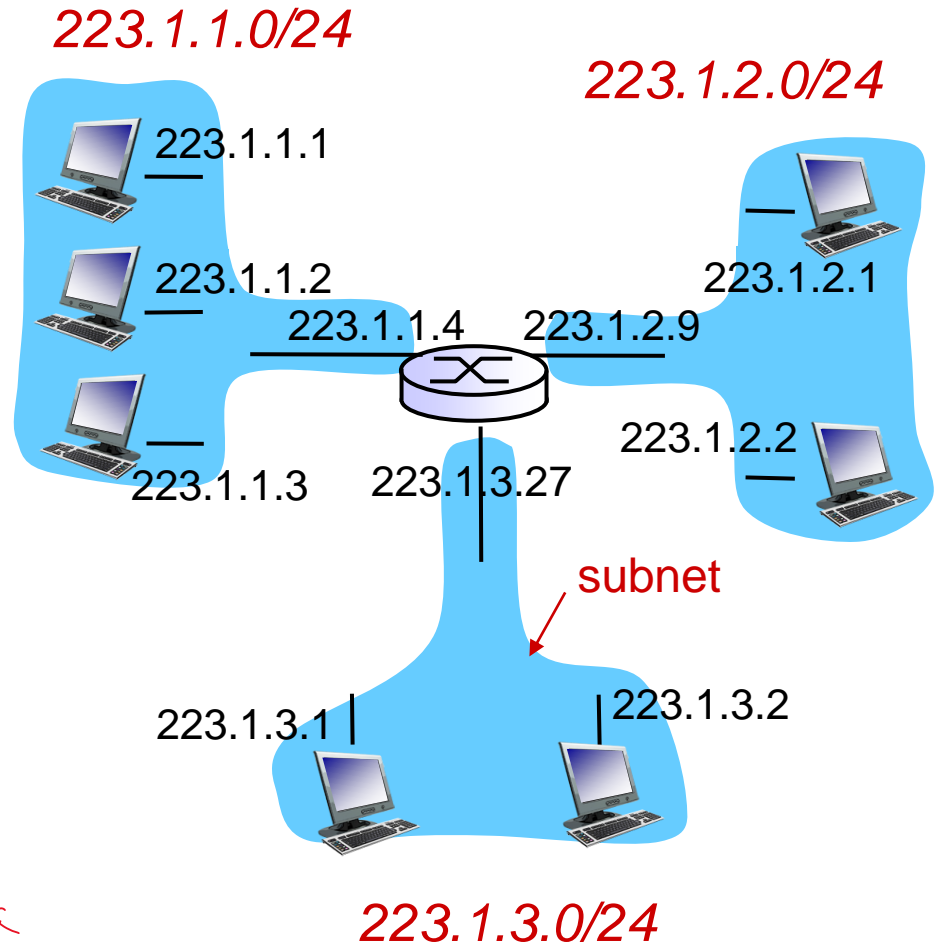- ❖ 20 bytes of TCP
- ❖ 20 bytes of IP
- ❖ = 40 bytes + app layer overhead

Subnets

recipe

- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks

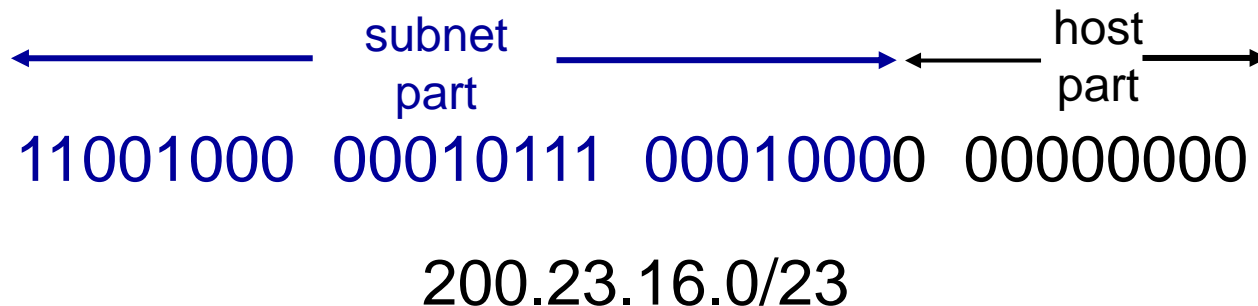- each isolated network is called a *subnet*

Building # : Subnet mask

Room # within the building : Specific address within its subnet

223.1.1.0/24

223.1.2.0/24

223.1.1.1

223.1.1.2

223.1.1.4   223.1.2.9

223.1.2.1

223.1.1.3   223.1.3.27

223.1.2.2

subnet

223.1.3.1   223.1.3.2

223.1.3.0/24

subnet mask: /24

IP addressing: CIDR

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

$\overset{\text{subnet}}{\overset{\text{part}}{\longleftarrow\hspace{3cm}\longrightarrow}}$ $\overset{\text{host}}{\overset{\text{part}}{\longleftrightarrow}}$

11001000 00010111 00010000 00000000

200.23.16.0/23

DHCP client-server scenario

App

DHCP server: 223.1.2.5

67 : DHCP Server

68 : DHCP Client

UDP

DHCP discover

arriving client

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
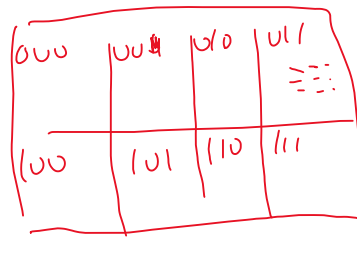yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

