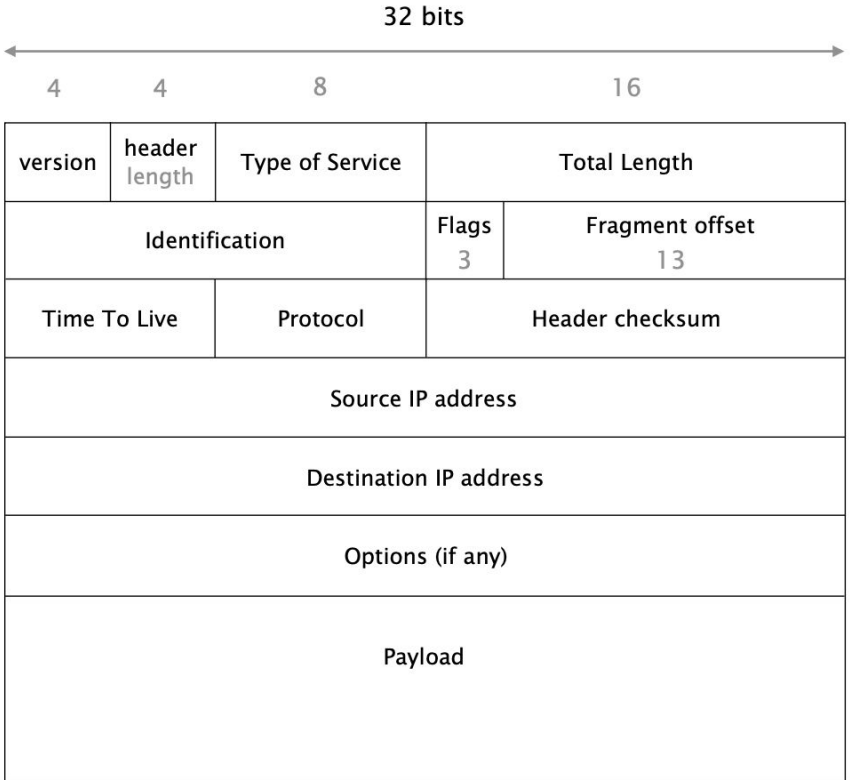
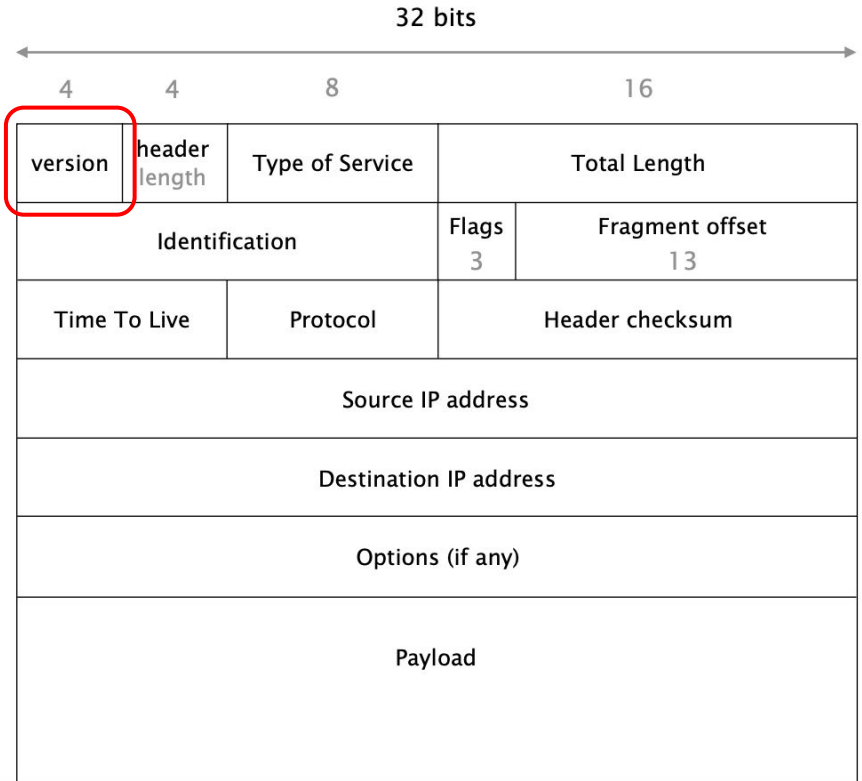
Network (IP) Layer

1. IP addresses
    - use, structure, allocation
2. IP forwarding
    - longest prefix match rule
3. IP header
    - IPv4 and IPv6, wire format

