

# Network layer: “data plane” roadmap

- Network layer: overview

- data plane
- control plane

- What’s inside a router

- input ports, switching, output ports
- buffer management, scheduling

- IP: the Internet Protocol

- datagram format
- addressing
- network address translation
- IPv6



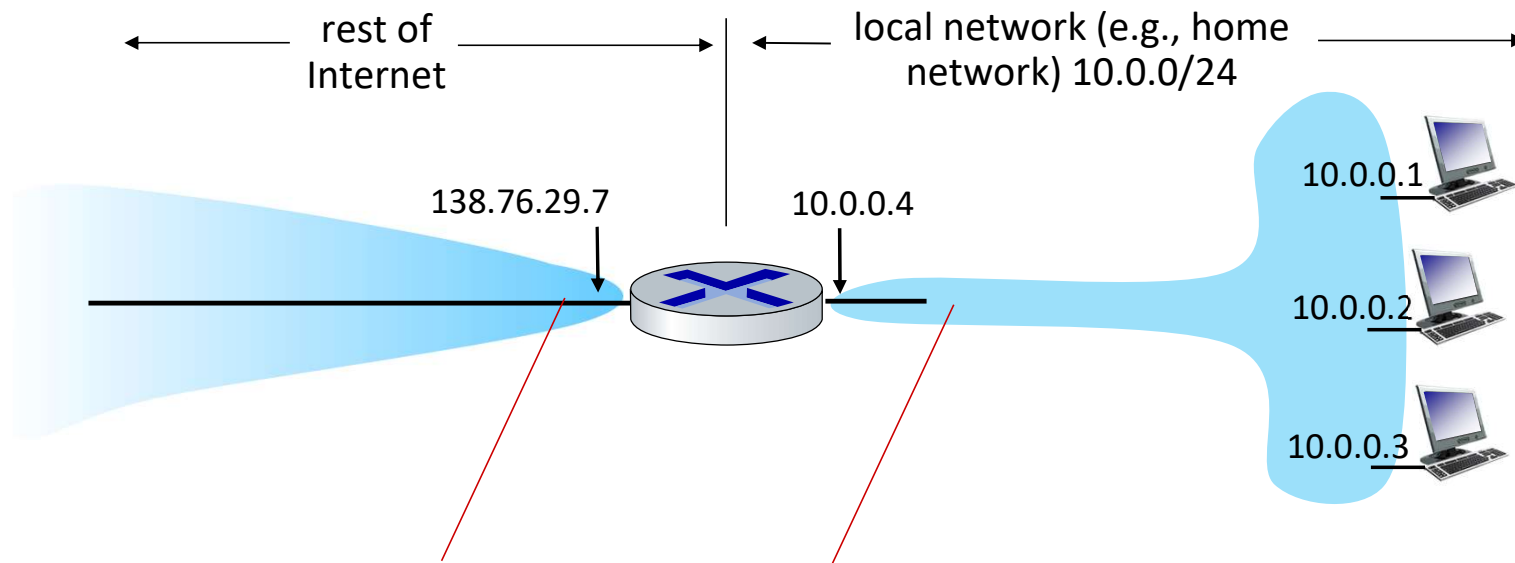
- Generalized Forwarding, SDN

- match+action
- OpenFlow: match+action in action

- Middleboxes

# NAT: network address translation

**NAT:** all devices in local network share just **one** IPv4 address as far as outside world is concerned



*all* datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *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

- all devices in local network have 32-bit addresses in a “private” IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just **one** IP address needed from provider ISP for *all* devices
  - can change addresses of host in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - security: devices inside local net not directly addressable, visible by outside world

# NAT: network address translation

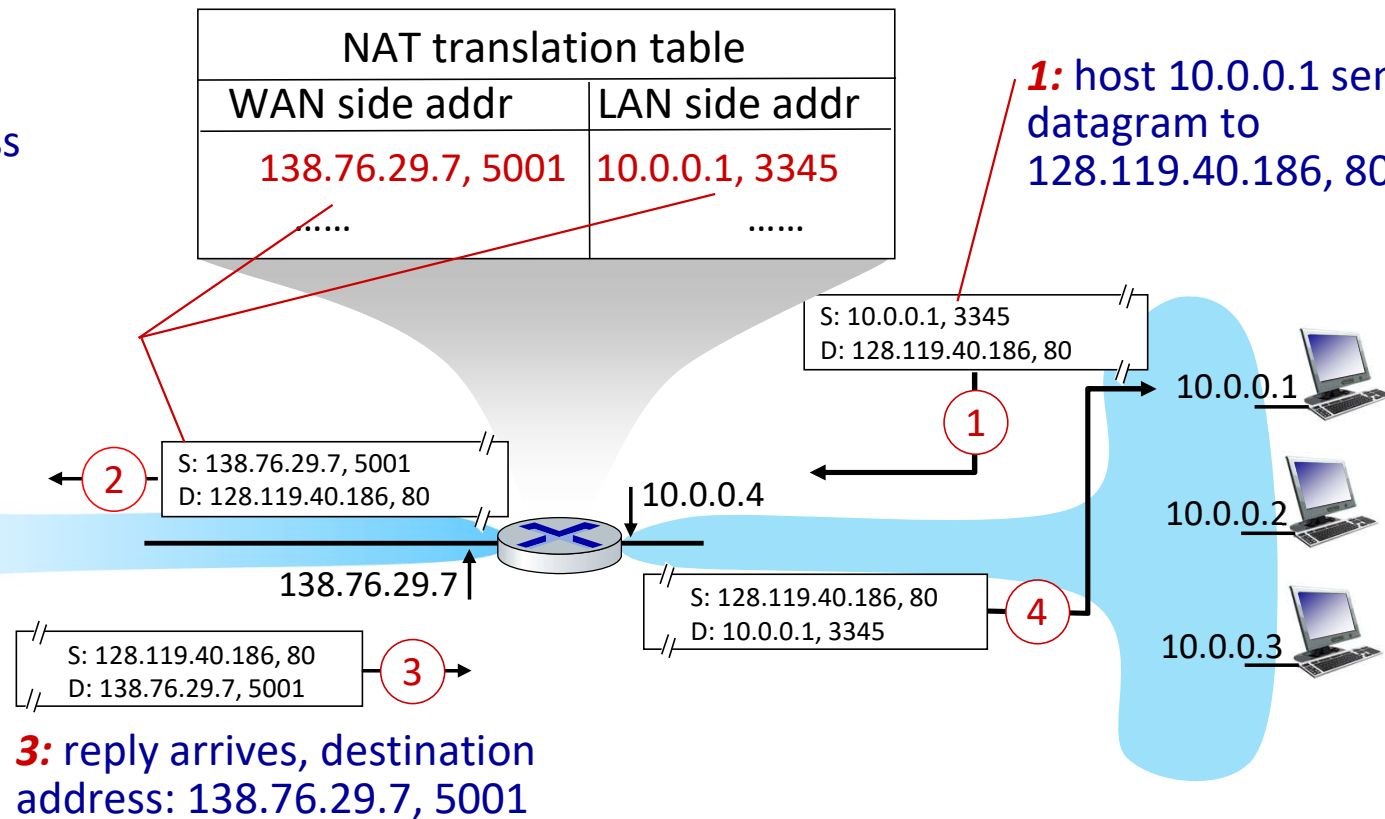
**implementation:** NAT router must (transparently):

- **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 address
- **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 destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