time

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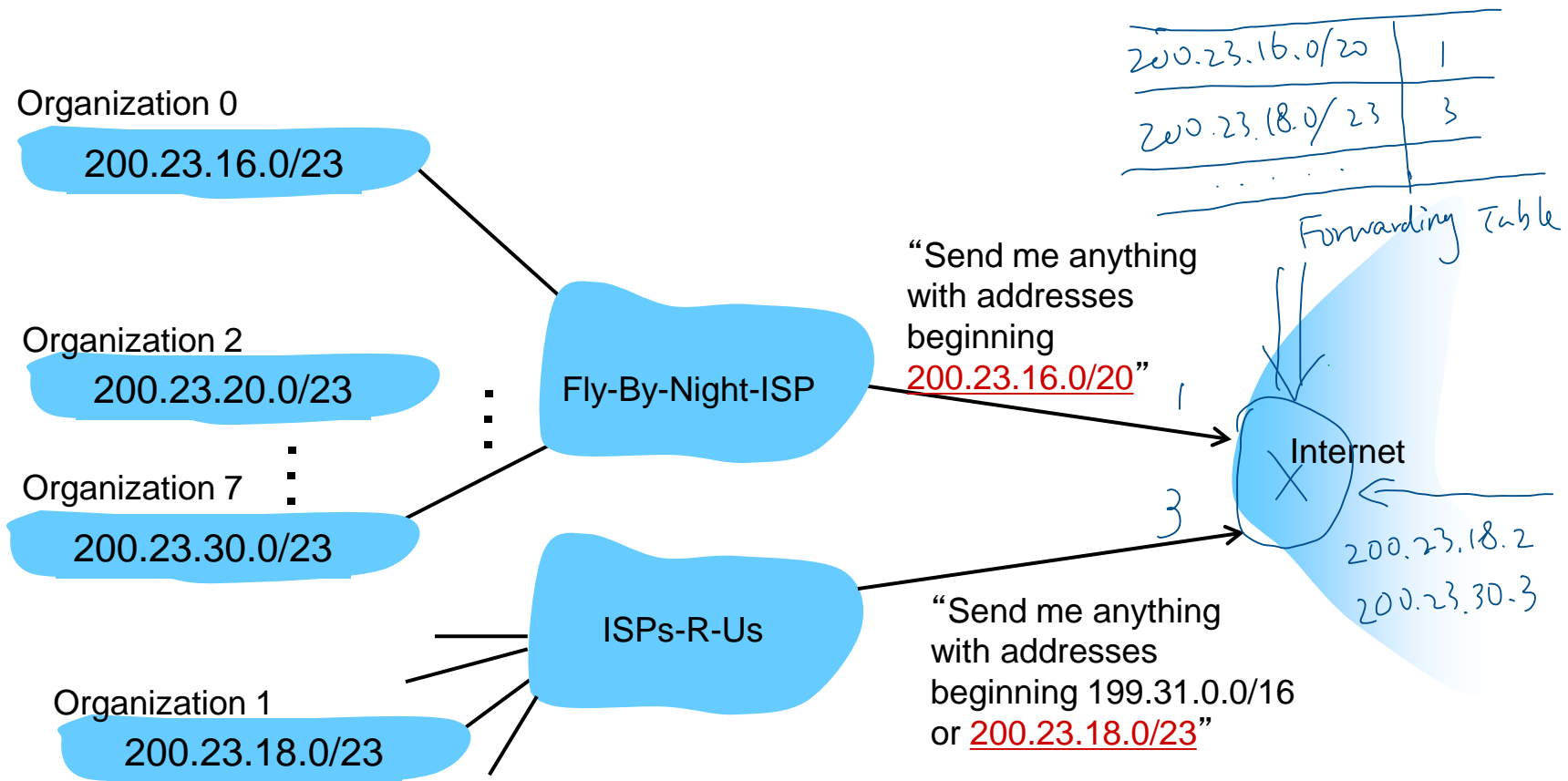
Q: how does *network* get subnet part of IP addr?

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

Rout aggregation and longest prefix match

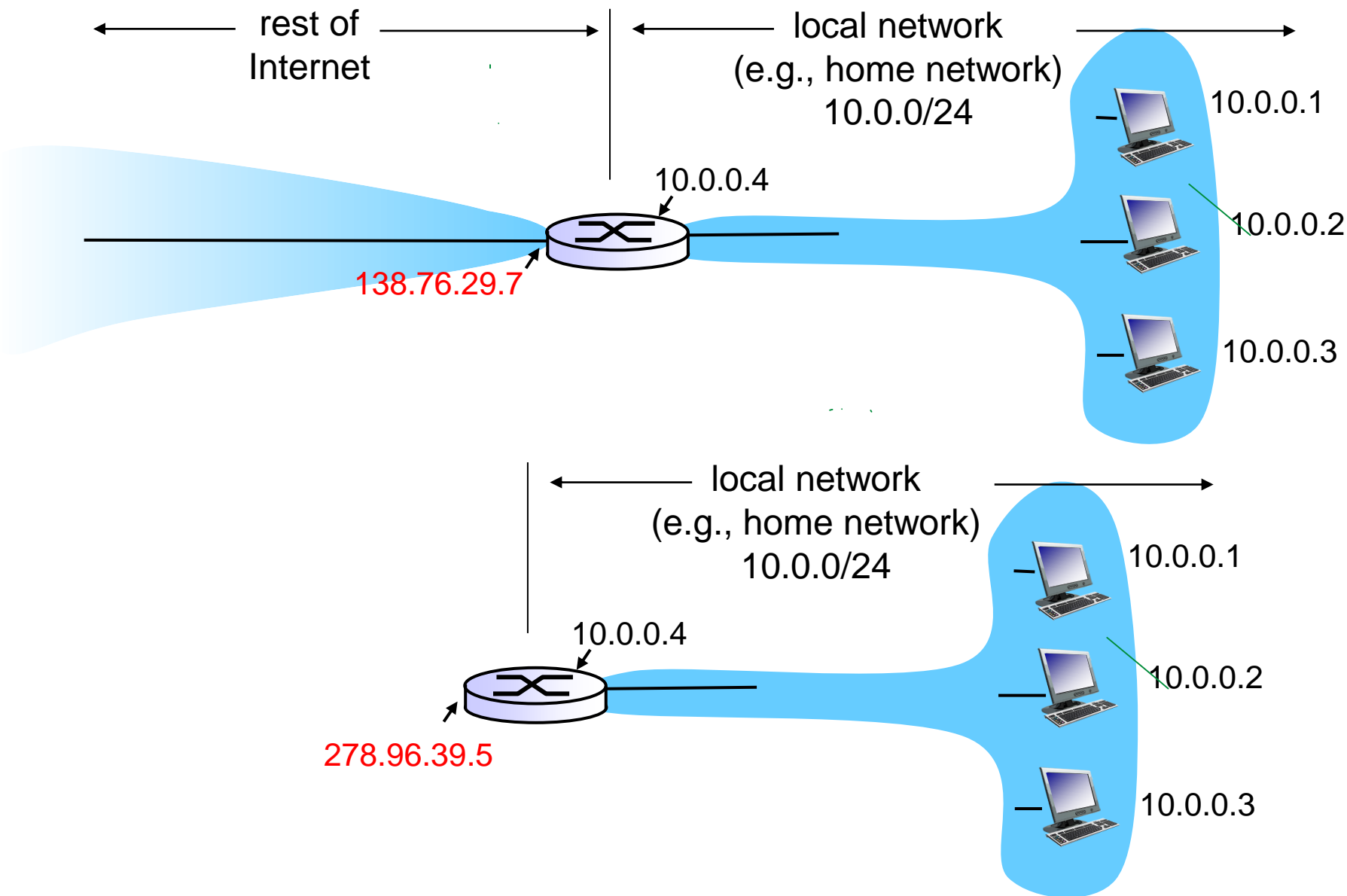ISPs-R-Us has a more specific route to Organization 1

Organization 0
200.23.16.0/23

Organization 2
200.23.20.0/23

Organization 7
200.23.30.0/23

Fly-By-Night-ISP

ISPs-R-Us

Organization 1
200.23.18.0/23

"Send me anything
with addresses
beginning
200.23.16.0/20"

"Send me anything
with addresses
beginning 199.31.0.0/16
or 200.23.18.0/23"

Internet

200.23.16.0/20 | 1
200.23.18.0/23 | 3
. . . . . .