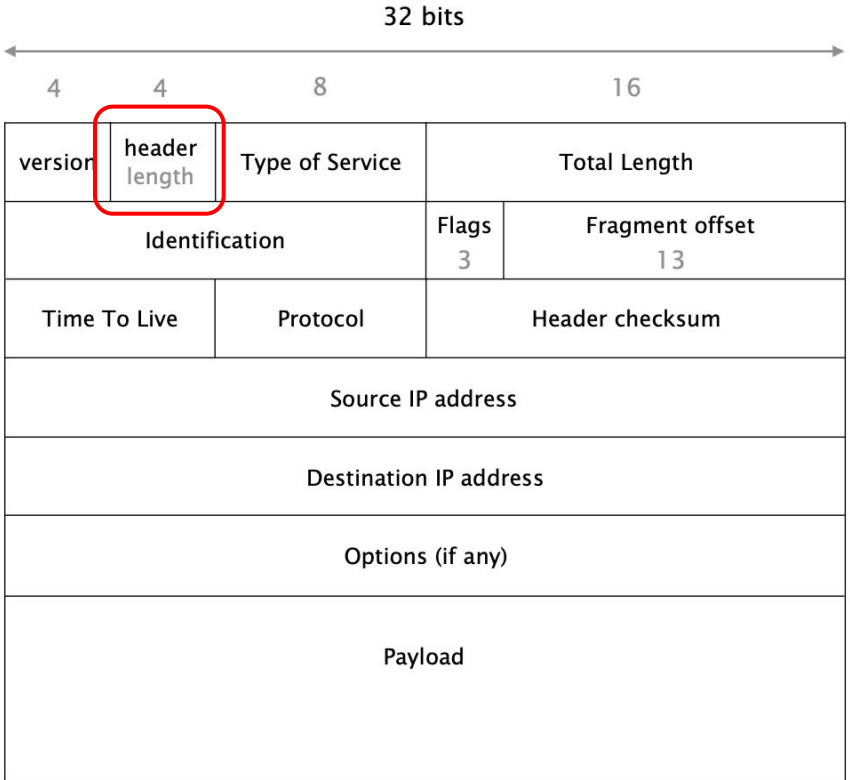
IPv4 Packet Looks Like This



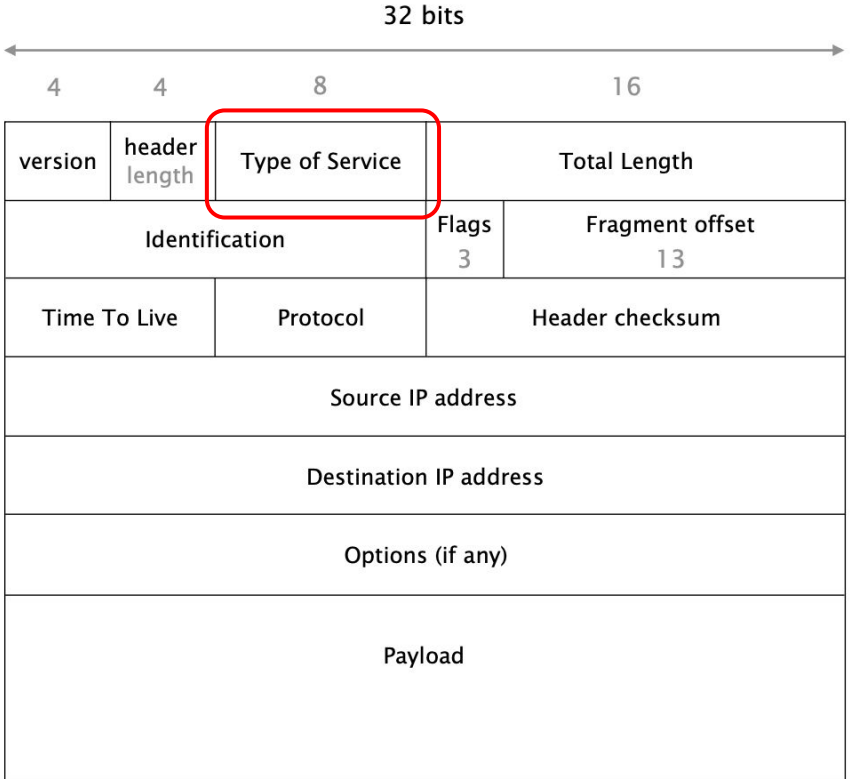
Version Lets Us Know What Fields to Expect (either 4 or 6)



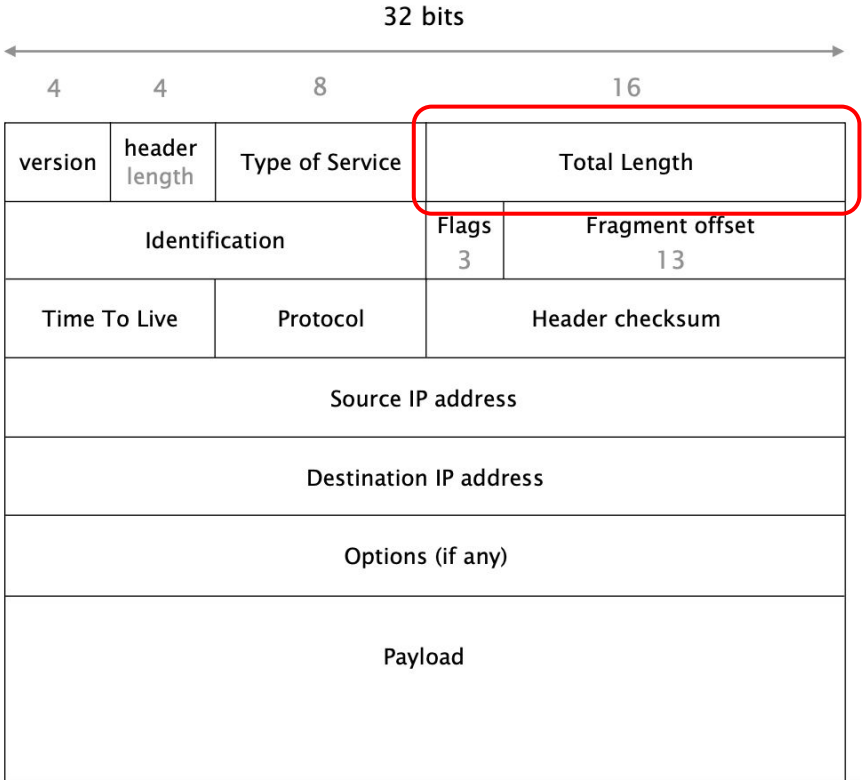
Header Length is the Number of 32-bit Words in the Header (typically 5 for 20 bytes)



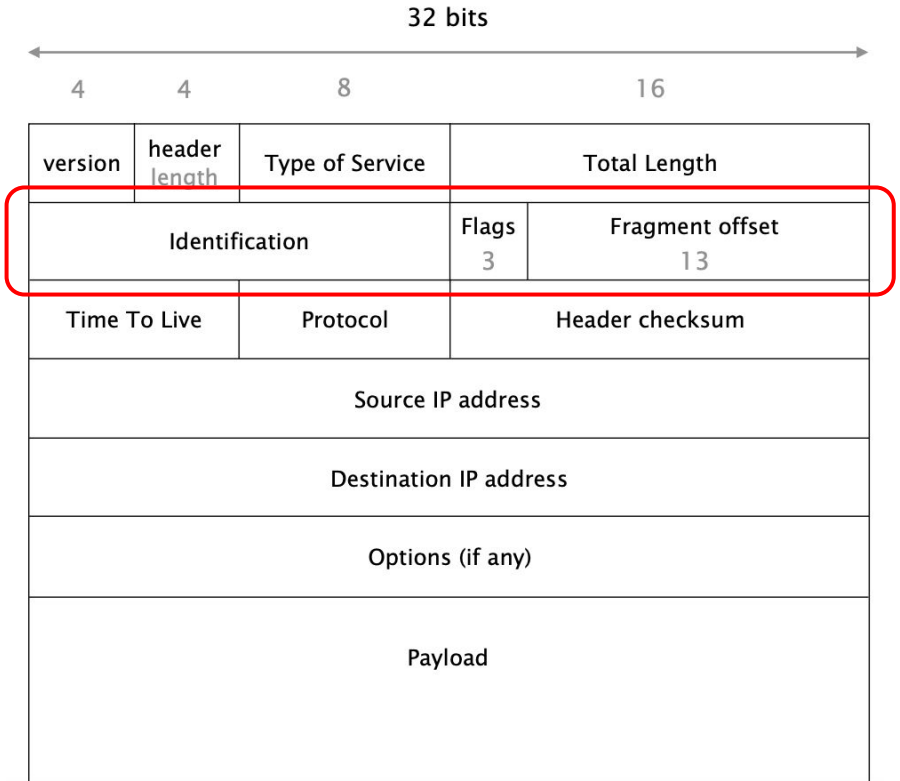
ToS Allows for Different Types of Packets to be Treated Differently (low delay for voice, high bandwidth for video, etc.)



