# IP DATAGRAMS AND DATAGRAM FORWARDING

# Virtual Packets

- The goal of internetworking is to provide a packet communication system that allows a program running on one computer to send data to a program running on another computer
- Application programs can send and receive data without knowing
  - The details of local network to which a computer connects
  - The details of remote network to which the destination connects
  - The details of interconnection between the two
- In general,
  - The source host
    - Creates a packet
    - Places the destination address in the packet header
    - Gives it to the local network to forward it to the destination host
  - The local network
    - Sends the packet to a nearby router

- The router
  - When receives a packet, it uses the destination address to select the next router on the path to the destination
  - Transmit the packet to this next router
- The destination network
  - When receives a packet, it delivers it to the destination host

- Note that, routers
  - May connect heterogeneous networks
  - Can not transfer a copy of a frame from one type of network to another because the frame formats differ
  - Can not simply reformat the frame header because the two networks may use incompatible address formats
- To overcome heterogeneity, the internet protocol software must define an internet packet format that is independent of the underlying hardware, where
  - Protocol software creates and handles these internet packets (*virtual*)
  - The underlying hardware does not understand or recognize the internet packet format

  - Each host, or router, in the internet contains protocol software that understands internet packets (*universal*)
- The result is a *universal, virtual* packet that can be transferred across the underlying hardware intact format
- TCP/IP protocols use the name *IP datagram* to refer to an internet packet

# The IP Datagram

- An IP datagram has the same general format as a hardware frame; both of them begin with a header followed by a data area
- The size of a datagram is determined by the size of the data which is sent by the application
- Allowing the size of datagrams to vary makes IP adaptable to a variety of applications
- In the current version of IP protocol (IPv4):
  - The datagram header contains information to route the datagram across the internet, i.e., the source and destination IP addresses
  - A datagram is variable in size; at most 64K bytes, including the header
  - In most datagrams, the header (overhead) is much smaller than the data area
- *Compare a hardware frame with an IP datagram, based on the above IPv4 characteristics*

# The IP Datagram Header Format

<table>
<tr><th>0</th><th>4</th><th>8</th><th>16</th><th>19</th><th>24 31</th></tr>
<tr><td>VERS</td><td>H. LEN</td><td>SERVICE TYPE</td><td colspan="3">TOTAL LENGTH</td></tr>
<tr><td colspan="3">IDENTIFICATION</td><td>FLAGS</td><td colspan="2">FRAGMENT OFFSET</td></tr>
<tr><td colspan="2">TIME TO LIVE</td><td>TYPE</td><td colspan="3">HEADER CHECKSUM</td></tr>
<tr><td colspan="6">SOURCE IP ADDRESS</td></tr>
<tr><td colspan="6">DESTINATION IP ADDRESS</td></tr>
<tr><td colspan="5">IP OPTIONS (MAY BE OMITTED)</td><td>PADDING</td></tr>
<tr><td colspan="6">BEGINNING OF DATA<br>⋮</td></tr>
</table>

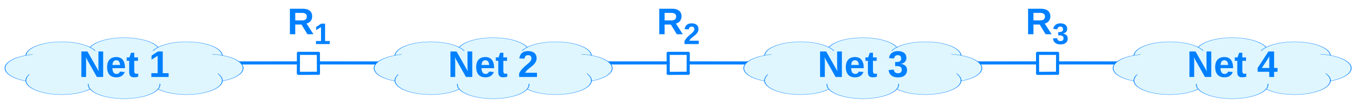
- VERS: Protocol version number (the current version is 4)
- H.LEN: Header length (multiple of 32-bit words)
- SERVICE TYPE: Specifies whether the sender prefers the datagram to be sent through a route with a minimal delay or a route with a maximal throughput, if possible

- TOTAL LENGTH: Specifies the total number of bytes in the datagram, including the header
- IDENTIFICATION, FLAGS, and FRAGMENT OFFSET: will be explained later
- TIME TO LIVE: Specifies the maximum number of routers to be visited before reaching the destination; to prevent a datagram from traveling forever around a path that contains a loop, due to software malfunction or when a network manager misconfigures routes
- TYPE: Specifies the type of data and determines which transport protocol should receive the datagram
- HEADER CHECKSUM: ensures that bits of the header are not changed in transit; *Why is not the checksum calculated for the whole IP Datagram?*
- SOURCE IP ADDRESS
- DESTINATION IP ADDRESS

# Forwarding An IP Datagram

- Performed by routers
- Similar to WAN forwarding
  - Table-driven
  - Entry specifies next hop
- Unlike WAN forwarding
  - Uses IP addresses
  - Next-hop may be a router IP address, not an interface number