Forwarding Table

200.23.18.2
200.23.30.3

Class Today

NAT: network address translation

rest of Internet

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

10.0.0.1

10.0.0.2

10.0.0.3

10.0.0.4
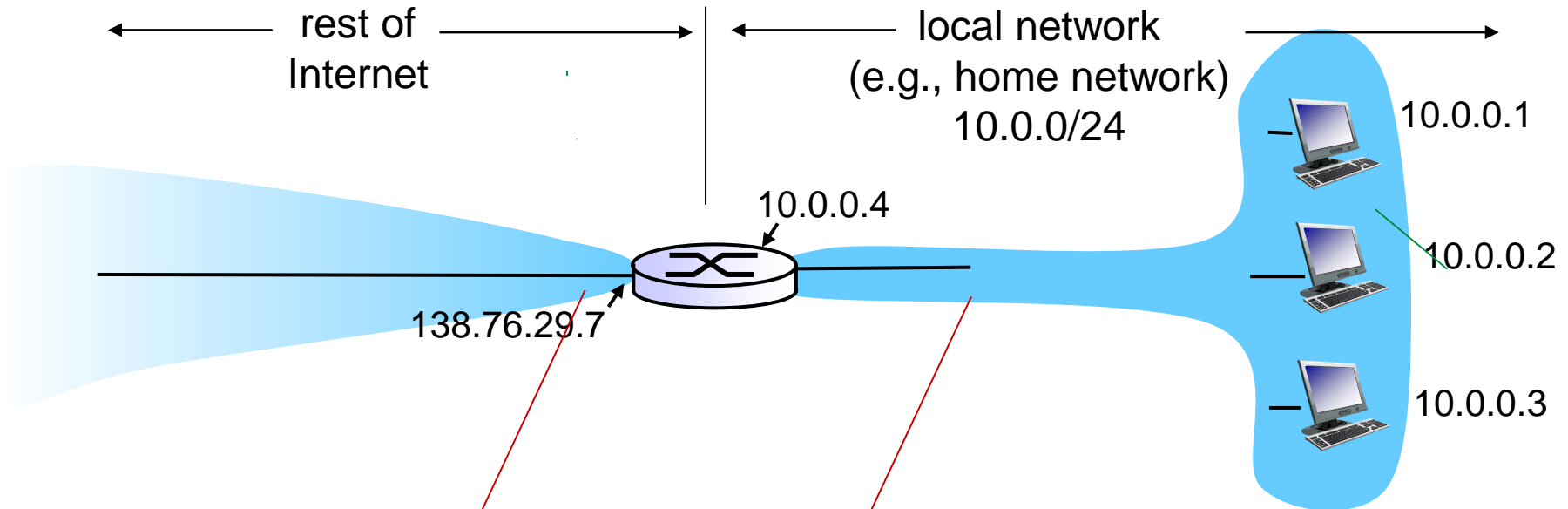
138.76.29.7

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)

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

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

10.0.0.1

10.0.0.4

10.0.0.2

278.96.39.5

10.0.0.3

IP addresses for local/private networks

- 192.168.0.0 – 192.168.255.255 (65,536 **IP addresses, 16bits**)
- 172.16.0.0 – 172.31.255.255 (1,048,576 **IP addresses, 20bits**)
- 10.0.0.0 – 10.255.255.255 (16,777,216 **IP addresses, 24bits**)

NAT: network address translation

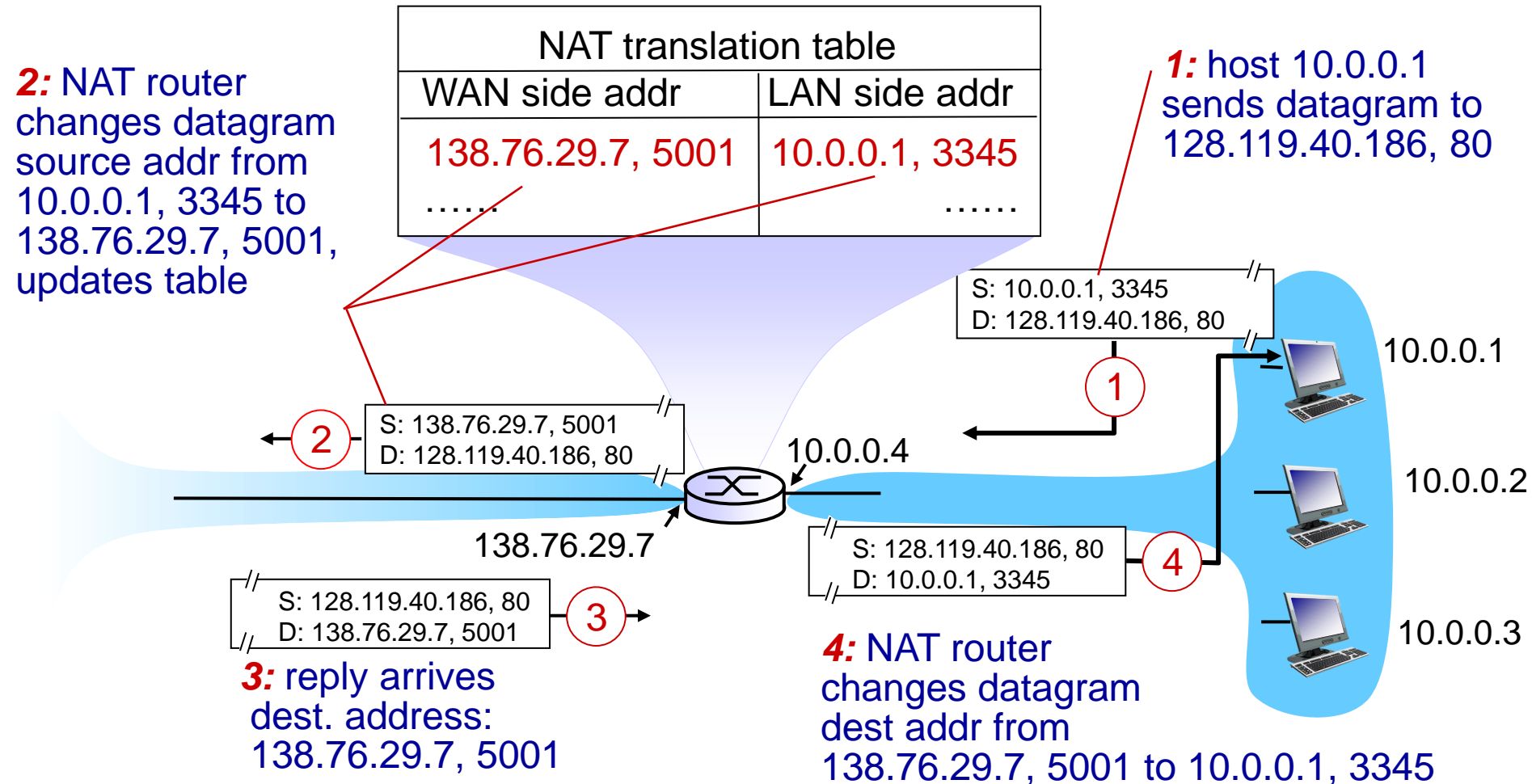*motivation:* local network uses just one IP address as far as outside world is concerned:

- 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
……	……

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

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

S: 10.0.0.1, 3345
D: 128.119.40.186, 80

1

10.0.0.1

S: 138.76.29.7, 5001
D: 128.119.40.186, 80

2

10.0.0.4

138.76.29.7

S: 128.119.40.186, 80
D: 10.0.0.1, 3345

4

10.0.0.2

S: 128.119.40.186, 80
D: 138.76.29.7, 5001

3

3: reply arrives dest. address: 138.76.29.7, 5001

10.0.0.3

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

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

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
  - address shortage should be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
 - NAT traversal: what if client wants to connect to server behind NAT?

