National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 18

Chapter 4

24^{th} October, 2023

Nauman Moazzam Hayat

nauman.moazzam@lhr.nu.edu.pk

Office Hours: 02:30 pm till 06:00 pm (Every Tuesday & Thursday)

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

match!

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

match!

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

Longest prefix matching

longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range				Link interface
11001000	00010111	00010***	********	0
11001000	00010111	00011000	********	1
11001000	00010111	00011***	********	2
otherwise				3

match!

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

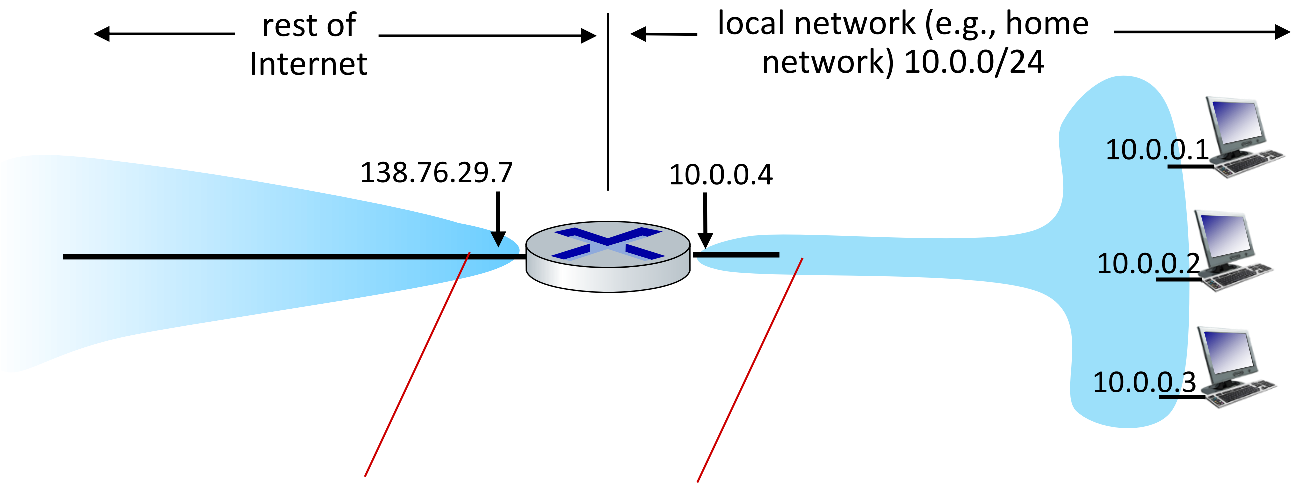
Network layer: "data plane" roadmap

- 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 (PAT)

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

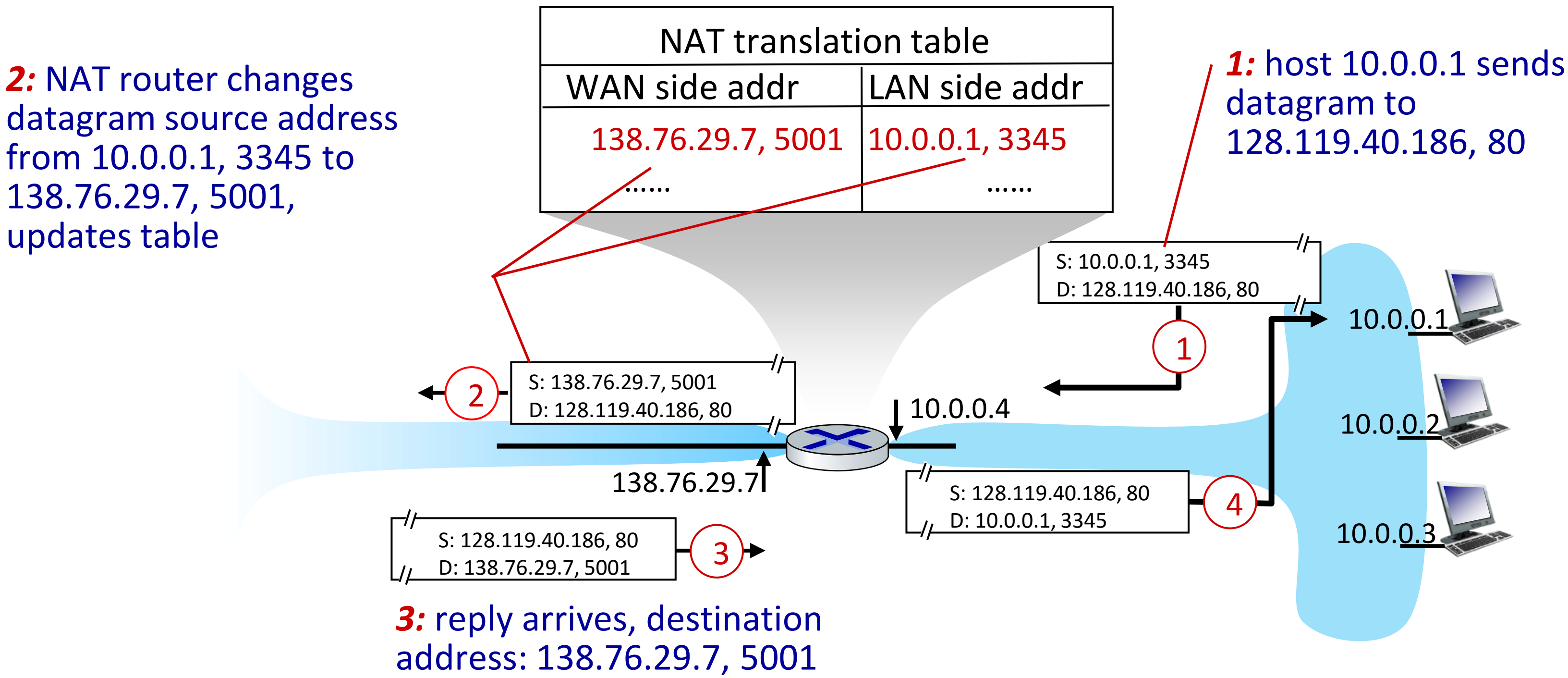
- all devices in local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just one IP address needed from provider ISP for *all* devices
  - can change addresses of host in local network without 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
- Implemented in the border (access) router separating the private & the public network
- Was introduced with Windows 2000

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



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
  - ***Study*** NAT Traversal Problem & Solutions (including static configuration, UPnP / IGD & relaying)