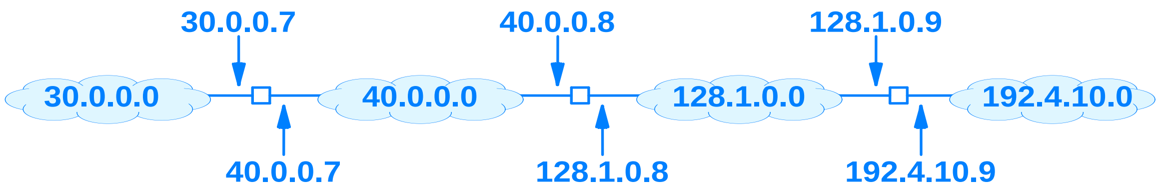



| Destination | Next Hop |
|---|---|
| net 1 | $R_1$ |
| net 2 | deliver direct |
| net 3 | deliver direct |
| net 4 | $R_3$ |

*R2 table*

# IP Addresses and Routing Table Entries

- Routing table entries include
  - Destination field: The destination network IP address
  - Mask field: specifies which bits of the destination field correspond to the network address
  - Next Hop field: an IP address denotes a router or a host
- Note that: although the same host suffix has been assigned to both routers ports, IP does not require uniformity, and a network administrator is free to assign different values to each port




| Destination | Mask | Next Hop |
|---|---|---|
| 30.0.0.0 | 255.0.0.0 | 40.0.0.7 |
| 40.0.0.0 | 255.0.0.0 | deliver direct |
| 128.1.0.0 | 255.255.0.0 | deliver direct |
| 192.4.10.0 | 255.255.255.0 | 128.1.0.9 |

*The routing table found in the central router*

- Datagram forwarding for a given datagram
  - Extract destination address filed, D
  - Look up D in routing table
  - Find next hop address, N
    The computation to examine the $i^{th}$ entry in the table can be expressed as follow:
    $$\text{IF } (Mask[i]\&D == Destination[i])$$
    $$N = Next\ Hop[i]$$
  - Send datagram to N
- Note that:
  - The destination address in a datagram header always refers to the ultimate destination
  - When a router forwards the datagram to another router, the address of the next hop does not appear in the datagram header
  - Because each destination in a routing table corresponds to a network, the number of entries in a routing table is proportional to the number of networks in an internet

# IP ENCAPSULATION, FRAGMENTATION, AND REASSEMBLY

## Datagram Transmission and Frames

- Internet transmission paradigm:
  - When a host wants to send data to another host, it
    - Forms a datagram and includes in it
      - Destination address
      - The data wanted to be send
    - Sends this datagram to a nearby router
  - When an intermediate router receives a datagram, it
    - Forwards it to the next router
  - When a final router receives a datagram, it
    - Delivers it to the destination host
- A datagram travels across various networks
- Network hardware does not understand datagram format or internet addressing

- Instead, each hardware technology defines its own
  - Frame format
  - Physical address
- A hardware
  - Accepts and delivers frames that stick with its own specified frame format
  - Uses its own specified hardware addressing scheme
- Because an internet may contain heterogeneous network technology, the frame format which needs to cross a network may differ from the frame format of the previous network
- Now, the question is:
  How can a datagram be transmitted across a physical network that does not understand the datagram format?
- The answer lies in a technique known as *encapsulation*

# IP Datagram Encapsulation

- When an IP datagram is encapsulated in a frame, the entire datagram is placed in the data area of this frame

