523454

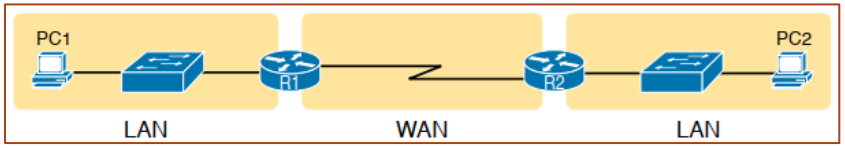
Computer Network Programming

Lecture 1: Network Layer (IPv4 and IPv6)

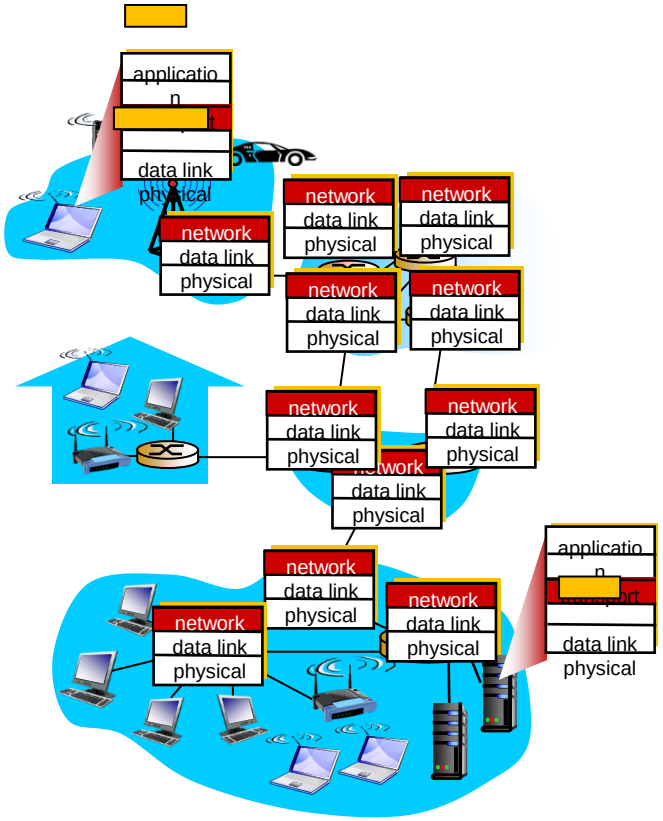
Dr. Parin Sornlertlamvanich,

parin.s@sut.ac.th

Network layer



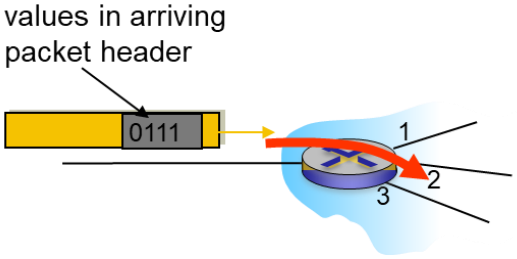
- The internetwork uses a router connected to each LAN, with a WAN (Wire-Area Networks) link between the routers
- Transport segment from sending to receiving host
  - On sending side encapsulates segments into datagrams
  - On receiving side, delivers segments to transport layer
- Network layer protocols in *every* Host, Router
  - Router examines header fields in all IP datagrams passing through it



Network layer: data plane, control plane

Data plane

- Local, per-router function
- Determines how datagram arriving on router input port is forwarded to router output port
- Forwarding function

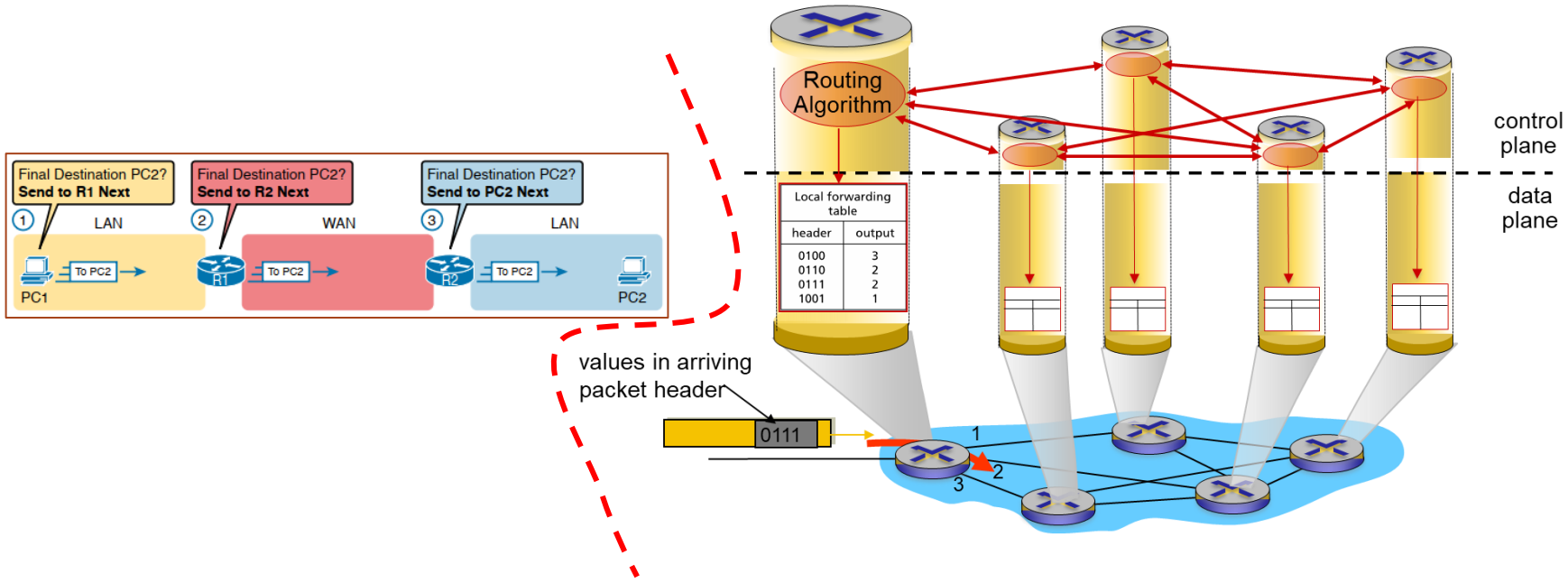


Control plane

- Network-wide logic
- Determines how datagram is routed among routers along end-end path from source host to destination host
- Two control-plane approaches:
  - *traditional routing algorithms:* implemented in Routers
  - *software-defined networking (SDN):* implemented in (remote) Servers
    - Controller

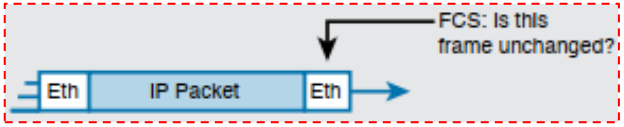
Per-router control plane

- Individual routing algorithm components in each and every router interact in the control plane



Sending a IP datagram (1)

- **Step1:** Use the data-link Frame Check Sequence (FCS) field to ensure that the frame had no errors; if errors occurred, discard the frame.
- **Step2:** To discard the old data-link header and trailer, leaving the IP packet.
- **Step3:** Compare the IP packet's destination IP address to the routing table, and find the route that best matches the destination address
- **Step4:** Encapsulate the IP packet inside a new data-link header and trailer, and forward the frame.



R1 Routing Table

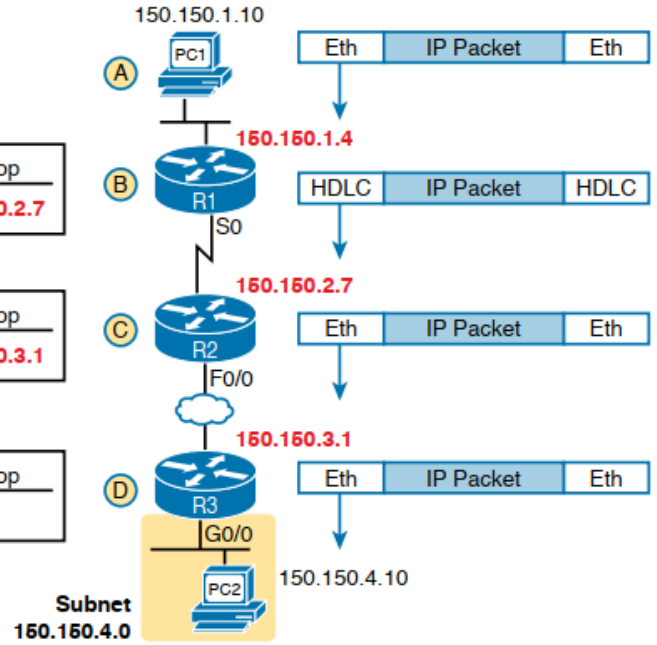
Subnet	Interface	Next Hop
150.150.4.0	Serial0	150.150.2.7