Total Length is Entire Size of Packet (max 65535 bytes)



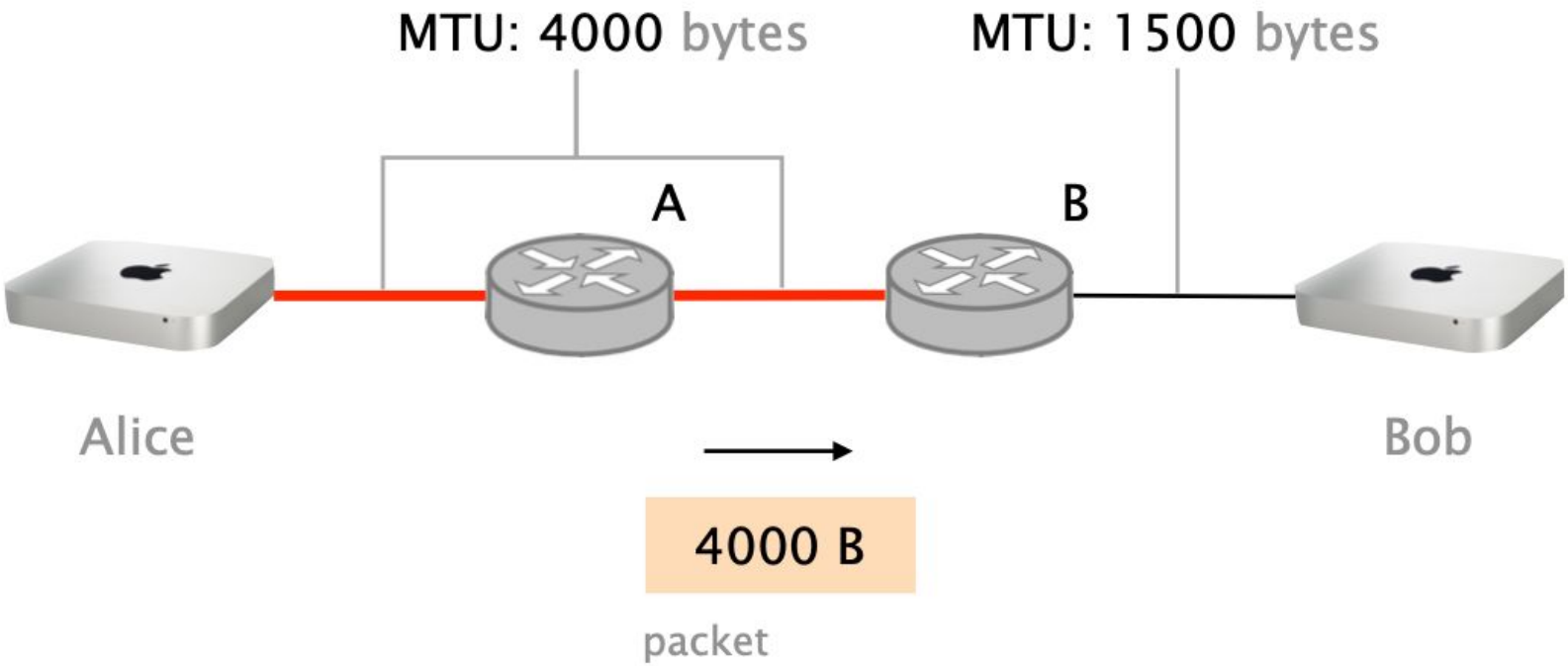
Next Three are About Fragmentation



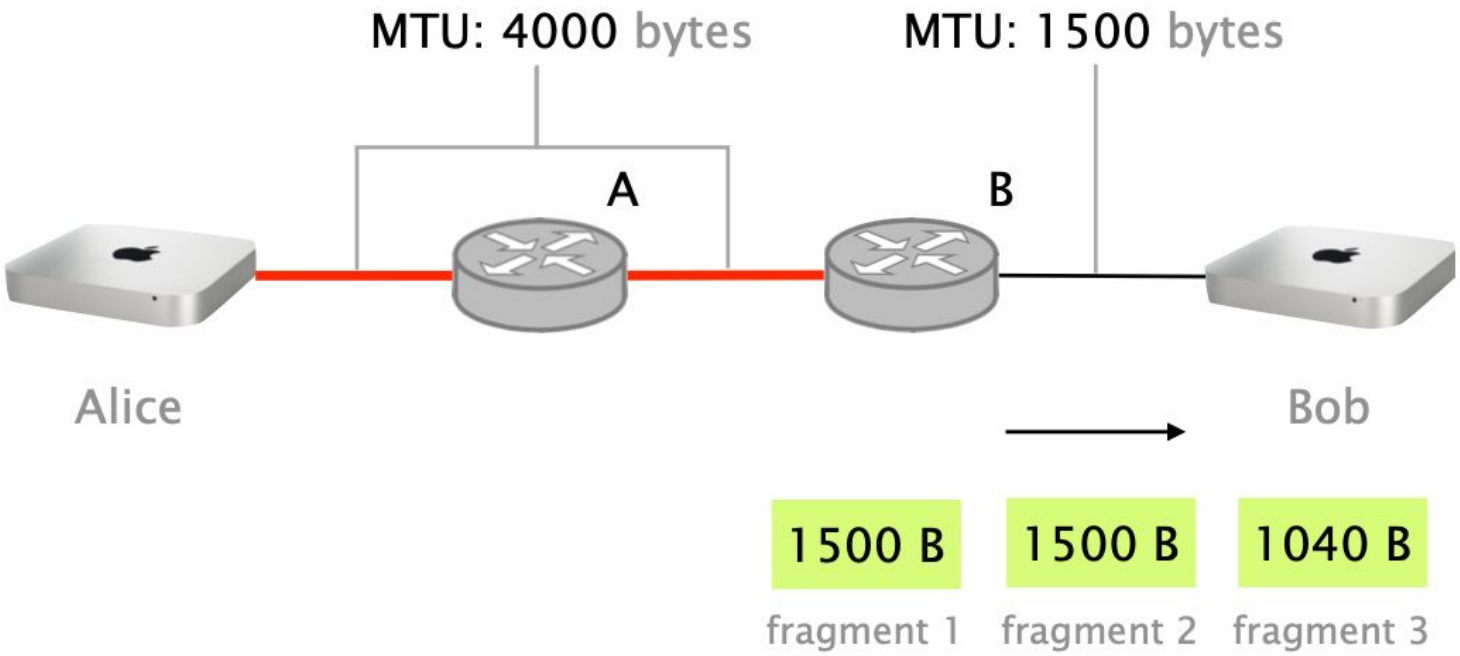
Every Link has a Maximum Transmission Unit (MTU)

- MTU is the max. # of bytes a link can carry as one unit
  - e.g., 1500 bytes for normal Ethernet
- A router can fragment a packet if the outgoing link MTU is smaller than the total packet size
- Fragmented packets are recomposed at the destination