# NAT: network address translation

**2:** NAT router changes datagram source address 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



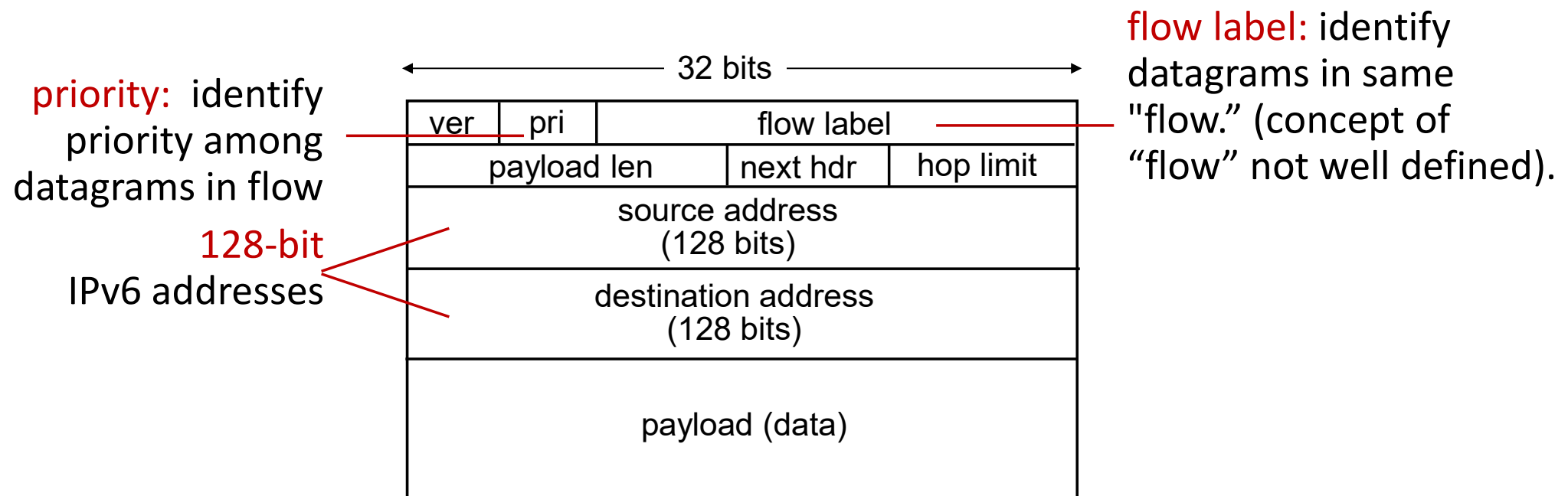
# NAT: network address translation

- NAT has been controversial:
  - routers “should” only process up to layer 3
  - address “shortage” should be solved by IPv6
  - violates end-to-end argument (port # manipulation by network-layer device)
  - NAT traversal: what if client wants to connect to server behind NAT?
- but NAT is here to stay:
  - extensively used in home and institutional nets, 4G/5G cellular nets

# IPv6: motivation

- **initial motivation:** 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of “flows”

# IPv6 datagram format



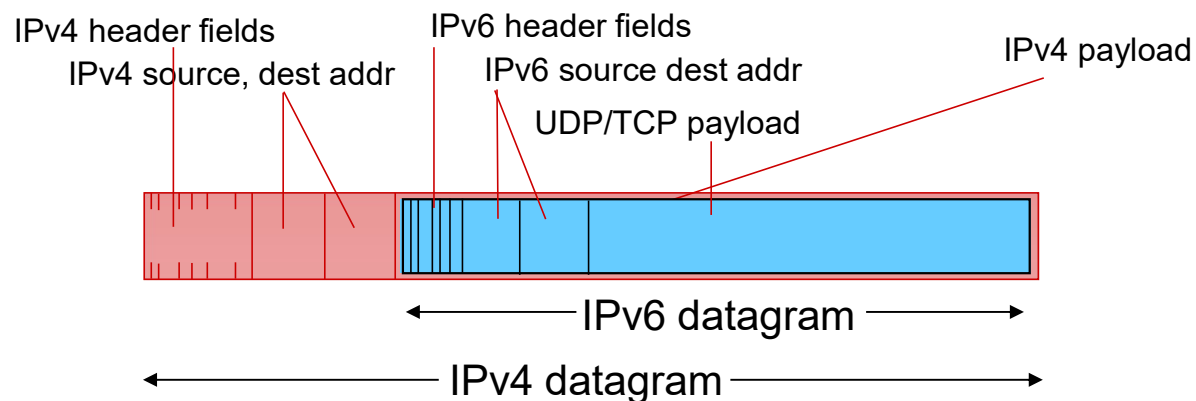
What's missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)



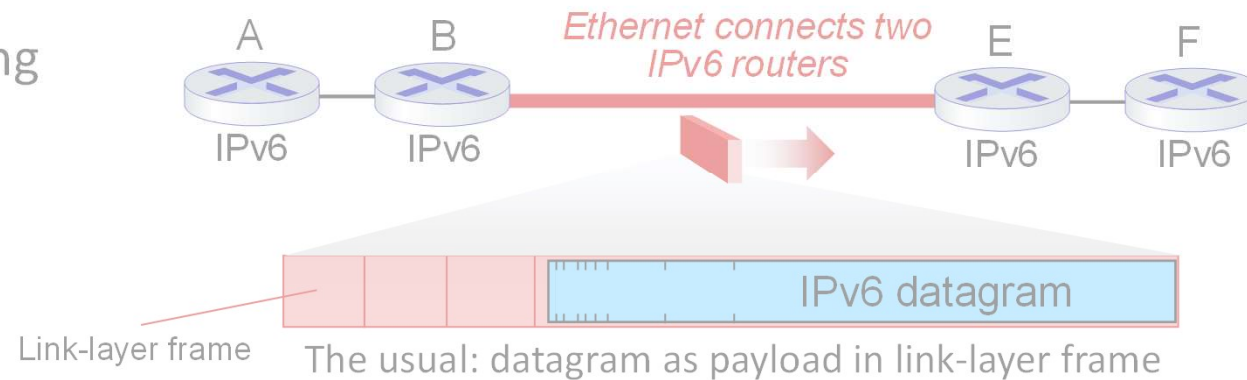
# Transition from IPv4 to IPv6

- 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 (“packet within a packet”)
  - tunneling used extensively in other contexts (4G/5G)

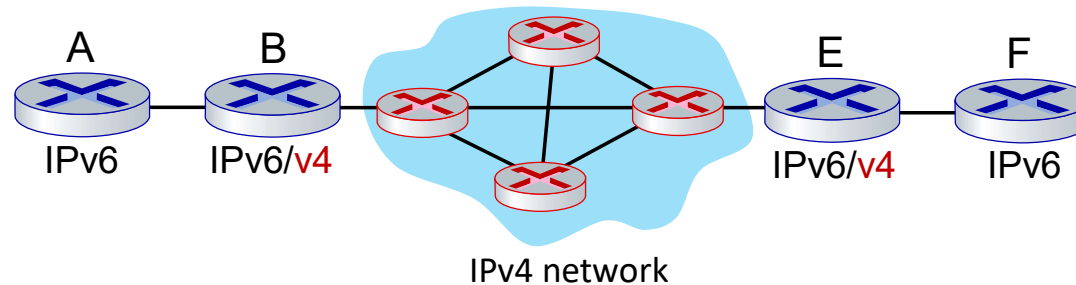


# Tunneling and encapsulation

Ethernet connecting two IPv6 routers:

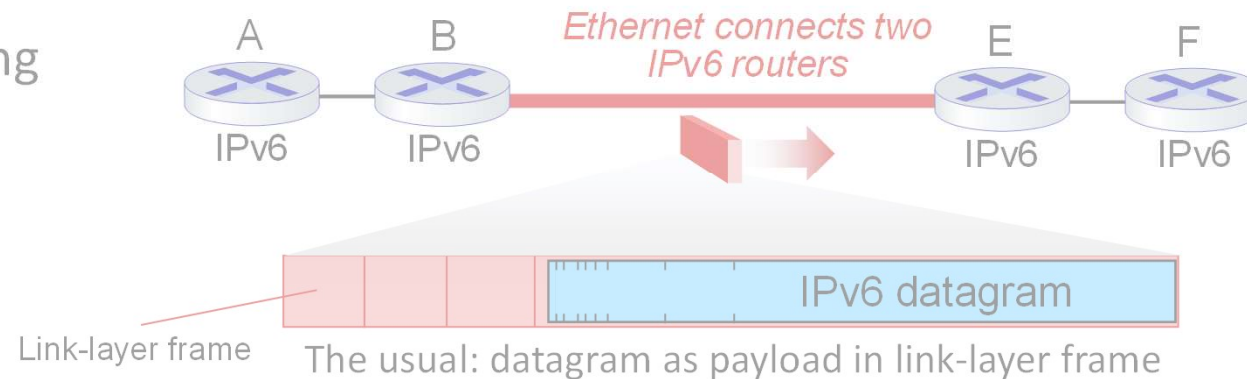


IPv4 network connecting two IPv6 routers

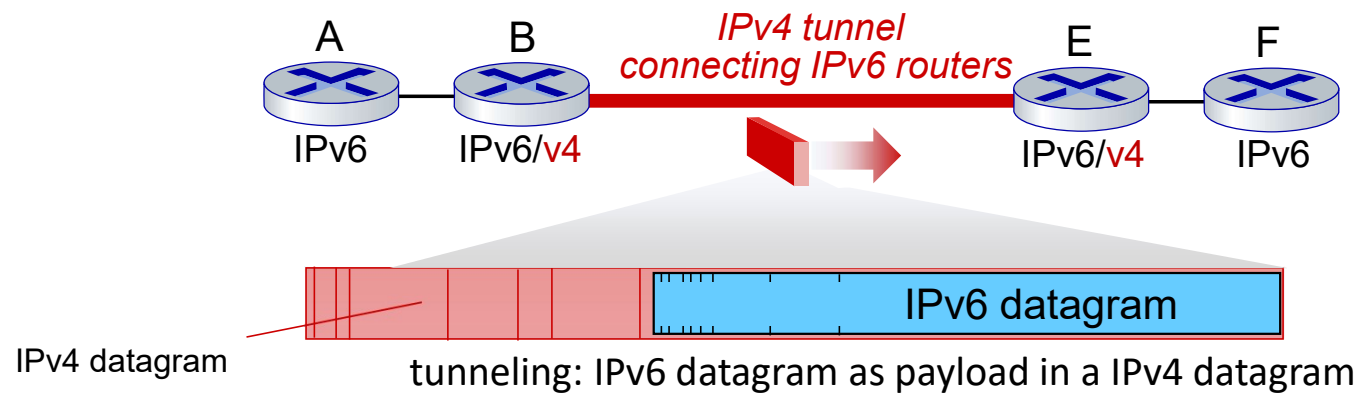


# Tunneling and encapsulation

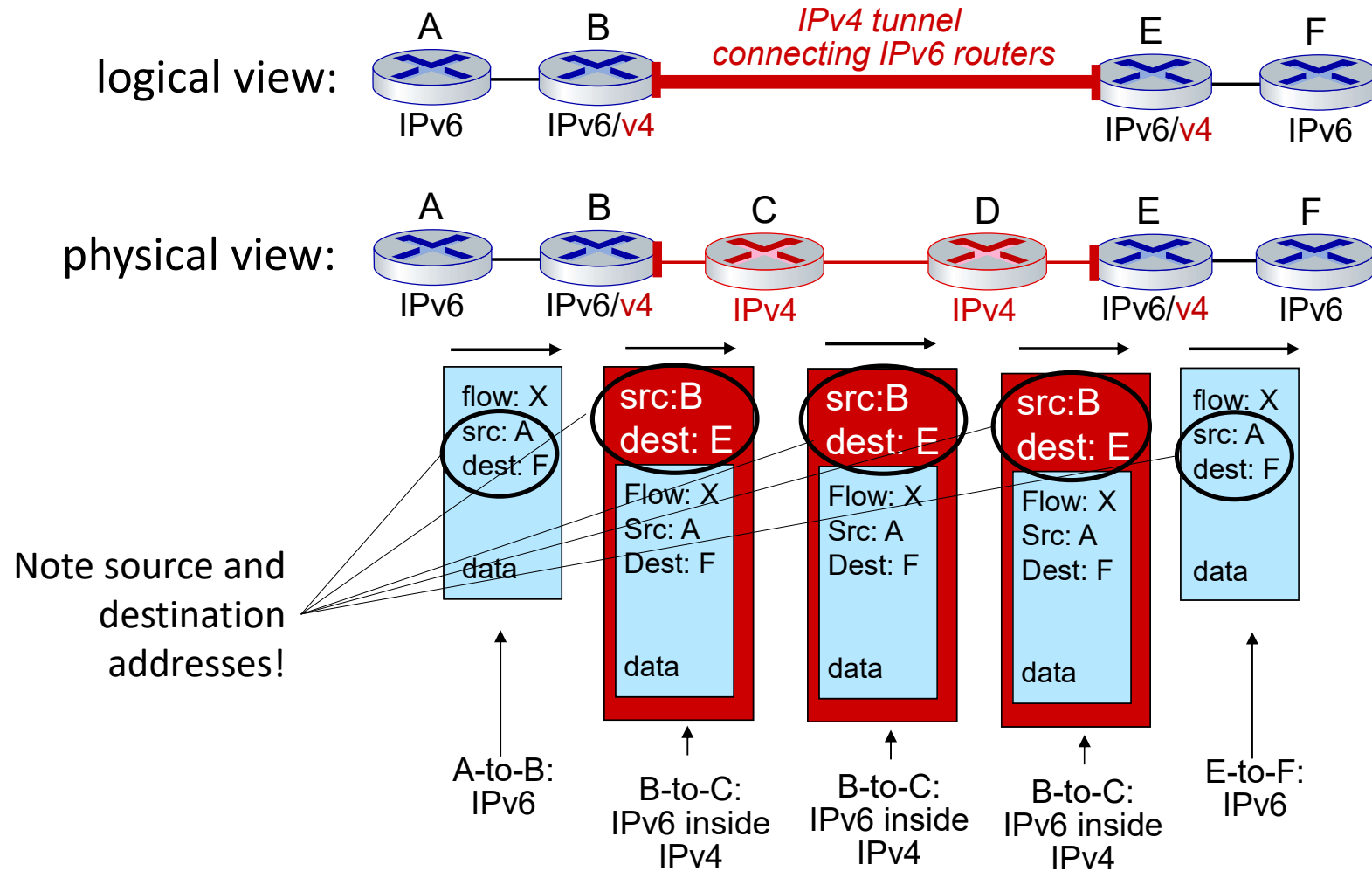
Ethernet connecting two IPv6 routers:



IPv4 tunnel connecting two IPv6 routers



# Tunneling

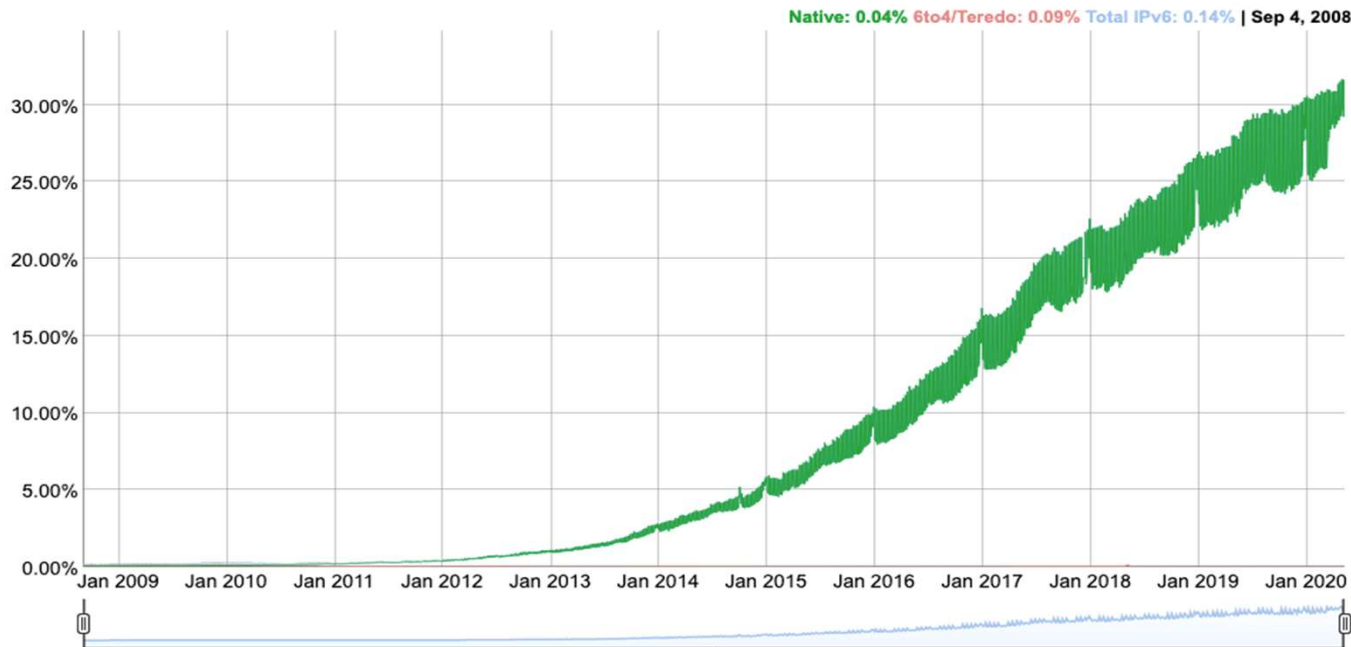


# IPv6: adoption

- Google<sup>1</sup>: ~ 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

## IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.



1

<https://www.google.com/intl/en/ipv6/statistics.html>

# IPv6: adoption

- Google<sup>1</sup>: ~ 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable
- Long (long!) time for deployment, use
  - 25 years and counting!
  - think of application-level changes in last 25 years: WWW, social media, streaming media, gaming, telepresence, ...
  - *Why?*

<sup>1</sup> <https://www.google.com/intl/en/ipv6/statistics.html>

# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

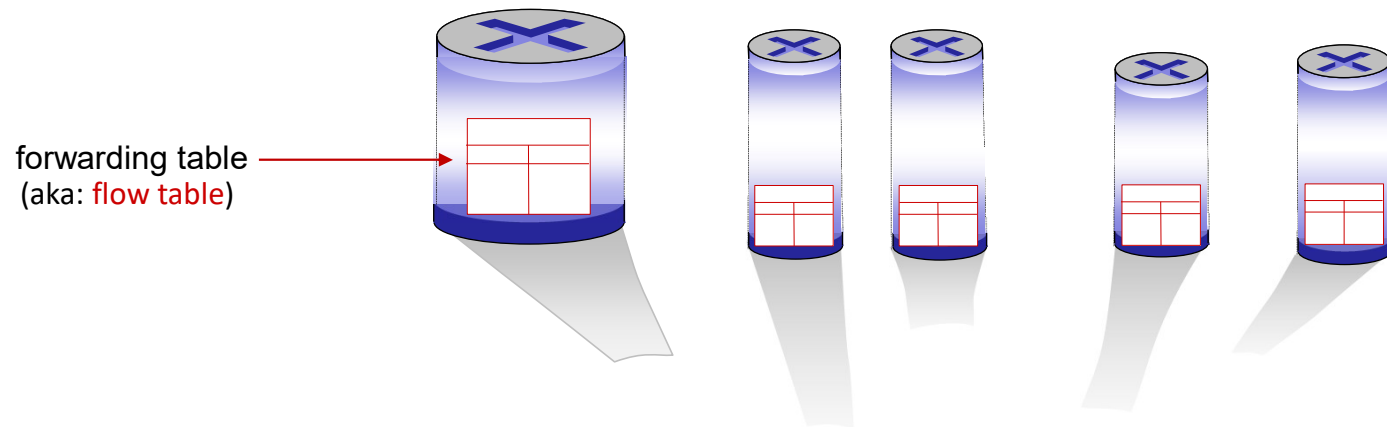


- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes

# Generalized forwarding: match plus action

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

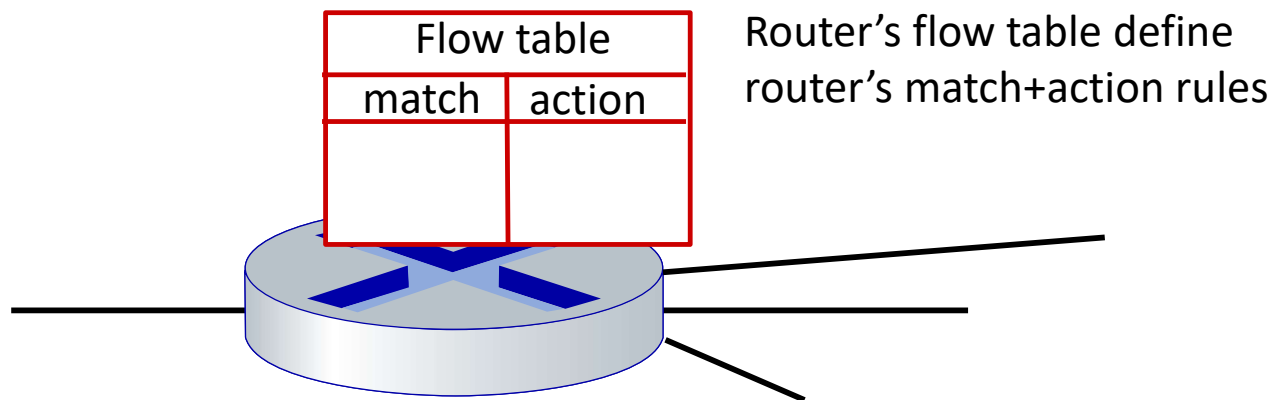
- “**match plus action**” abstraction: match bits in arriving packet, take action
  - *destination-based forwarding*: forward based on dest. IP address
  - *generalized forwarding*:
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet





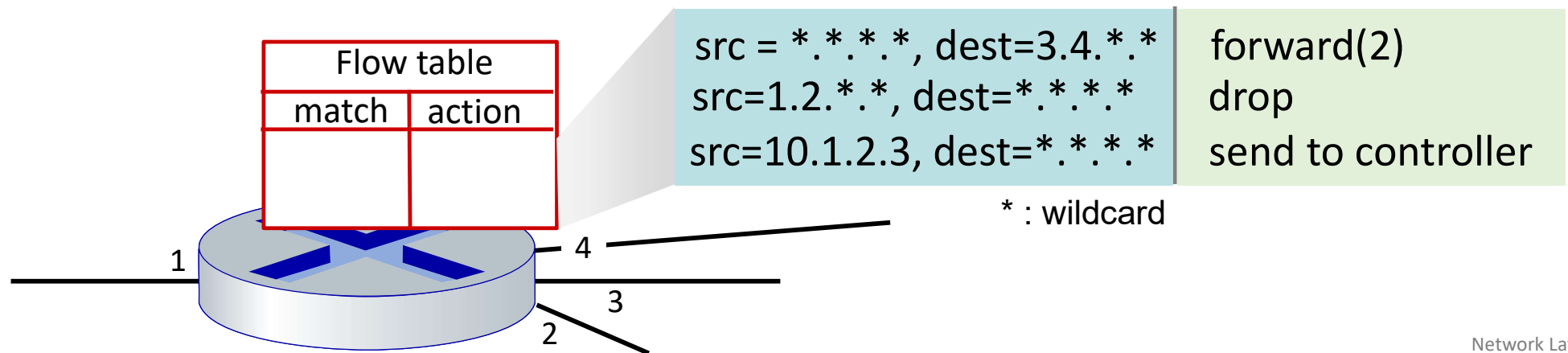
# Flow table abstraction

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

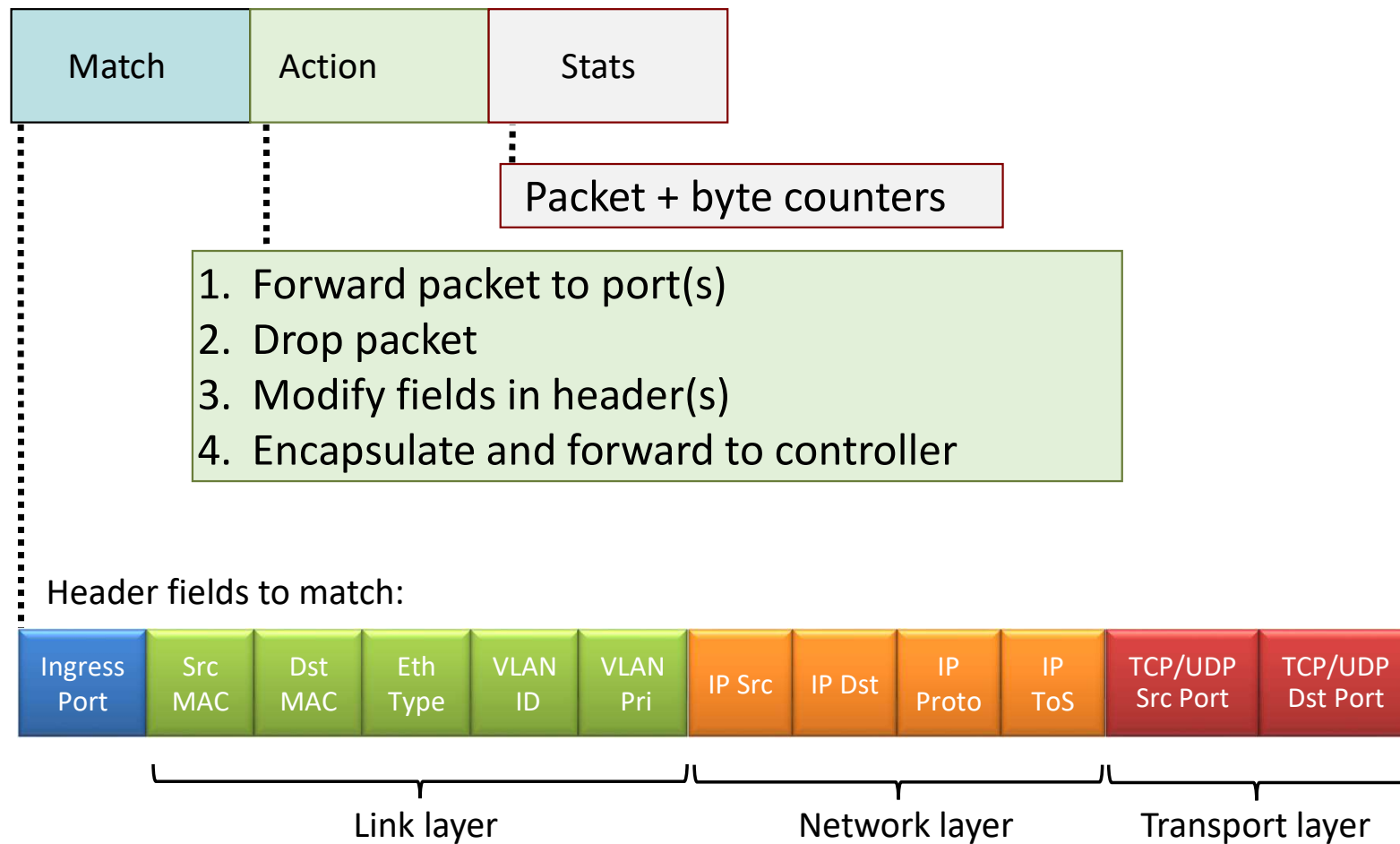


# Flow table abstraction

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



# OpenFlow: flow table entries



# OpenFlow: examples

## Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

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

## Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

# OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

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

# OpenFlow abstraction

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

## Router

- *match*: longest destination IP prefix
- *action*: forward out a link

## Switch

- *match*: destination MAC address
- *action*: forward or flood

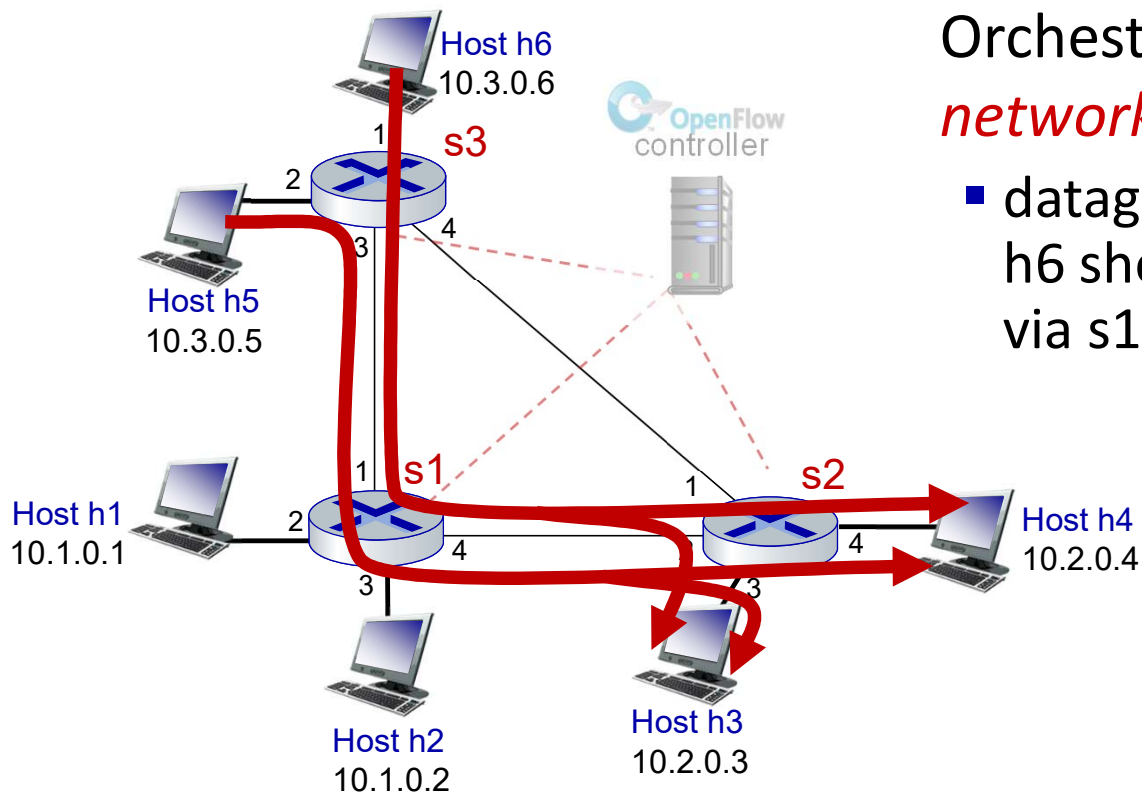
## Firewall

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

## NAT

- *match*: IP address and port
- *action*: rewrite address and port

# OpenFlow example

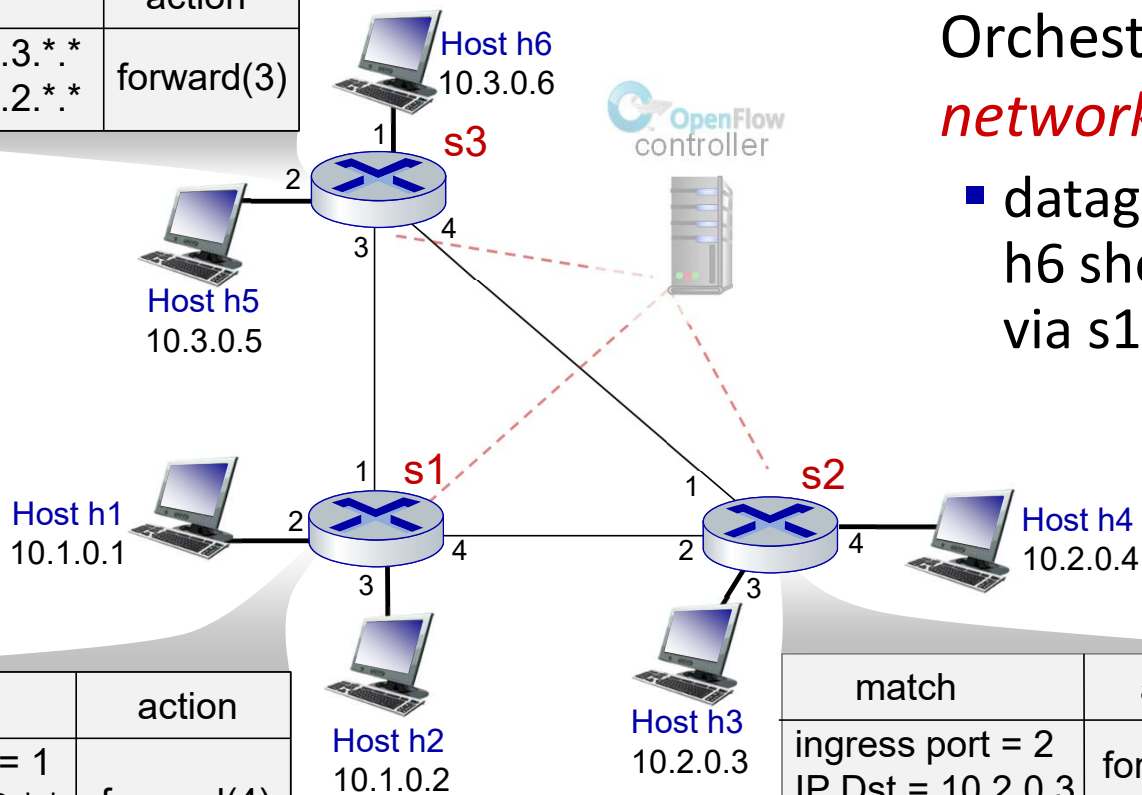


Orchestrated tables can create *network-wide* behavior, e.g.,:

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

# OpenFlow example

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

Orchestrated tables can create *network-wide* behavior, e.g.,:

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



# Generalized forwarding: summary

- “match plus action” abstraction: match bits in arriving packet header(s) in any layers, take action
  - matching over many fields (link-, network-, transport-layer)
  - local actions: drop, forward, modify, or send matched packet to controller
  - “program” *network-wide* behaviors
- simple form of “network programmability”