R2 Routing Table

Subnet	Interface	Next Hop
150.150.4.0	FastEth0/0	150.150.3.1

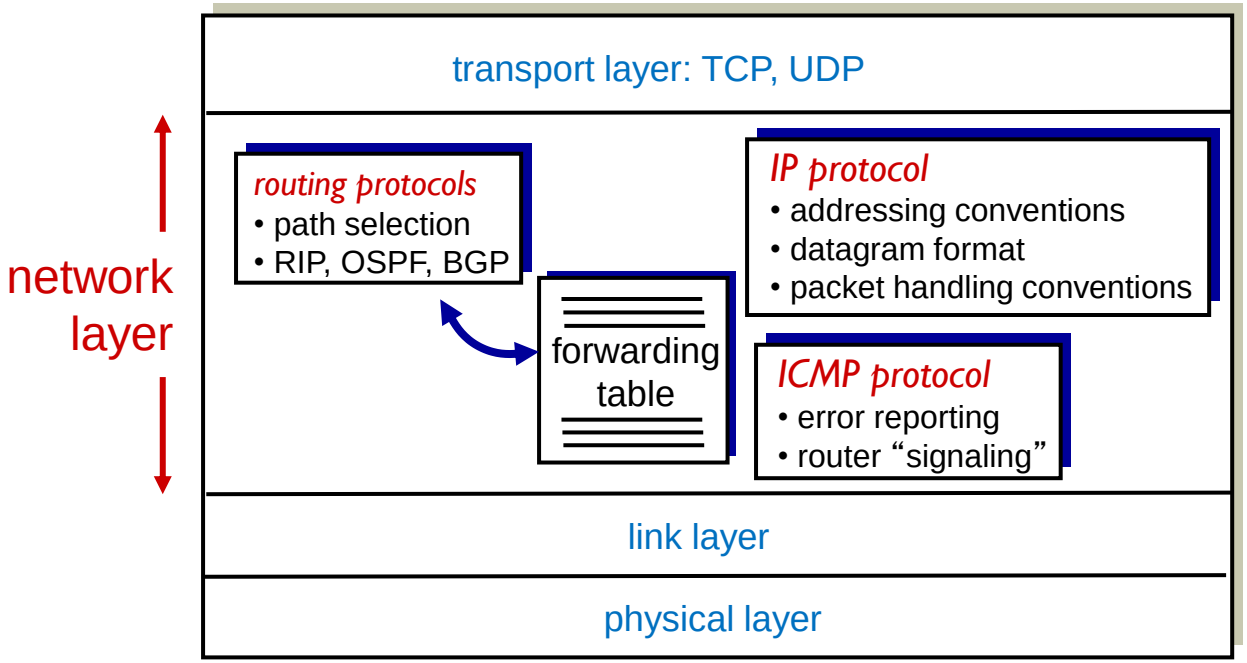
R3 Routing Table

Subnet	Interface	Next Hop
150.150.4.0	Gigabit0/0	N/A

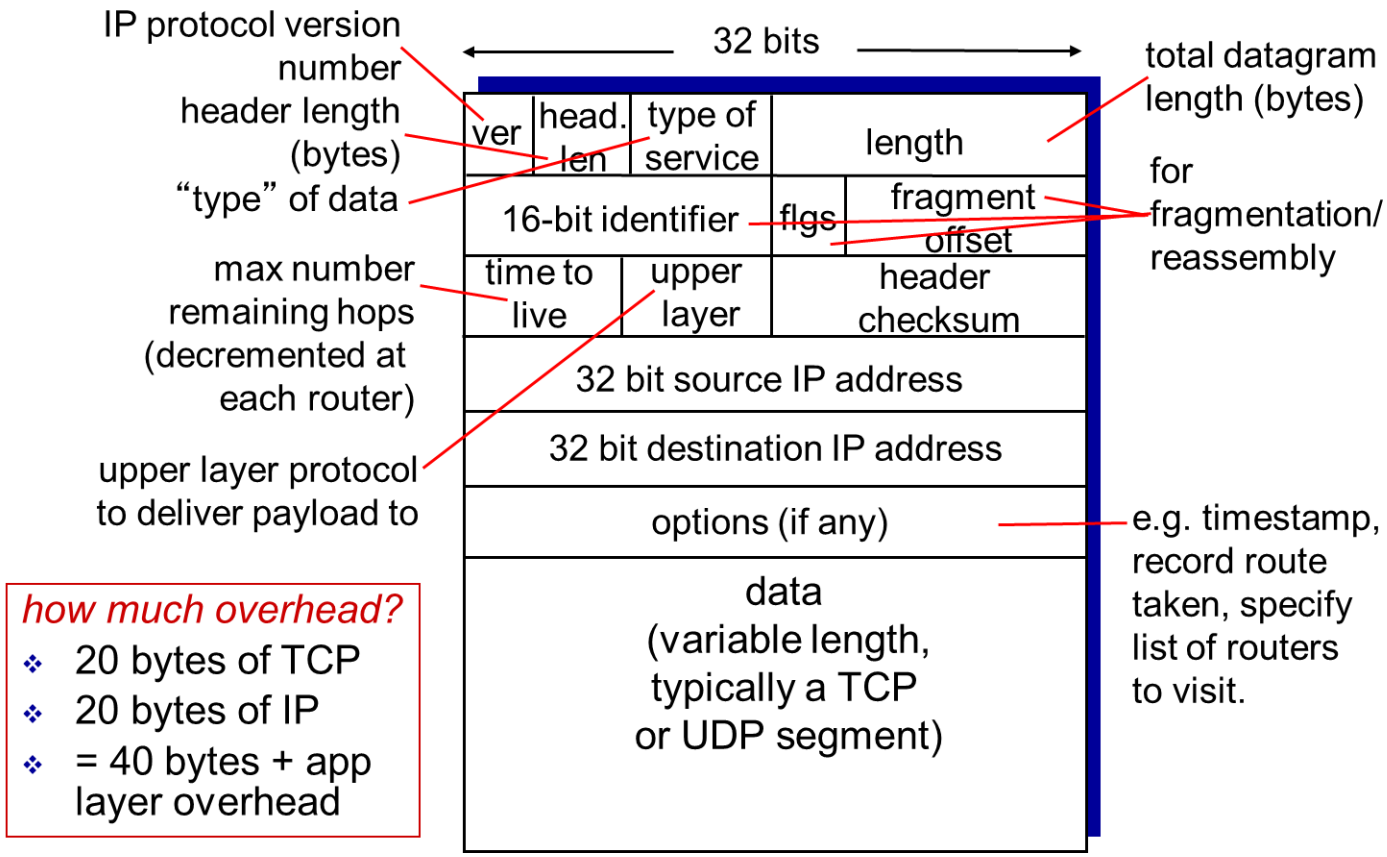


The Internet network layer

Host, Router network layer functions:



IP Datagram format

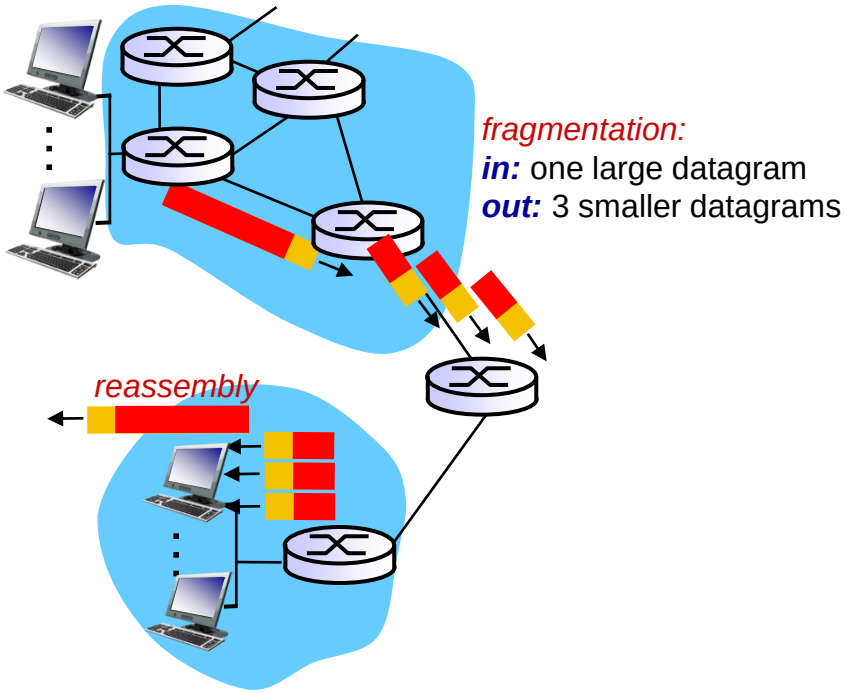


how much overhead?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

IP Fragmentation, reassembly

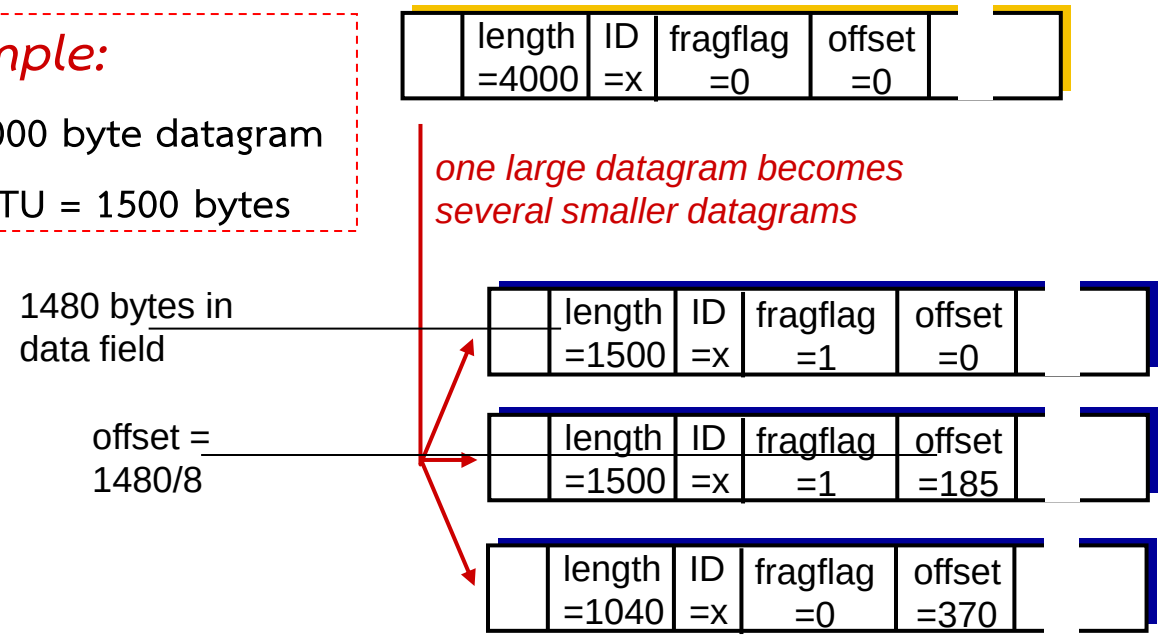
- Network links have MTU (max.transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order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



IP Fragmentation, reassembly

example:

- 4000 byte datagram
- MTU = 1500 bytes



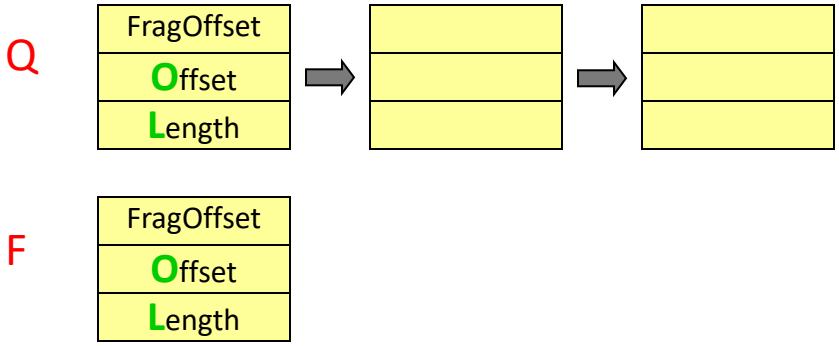
Reassembly Algorithm

- One fragment queue
  - For each different packet
  - Where fragment has arrived
    - deliver complete packets, don't queue them
- Queue ordered
  - by order of fragment offset
    - smallest first
    - biggest last
  - not arrival order
- Merge fragments on queue
  - whenever possible
- When queue contains complete packet
  - Deliver packet
    - no more need for this queue

Fragment Merge Decision

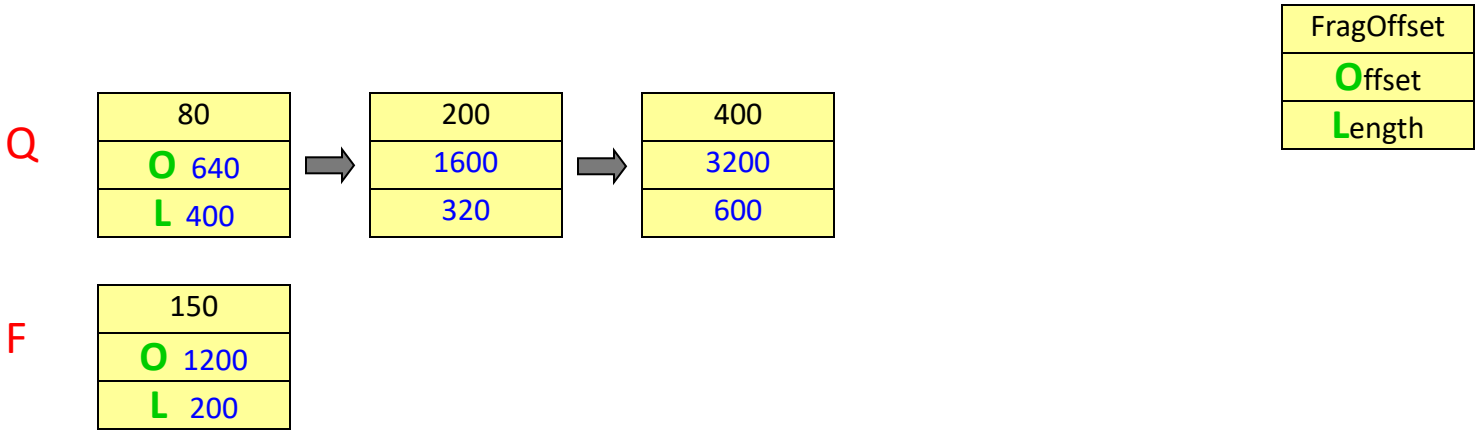
- Fragment F arrives
  - Fragment offset — O — (in bytes)
  - Fragment Length — L — (excluding header)
- Find the appropriate queue
  - If none
    - create one -- nothing to merge, put F in queue
- Scan queue in order (increasing fragment offsets)
  - For each packet Q on the queue in order
    - if O(F) < O(Q) && O(F) + L(F) >= O(Q)
      - F merges into Q (details later)
      - done
    - if O(F) < O(Q)
      - F inserted in queue before Q
      - done
    - if O(Q) + L(Q) >= O(F)
      - F merges into Q
      - use merged packet as F
      - continue
    - if reach end of queue, append F