NAT traversal problem

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

Client ?

10.0.0.1

10.0.0.4

138.76.29.7   NAT router
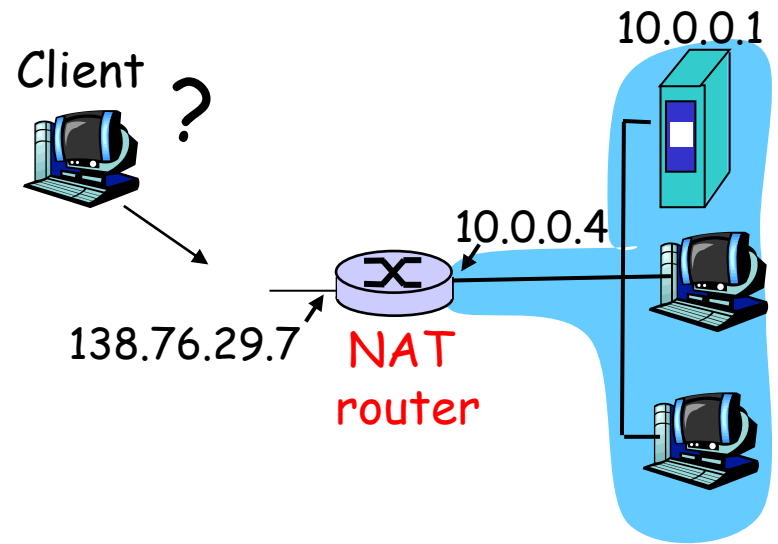
NAT traversal problem

- 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

10.0.0.1

UPnP/IGD

10.0.0.4

138.76.29.7   NAT router
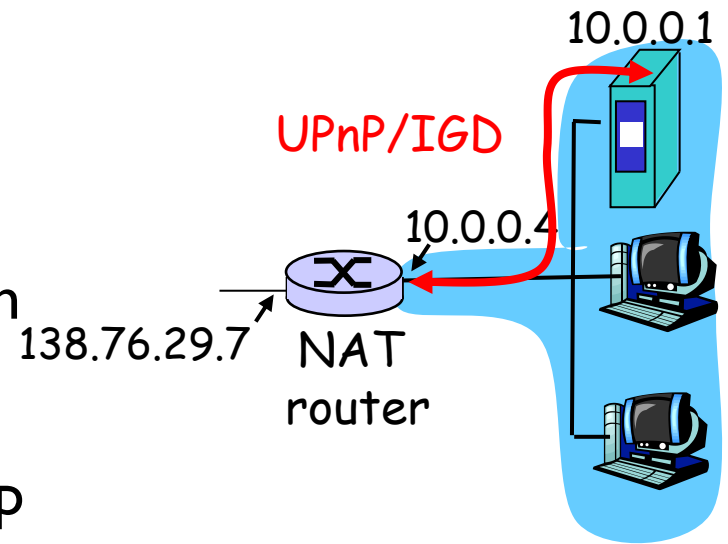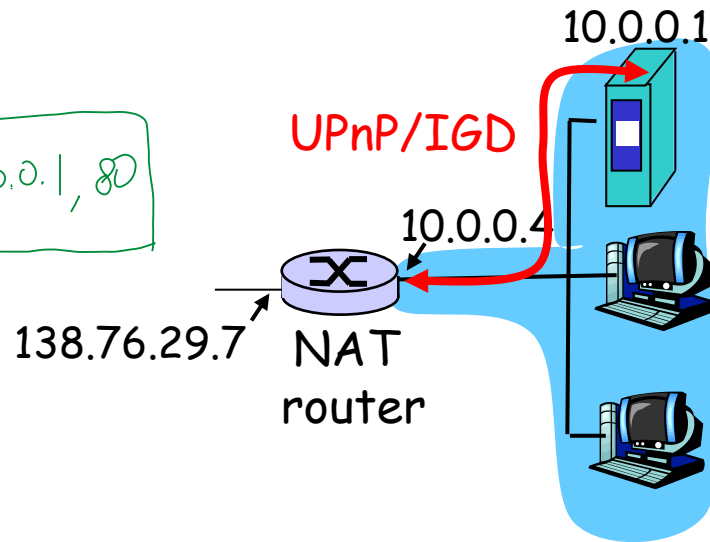
Alicesolution.com
10.0.0.1 , 80

10.0.0.1

UPnP/IGD

| 138.76.29.7, 80 | 10.0.0.1, 80 |

10.0.0.4

138.76.29.7   NAT
router

① Alice open startup Alicesolution.com, run company
   web at 10.0.0.1, with port 80

② UPnP/IGD automatically establish record

| 138.76.29.7, 80 | 10.0.0.1, 80 |

③ Insert DNS record (Alicesolution.com , 138.76.29.7)

NAT traversal problem

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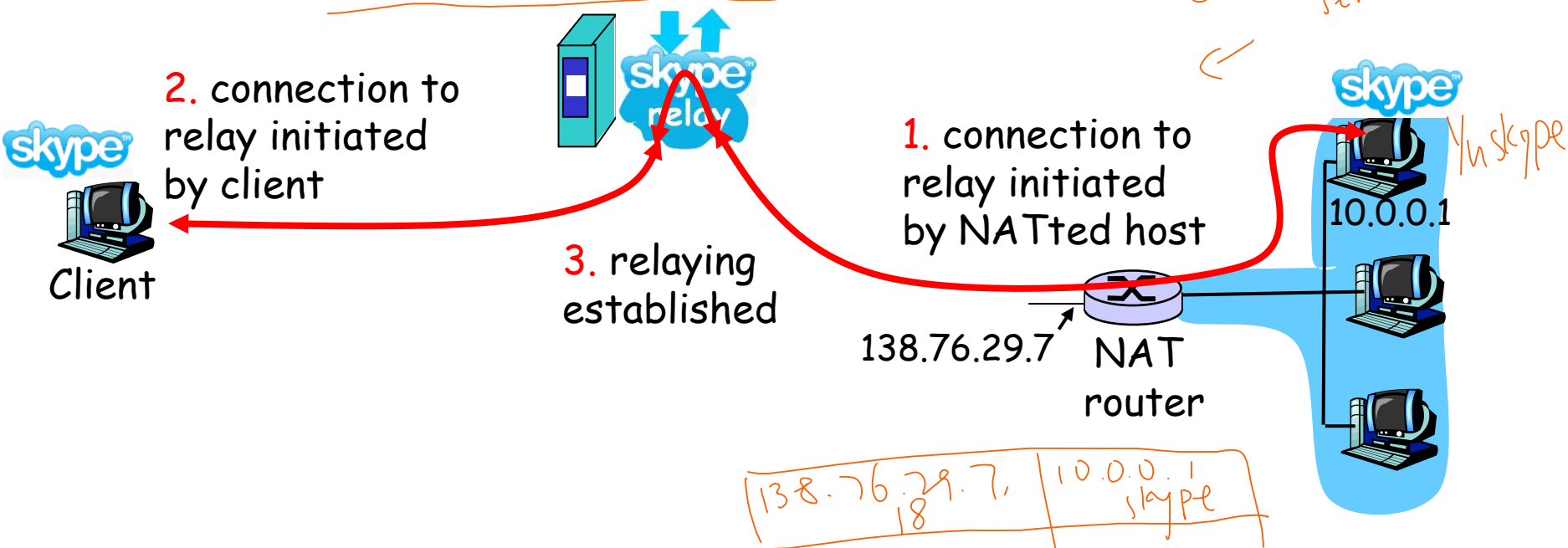 - relay bridges packets between to connections