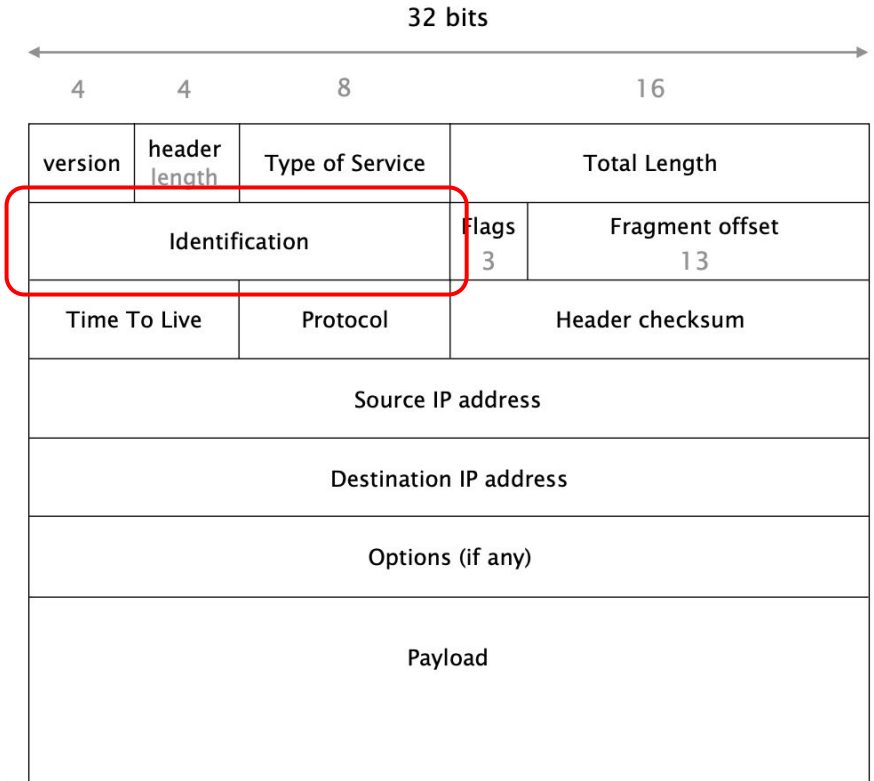
Fragmentation



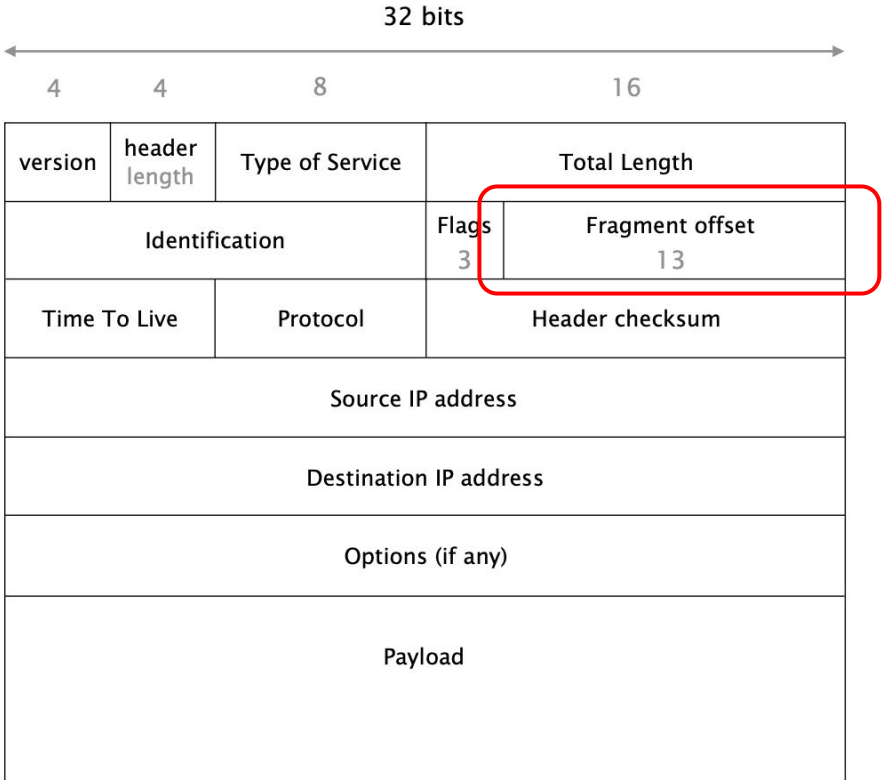
Fragmentation - Packet Larger than the MTU, Router B Fragments Packet



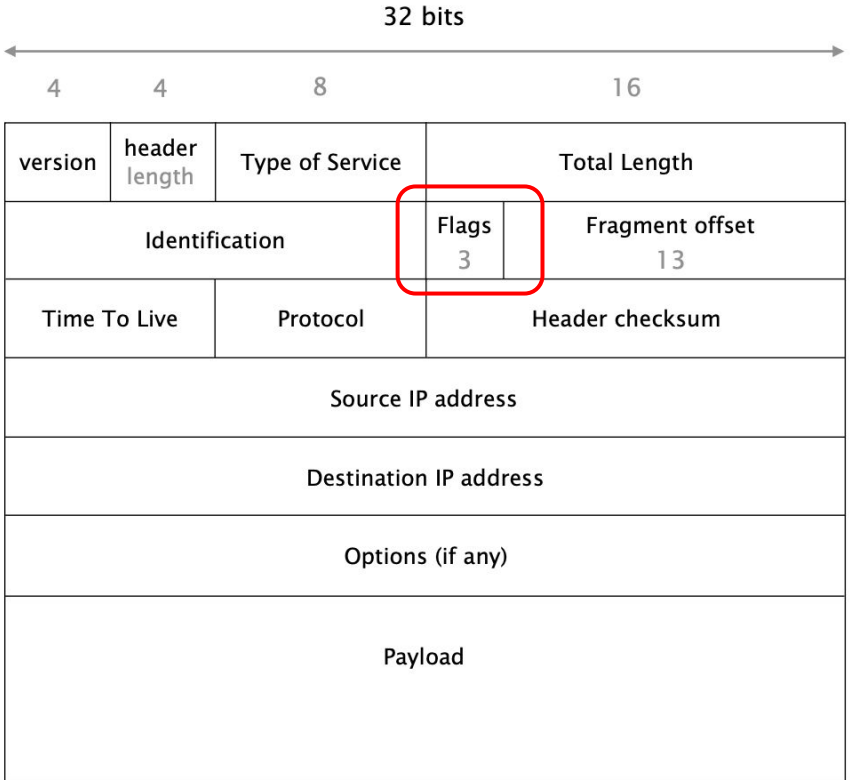
Identification Uniquely Identifies Fragments of a Given Packet