Merge Decision Examples



- Queue of waiting Fragments
- Fragment arriving
- Already tested, and know
  - all fragments from same packet
- i.e.:
  - same source & destination addresses
  - same protocol
  - same packet identifier

Merge Example 1

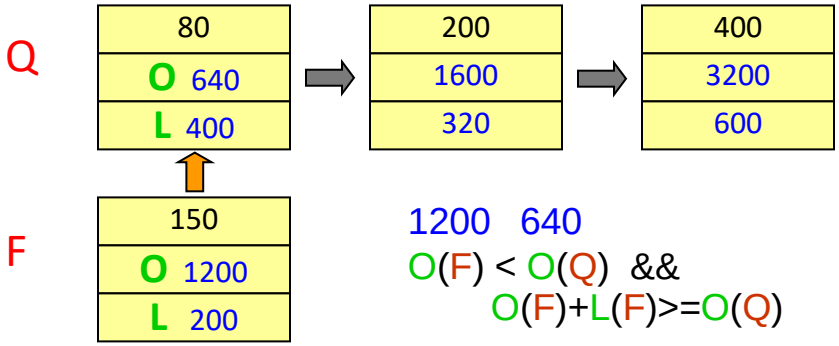


- Queue of waiting fragments
  - with particular Offsets and Lengths
- Fragment arrives
  - It also has Offset and Length

- We run the merge algorithm

Merge Example 1 (2)

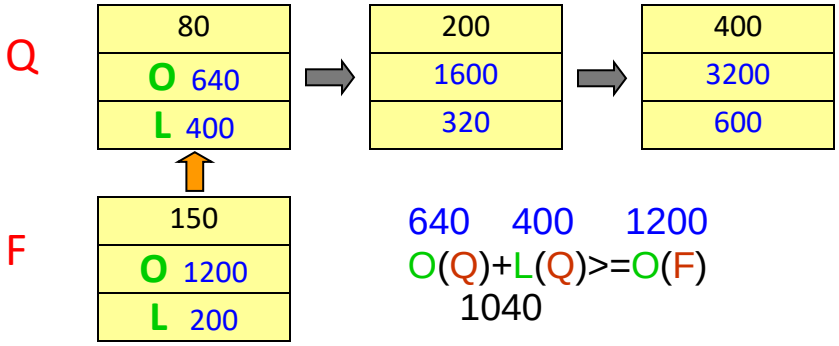
FragOffset
Offset
Length

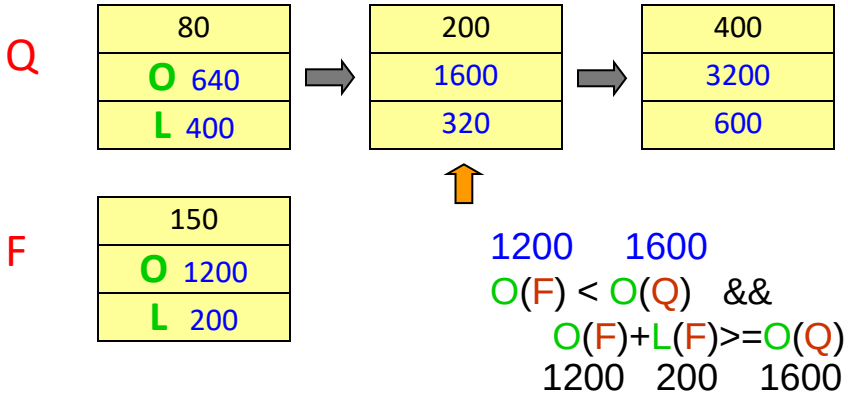


1200 640
O(F) < O(Q) &&
O(F)+L(F)>=O(Q)

if O(F) < O(Q) && O(F) + L(F) >= O(Q)
F merges into Q (details later)
done
if O(F) < O(Q)
F inserted in queue before Q
done
if O(Q) + L(Q) >= O(F)
F merges into Q
use merged packet as F
continue

Merge Example 1 (3)



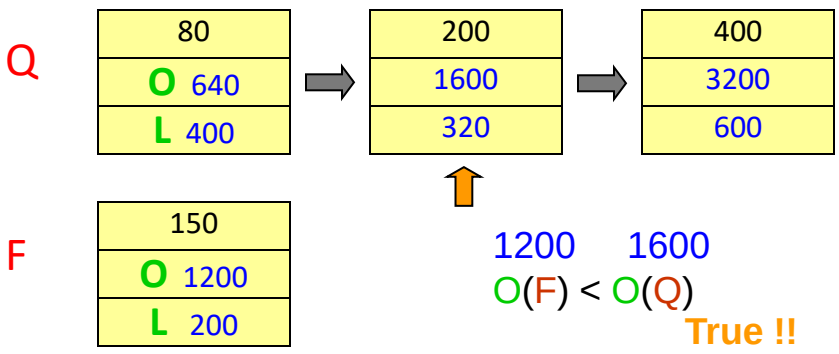


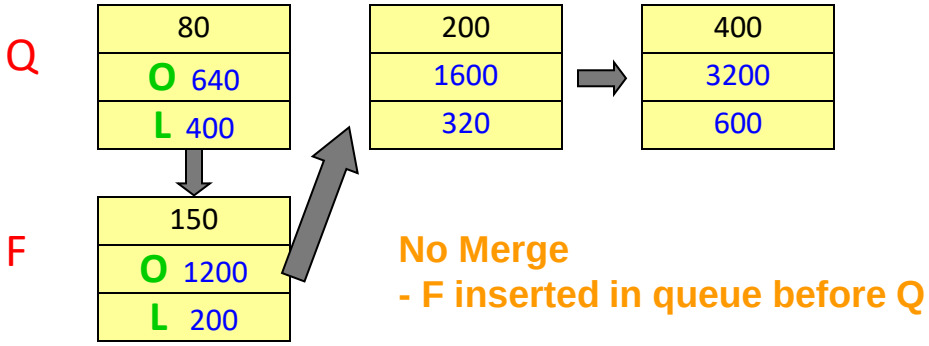
FragOffset
Offset
Length

if O(F) < O(Q) && O(F) + L(F) >= O(Q)
    F merges into Q (details later)
    done
if O(F) < O(Q)
    F inserted in queue before Q
    done
if O(Q) + L(Q) >= O(F)
    F merges into Q
    use merged packet as F
    continue

Merge Example 1 (4)

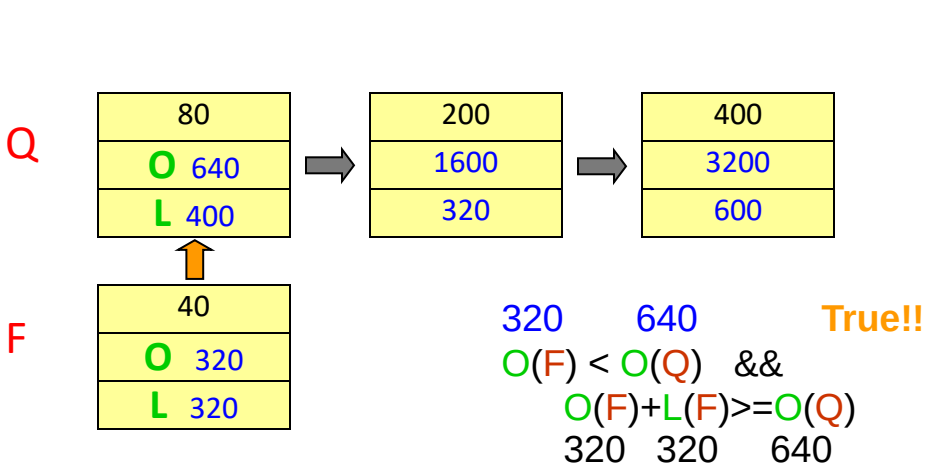
FragOffset
Offset
Length

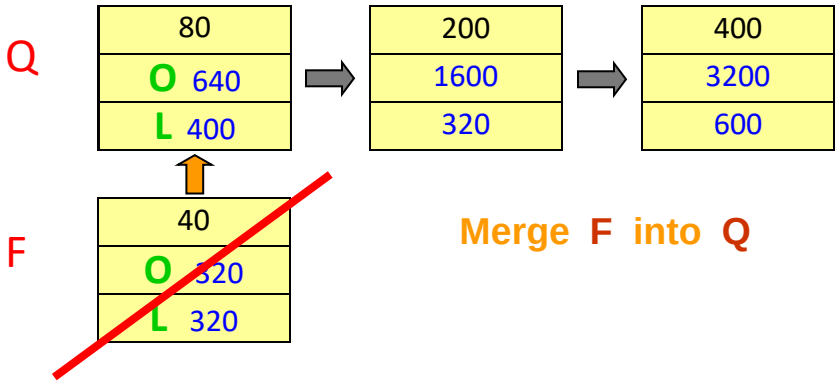




if O(F) < O(Q) && O(F) + L(F) >= O(Q)
    F merges into Q (details later)
    done
if O(F) < O(Q)
    F inserted in queue before Q
    done
if O(Q) + L(Q) >= O(F)
    F merges into Q
    use merged packet as F
    continue

Merge Example 2

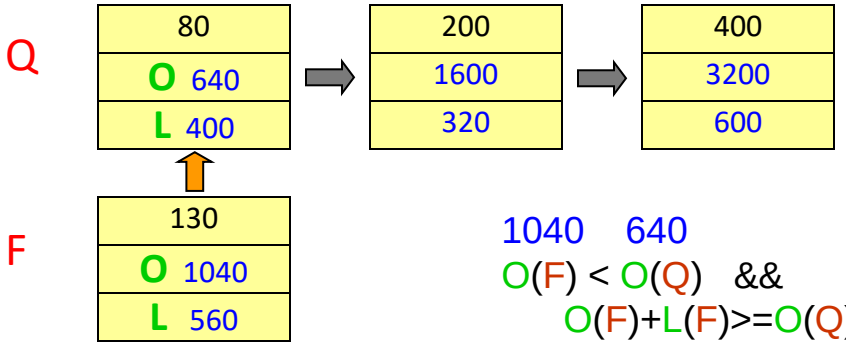




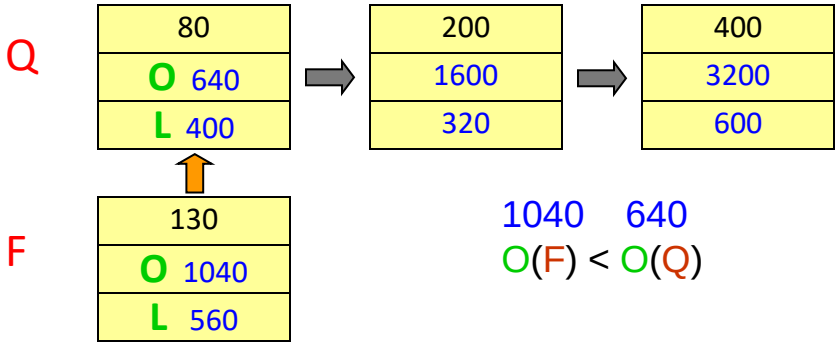
if O(F) < O(Q) && O(F) + L(F) >= O(Q)
    F merges into Q (details later)
    done
if O(F) < O(Q)
    F inserted in queue before Q
    done
if O(Q) + L(Q) >= O(F)
    F merges into Q
    use merged packet as F
    continue

Merge Example 3

FragOffset
Offset
Length



1040 640
O(F) < O(Q) &&
O(F)+L(F)>=O(Q)