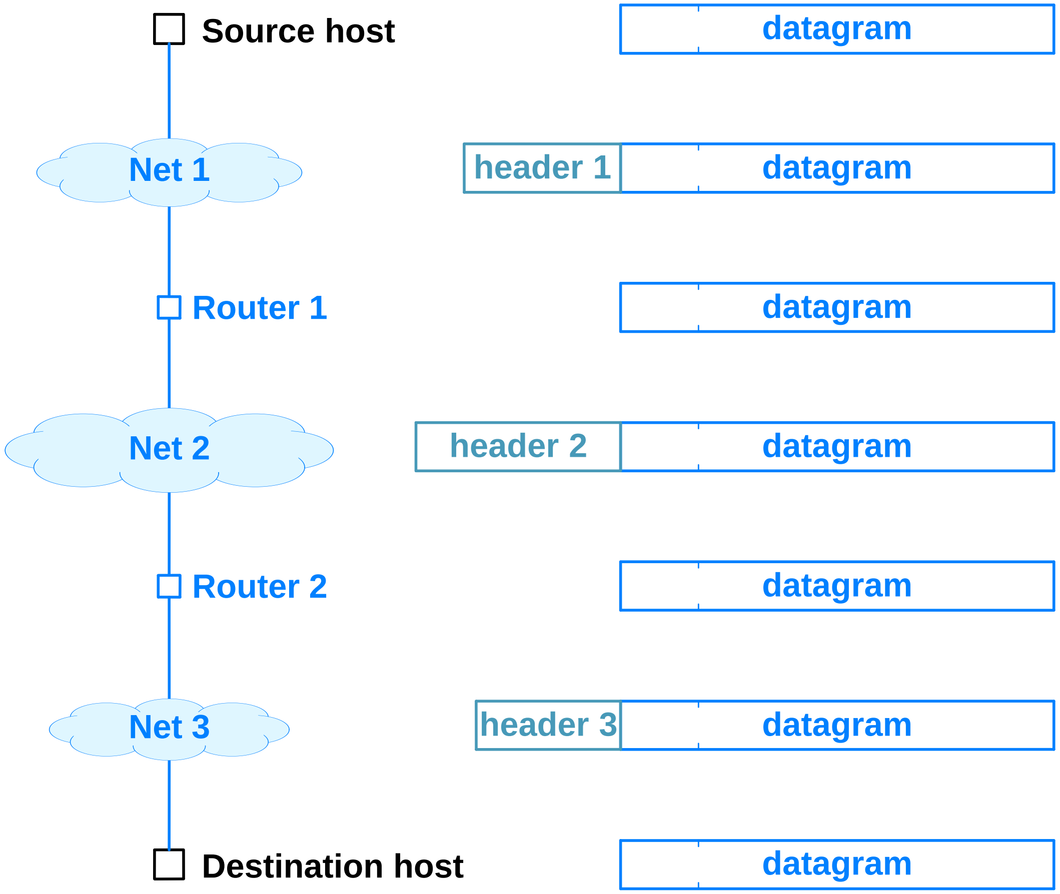

- The network hardware treats a frame that contains a datagram exactly like any other frame
- The hardware does not examine or change the contents of the frame data area
- To identify that the data area contains an IP datagram, or any other data, the sender and receiver must agree on the value used in the frame type field
- The encapsulation process requires a sender (a host or a router) to supply the physical address of the next computer to which the datagram should be sent

- The sender may perform address binding to get the next computer's physical address; for more details about *address binding*, see the *Address Resolution Protocol (ARP)* lecture
- To conclude:
  - The datagram address is the IP address of the ultimate destination
  - A datagram is encapsulated in a frame for transmission across a physical network
  - The frame address is the physical address of the next hop
  - Address binding is needed to translate the next hop IP address to an equivalent hardware address

## Transmission Across an Internet

- Encapsulation is applied to one transmission at a time
- After the sender selects a next hop, the sender encapsulates the datagram in a frame and transmits the result across the physical network to the next hop
- When the frame reaches the next hop, the receiving software removes the IP datagram and discards the frame
- If the datagram must be forwarded across another network, a new frame is created
- Since each network can use a different hardware technology than the others, the frame formats can differ from one network to another as well
- Frame headers do not accumulate during a trip through the internet
- When a datagram reaches its final destination, it appears exactly the same as when it was originally sent

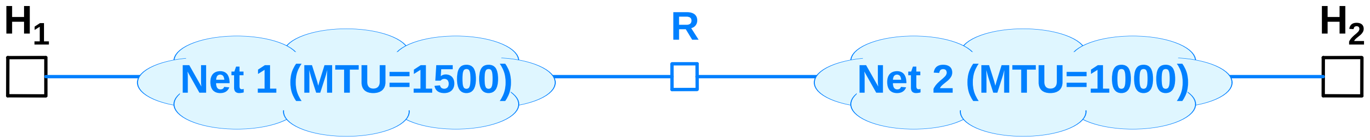



Whenever a datagram travels across a physical network, it is encapsulated in a frame appropriate to the network

- To conclude:
  - A datagram survives the entire trip across the internet
  - A frame survives only one hop

# Datagram Size and Encapsulation

- Each hardware technology specifies the maximum amount of data that a frame can carry; this amount is called *maximum transmission unit* (MTU)
- There is no exception to the MTU limit; the network hardware is not designed to accept frames that carry more data than the MTU allows
- Thus, a datagram must be smaller than, or equal to, the network MTU, or it can not be encapsulated for transmission
- In an internet that contains heterogeneous networks, MTU restrictions can cause a problem




A router can receive a datagram over one network that can not be sent over another

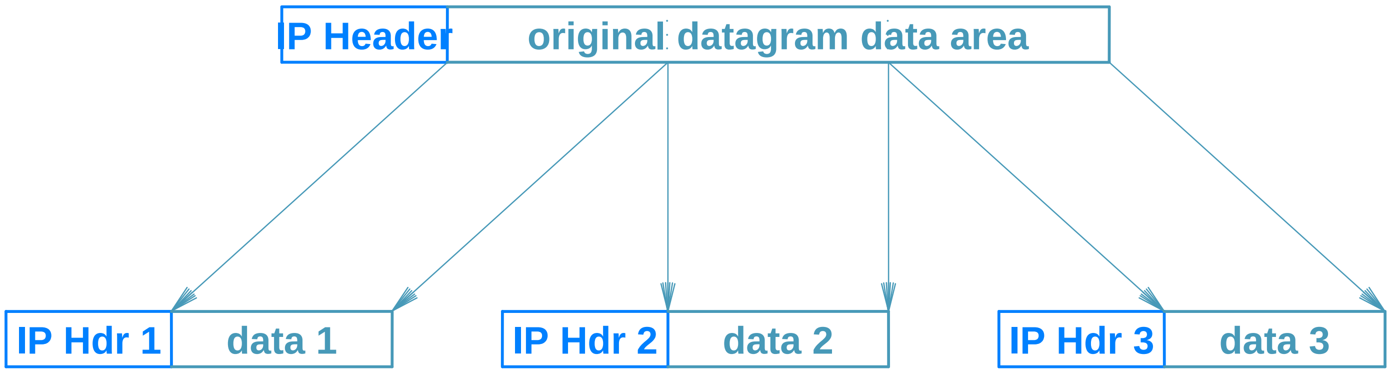
# Datagram Fragmentation

- An IP router uses a technique called *fragmentation* to solve the problem of heterogeneous MTUs
- When a datagram is larger than the MTU of the network over which it must be sent, the router divides the datagram into smaller pieces called *fragment*, and sends each fragment independently
- A fragment has the same format as other datagrams