Fragment Offset Used to Put Fragments Back Together in Order



Flags Say Whether More Fragments are Coming

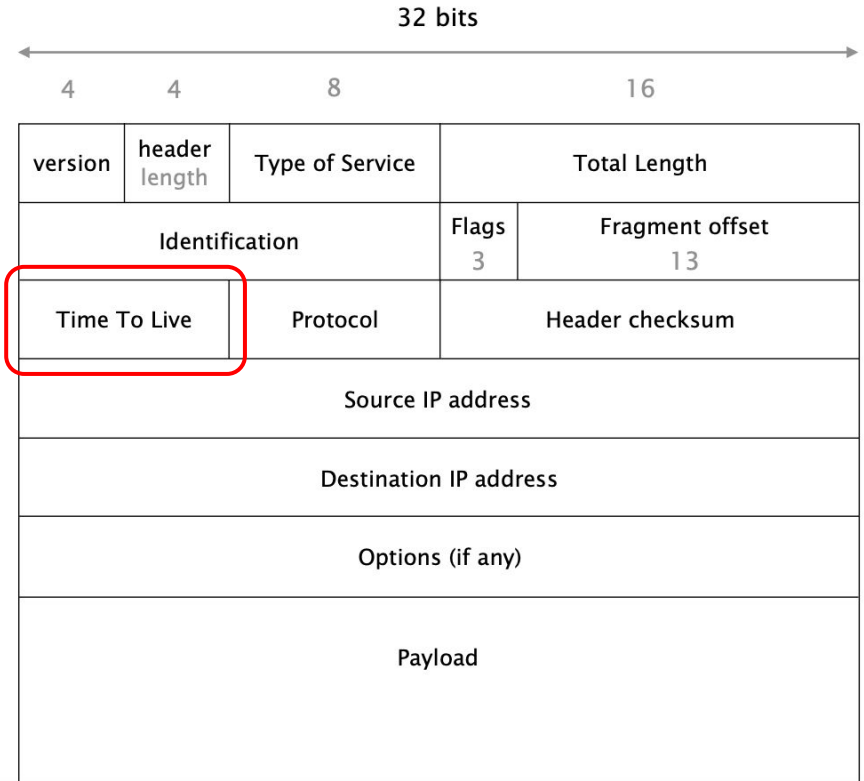


TTL Identifies Looping Packets, Dropped if They Reach Zero

Default TTL values:

Linux/MAC: 64
Windows: 128

Why would this be useful?

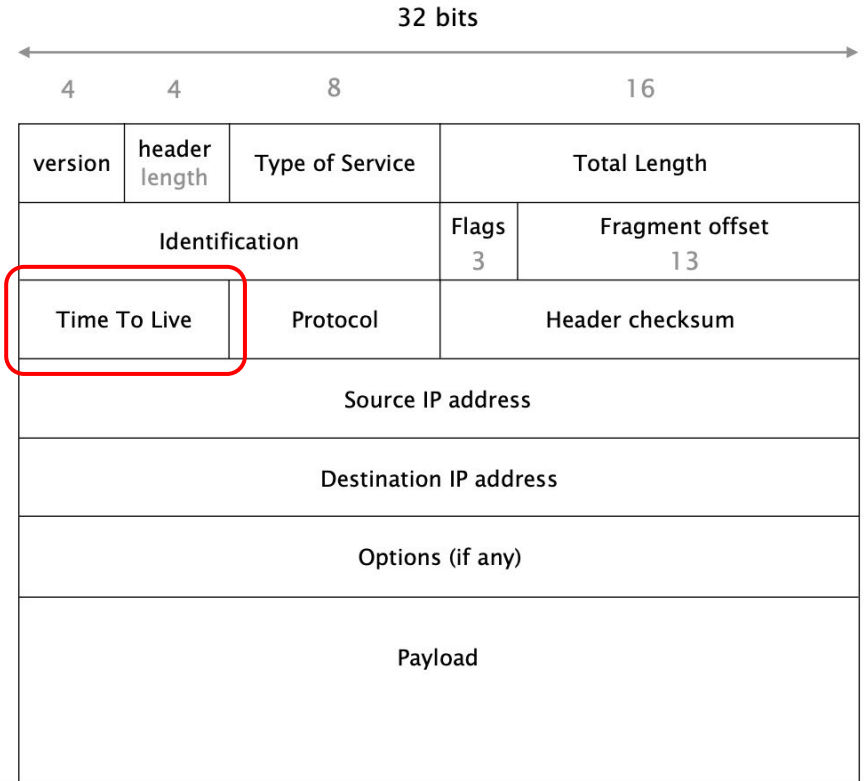


TTL Identifies Looping Packets, Dropped if They Reach Zero

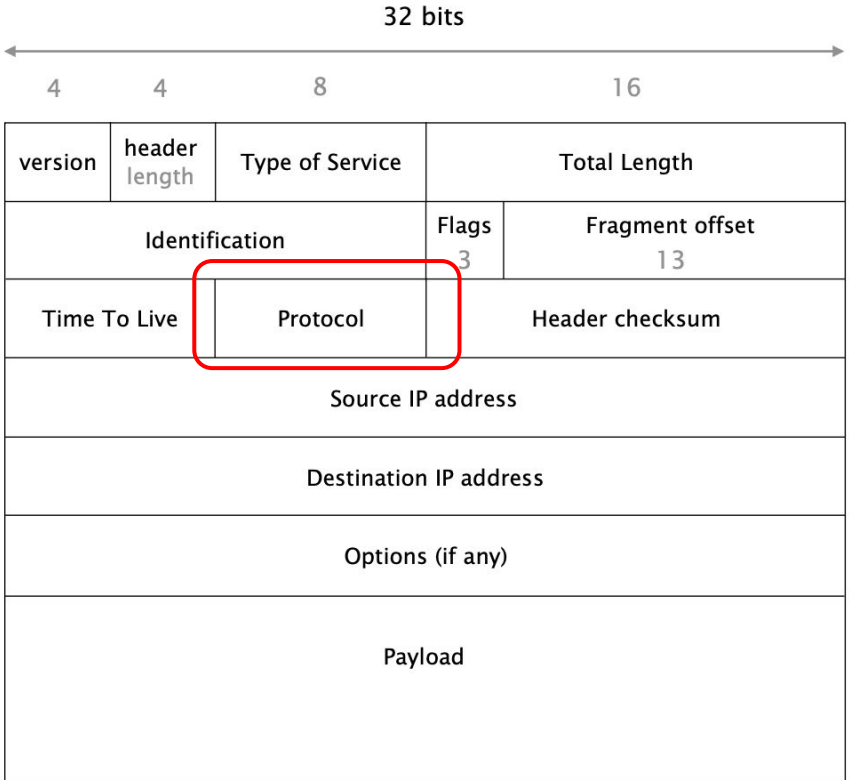
Default TTL values:

Linux/MAC: 64
Windows: 128

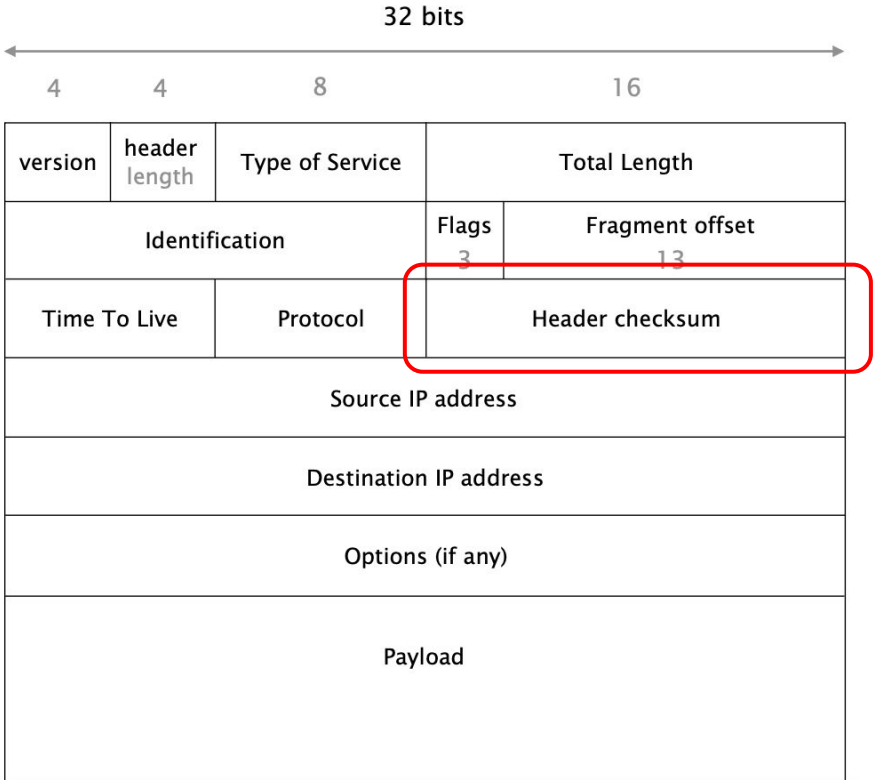
This can be used to fingerprint OSes



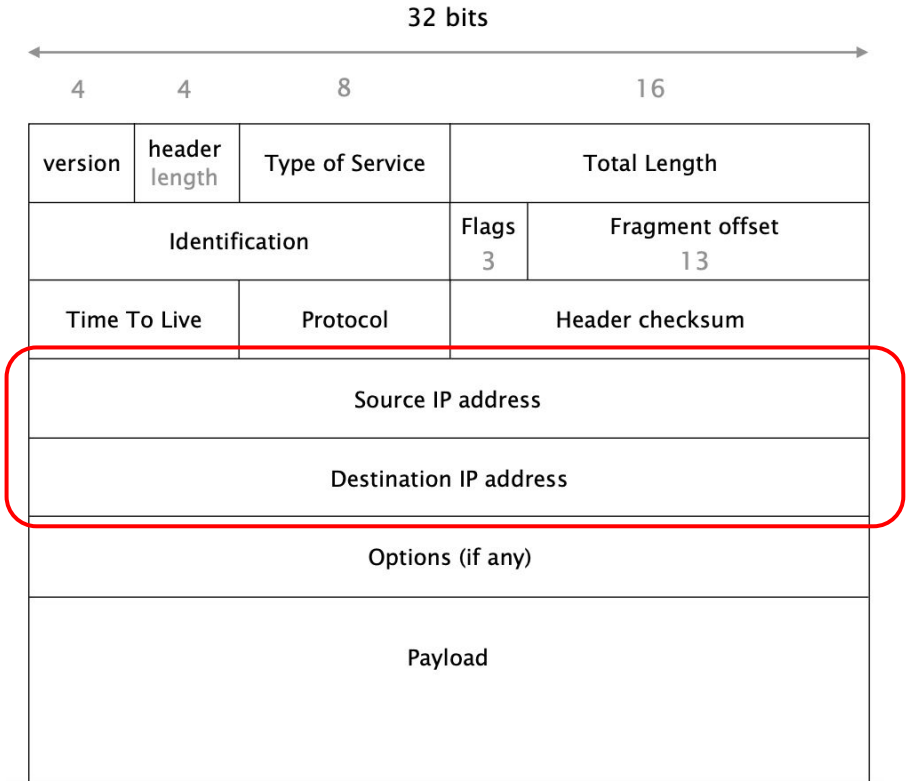
Protocol Identifies Higher Layer Protocol: 6 = TCP, 17 = UDP



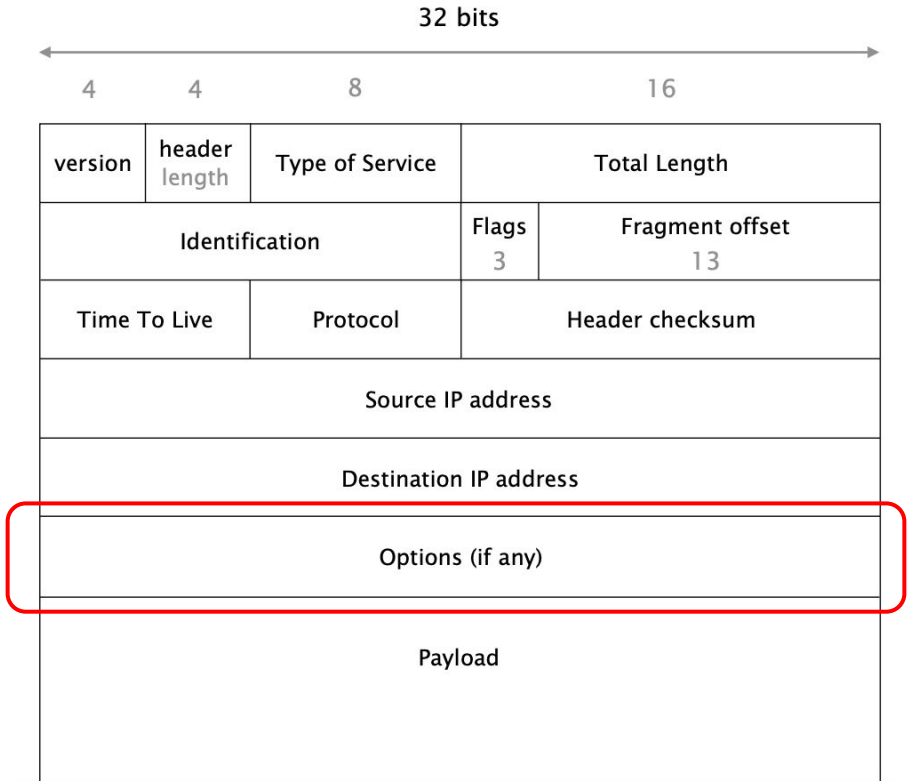
Checksum is the Sum of all 16 bit Words in the Header (NOT the payload)



Source and Destination IPs are the Host Endpoints



Options Provide Flexibility, They are Often Disabled



IPv4 vs IPv6

v4 still dominates the Internet (for now), even though we ran out of addresses

IPv6

Simpler than IPv4

- IPv6 was motivated by address exhaustion
  - IPv6 addresses are 128 bits long, huge address space
- IPv6 got rid of anything that wasn’t necessary
  - Fragmentation - leave problems to the endpoints
  - Checksum - leave problems to the endpoints
  - Header length - simplify handling
- Result is an elegant, boring, protocol
  - Allows for arbitrary options (IPv4 options have to be processed by every router - SLOW)

IPv6

Why hasn’t IPv6 taken over?