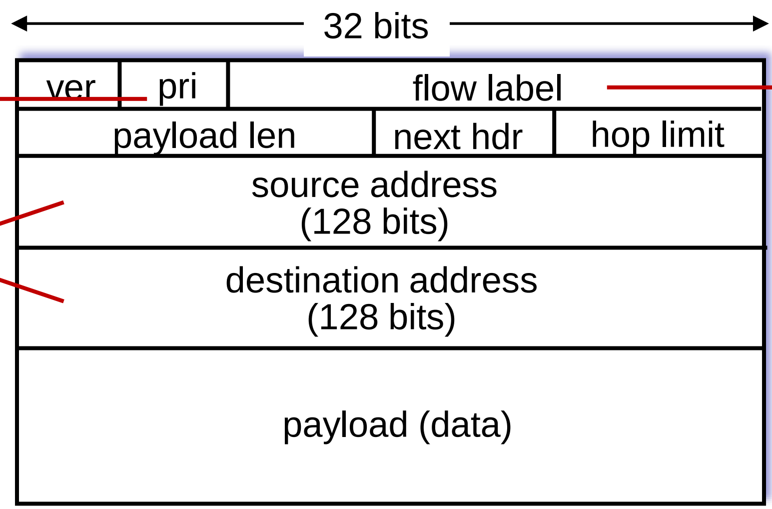
IPv6: motivation

- 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

traffic class / priority: identify priority among datagrams in flow

128-bit IPv6 addresses

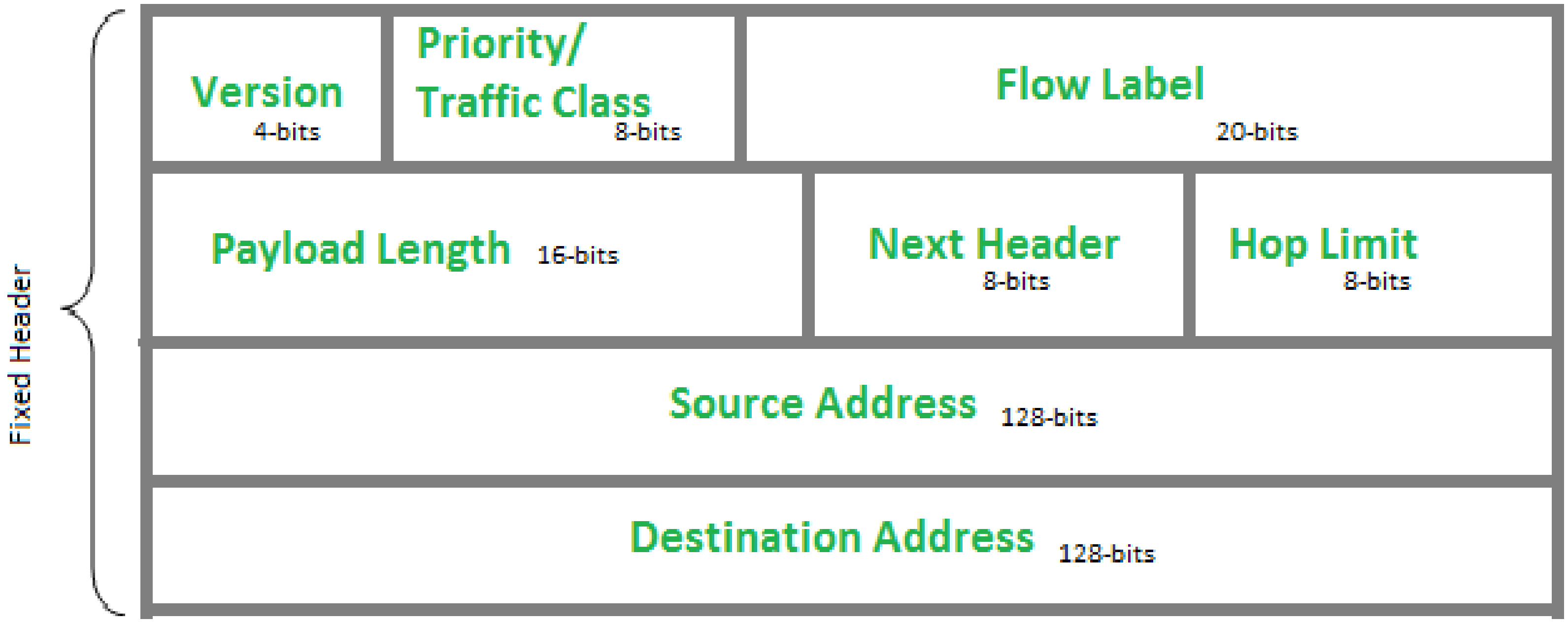


flow label: identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4): (Changes from IPv4)

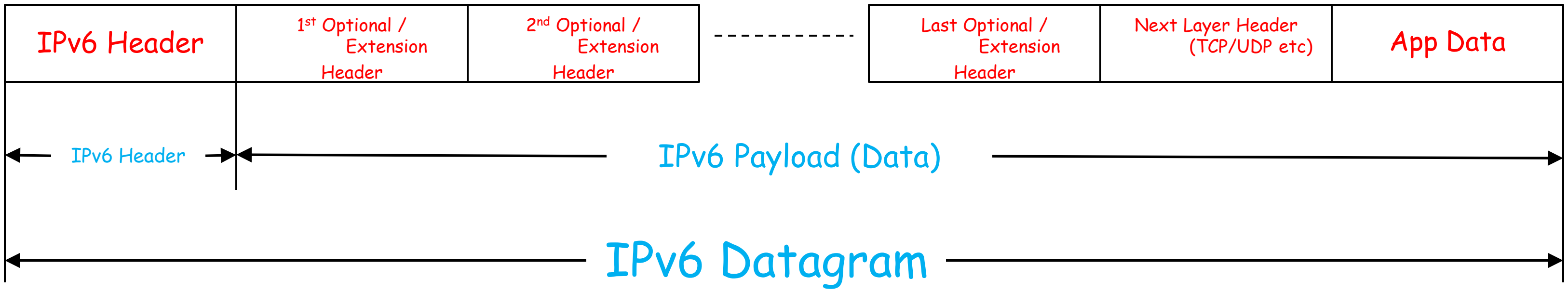
- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (not part of the standard IP header but can be outside of header, indicated by "Next Header" field.)