0 4 8 16 19 24 31

<table>
<tr><td>VERS</td><td>H. LEN</td><td>SERVICE TYPE</td><td colspan="3">TOTAL LENGTH</td></tr>
<tr><td colspan="3">IDENTIFICATION</td><td>FLAGS</td><td colspan="2">FRAGMENT OFFSET</td></tr>
<tr><td colspan="2">TIME TO LIVE</td><td>TYPE</td><td colspan="3">HEADER CHECKSUM</td></tr>
<tr><td colspan="6">SOURCE IP ADDRESS</td></tr>
<tr><td colspan="6">DESTINATION IP ADDRESS</td></tr>
<tr><td colspan="5">IP OPTIONS (MAY BE OMITTED)</td><td>PADDING</td></tr>
<tr><td colspan="6">BEGINNING OF DATA<br>⋮</td></tr>
</table>

- There is a bit in the *FLAGS* field of the header indicates whether this IP datagram carries the last part of the data or not, i.e., it indicates whether there is more fragment to come or not
- The *FRAGMENT OFFSET* field in the header of a fragment specifies where in the original datagram the segment belongs
- The *FRAGMENT OFFSET* is measured in 8-byte units; *WHY?*
- To fragment a datagram for transmission across a network, a router uses the network MTU and the datagram header size to calculate
  - The maximum amount of data that can be sent in each fragment
  - The number of fragments that will be needed



- Example: Consider the following IP datagram:

```
          1         2         3
01234567890123456789012345678901
```

| | | | |
|---|---|---|---|
| Ver=4 | H.L=5 | Type of service | Total Length = 40 |
| Identification | | Flg=0 | Fragment Offset = 0 |
| Time to live=7 | Protocol | | Header Checksum |
| Source IP address | | | |
| Destination IP address | | | |
| Data1 | Data2 | Data3 | Data4 |
| Data5 | Data6 | Data7 | Data8 |
| Data9 | Data10 | Data11 | Data12 |
| Data13 | Data14 | Data15 | Data16 |
| Data17 | Data18 | Data19 | Data20 |

If the MTU = 38, this datagram maybe fragmented as follows:

```
          1         2         3
01234567890123456789012345678901
```

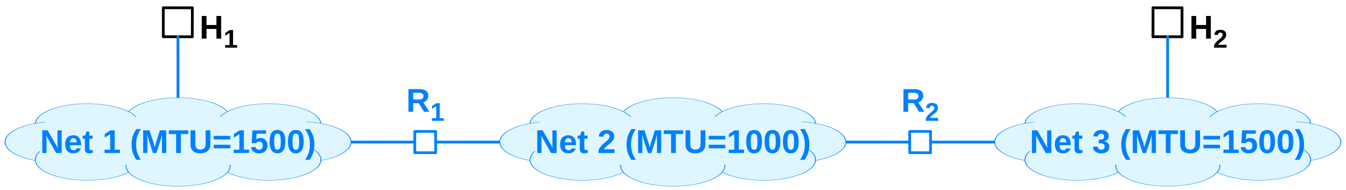
| | | | |
|---|---|---|---|
| Ver=4 | H.L=5 | Type of service | Total Length = 36 |
| Identification | | Flg=1 | Fragment Offset = 0 |
| Time to live=6 | Protocol | | Header Checksum (recalc.) |
| Source IP address | | | |
| Destination IP address | | | |
| Data1 | Data2 | Data3 | Data4 |
| Data5 | Data6 | Data7 | Data8 |
| Data9 | Data10 | Data11 | Data12 |
| Data13 | Data14 | Data15 | Data16 |

```
          1         2         3
01234567890123456789012345678901
```

| | | | |
|---|---|---|---|
| Ver=4 | H.L=5 | Type of service | Total Length = 24 |
| Identification | | Flg=0 | Fragment Offset = 2 |
| Time to live=6 | Protocol | | Header Checksum (recalc.) |
| Source IP address | | | |
| Destination IP address | | | |
| Data17 | Data18 | Data19 | Data20 |

# Datagram Reassembly

- The process of creating a copy of the original datagram from fragments is called *reassembly*
- Using the $FLAGS$ field, a receiver can tell whether all fragments have arrived successfully or not
- The IPv4 protocol specifies that the ultimate destination host, not the internal routers, should reassembly fragments




- The advantages of requiring the ultimate destination to reassemble fragments are:
  - Reduces the complexity of routers
  - Allows routers to change dynamically; i.e., IP is free to pass some fragments from a datagram along a different route than other fragment