Xuskype: 138.76.29.7, 18

login to skype server

Xuskype

2. connection to relay initiated by client

Client

3. relaying established

1. connection to relay initiated by NATted host

10.0.0.1

138.76.29.7   NAT router

138.76.29.7, 18	10.0.0.1 skype

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane

4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
- OpenFlow examples of match-plus-action in action

IPv6: motivation

- *initial motivation:* **32-bit** address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:
  - fixed-length 40 byte header
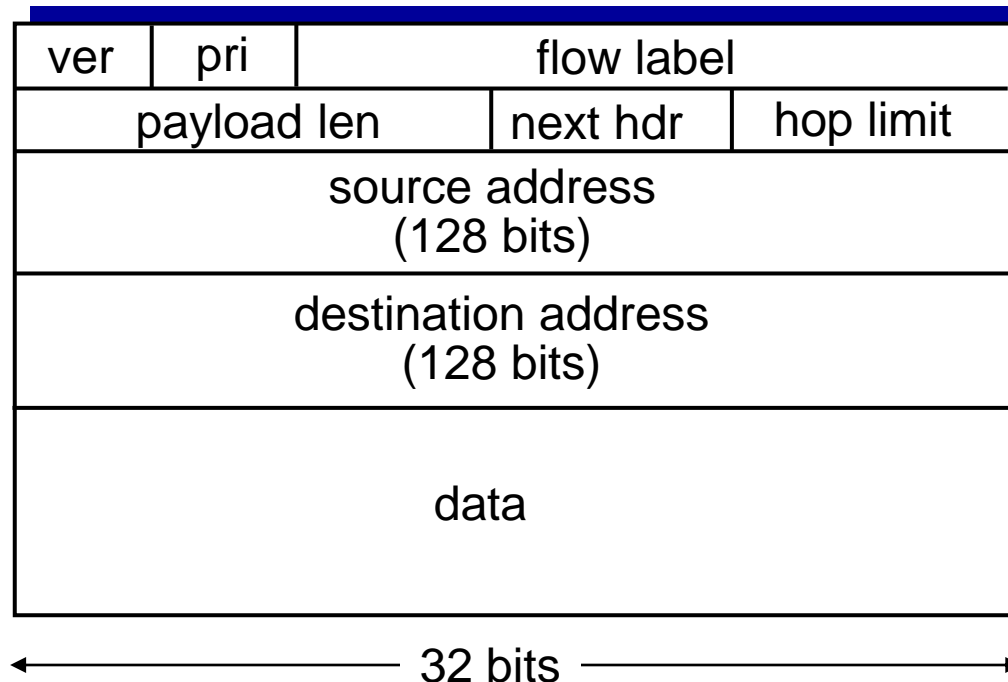 - no fragmentation allowed

IPv6 datagram format

priority: identify priority among datagrams in flow
flow Label: identify datagrams in same "flow."
(concept of "flow" not well defined).
next header: identify upper layer protocol for data

ver	pri	flow label		
payload len		next hdr		hop limit
source address (128 bits)				
destination address (128 bits)				
data				

⟵————— 32 bits —————⟶

Quality of Service

- *Pri bits:* can be used to indicate a service class: premium, assured, or best effort

- *QoS requirements for premium:*
  - In order delivery
  - Guaranteed bandwidth

- *Implementation for premium service*
  - virtual-circuit based resource allocation: fixed path with reserved bandwidth along the path
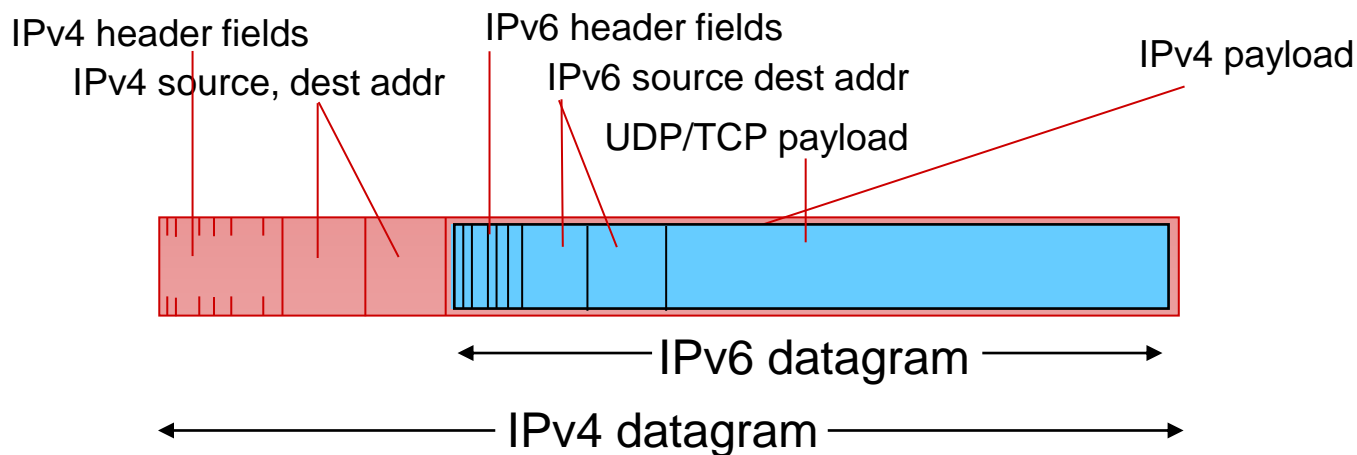 - Packets carrying the same *flow label* use the same virtual circuit

Other changes from IPv4

- *checksum:* removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by "Next Header" field
  - 8bits
  - TCP, UDP, ICMP, or an extension header for other functions (e.g., specifiy the path for a flow)
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
  - multicast group management functions

Transition from IPv4 to IPv6
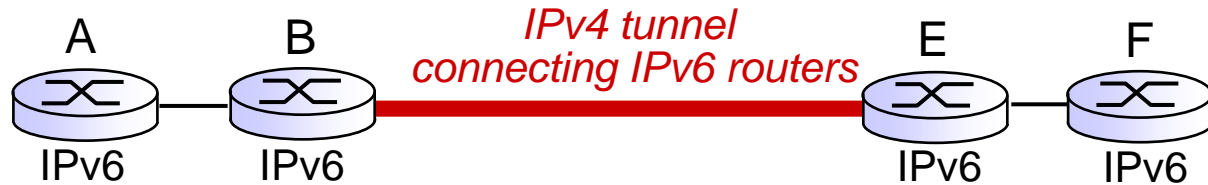
backward compatibility

- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
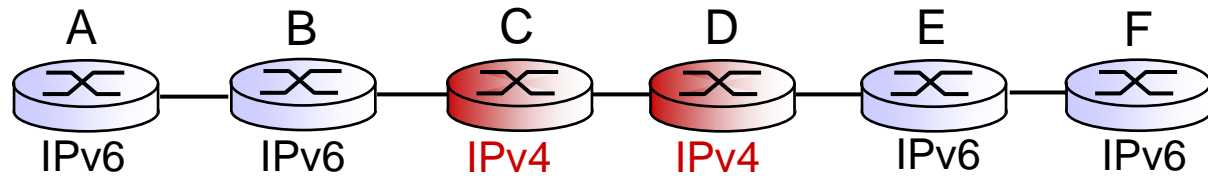- *tunneling:* IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers

IPv4 header fields

IPv4 source, dest addr

IPv6 header fields

IPv6 source dest addr

UDP/TCP payload

IPv4 payload

IPv6 datagram

IPv4 datagram

Tunneling

logical view:

A — B ======= *IPv4 tunnel connecting IPv6 routers* ======= E — F

IPv6   IPv6   IPv6   IPv6

physical view:

A — B — C — D — E — F

IPv6   IPv6   IPv4   IPv4   IPv6   IPv6

Tunneling

logical view:

A   B   *IPv4 tunnel*   E   F
                 connecting IPv6 routers

IPv6   IPv6   IPv6   IPv6

physical view:

A   B   C   D   E   F

IPv6   IPv6   IPv4   IPv4   IPv6   IPv6

flow: X
src: A
dest: F

data

src:B
dest: E

Flow: X
Src: A
Dest: F

data

src:B
dest: E

Flow: X
Src: A
Dest: F

data

flow: X
src: A
dest: F

data

A-to-B:
IPv6

B-to-C:
IPv6 inside
IPv4

B-to-C:
IPv6 inside
IPv4

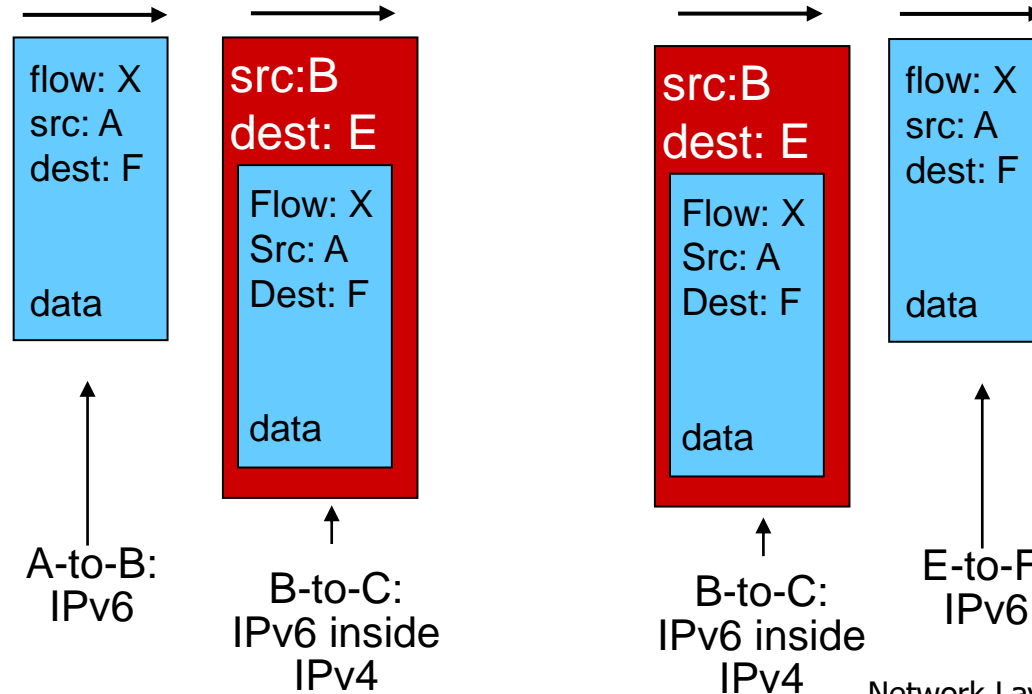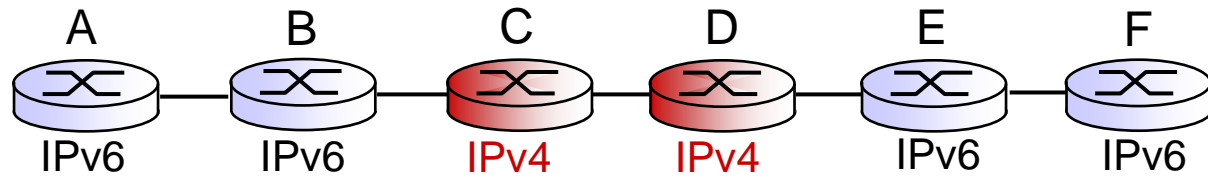E-to-F:
IPv6

IPv6: adoption

- Google: 8% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

- *Long (long!) time for deployment, use*
  - 20 years and counting!
  - think of application-level changes in last 20 years: WWW, Facebook, streaming media, Skype, …