IPv6 Header



Fixed Header
Version 4-bits
Priority/ Traffic Class 8-bits
Flow Label 20-bits
Payload Length 16-bits
Next Header 8-bits
Hop Limit 8-bits
Source Address 128-bits
Destination Address 128-bits

IPv6 Next Header Field



- An IPv6 packet header always present and of fixed size (i.e. 40 bytes)

- Zero or more optional / extension header(s) can be present (all can be of varying lengths)

- The Next Header Field is present in all the headers, including the IPv6 fixed header and any optional / extension header(s)

- The Next Header Field in the last optional / extension header (or in the IPv6 Fixed header in case there is no optional / extension header) indicates the upper layer protocol (such as TCP, UDP, or ICMPv6 etc.)

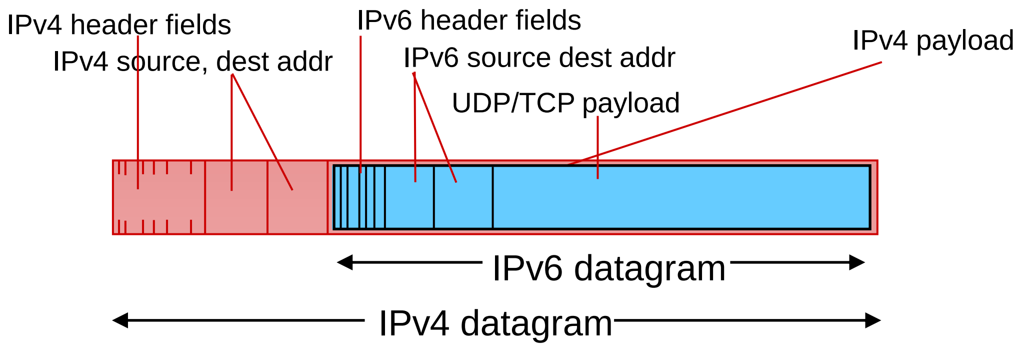
- Unlike options in the IPv4 header, IPv6 optional / extension headers have no maximum size and can expand to accommodate all the extension data needed for IPv6 communication.

- While 256 next header values are possible, some typical Next Header values are given below:

Value in Decimal	Header
6	TCP
17	UDP
58	ICMPv6
59	No Next Header

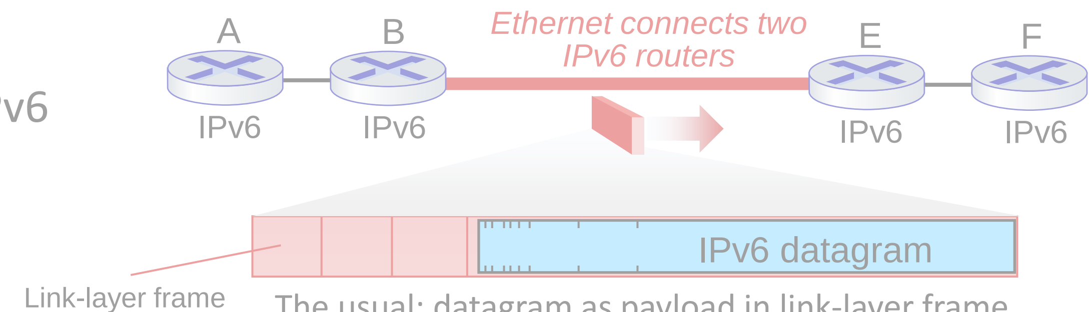
Transition from IPv4 to IPv6

- 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)
  - Also study dual stack approach (& it's issue) for transitioning from IPv4 to IPv6



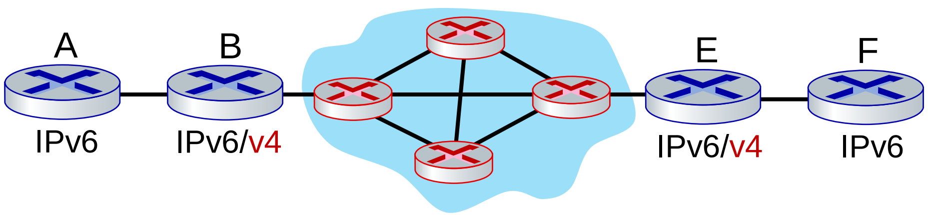
Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



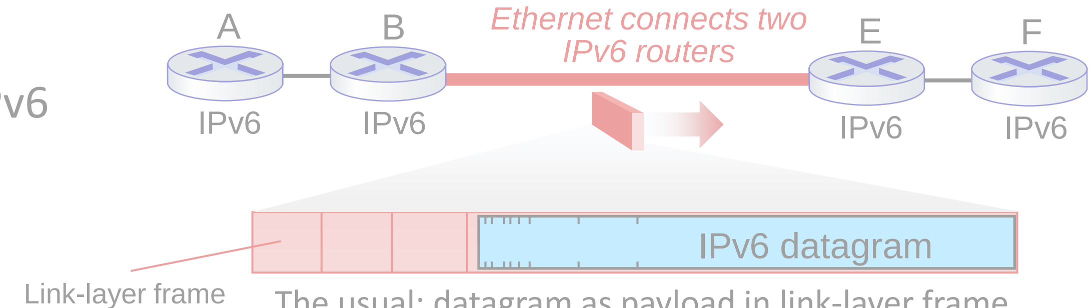
The usual: datagram as payload in link-layer frame

IPv4 network connecting two IPv6 routers



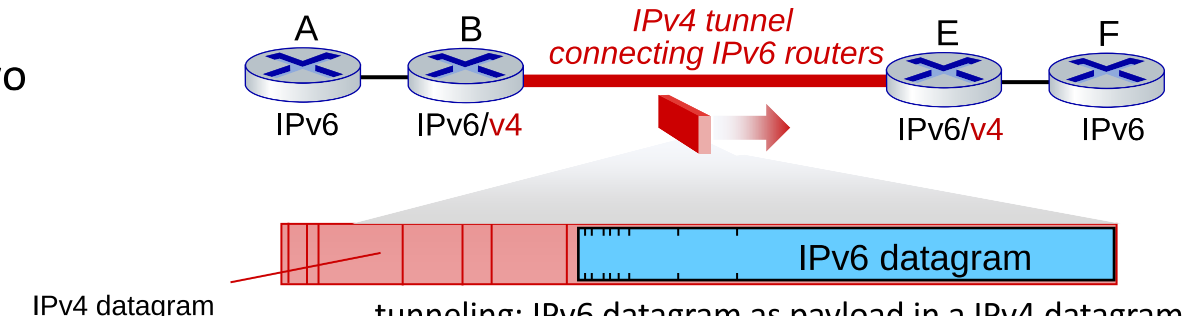
Tunneling and encapsulation

Ethernet connecting two IPv6 routers:



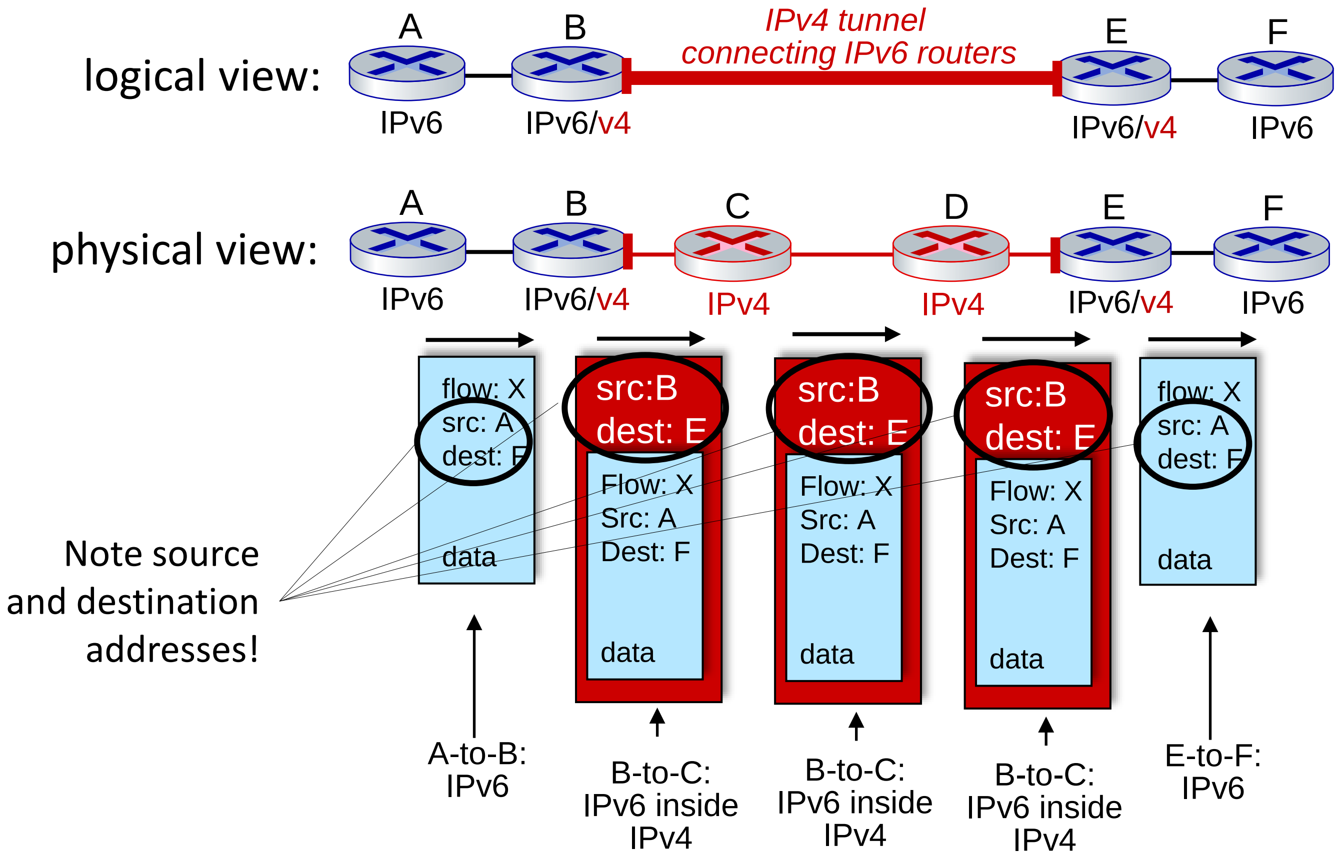
The usual: datagram as payload in link-layer frame

IPv4 tunnel connecting two IPv6 routers



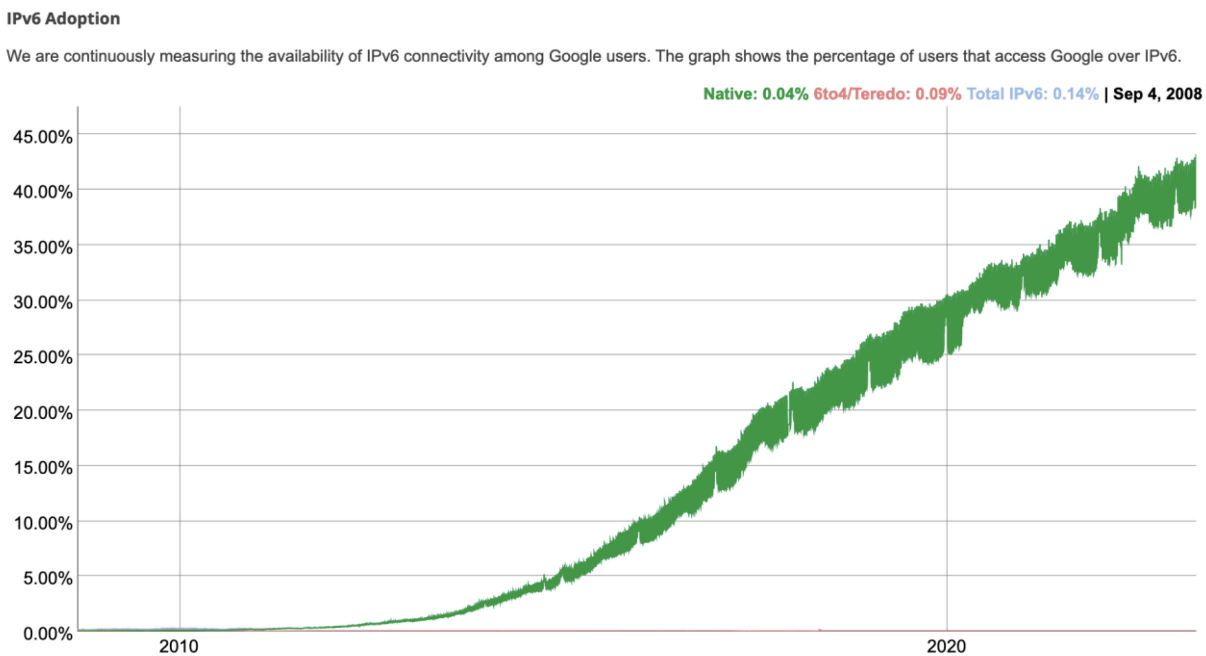
tunneling: IPv6 datagram as payload in a IPv4 datagram