## Identifying Datagram

- Individual fragments can be lost or arrive out of order
- If a source sends multiple datagrams to a given destination, the fragments from these datagrams can arrive in arbitrary order
- To overcome this problem,
  - A sender places a unique identification number in the *IDENTIFICATION* field of each outgoing datagram
  - The router copies the identification number into each fragment
  - A receiver uses
    - The identification number and IP source address in an incoming fragment to determine the datagram to which the fragment belongs
    - The *FRAGMENT OFFSET* field to order fragments within a given datagram

# Fragment Loss

- An underlying network may drop an encapsulated datagram, or fragment
- When the first fragment arrives from a given datagram, the receiver starts a timer
- If all fragments of a datagram arrive before the timer expires, the receiver cancels the timer and reassembles the datagram
- However, if the timer expires before all fragments arrive, the receiver discards those fragments that have arrived
- Note that, in case of discarding expired fragment, the receiver does not send to the sender a request for the missing parts, because:
  - The sender does not know any thing about the fragmentation process
- To conclude, IP's reassembly timer is all-or nothing process

# Fragmenting A Fragment

- After performing fragmentation, a router forwards each fragment to its destination
- If a fragment eventually reaches another network that has a smaller MTU, the fragment will be divided into smaller fragments
- IP does not distinguish between original fragments and sub-fragments
- In particular, a receiver can not know whether an incoming fragment was the result of one router fragmenting a datagram, or multiple routers fragmenting fragments
- Advantage: A receiver can perform reassembly of the original datagram without first reassembling sub-fragments, and hence
  - Saves the CPU time
  - Reduces the amount of information needed in the headers of each fragment

– Example: In the last example, if the first fragment goes through another router which has an MTU less than 36, it will be re-fragmented again as follows:

```
          1         2         3
01234567890123456789012345678901
```

| Ver=4 | H.L=5 | Type of service | Total Length = 28 | |
|---|---|---|---|---|
| Identification | | | Flg=1 | Fragment Offset = 0 |
| Time to live=5 | | Protocol | Header Checksum (recalc.) | |
| Source IP address | | | | |
| Destination IP address | | | | |
| Data1 | Data2 | Data3 | Data4 | |
| Data5 | Data6 | Data7 | Data8 | |

```
          1         2         3
01234567890123456789012345678901
```

| Ver=4 | H.L=5 | Type of service | Total Length = 28 | |
|---|---|---|---|---|
| Identification | | | Flg=1 | Fragment Offset = 1 |
| Time to live=5 | | Protocol | Header Checksum (recalc.) | |
| Source IP address | | | | |
| Destination IP address | | | | |
| Data9 | Data10 | Data11 | Data12 | |
| Data13 | Data14 | Data15 | Data16 | |

– Note that:

- The Flag in both fragments is 1
- The time to live field has been reduced once more
- The header checksum has been recalculated once more
- The fragment offset has been adjusted relative to the original IP datagram
- I did not mention the second fragment in the last example, since its size is Ok.

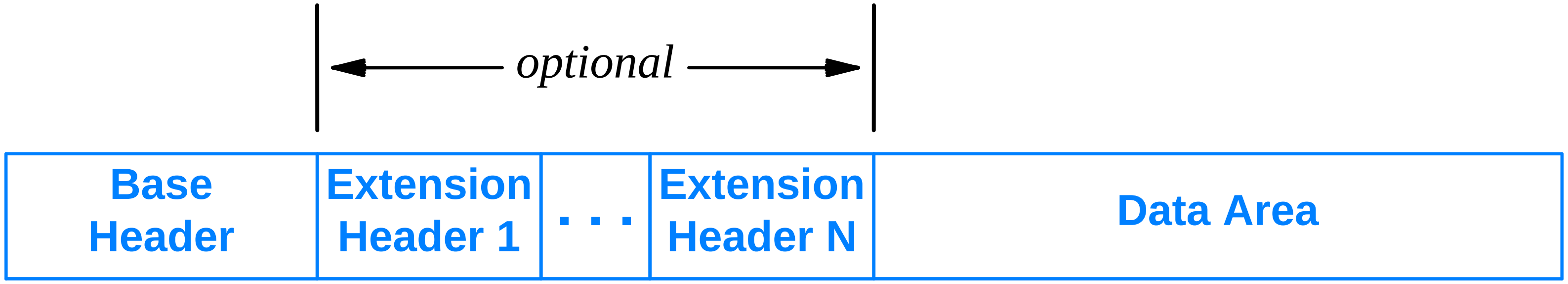
# THE FUTURE IP (IPV6)

# Characterization of Features in IPv6

- IPv6 retains many design features that have made IPv4 so successful, e.g.,
  - Connectionless
  - Specifying in the datagram header the maximum number of hops the datagram can take before discarded
  - Options facility
- Despite retaining the basic concepts from the current IP version (i.e., IPv4), IPv6 changes all the details
  - Instead of 32 bits, each IPv6 address contains 128 bits;
    The resulting address space is large enough to accommodate continued growth of the world-wide internet for many decades
  - IPv6 uses an entirely new datagram header
  - IPv6 includes a mechanism that allows a sender and receiver to establish a high-quality low-cost path through the underlying network and to associate datagrams with the path
  - The designers have provided a scheme that allows a sender to add additional information to a datagram, i.e., new features can be added to the design as needed