- Every device needs to support it (routers, middleboxes, end hosts, applications, ...)
- Most features we backported to IPv4 - no obvious advantage
- Network Address Translation (NAT) works
  - Most people have no idea we ran out of IPv4 addresses

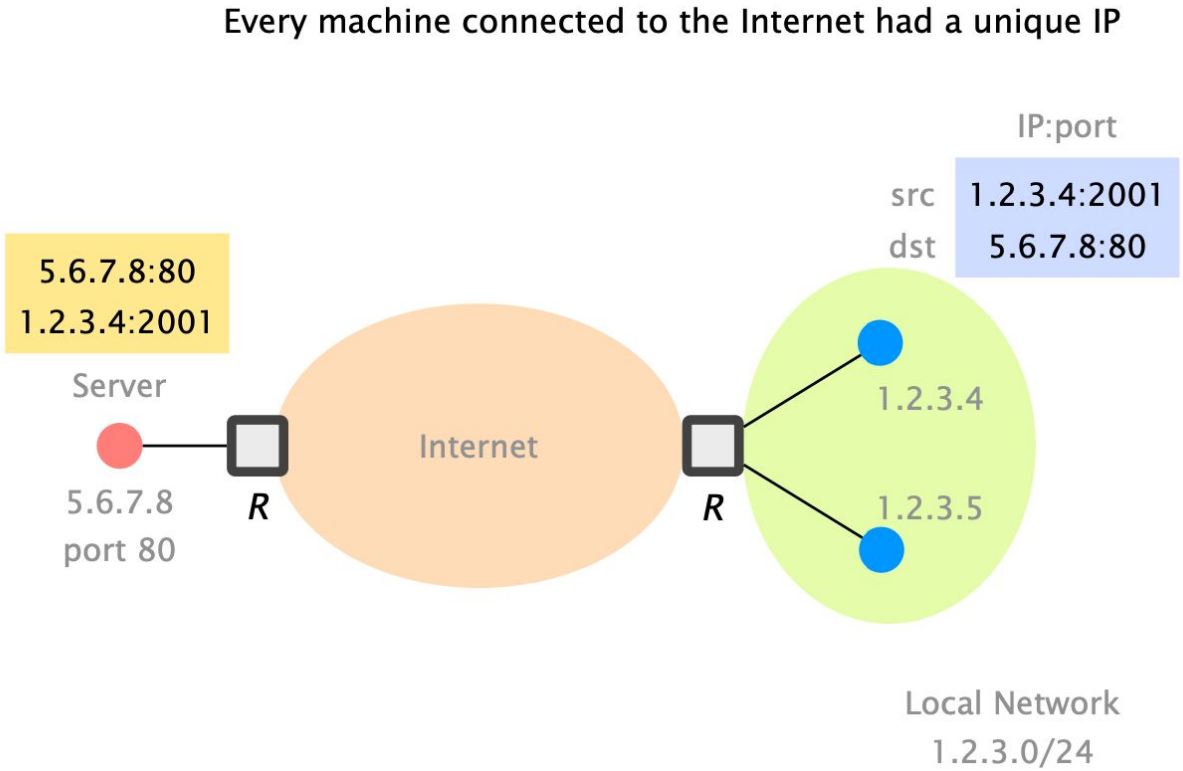
NAT

- Share a single (public) address or a pool (CG NATs) between hosts
  - Port numbers (transport layer) are used to distinguish

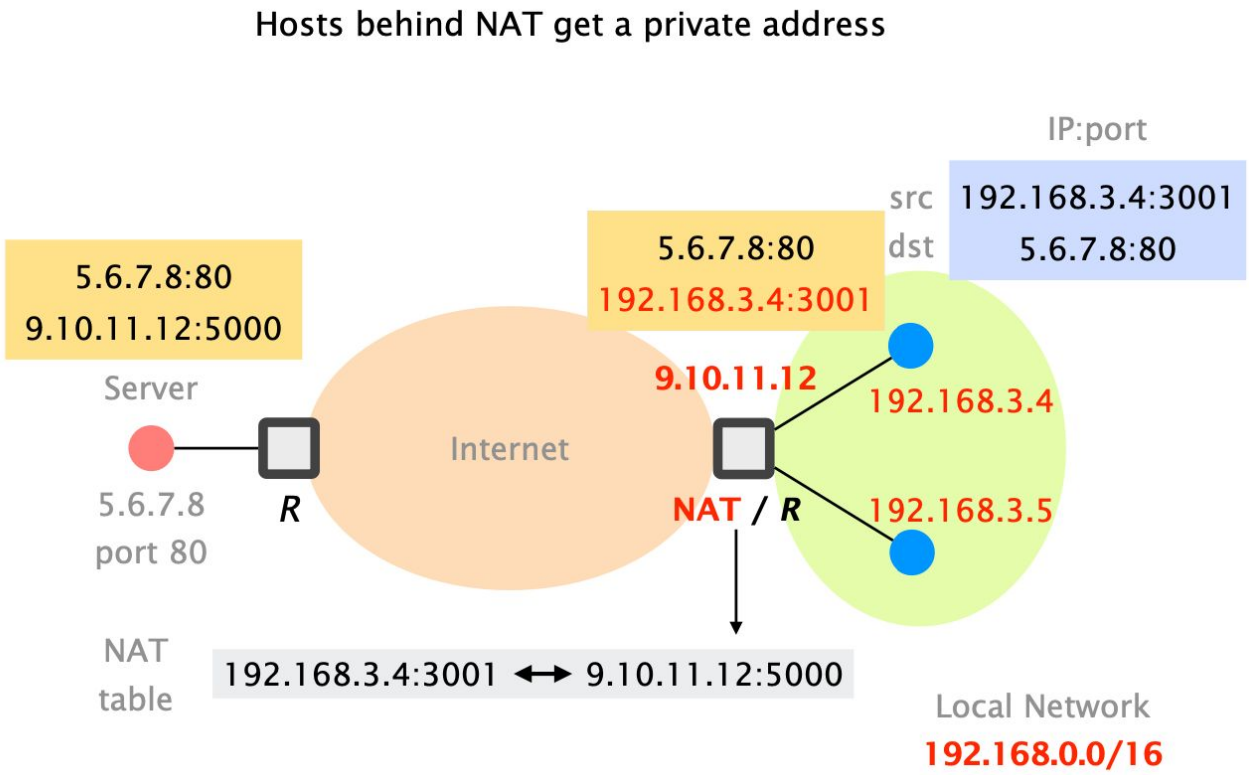
- One of the main reasons why we can still use IPv4
  - Saved us from address depletion

- Violates the general end-to-end principle of the Internet
  - A NAT box adds a layer of indirection