  - programmable, per-packet “processing”
  - *historical roots*: active networking
  - *today*: more generalized programming: P4 (see [p4.org](http://p4.org)).

# Network layer: “data plane” roadmap

- Network layer: overview
- What’s inside a router
- IP: the Internet Protocol
- Generalized Forwarding
- **Middleboxes**
  - middlebox functions
  - evolution, architectural principles of the Internet



# Middleboxes

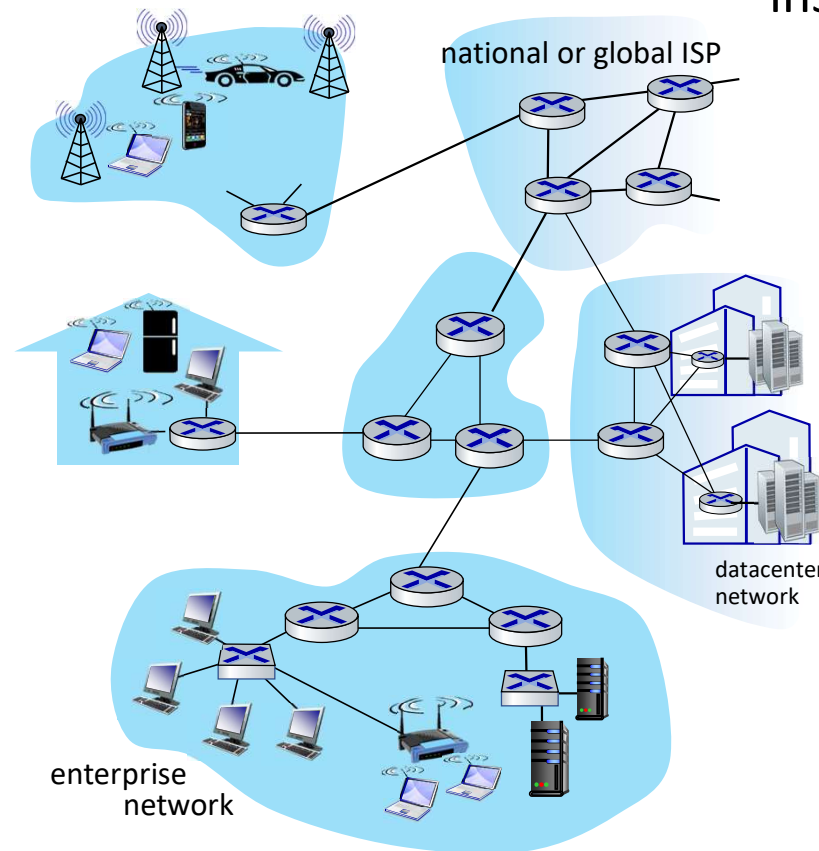
Middlebox (RFC 3234)

“any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host”

# Middleboxes everywhere!

**NAT:** home,  
cellular,  
institutional

**Application-specific:** service  
providers,  
institutional,  
CDN



**Firewalls, IDS:** corporate,  
institutional, service providers,  
ISPs

**Load balancers:**  
corporate, service  
provider, data center,  
mobile nets

**Caches:** service  
provider, mobile, CDNs

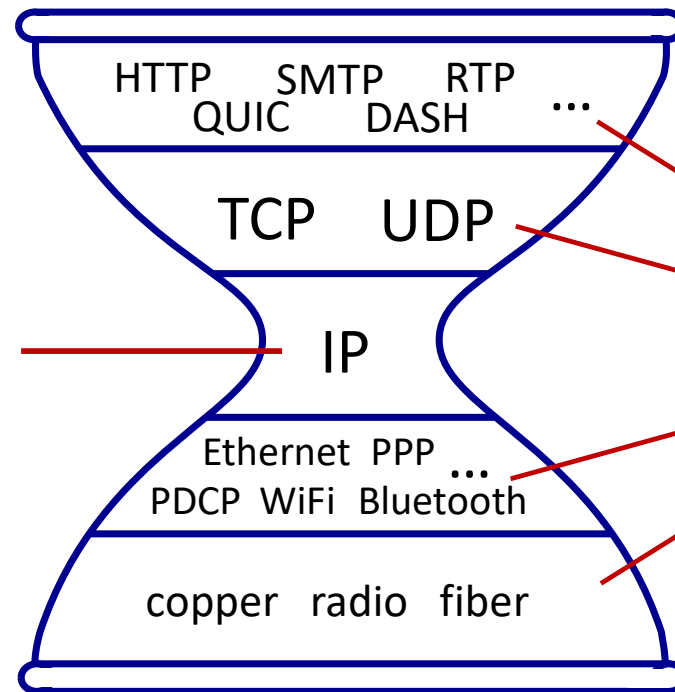
# Middleboxes

- initially: proprietary (closed) hardware solutions
- move towards “whitebox” hardware implementing open API
  - move away from proprietary hardware solutions
  - programmable local actions via match+action
  - move towards innovation/differentiation in software
- SDN: (logically) centralized control and configuration management often in private/public cloud
- network functions virtualization (NFV): programmable services over white box networking, computation, storage

# The IP hourglass

## Internet's "thin waist":

- *one* network layer protocol: IP
- *must* be implemented by every (billions) of Internet-connected devices

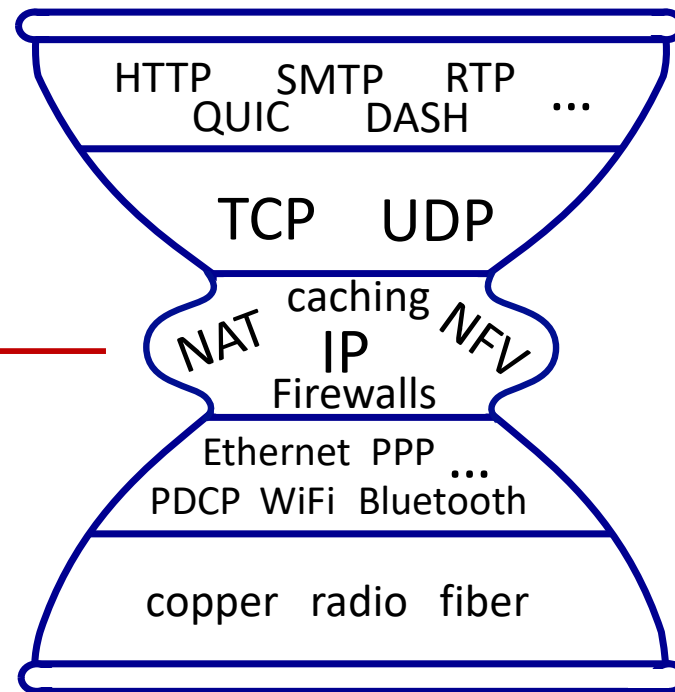


*many* protocols  
in physical, link,  
transport, and  
application  
layers

# The IP hourglass, at middle age

Internet's middle age  
"love handles"?