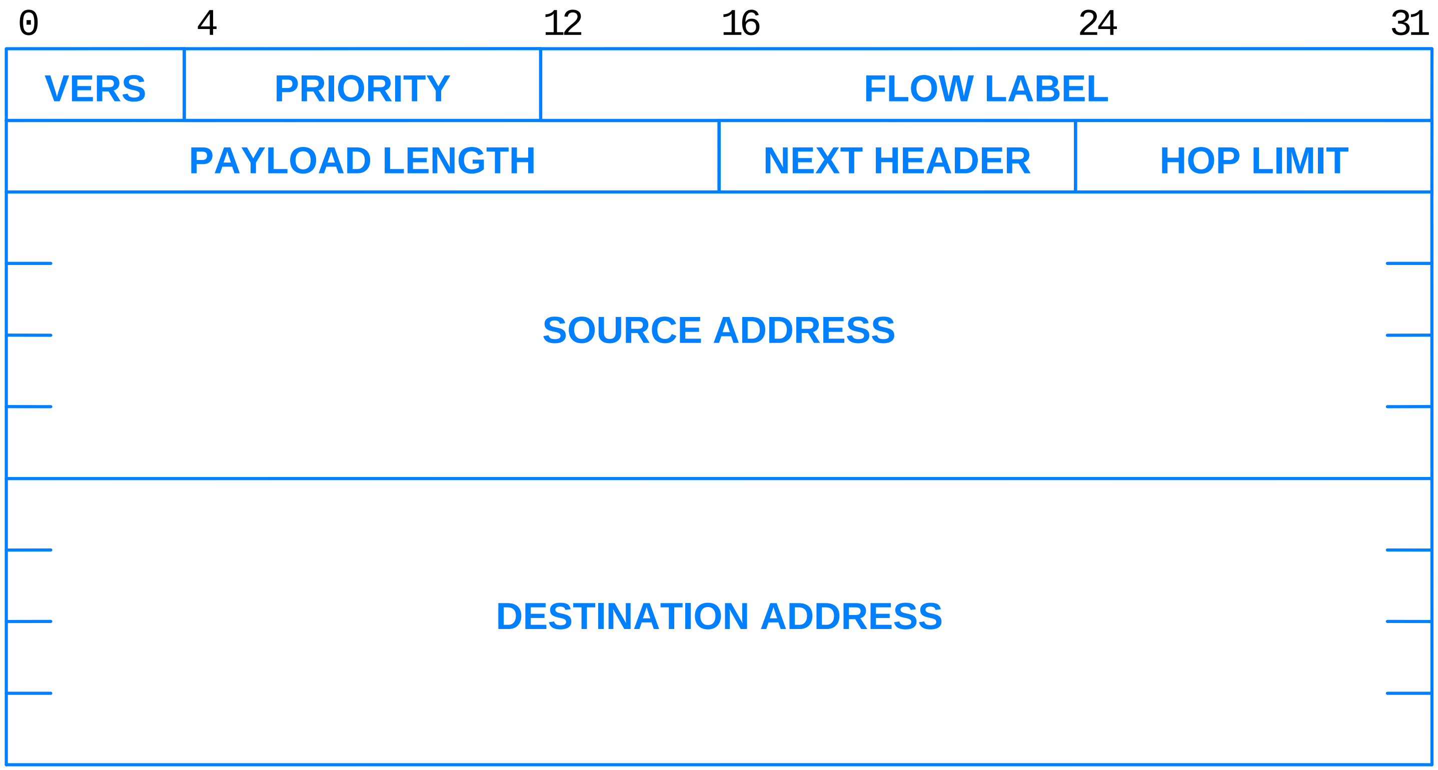
 

# IPv6 Datagram Format



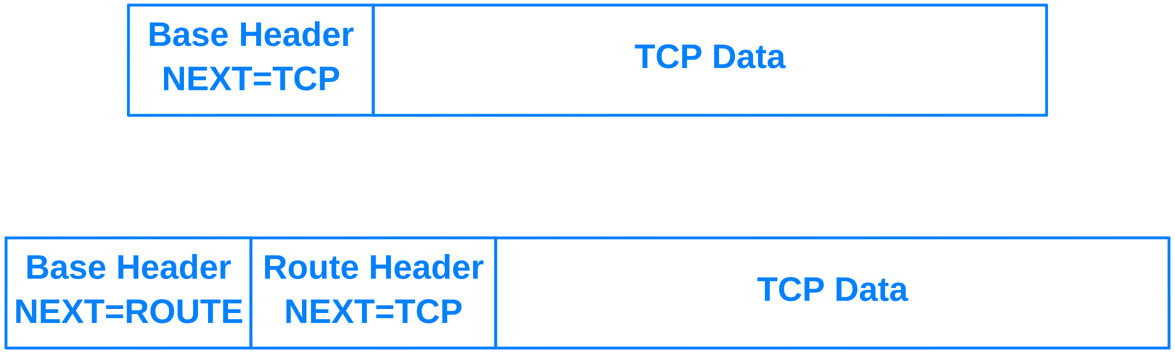

- An IP datagram
  - Begins with a base header
  - Followed by zero, or more, extension headers (optional)
  - Followed by data
- Note that, this IP datagram illustration is not drawn to scale