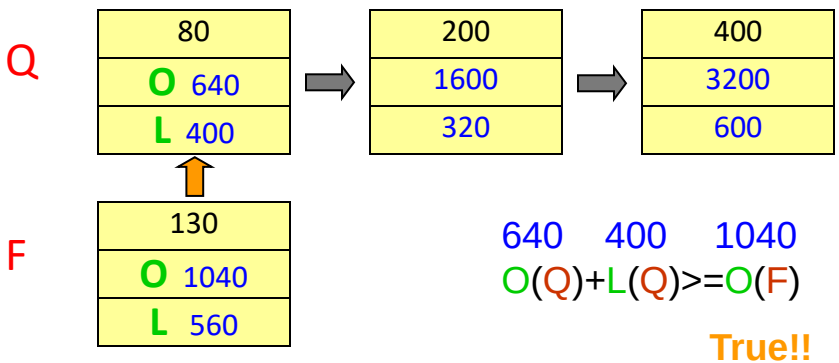


1040 640
O(F) < O(Q)

if O(F) < O(Q) && O(F) + L(F) >= O(Q)
    F merges into Q (details later)
    done
if O(F) < O(Q)
    F inserted in queue before Q
    done
if O(Q) + L(Q) >= O(F)
    F merges into Q
    use merged packet as F
    continue

Merge Example 3 (2)

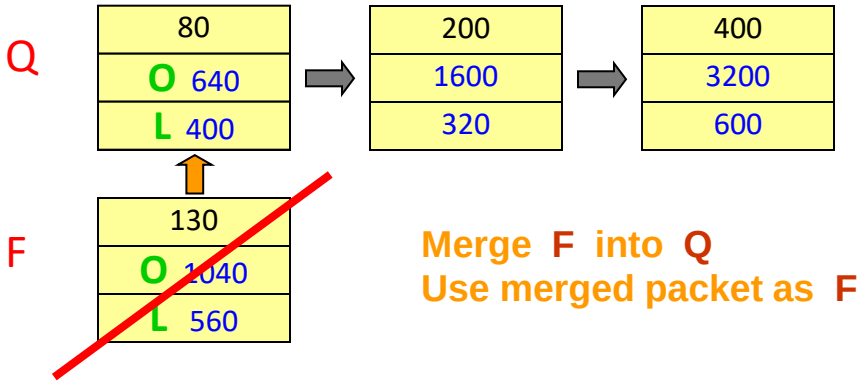
FragOffset
Offset
Length



640 400 1040

O(Q)+L(Q)>=O(F)

True!!



Merge F into Q
Use merged packet as F

if O(F) < O(Q) && O(F) + L(F) >= O(Q)
    F merges into Q (details later)
    done
if O(F) < O(Q)
    F inserted in queue before Q
    done
if O(Q) + L(Q) >= O(F)
    F merges into Q
    use merged packet as F
    continue

Merge Example 3 (3)

FragOffset
Offset
Length

Q

200	400
1600	3200
320	600

F

80
O 640
L 960

Merge F into Q
Use merged packet as F

Q

200	400
1600	3200
320	600

F

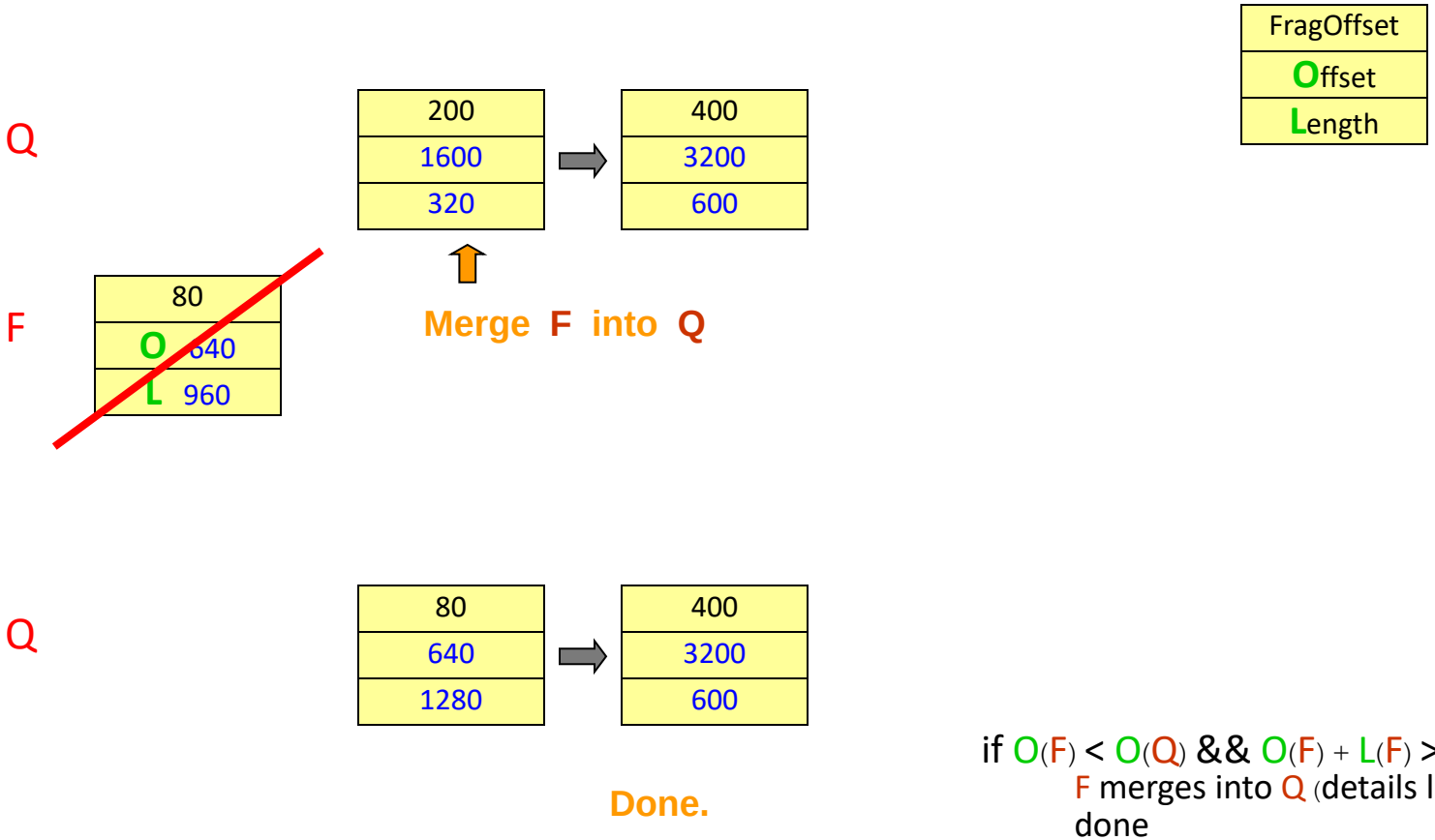
80
O 640
L 960

640 1600
O(F) < O(Q) &&
O(F)+L(F)>=O(Q)
640 960 1600
1600

True!!

if O(F) < O(Q) && O(F) + L(F) >= O(Q)
    F merges into Q (details later)
    done
if O(F) < O(Q)
    F inserted in queue before Q
    done
if O(Q) + L(Q) >= O(F)
    F merges into Q
    use merged packet as F
    continue

Merge Example 3 (4)

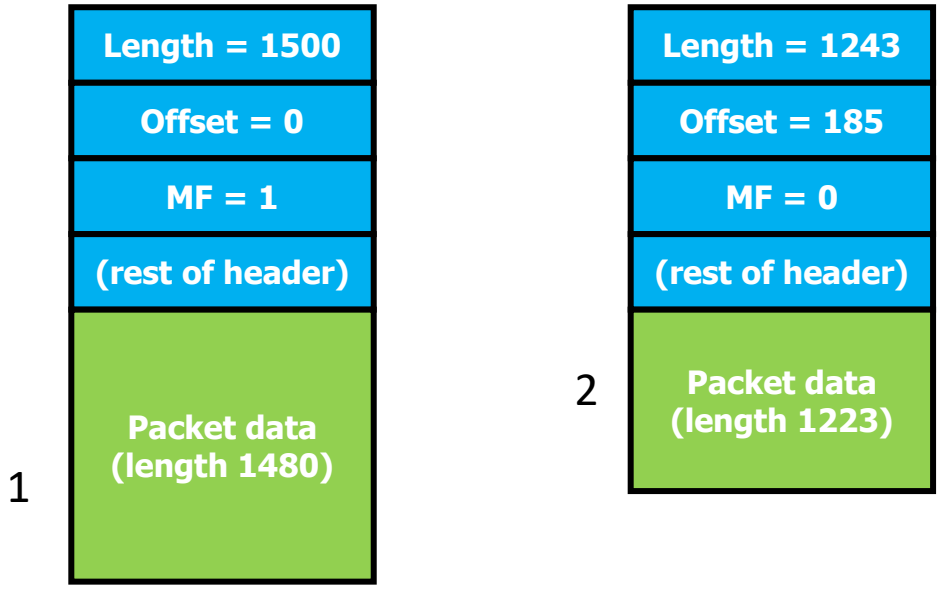


if O(F) < O(Q) && O(F) + L(F) >= O(Q)
    F merges into Q (details later)
    done
if O(F) < O(Q)
    F inserted in queue before Q
    done
if O(Q) + L(Q) >= O(F)
    F merges into Q
    use merged packet as F
    continue

Merge Example Summary

- Example 1
  - Fragment inserted in queue
  - In correct location
    - So offsets in queue remain sorted
  - No merge possible
- Example 2
  - Fragment merged into fragment in queue
    - Data fit before queue fragment
- Example 3
  - Fragment merged into fragment in queue
    - Data fit after queue fragment
  - Resulting merged fragment
    - Merged into following queue fragment
  - Original fragment filled the gap
    - between two fragments on the queue
- Many other variations possible

Fragment Reassembly Example

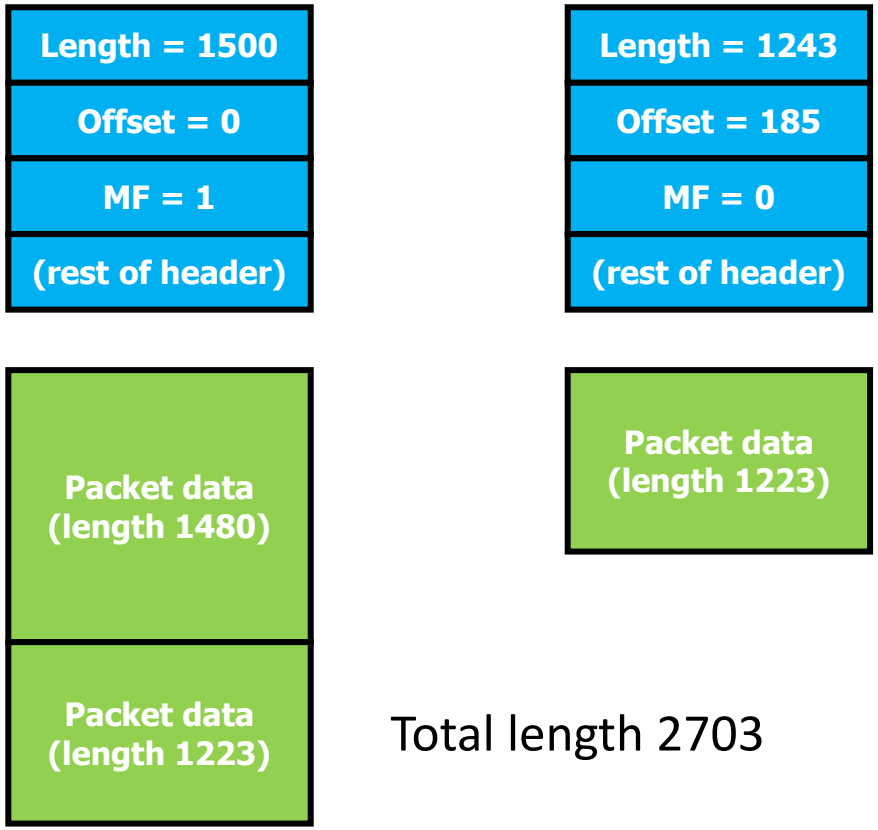


Length(1) + Offset(1) >= Offset(2) ➔ Merge

1480 0(*8) 185(*8)