Tunneling


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6/v4
IPv6/v4
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6/v4
IPv4
IPv4
IPv6/v4
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
src:B
dest: E
Flow: X
Src: A
Dest: F
data
flow: X
src: A
dest: F
data
Note source
and destination
addresses!
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6

IPv6: adoption

- Google[1]: ~ 40% of clients access services via IPv6 (2023)
- NIST: 1/3 of all US government domains are IPv6 capable



IPv6: adoption

- Google[1]: ~ 40% of clients access services via IPv6 (2023)
- 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? (Expensive, Solutions like NAT take some of the pressure off.)*
  - *(Think of network layer changes akin to changing the foundation of a house while application layer changes are rapid, akin to applying a new layer of paint to a house)*

[1] https://www.google.com/intl/en/ipv6/statistics.html

Network layer: “data plane” roadmap

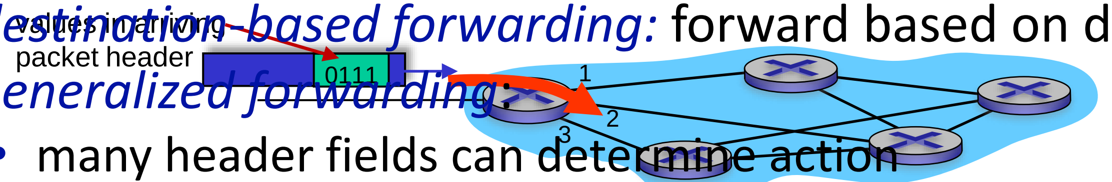
- Network layer: overview
  - data plane
  - control plane
- What’s inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

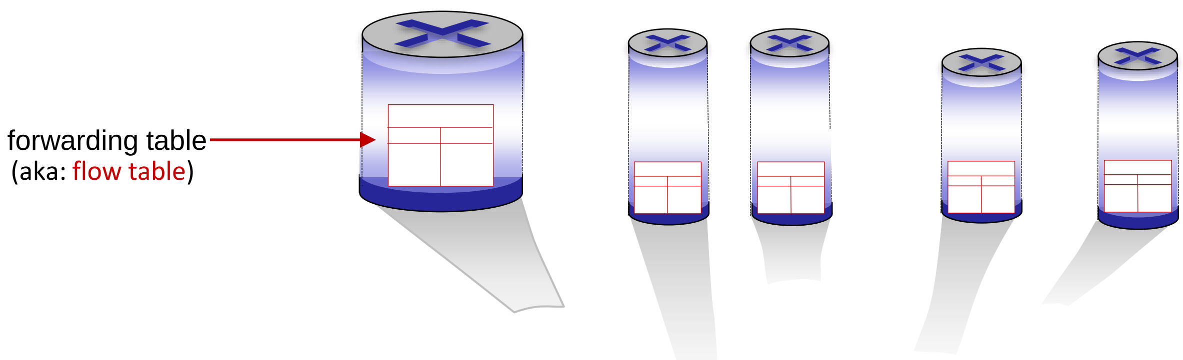
- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes

Generalized forwarding: match plus action

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

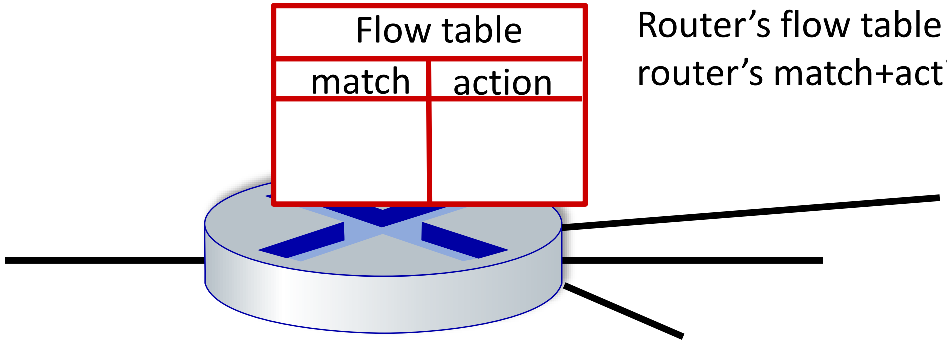
- "match plus action" abstraction: match bits in arriving packet, take action
  - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding:*
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet





Flow table abstraction

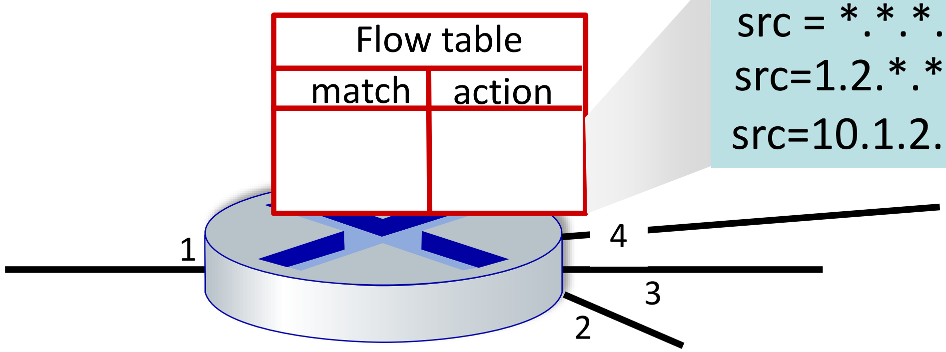
- flow: defined by header field values (in link-, network-, transport-layer fields) (violation of layering principle)
- Open flow device can easily perform as a router (layer 3 device) or a switch (layer 2 device), thus a new term "*packet switch*" is being used (a terminology that is gaining widespread adoption in SDN literature)
- 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