# IPv6 Base Header Format




- An IPv6 base datagram header is 10 words
- Although the IPv6 base datagram header is twice as large as an IPv4 datagram header, it contains less information
  - Most of the space in the header is devoted to two fields that identify the sender and the recipient

- The other fields include
  - *VERS*: To identify the protocol as version 6
  - *PRIORITY*: To specify the routing priority class
  - *PAYLOAD LENGTH*: To specify the size of the data being carried
  - *HOP LIMIT*: Corresponding to the IPv4 TIME-TO-LIVE field
  - *FLOW LABEL*: Intended for use with new applications that require delay performance guarantee
  - *NEXT HEADER*: To specify the type of information that follows the current header (i.e., the extension header), if any

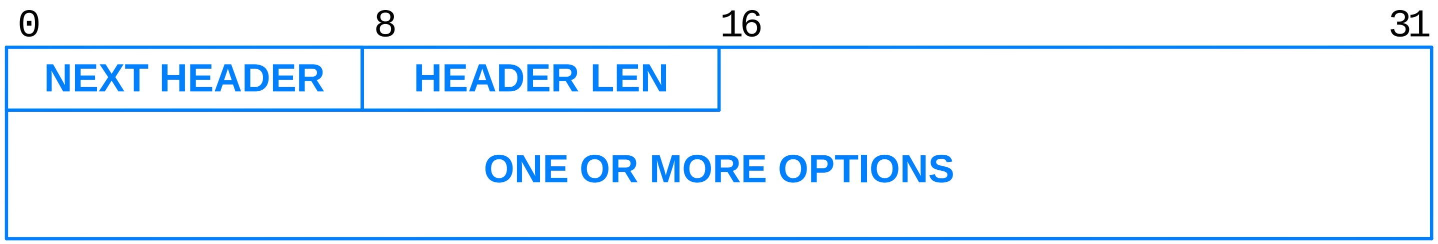



Two IPv6 datagrams: without and with extended header

- Extension headers include
  - *Routing*: Full or partial route to follow
  - *Fragmentation*: Management of datagram fragments
  - *Authentication*: Verification of the sender's identity
  - *Encrypted security payload*: Information about the encrypted contents
- The reason for having multiple headers include:
  - Economy reason
    * Use only whatever you need
  - Extensibility reason
    * The ability to add features without redesigning the protocol
- Since most of the communication lines do not produce that much transmission errors, the *header checksum* field is gone, because calculating it greatly reduces the performance

# How IPv6 Handles Multiple Headers

- Because the standard specifies a unique value for each possible header type, there is never ambiguity about the interpretation of the *NEXT HEADER* filed
- A receiver uses the *NEXT HEADER* filed in each header to determine what follows
- If the option extension header length is not fixed, *HEADER LEN* field is used to identify the exact length of the option



# Fragmentation and Reassembly

- Like IPv4
  - A prefix of the original datagram is copied into each fragment
  - The payload length is modified to be the length of the fragment
- Unlike IPv4
  - Fields for fragmentation information are not included in the base header, *Why?*
  - These fragmentation information is placed is a separate fragment extension header
  - The presence of this fragmentation header identifies the datagram as a fragment