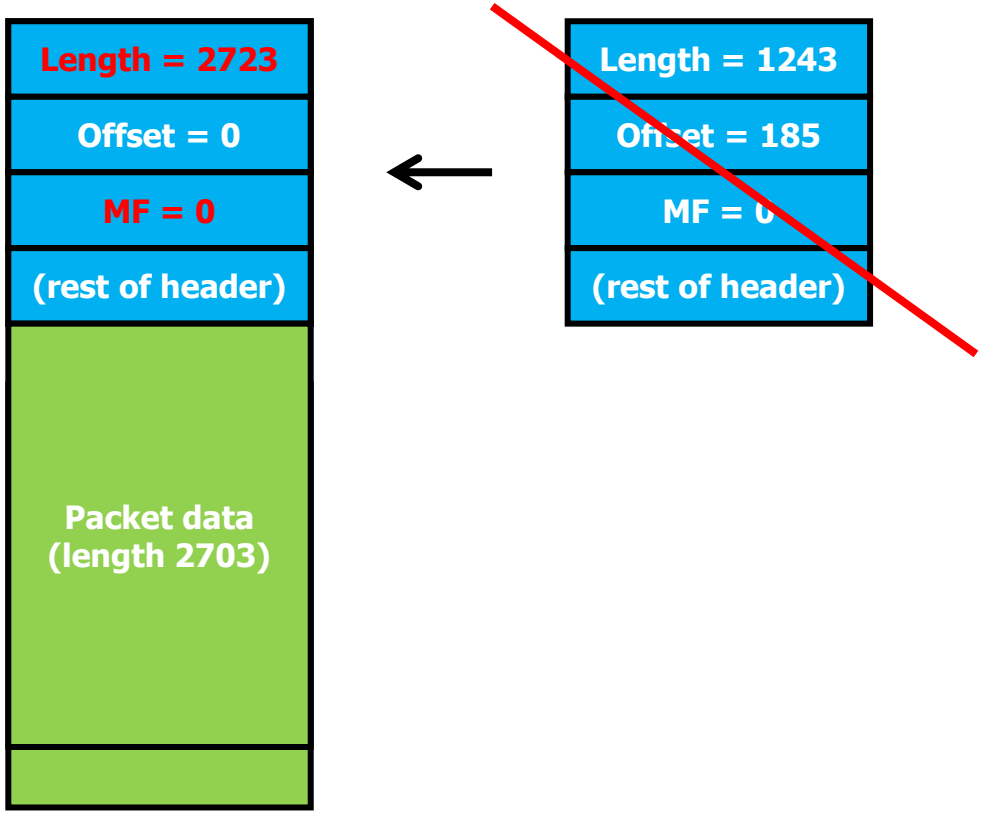
185*8 =1480

Fragment Reassembly Eg (cont.)

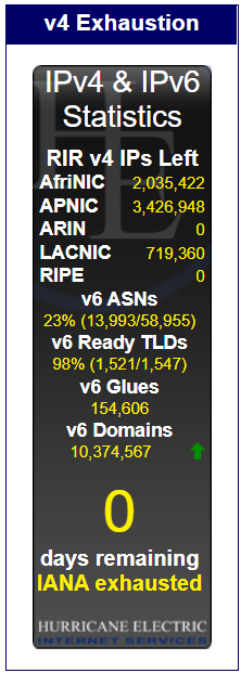


Total length 2703

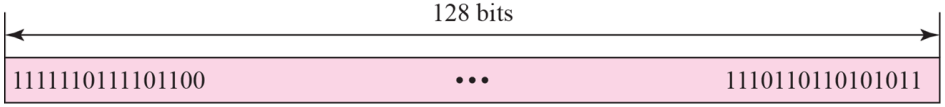
Fragment Reassembly Eg (cont.)



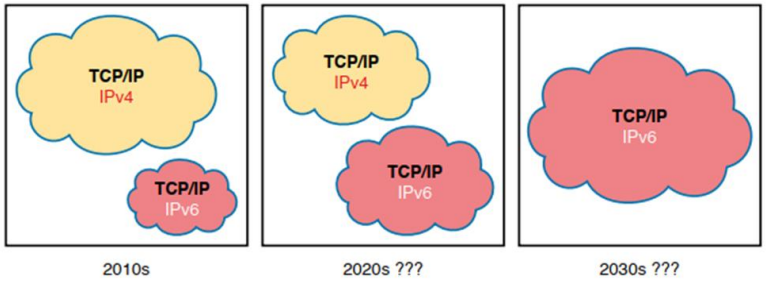
IPv6: motivation



- Initial motivation:
  - Run Out of IPv4 Addresses
- Additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS
- IPv6 datagram format:
  - 128 bits – IPv6 address
  - fixed-length 40 byte header
  - no fragmentation allowed
    - supporting the cc[illegible] the path MTU

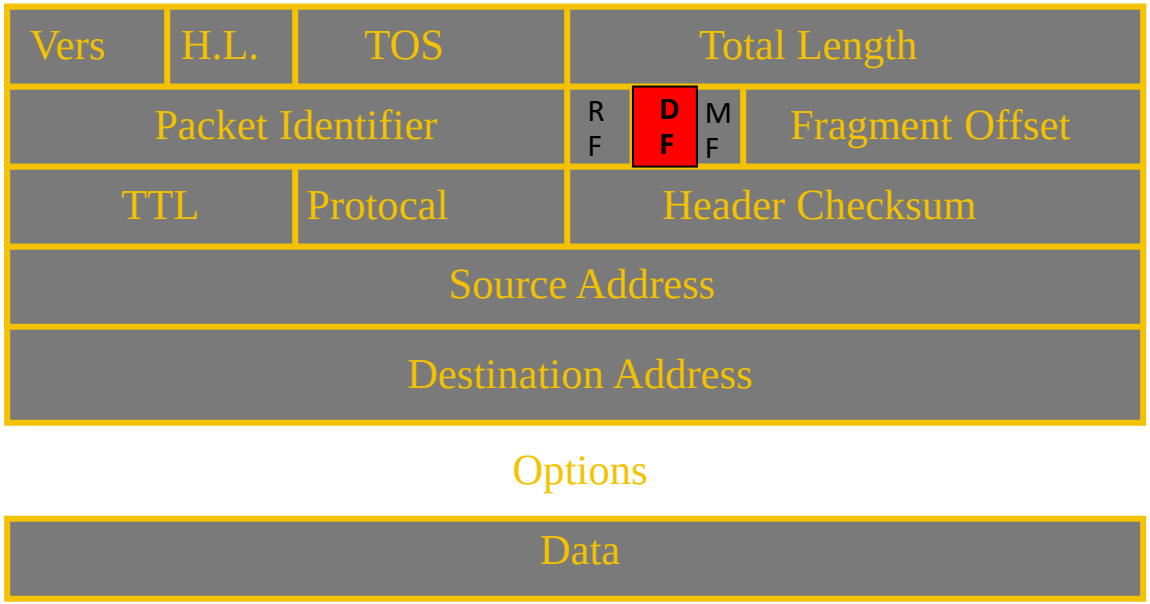


FDEC : BA98 : 7654 : 3210 : ADBF : BBFF : 2922 : FFFF



Path MTU Discovery (PMTUD)

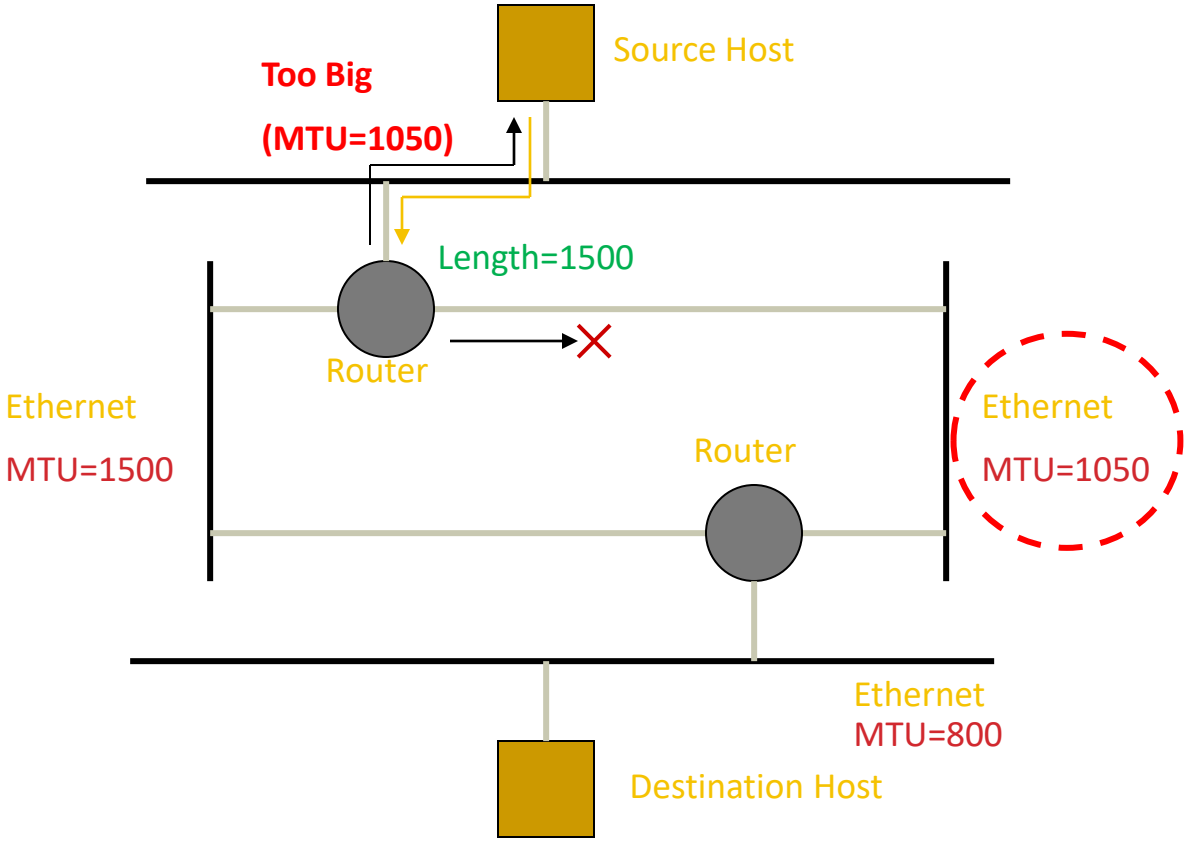
- Sending Host discovers the minimum MTU
  - of the Path used
  - and then sends its packets
  - no bigger than that
  - DF - Do-Not-Fragment (DF) bit