- middleboxes, — operating inside the network



# Architectural Principles of the Internet

RFC 1958

“Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that **the goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network.**”

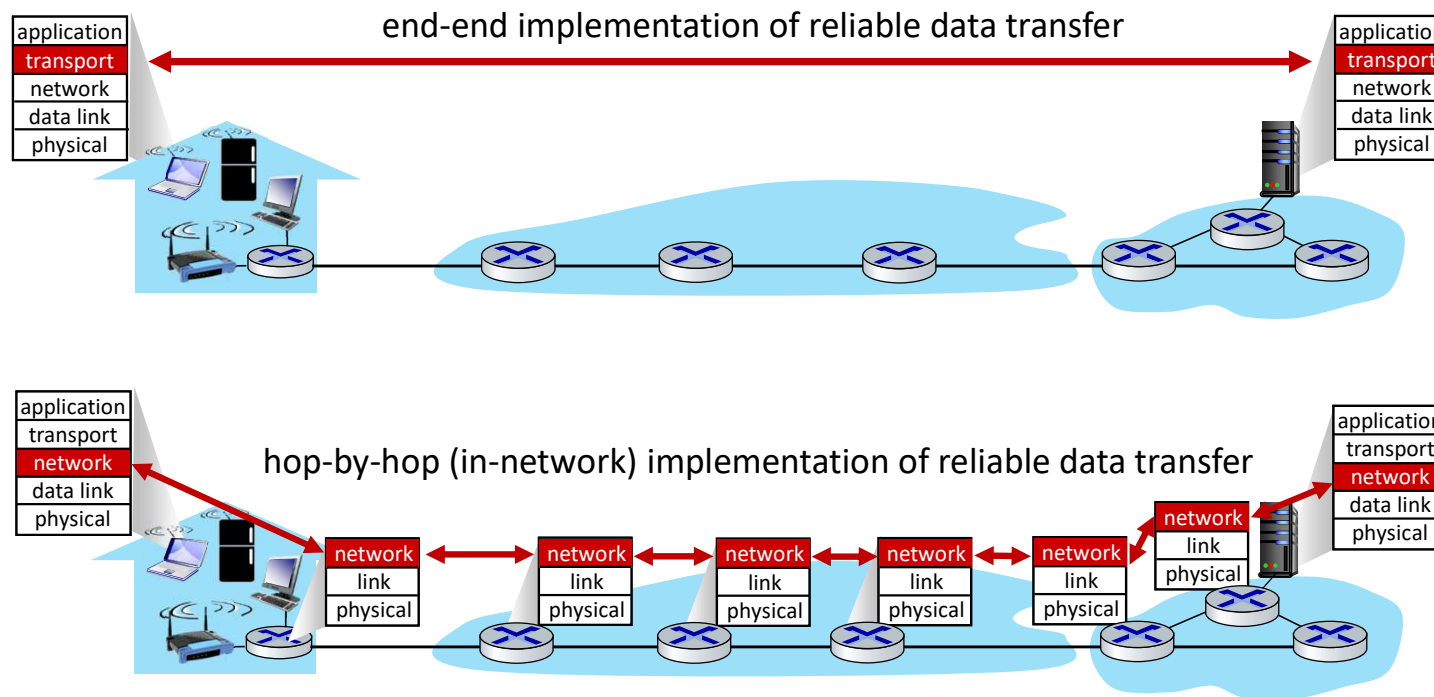
Three cornerstone beliefs:

- simple connectivity
- IP protocol: that narrow waist
- intelligence, complexity at network edge



# The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented **in network**, or at **network edge**



# The end-end argument

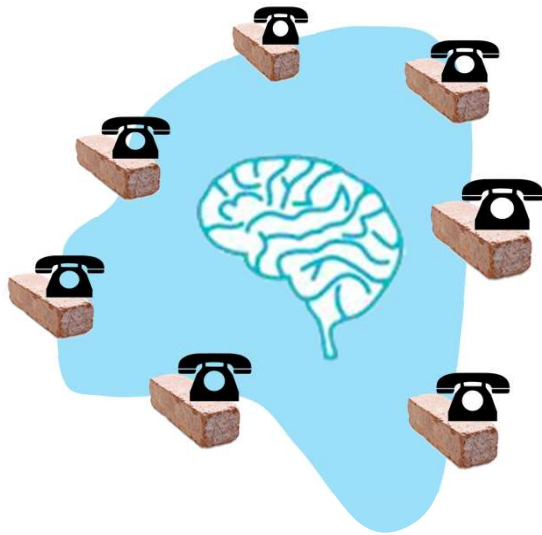
- some network functionality (e.g., reliable data transfer, congestion) can be implemented **in network**, or at **network edge**

“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

We call this line of reasoning against low-level function implementation the “end-to-end argument.”

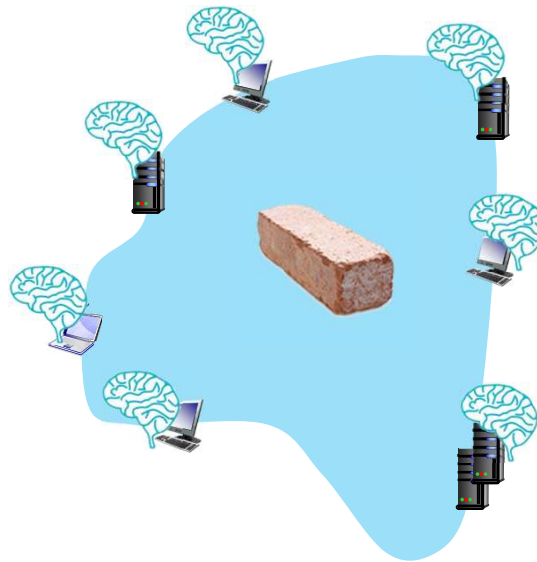
— Saltzer, Reed, Clark 1981 —

# Where's the intelligence?



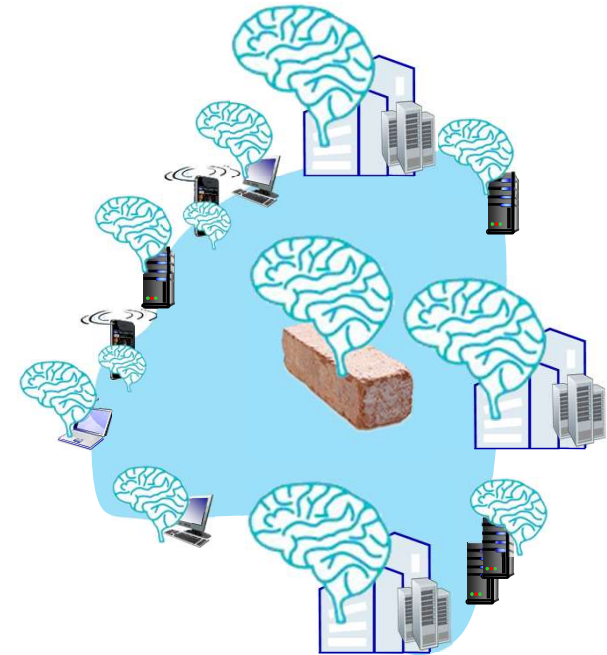
## 20<sup>th</sup> century phone net:

- intelligence/computing at network switches



## Internet (pre-2005)

- intelligence, computing at edge



## Internet (post-2005)

- programmable network devices
- intelligence, computing, massive application-level infrastructure at edge