Router's flow table define router's match+action rules

Flow table abstraction

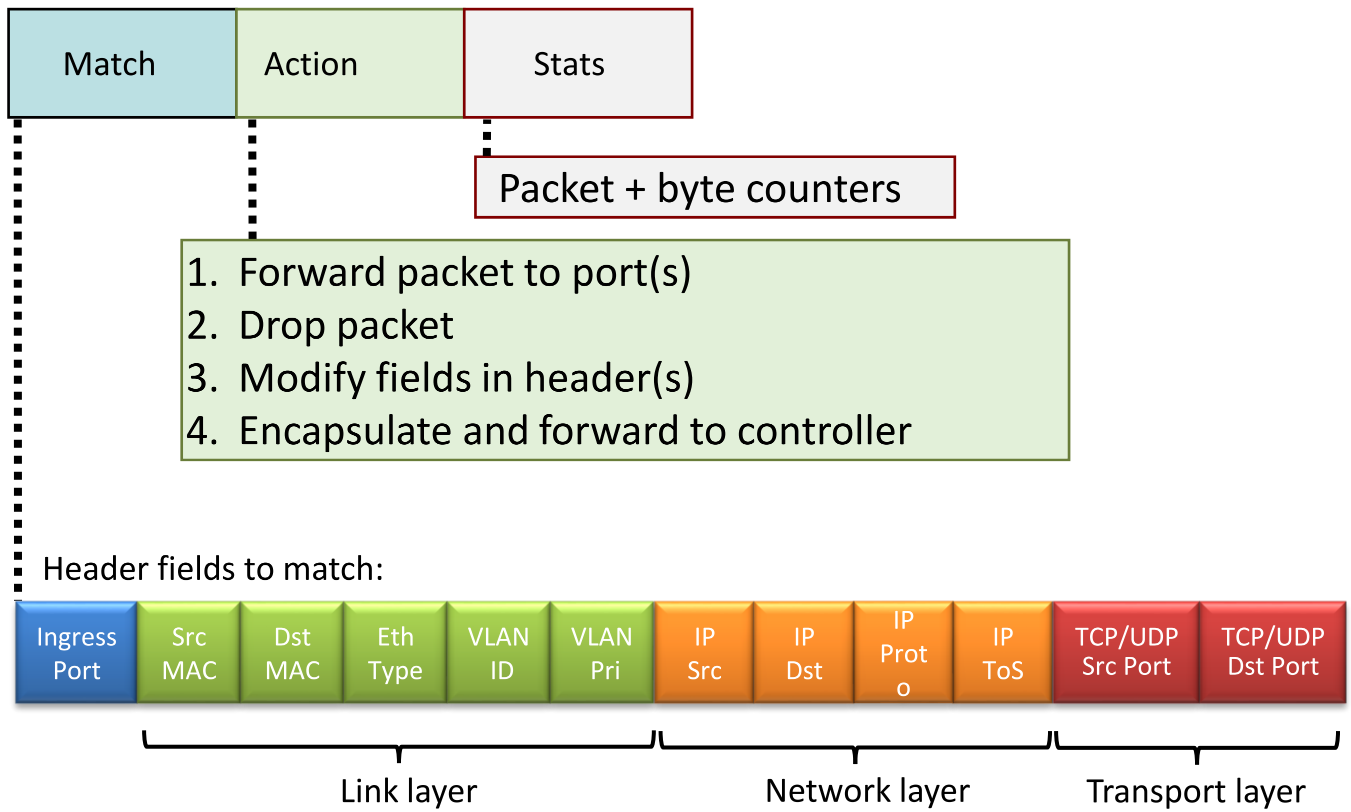
- flow: defined by header fields
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - priority: disambiguate overlapping patterns
  - counters: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



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

* : wildcard

OpenFlow: flow table entries (11 header entries in Open Flow 1.0, more in newer specifications of Open Flow)



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

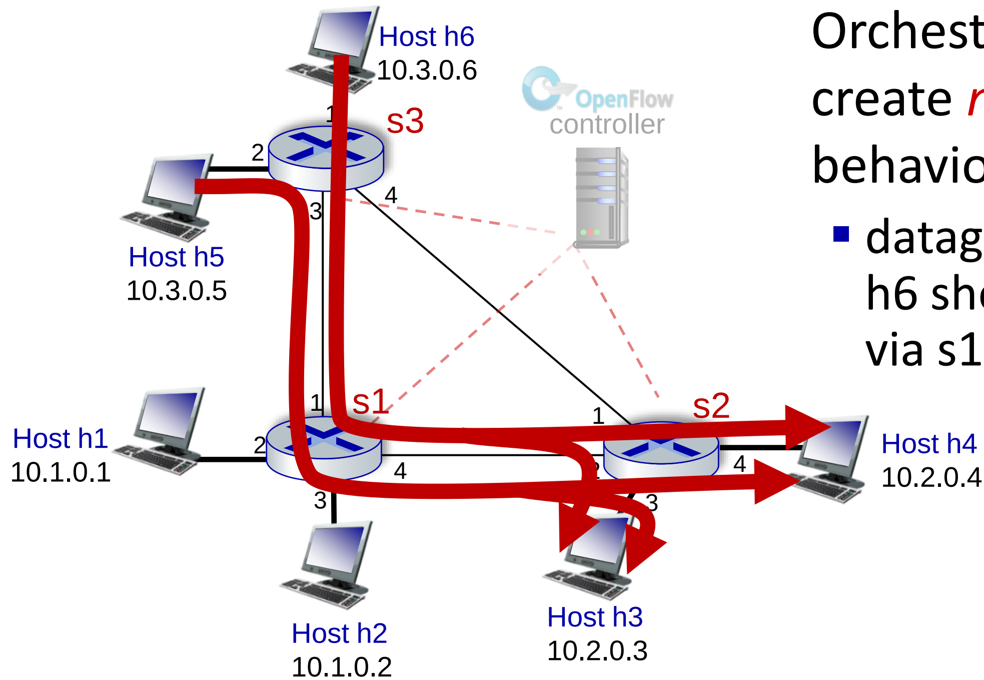
Router
- *match:* longest destination IP prefix
- *action:* forward out a link