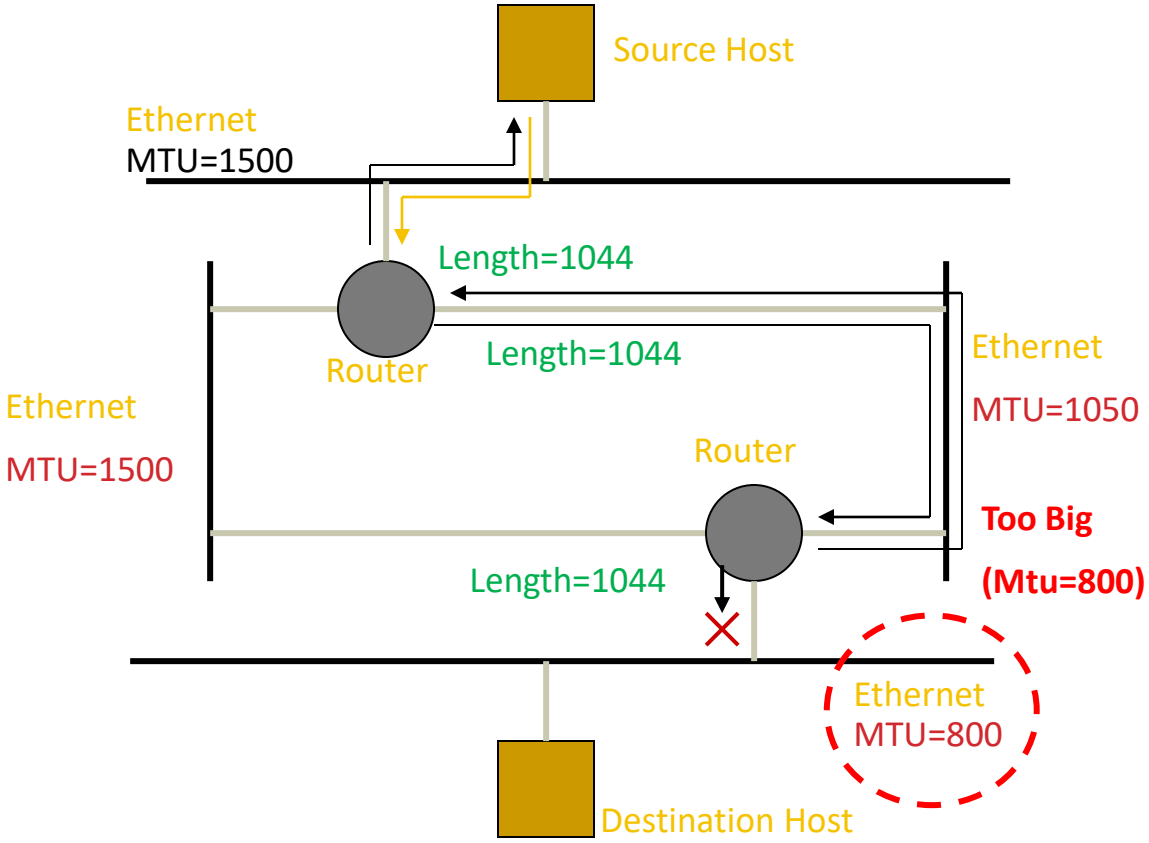
PMTUD (cont)

- If a packet arrives at a router
  - The DF bit is set in the header
  - If fragmentation is required to send the packet
- Router discards the packet
  - Sends ICMP "packet too big and DF set" to source
    - Includes the MTU of the link to be used
    - Source host knows MTU of this link
      - smallest link so far

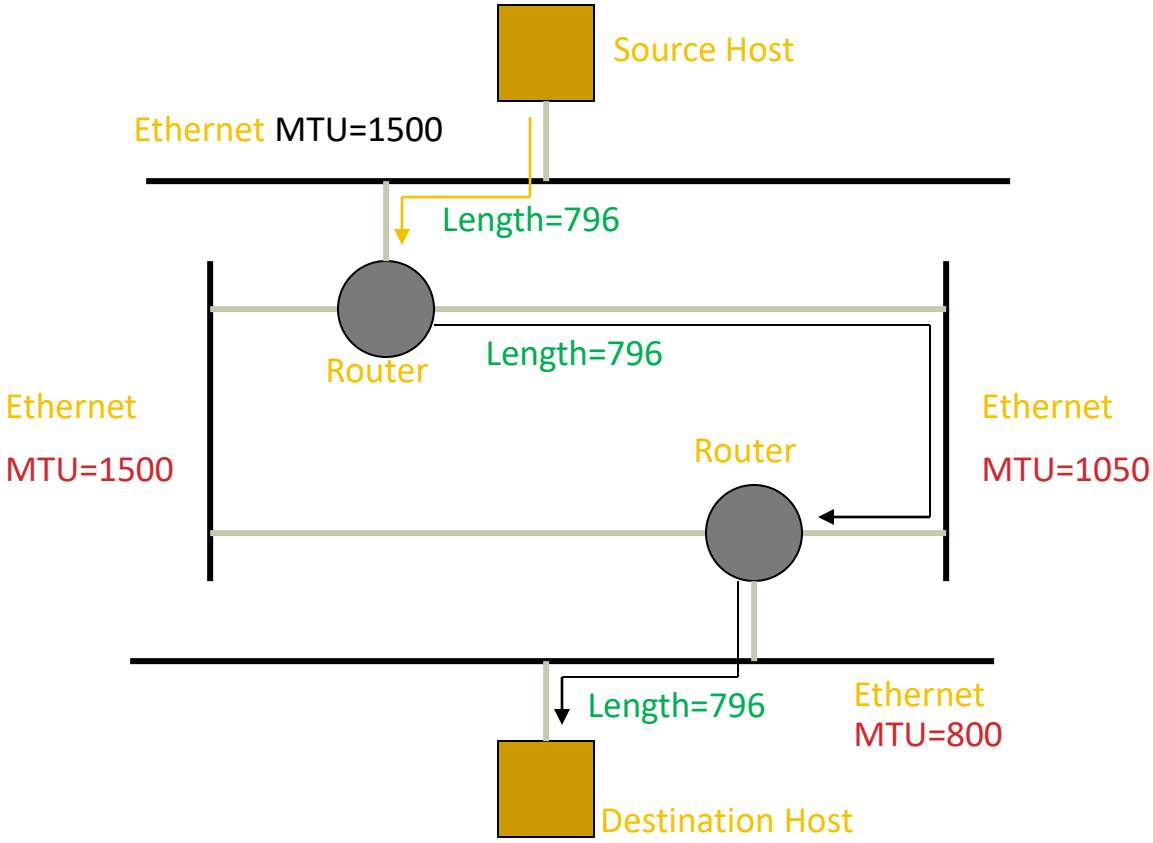
PMTUD (cont)


Source Host
Too Big
(MTU=1050)
Length=1500
Router
Ethernet
MTU=1500
Router
Ethernet
MTU=1050
Ethernet
MTU=800
Destination Host

PMTUD (cont)


Source Host
Ethernet
MTU=1500
Length=1044
Router
Length=1044
Ethernet
MTU=1050
Ethernet
MTU=1500
Router
Too Big
(Mtu=800)
Length=1044
Ethernet
MTU=800
Destination Host

PMTUD (cont)


Source Host
Ethernet MTU=1500
Length=796
Router
Length=796
Ethernet
MTU=1500
Router
Ethernet
MTU=1050
Length=796
Ethernet
MTU=800
Destination Host

Final PMTUD

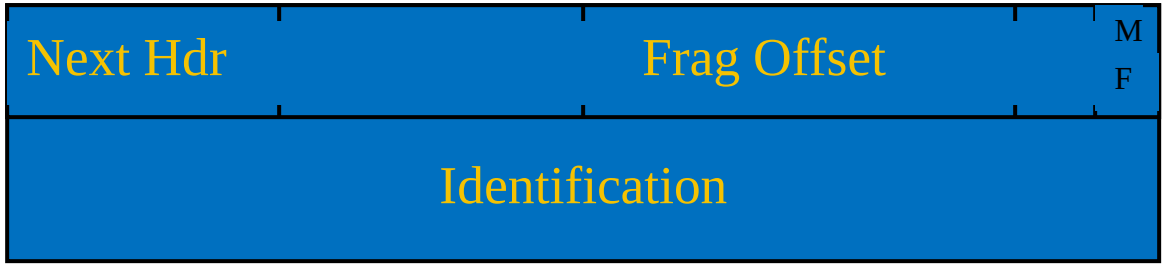
- To allow for possibility
  - ➢ that PMTU has increased
- Sender occasionally
  - ➢ sends slightly bigger packet
- If that fails
  - ICMP Too Big is returned
  - ➢ Then nothing has changed
    - Determined PMTU remains as found earlier

- If bigger packet reaches Destination
  - ➢ Then PMTU has increased
    - Sender remembers bigger packets work
    - Tries again with even bigger packet
      - Until local link MTU reached
      - Or until ICMP is returned
  - ➢ Now the new PMTU is known

Fragmentation Lessons

- Fragmentation wastes bandwidth
  - Entire packets transmitted if fragment lost
- Fragmentation requires router complexity
  - More than just forward or drop decision
- Avoid fragmentation if possible
  - Use PMTUD
    - Discover max possible packet size
    - Use that as MSS for TCP
  - Keep UDP packets small
    - Avoid fragmentation

IPv6 Fragment Header



- Fragment Offset & MF
  - Identical to IPv4
- Identification
  - Identical purpose, but 32 bits
    - Less chance of accidental collision
- DF
  - Not needed

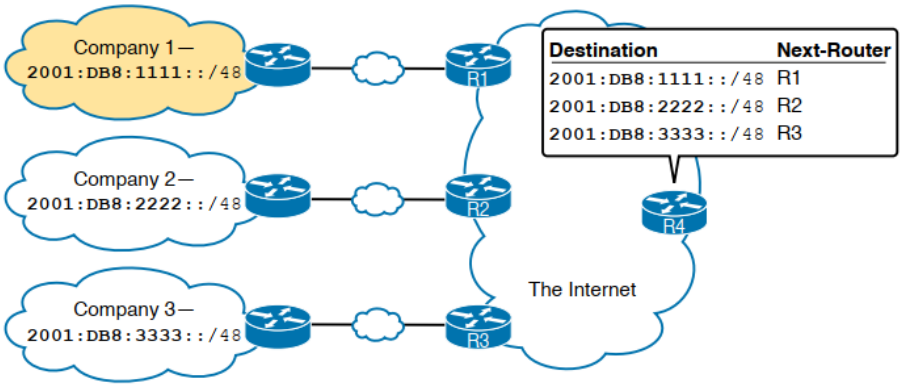
IPv6

- Requires use of Fragment header
  - Only source nodes add headers
  - Only source node can fragment packets
    - No router complexity
- No overheads when no fragmentation
- PMTUD is required
  - Or packets must remain smaller
  - than guaranteed PMTU
    - 1280 for IPv6

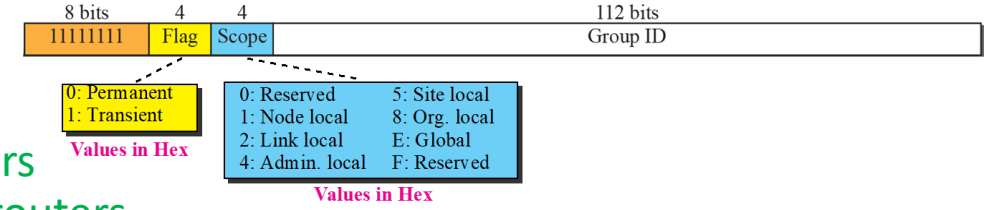
Other changes from IPv4

- Checksum: removed entirely to reduce processing time at each hop
  - The transport-layer (for example, TCP and UDP) and link-layer (for example, Ethernet) protocols in the Internet layers perform check-summing
- Options: allowed, but outside of header, indicated by “Next Header” field
- ARP Replaced by Neighbor Discovery Protocol:
  - For IPv4, ARP discovers the MAC address used by neighbors
  - IPv6 replaces ARP with a more general Neighbor Discovery Protocol (NDP).
- ICMPv6: new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions
- Older OSPF Version 2 Upgraded to OSPF Version 3:
  - OSPF version 3 was created to support IPv6

Address type



- Global (public, unicast)
  - IPv4: 203.11.22.33
  - IPv6: 2xxx:… - 3xxx :…
- Broadcast (To all)
  - IPv4: 203.11.22.255 (host address = all 1's)
  - IPv6: does not exist
- Multicast (to a group)
  - IPv4: 224.0.0.1 (224.x.x.x, 225.x.x.x, 226.x.x.x)
  - IPv6: FFxx:…
    - ff02::1 All IPv6 devices
    - ff02::2 All IPv6 routers
    - ff02::5 All OSPFv3 routers
    - ff02::a All EIGRP (IPv6) routers



- Anycast (one-to-one-of-many)

Address type (cont.)