# Chapter 4: done!

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding, SDN
- Middleboxes



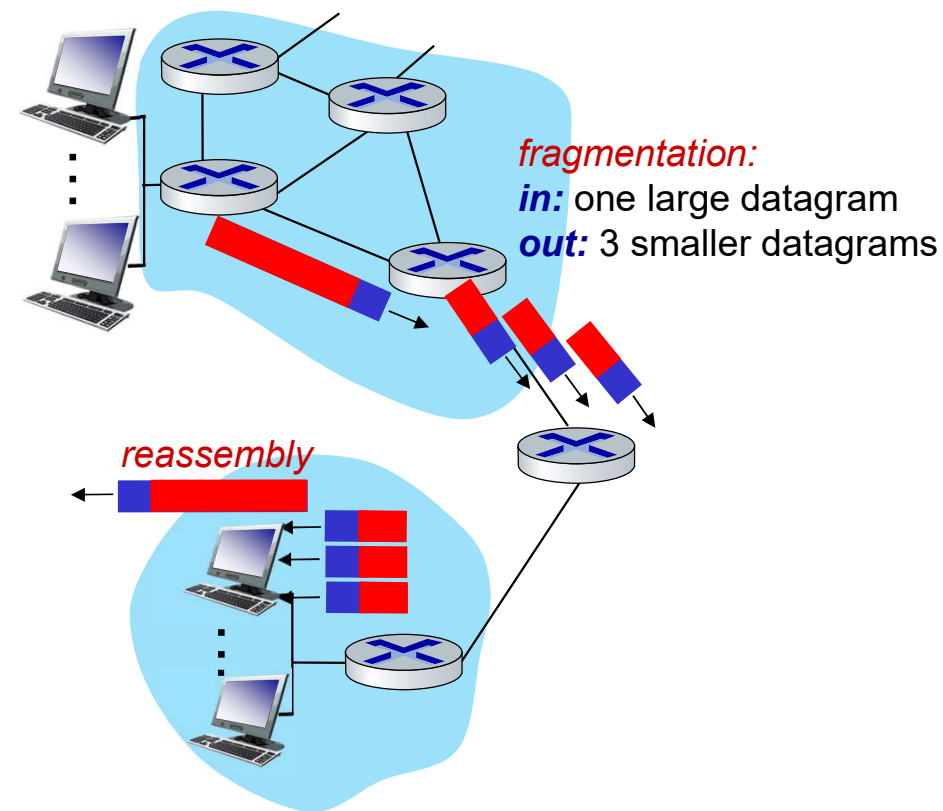
*Question:* how are forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

*Answer:* by the control plane (next chapter)

# Additional Chapter 4 slides

# IP fragmentation/reassembly

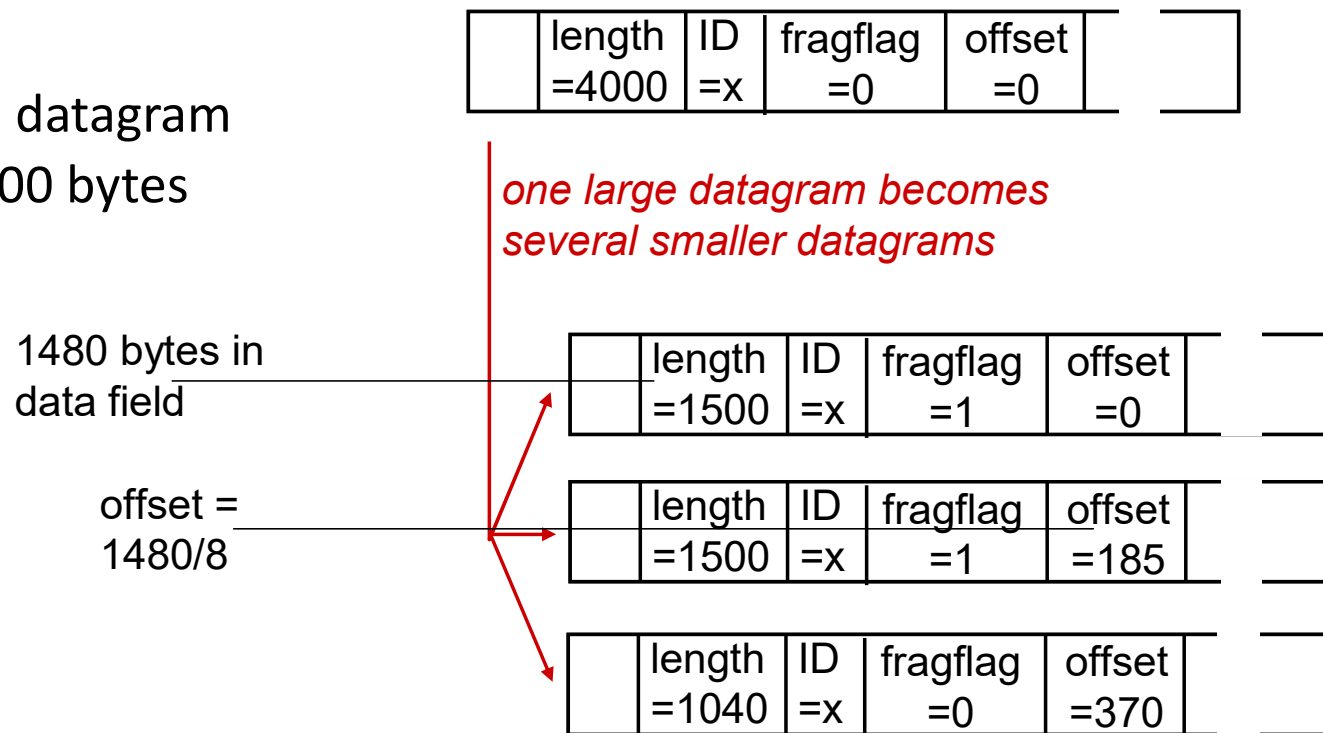
- network links have MTU (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided (“fragmented”) within net
  - one datagram becomes several datagrams
  - “reassembled” only at *destination*
  - IP header bits used to identify, order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



# IP fragmentation/reassembly

## example:

- 4000 byte datagram
- MTU = 1500 bytes



# DHCP: Wireshark output (home LAN)

Message type: **Boot Request (1)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

request

**Transaction ID: 0x6b3a11b7**

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

Client IP address: 0.0.0.0 (0.0.0.0)

Your (client) IP address: 0.0.0.0 (0.0.0.0)

Next server IP address: 0.0.0.0 (0.0.0.0)

Relay agent IP address: 0.0.0.0 (0.0.0.0)

**Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)**

Server host name not given

Boot file name not given

Magic cookie: (OK)

Option: (t=53,l=1) **DHCP Message Type = DHCP Request**

Option: (61) Client identifier

Length: 7; Value: 010016D323688A;

Hardware type: Ethernet

Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)

Option: (t=50,l=4) Requested IP Address = 192.168.1.101

Option: (t=12,l=5) Host Name = "nomad"

**Option: (55) Parameter Request List**

Length: 11; Value: 010F03062C2E2F1F21F92B

**1 = Subnet Mask; 15 = Domain Name**

**3 = Router; 6 = Domain Name Server**

44 = NetBIOS over TCP/IP Name Server

.....

Message type: **Boot Reply (2)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

reply

**Transaction ID: 0x6b3a11b7**

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

**Client IP address: 192.168.1.101 (192.168.1.101)**

Your (client) IP address: 0.0.0.0 (0.0.0.0)

**Next server IP address: 192.168.1.1 (192.168.1.1)**

Relay agent IP address: 0.0.0.0 (0.0.0.0)

Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)

Server host name not given

Boot file name not given

Magic cookie: (OK)

**Option: (t=53,l=1) DHCP Message Type = DHCP ACK**

**Option: (t=54,l=4) Server Identifier = 192.168.1.1**

**Option: (t=1,l=4) Subnet Mask = 255.255.255.0**

**Option: (t=3,l=4) Router = 192.168.1.1**

**Option: (6) Domain Name Server**

Length: 12; Value: 445747E2445749F244574092;

IP Address: 68.87.71.226;

IP Address: 68.87.73.242;

IP Address: 68.87.64.146

**Option: (t=15,l=20) Domain Name = "hsd1.ma.comcast.net."**