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

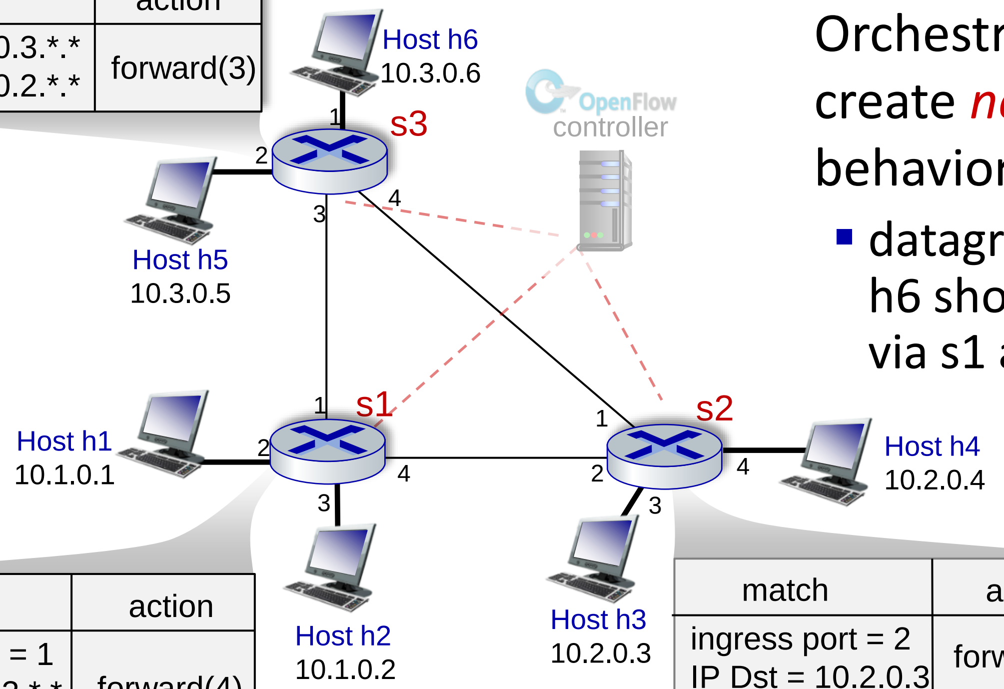


Orchestrated tables can create *network-wide* behavior, e.g.,:

- 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).

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

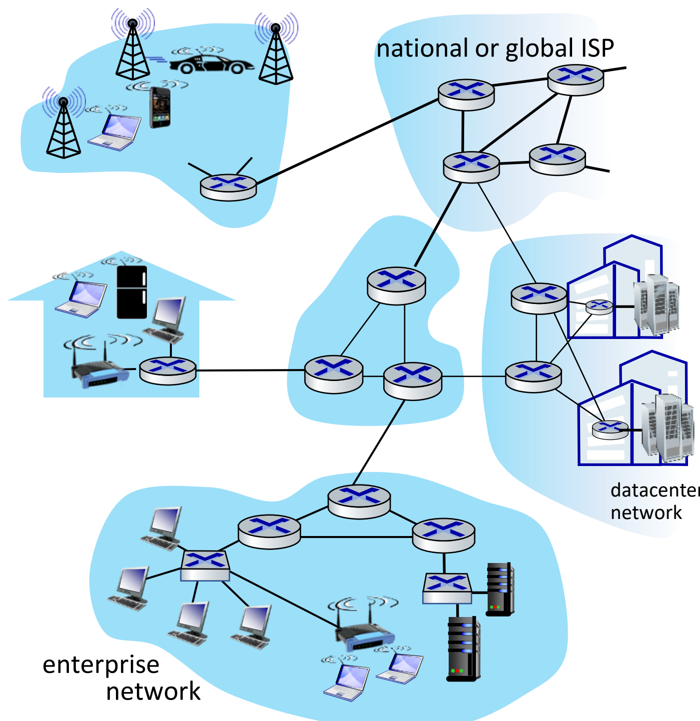
Middlebox (RFC 3234)

"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 (Intrusion Detection Systems): corporate, institutional, service providers, ISPs

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

Caches: service provider, mobile, CDNs

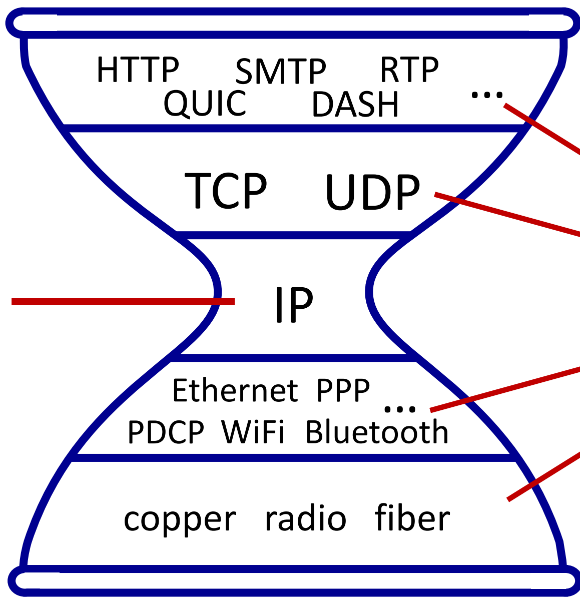
Middleboxes

- 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 Researchers are exploring the use of commodity hardware (networking, computing, and storage) with specialized software built on top of a common software stack to implement these services (Network translation, Security Services & Performance Enhancement.) This approach has become known as **network function virtualization (NFV)**. An alternate approach that has also been explored is to outsource middlebox functionality to the cloud.

The IP hourglass (initially)

Internet's "thin waist":

- *one* network layer protocol: IP (also called spanning layer)
- *must* be implemented by every (billions) of Internet-connected devices
- This narrow waist has played a critical role in the phenomenal growth of the Internet

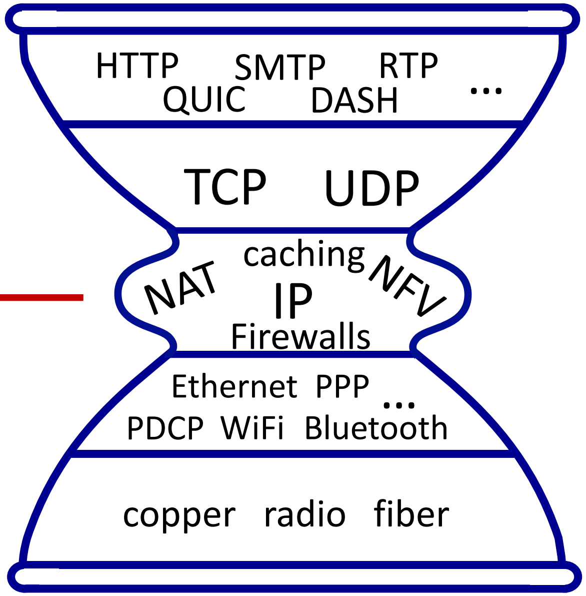


many protocols in physical, link, transport, and application layers

The IP hourglass, at middle age

Internet's middle age "love handles"?