- Link-Local (unique on a segment)
  - IPv4: 169.254.22.33
  - IPv6: FE8x:... – FEBx:...
- Private (unique on a LAN)
  - IPv4: 192.168.x.x, 10.x.x.x, 172.16.x.x, 172.31.x.x
  - IPv6: FC00:... , FD00:...
    (Unique Local Address)
- Loopback
  - IPv4: 127.0.0.1
  - IPv6: ::1

Prefix

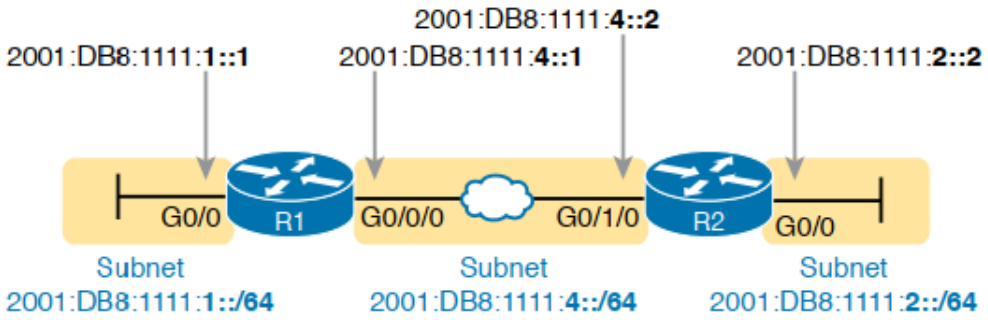
- Prefix concept replaces netmask of IPv4
- Prefix = number of bits for network address

2400:cb00:2048:0001:0000:0000:c71b:87a7/48

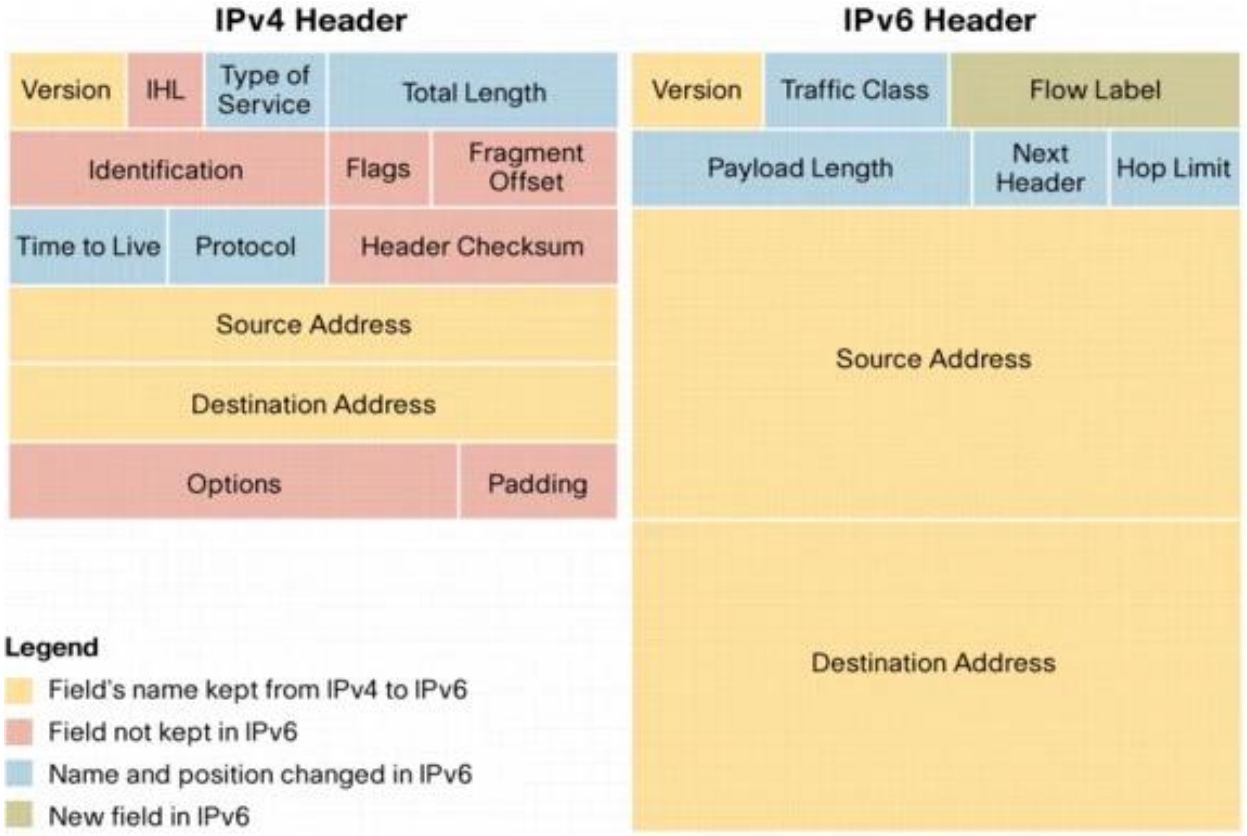
Network address (48 bits)

host address (80 bits)

2400:cb00:2048::/48 indicates network number



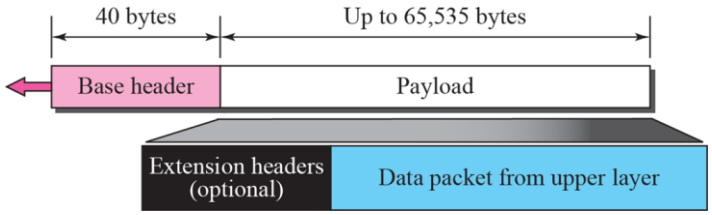
IPv4 and IPv6 Headers

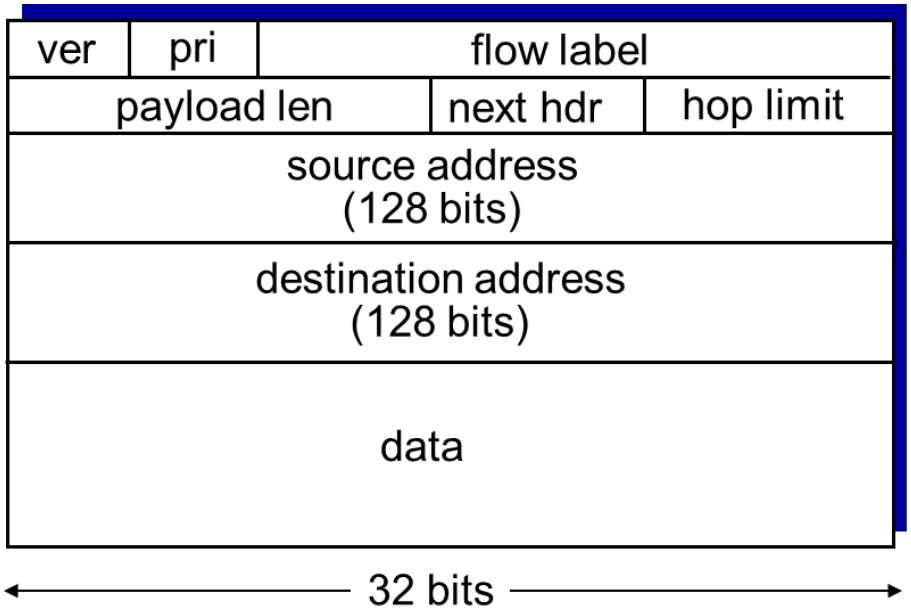


- Removed:
  - Header Length field
  - Header Checksum
  - Identification, Flags, Fragment Offset
  - TOS

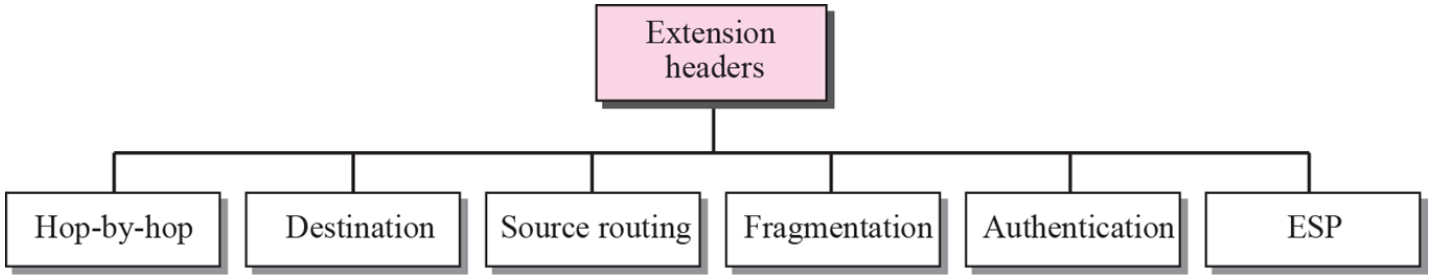
IPv6 datagram format

- Priority: identify priority among datagrams in flow
- Flow Label: identify datagrams in same flow.
- Next header: identify upper layer protocol for data

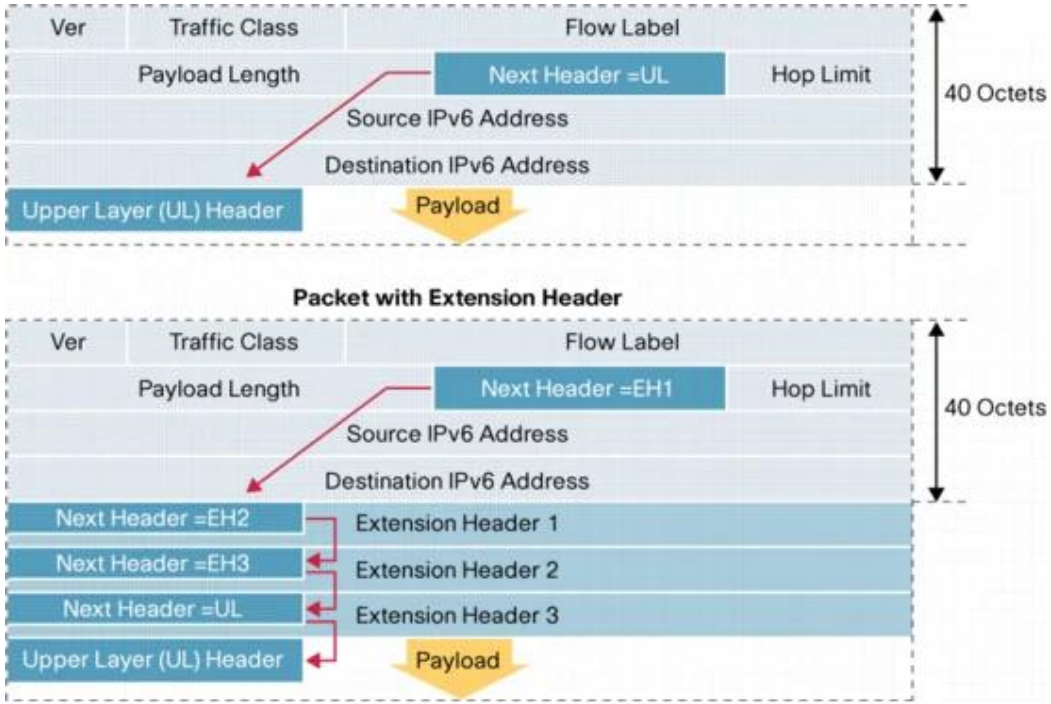




IPv6 Next Header

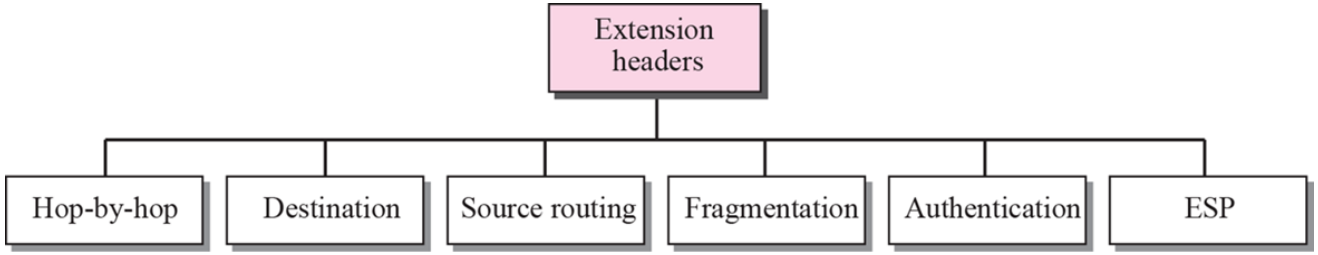


- IPv6 is using two distinct types of headers: Main/Regular IPv6 Header and IPv6 Extension Headers.
  - The main IPv6 header is equivalent to the basic IPv4 one despite some field differences
  - Extension headers are daisy-chained by the "next header" field



https://www.cisco.com/en/US/technologies/tk648/tk872/technologies_white_paper0900aecd8054d37d.html

Commonly Used Extension Headers



- Hop-by-Hop Extension Header (EH)
- The Hop-by-Hop Extension Header is the ONLY EH that MUST be fully processed by all network devices
  - This EH MUST be the first in a chain of extension headers.
  - Hop-by-Hop EH is used for the support of Jumbo-grams or, with the Router Alert option.