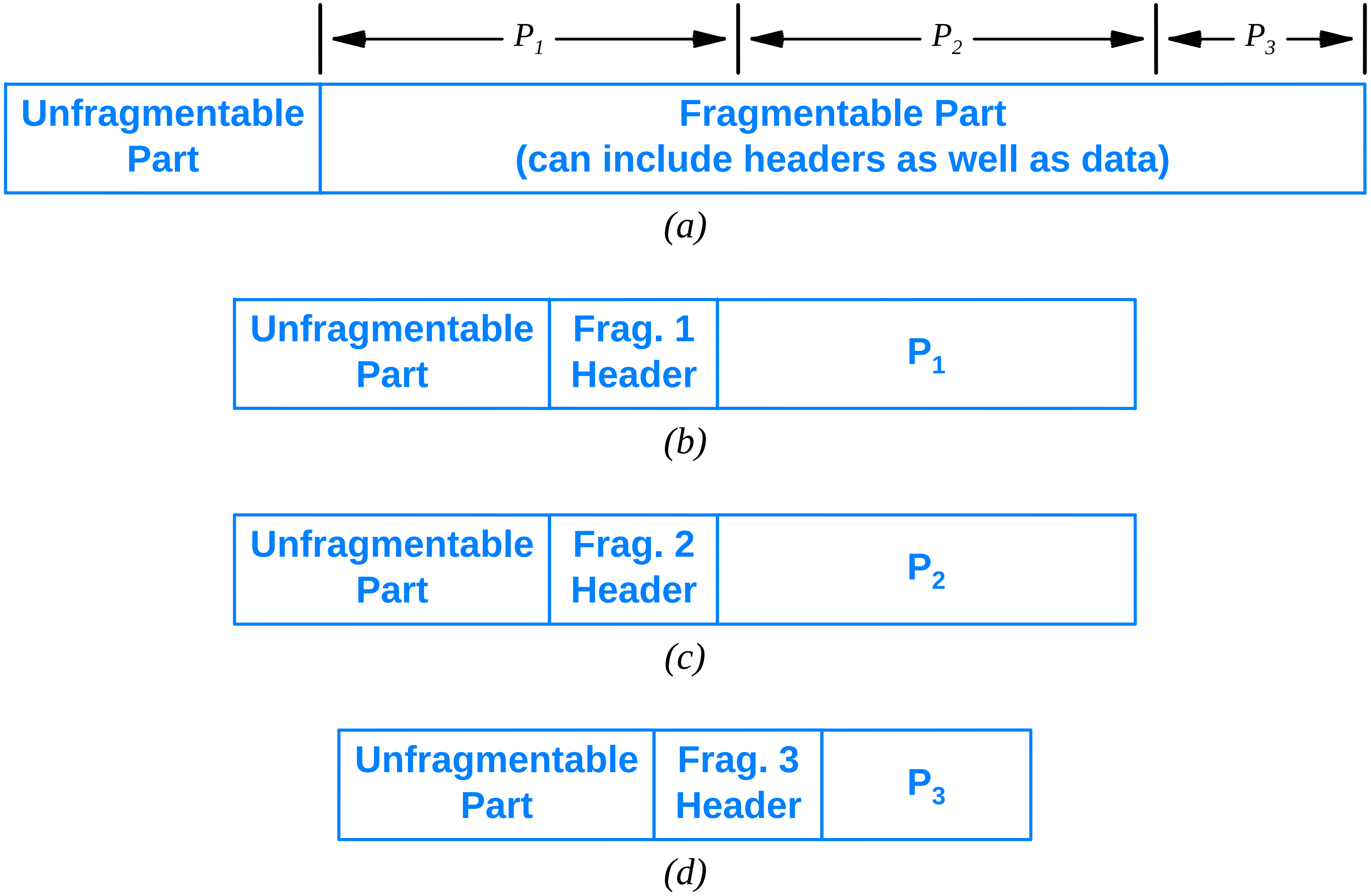



Illustration of fragmentation

Each fragment begins with a copy of the unfragmentable part (i.e., base header) and a fragment extension header

# Path MTU

- In IPv4, a router performs fragmentation when it receives a datagram that is too large for the network over which the datagram must be sent
- In IPv6, a sending host is responsible for fragmentation
- To do so, the host must
  - Learn the MTU of each network along the path to the destination
  - Choose a datagram size to fit the smallest MTU
- The minimum MTU along a path from a source to a destination is known as the *path MTU*
- The process of learning the path MTU is known as *path MTU discovery*
- For more details about *path MTU* and *path MTU discovery*, see the *Internet Control Message Protocol (ICMP)* lecture

## IPv6 Addressing

- Like IPv4, the address consists of
  - Prefix part, to identify the network
  - Suffix part, to identify a particular computer on the network
- Unlike IPv4, addresses did not define classes, i.e., the boundary between the prefix and suffix can fall anywhere within the address and cannot be determined from the address alone

## IPv6 Destination Addressing Types

- IPv6 does not include a special address for broadcasting on a given network
- the IPv6 destination addressing types include
  - *Unicast*: As in IPv4
  - *Multicast*: As in IPv4
  - *Anycast*:
    - Originally known as a *cluster* addressing
    - The address corresponds to a set of computers
    - A datagram sent to the address is routed along a shortest path and then delivered to exactly one of the computer (any of them)
    - This addressing type is useful when a corporation offers a service over the network
      - An *anycast* address is assigned to several hosts that can provide the service
      - When a user sends a datagram to the anycast address, IPv6 routes the datagram to one of the computers in the set

# IPv6 Colon Hexadecimal Notation

- Although an address that occupies 128 bits can accommodate internet growth, writing such numbers is not that easy; here it is an example 105.220.136.100.255.255.255.255.0.0.18.128.140.10.255.255
- To help reduce the number of characters used to write an address, the designers of IPv6 propose using a more compact syntactic form known as *colon hexadecimal notation* or simply *colon hex*
- In *colon hex*, each group of 16 bits is written in hexadecimal with a colon separating groups, for example, 69DC:8864:FFFF:FFFF:0:1280:8C0A:FFFF
- An additional optimization known as *zero compression*, which replaces sequences of zeroes with two colons, further reduces the size; for example, 2002:0:0:0:0:0:0:357 can be written as 2002::357
- To help ease the transition to the new protocol, the designers mapped existing IPv4 addresses into IPv6 address space, by adding 96 zero bits to the left of the current IPv4, for example, 192.5.48.85 will be 0:0:0:0:0:0:C005:3055, which can be written as ::C005:3055