Chapter 4: outline

4.1 Overview of Network layer
- data plane
- control plane
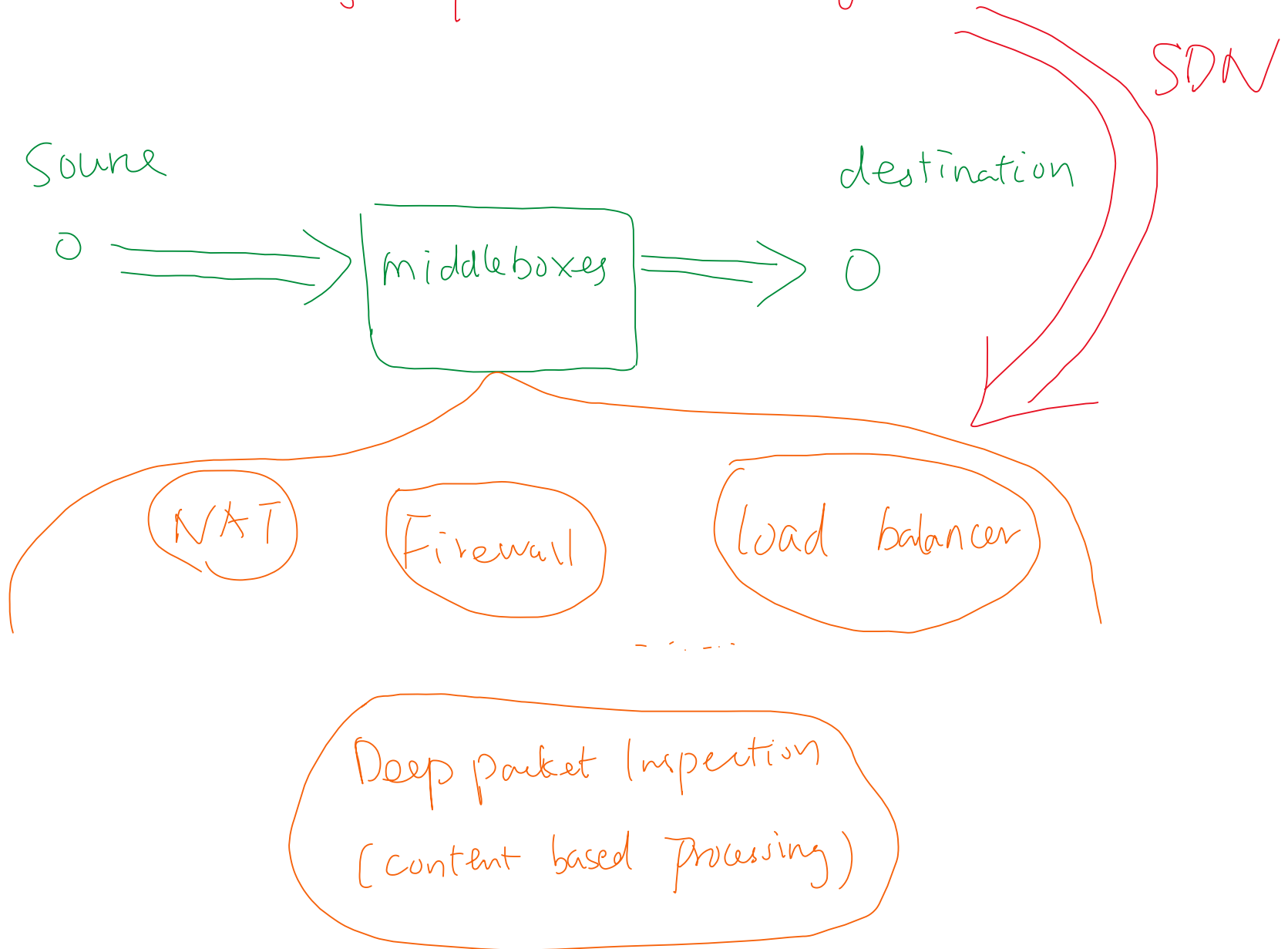
4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- network address translation
- IPv6

4.4 Generalized Forward and SDN
- match
- action
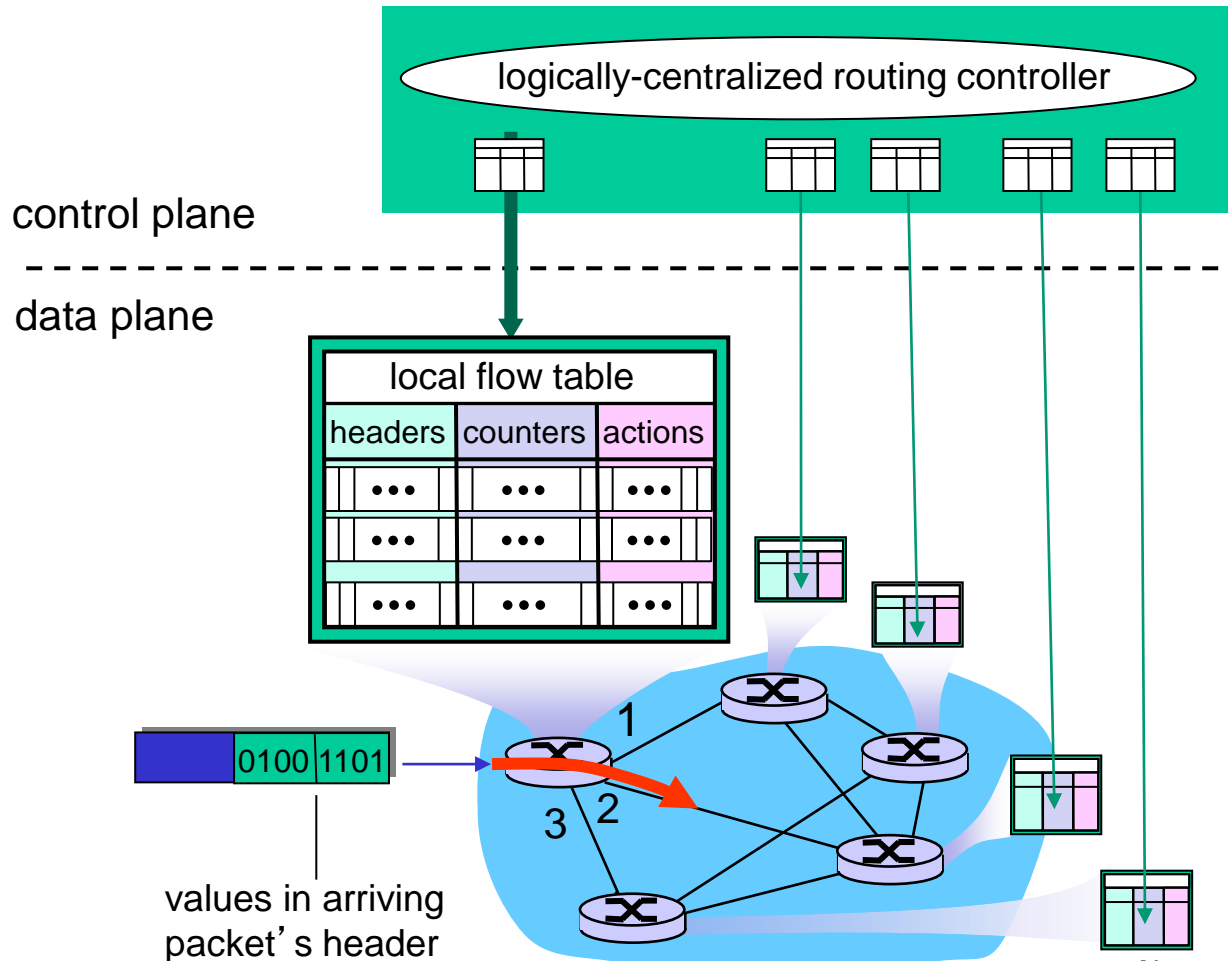- OpenFlow examples of match-plus-action in action

Unified / streamlined Design

SDN

Source

destination

O $\longrightarrow$ | middleboxes | $\Longrightarrow$ O

- NAT
- Firewall
- load balancer

Deep packet Inspection (content based Processing)

Generalized Forwarding and SDN

Each router contains a *flow table* that is computed and distributed by a *logically centralized* routing controller

logically-centralized routing controller

control plane

data plane

local flow table

headers	counters	actions
...	...	...
...	...	...
...	...	...