Recommended Order in a Packet

Order	Header Type	Next Header Code
1	**Basic IPv6 Header**	-
2	**Hop-by-Hop Options**	0
3	**Destination Options (with Routing Options)**	60
4	**Routing Header**	43
5	**Fragment Header**	44
6	**Authentication Header**	51
7	**Encapsulation Security Payload Header**	50
8	**Destination Options**	60
9	**Mobility Header**	135
	No next header	59
Upper Layer	TCP	6
Upper Layer	UDP	17
Upper Layer	ICMPv6	58

https://www.cisco.com/en/US/technologies/tk648/tk872/technologies_white_paper0900aecd8054d37d.html

Commonly Used Extension Headers

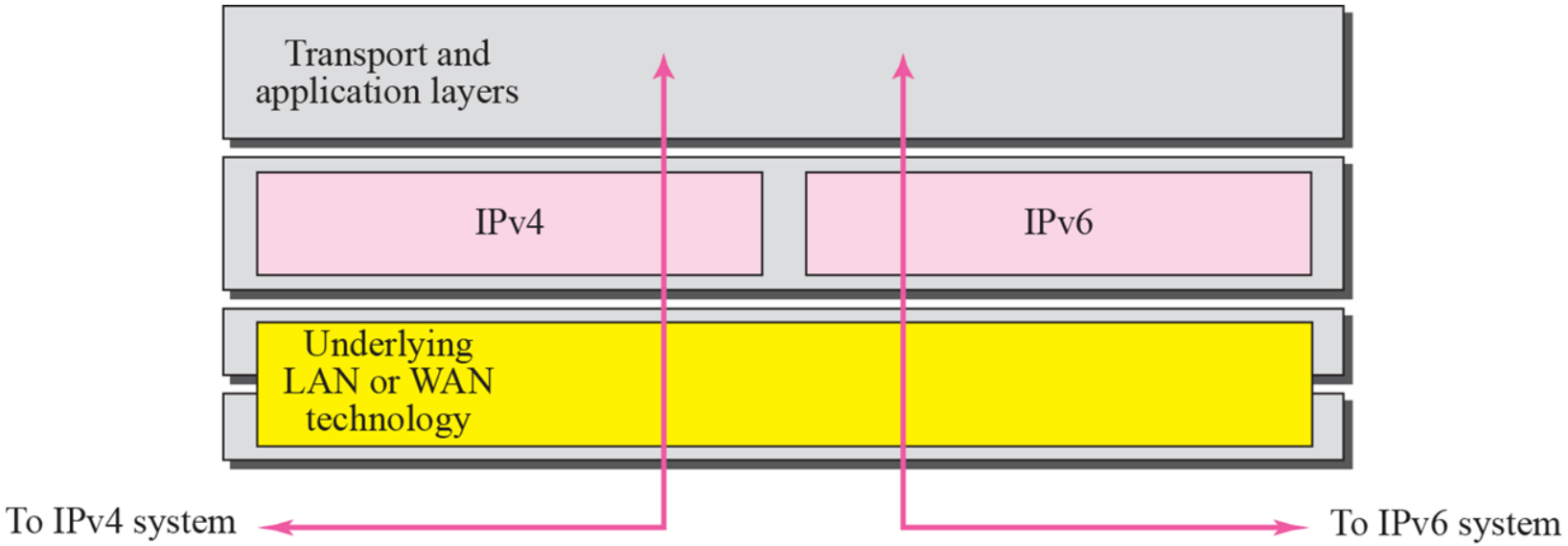
- **Fragmentation EH**
  - is critical in support of communication using fragmented packets (in
    - IPv6, the traffic source must do fragmentation
    - Routers do not perform fragmentation of the packets they forward
- **Mobility EH**
  - is used in support of Mobile IPv6 service
- **Authentication EH**
  - is similar in format and use to the IPv4 authentication header
- **Encapsulating Security Payload EH**
  - is similar in format and use to the IPv4 ESP header. All information following ESP Header is encrypted. The ESP Header can be followed by an additional Destination Options EH and the upper layer datagram.

Transition from IPv4 to IPv6

- **Because of the huge number of systems on the Internet, the transition from IPv4 to IPv6 cannot happen suddenly.**
  - It will take a considerable amount of time before every system in the Internet can move from IPv4 to IPv6.
  - The transition must be smooth to prevent any problems between IPv4 and IPv6 systems.

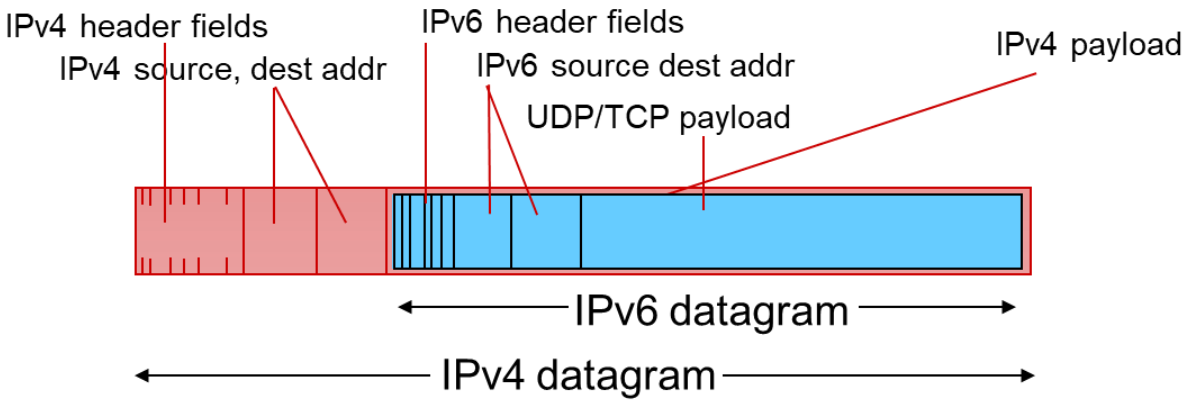
- **The strategies have been devised by the IETF to help the transition**
  - Dual Stack
  - Tunneling

Dual stack

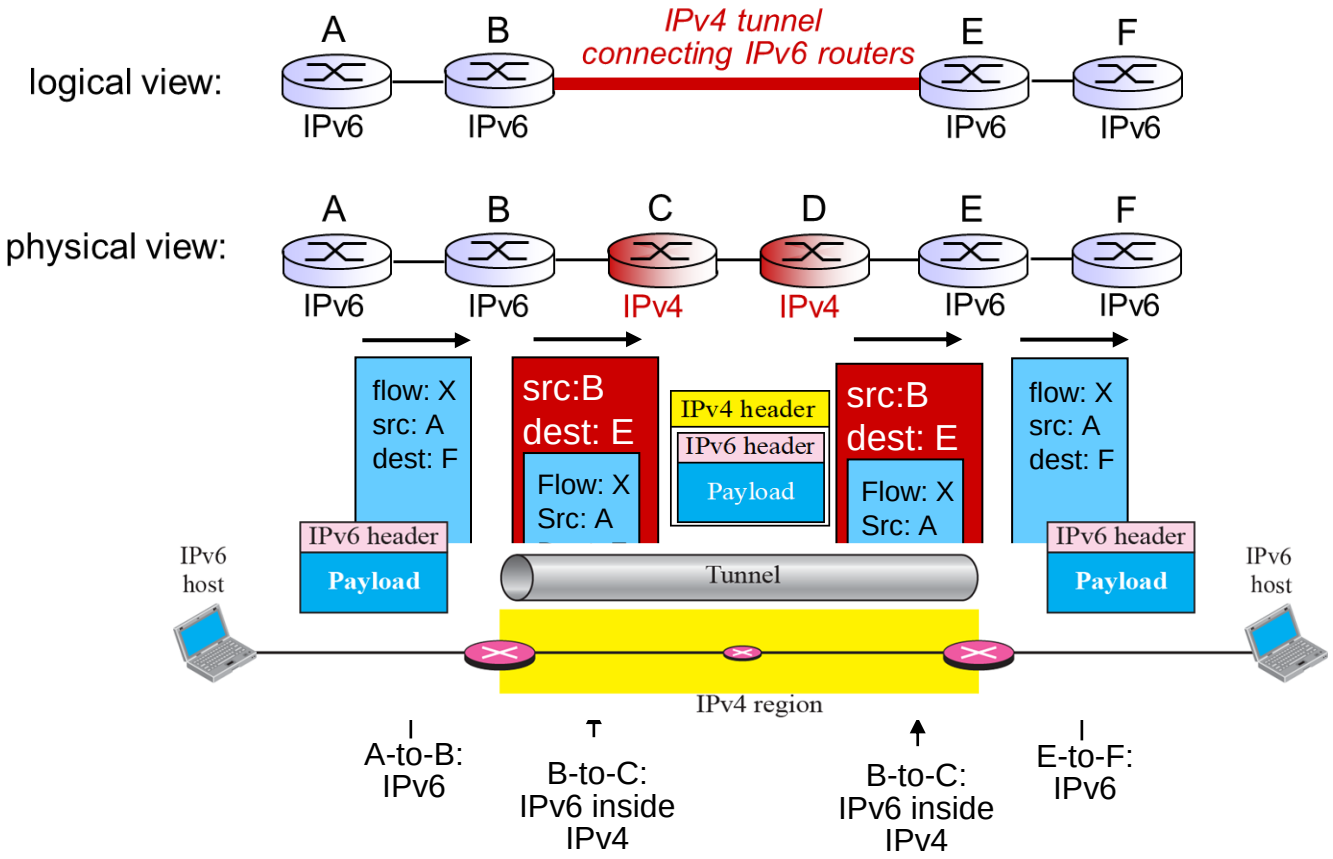

Transport and application layers
IPv4
IPv6
Underlying LAN or WAN technology
To IPv4 system
To IPv6 system

Tunneling

- Tunneling: IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers



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
flow: X
src: A
dest: F
src:B
dest: E
Flow: X
Src: A
IPv4 header
IPv6 header
Payload
src:B
dest: E
Flow: X
Src: A
flow: X
src: A
dest: F
IPv6 header
Payload
IPv6 header
Payload
IPv6
host
Tunnel
IPv6
host
IPv4 region
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6