- middleboxes, operating inside the network



Architectural Principles of the Internet (RFC 1958)

RFC 1958

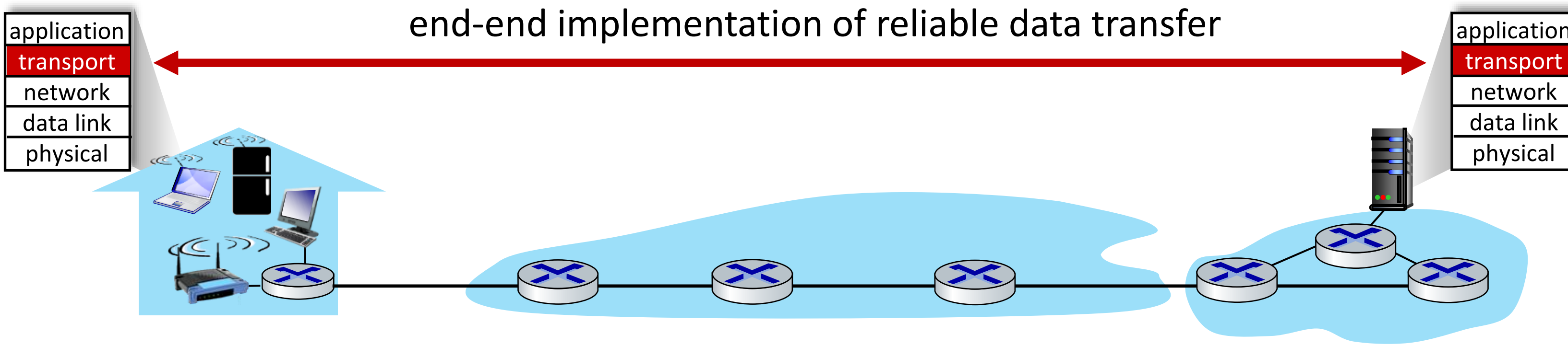
"Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that the goal is connectivity, the tool is the Internet Protocol,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

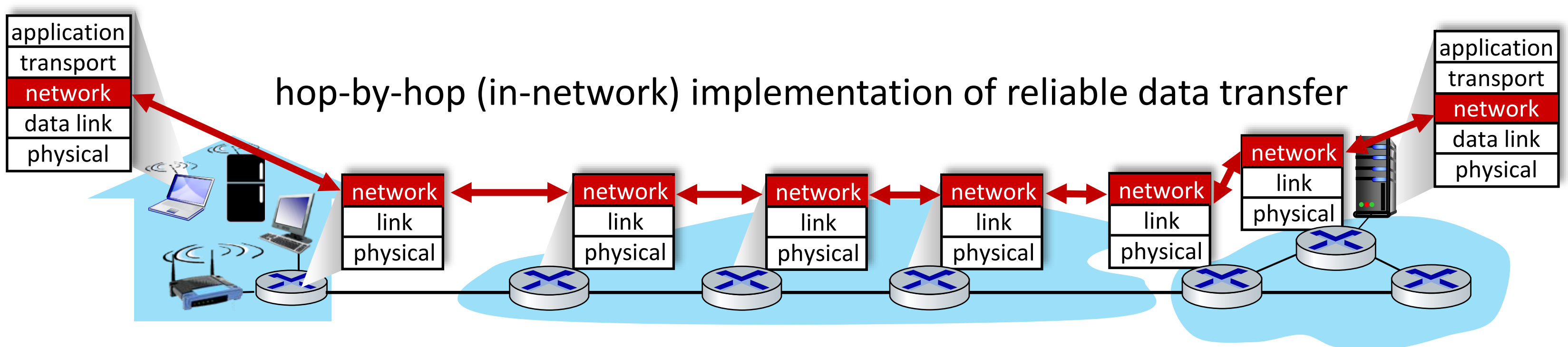
Three cornerstone beliefs: (i.e. three principles of RFC 1958)

- simple connectivity (Goal)
- IP protocol: that narrow waist (Just one protocol)
- intelligence, complexity at network edge (Rather than network core)

The end-end argument

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



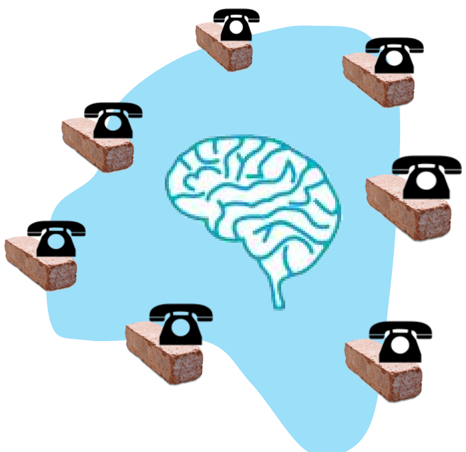


The end-end argument (Third principle)

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

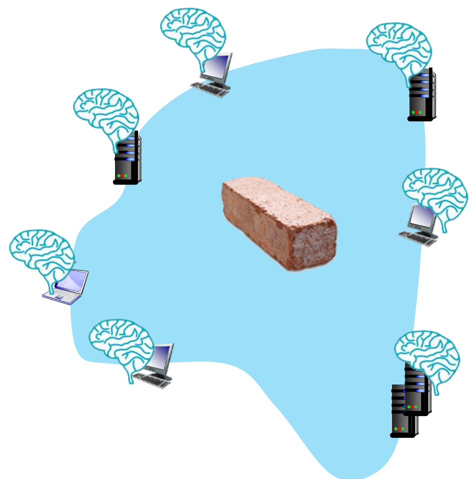
> "The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes 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?



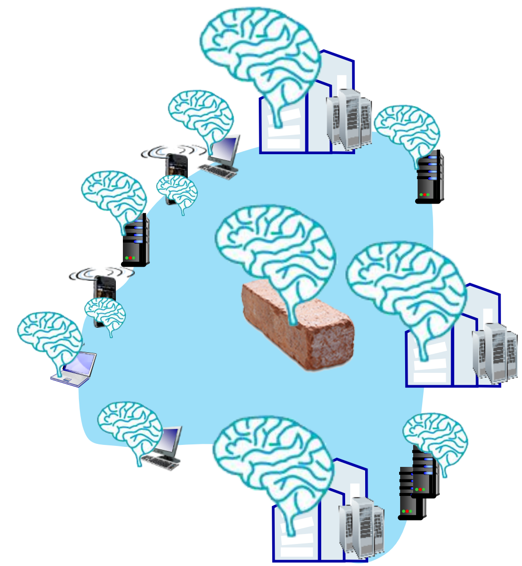
20th century phone net:

- 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)

Assignement # 4 (Chapter - 4)

- *4th Assignment will be uploaded on Google Classroom on Thursday, 26th October, 2023, in the Stream - Announcement Section*

- *Due Date: Thursday, 2nd November, 2023 (Handwritten solutions to be submitted during the lecture)*

- *Please read all the instructions carefully in the uploaded Assignment document, follow & submit accordingly*

Quiz # 4 (Chapter - 4)

- *On: Thursday, 2nd November, 2023 (During the lecture)*

- *Quiz to be taken during <u>own section class only</u>*