0100 1101

values in arriving packet's header

1

3 2

OpenFlow data plane abstraction

- *flow*: defined by header fields

- generalized forwarding: simple packet-handling rules

  - *Pattern:* match values in packet header fields

  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller

  - *Priority*: disambiguate overlapping patterns

  - *Counters:* #bytes and #packets
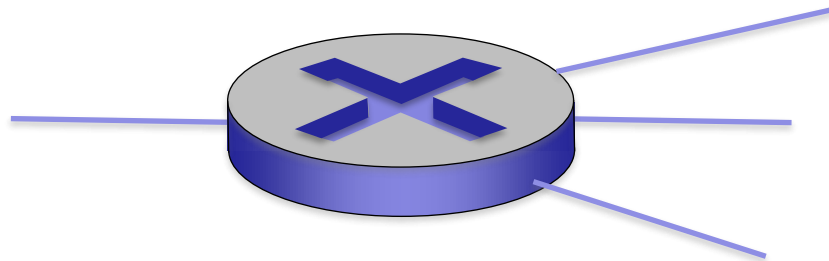
Flow table in a router (computed and distributed by controller) define router's match+action rules
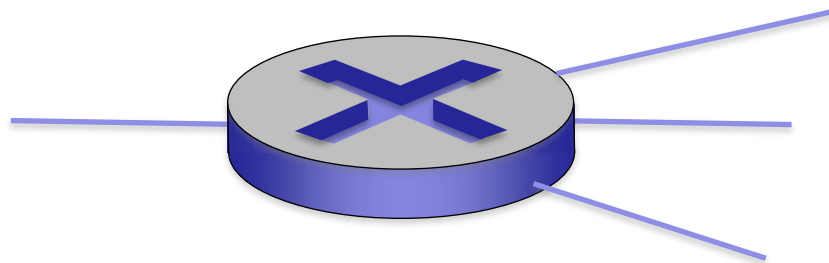
OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern:* match values in packet header fields
  - *Actions: for matched packet:* drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters:* #bytes and #packets

* : wildcard

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=*.*.*.* → send to controller

OpenFlow: Flow Table Entries

Rule	Action	Stats

Packet + byte counters

1. Send to normal processing pipeline (normal forwarding)
2. Forward packet to multiple port (load balancing)
3. Drop packet (firewall)
4. Modify Fields (NAT)
5. Encapsulate and forward to controller (deep packet inspection)

Switch Port	VLAN ID	MAC src	MAC dst	Eth type	IP Src	IP Dst	IP Prot	TCP sport	TCP dport

Link layer   Network layer   Transport layer

OpenFlow abstraction

- *match+action:* unifies different kinds of devices

- Router
  - *match:* longest destination IP prefix
  - *action:* forward out a link
- Switch
  - *match:* destination MAC address
  - *action:* forward or flood

- Firewall
  - *match*: IP addresses and TCP/UDP port numbers
  - *action:* permit or deny
- NAT
  - *match:* IP address and port
  - *action:* rewrite address and port

Examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

do not forward (block) all datagrams sent by host 128.119.1.1
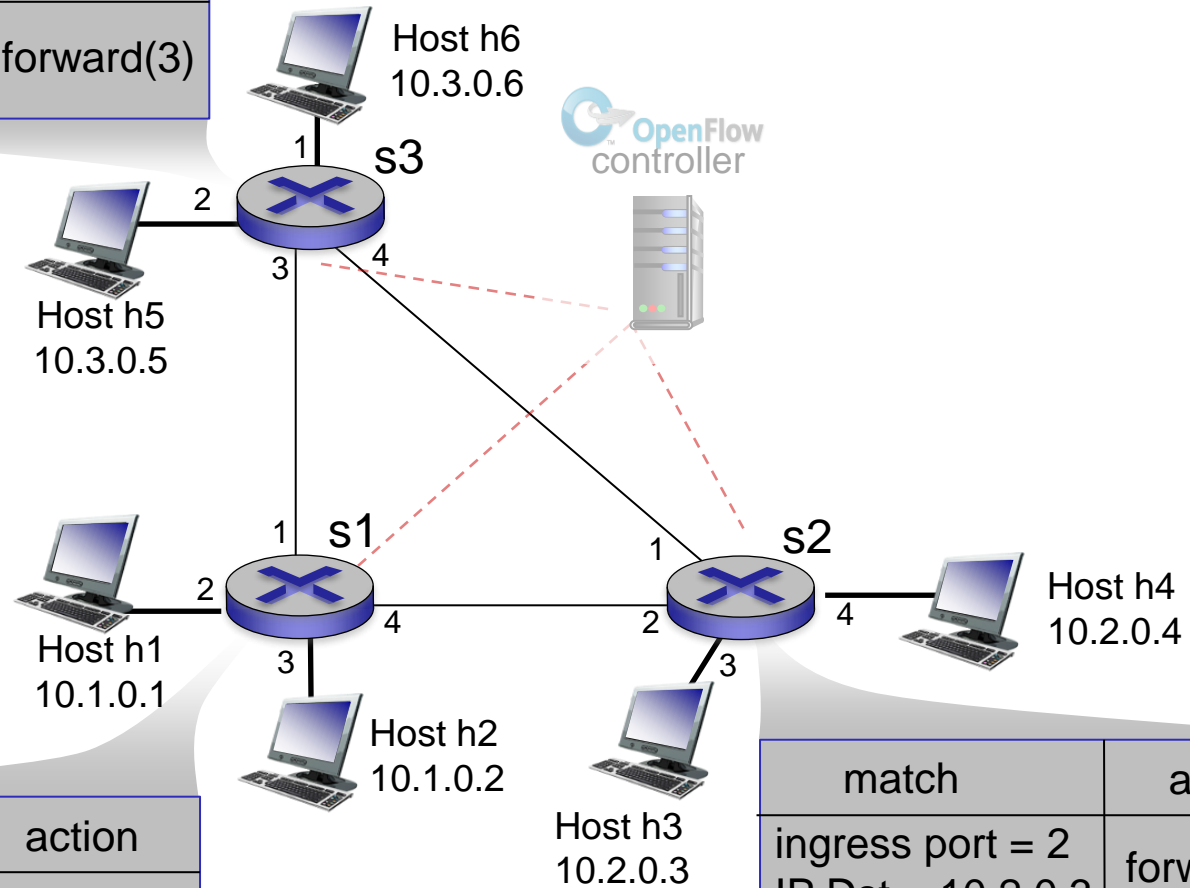
Examples

Destination-based layer 2 (switch) forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	port3

*layer 2 frames from MAC address 22:A7:23:11:E1:02
should be forwarded to output port 3*

OpenFlow example

Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

match	action
IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(3)

Host h6
10.3.0.6

OpenFlow
controller

s3

Host h5
10.3.0.5

s1

s2

Host h1
10.1.0.1

Host h4
10.2.0.4

Host h2
10.1.0.2

Host h3
10.2.0.3

match	action
ingress port = 1 IP Src = 10.3.*.* IP Dst = 10.2.*.*	forward(4)

match	action
ingress port = 2 IP Dst = 10.2.0.3	forward(3)
ingress port = 2 IP Dst = 10.2.0.4	forward(4)

Chapter 4: *done!*

4.1 Overview of Network layer: data plane and control plane

4.2 What's inside a router

4.3 IP: Internet Protocol
- datagram format
- fragmentation
- IPv4 addressing
- NAT
- IPv6

4.4 Generalized Forward and SDN
- match plus action
- OpenFlow example

Question: how do forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

Answer: by the control plane (next chapter)