Before NAT

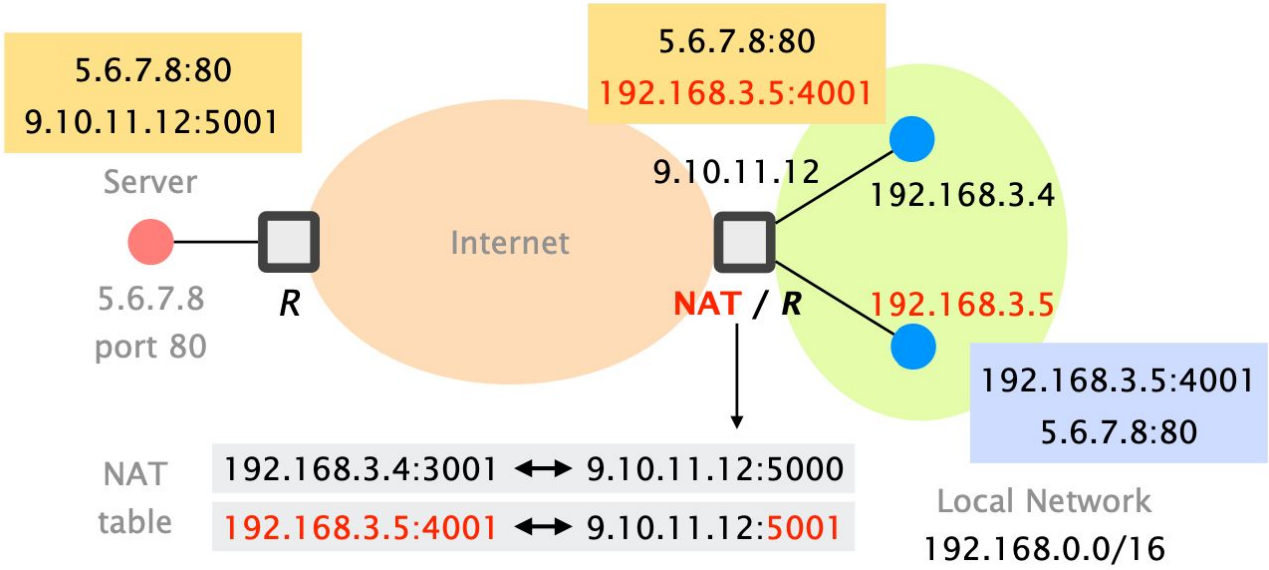

Every machine connected to the Internet had a unique IP
IP:port
src 1.2.3.4:2001
dst 5.6.7.8:80
5.6.7.8:80
1.2.3.4:2001
Server
5.6.7.8
port 80
R
Internet
R
1.2.3.4
1.2.3.5
Local Network
1.2.3.0/24

After NAT


Hosts behind NAT get a private address
IP:port
src
dst
192.168.3.4:3001
5.6.7.8:80
5.6.7.8:80
9.10.11.12:5000
5.6.7.8:80
192.168.3.4:3001
Server
9.10.11.12
192.168.3.4
Internet
5.6.7.8
port 80
R
NAT / R
192.168.3.5
NAT
table
192.168.3.4:3001 ↔ 9.10.11.12:5000
Local Network
192.168.0.0/16

After NAT

The port numbers are used to multiplex single addresses



NAT Pros and Cons

- Better privacy/anonymization
  - All hosts in one network get the same public IP
  - But, cookies, browser version, ... still identify hosts
- Better security
  - From the outside you cannot directly reach the hosts
  - Problematic e.g., for online gaming
- Limited scalability (size of the mapping table)
  - Example: Wi-Fi access problems in public places (e.g., lecture hall) often due to a full NAT table

NAT

There are two pcaps: NAT_home_side.pcap and NAT_ISP_side.pcap

Open the NAT_home_side file.

1. What is the IP address of the client?

2. The client actually communicates with several different Google servers in order to implement “safe browsing.” The main Google server that will serve up the main Google web page has IP address 64.233.169.104. In order to display only those frames containing HTTP messages that are sent to/from this Google, server, enter the expression “http && ip.addr == 64.233.169.104” (without quotes) into the Filter: field in Wireshark.

3. Consider now the HTTP GET sent from the client to the Google server (whose IP address is IP address 64.233.169.104) at time 7.109267. What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP GET?

4. At what time is the corresponding 200 OK HTTP message received from the Google server? What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP 200 OK message?

5. Before a GET command can be sent to an HTTP server, TCP must first set up a connection using the three-way SYN/ACK handshake. At what time is the client-to-server TCP SYN segment sent that sets up the connection used by the GET sent at time 7.109267? What are the source and destination IP addresses and source and destination ports for the TCP SYN segment? What are the source and destination IP addresses and source and destination ports of the ACK sent in response to the SYN. At what time is this ACK received at the client? (Note: to find these segments you will need to clear the Filter expression you entered above in step 2. If you enter the filter “tcp”, only TCP segments will be displayed by Wireshark).

NAT

There are two pcaps: NAT_home_side.pcap and NAT_ISP_side.pcap

Open the NAT_home_side file.

1. What is the IP address of the client?
   192.168.1.100
2. The client actually communicates with several different Google servers in order to implement “safe browsing.” The main Google server that will serve up the main Google web page has IP address 64.233.169.104. In order to display only those frames containing HTTP messages that are sent to/from this Google, server, enter the expression “http && ip.addr == 64.233.169.104” (without quotes) into the Filter: field in Wireshark.

3. Consider now the HTTP GET sent from the client to the Google server (whose IP address is IP address 64.233.169.104) at time 7.109267. What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP GET?
   192.168.1.100 -> 64.233.169.104, Ports 4335 -> 80
4. At what time is the corresponding 200 OK HTTP message received from the Google server? What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP 200 OK message?
   Time 7.158797, 64.233.169.104 -> 192.168.1.100, Ports 80 -> 4335
5. Before a GET command can be sent to an HTTP server, TCP must first set up a connection using the three-way SYN/ACK handshake. At what time is the client-to-server TCP SYN segment sent that sets up the connection used by the GET sent at time 7.109267? What are the source and destination IP addresses and source and destination ports for the TCP SYN segment? What are the source and destination IP addresses and source and destination ports of the ACK sent in response to the SYN. At what time is this ACK received at the client? (Note: to find these segments you will need to clear the Filter expression you entered above in step 2. If you enter the filter “tcp”, only TCP segments will be displayed by Wireshark).
   Time 7.07675, 192.168.1.100 -> 64.233.169.104, 4335 -> 80, ACK received 7.108986, reverse IPs and ports

NAT

Open the NAT_ISP_side file.

1. Find the HTTP GET message was sent from the client to the Google server at time 7.109267 (where t=7.109267 is time at which this was sent as recorded in the NAT_home_side trace file). At what time does this message appear in the NAT_ISP_side trace file? What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP GET (as recording in the NAT_ISP_side trace file)? Which of these fields are the same, and which are different, than in your answer to question 3 above?

2. Are any fields in the HTTP GET message changed? Which of the following fields in the IP datagram carrying the HTTP GET are changed: Version, Header Length, Flags, Checksum. If any of these fields have changed, give a reason (in one sentence) stating why this field needed to change.

3. In the NAT_ISP_side trace file, at what time is the first 200 OK HTTP message received from the Google server? What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP 200 OK message? Which of these fields are the same, and which are different than your answer to question 4 above?

4. In the NAT_ISP_side trace file, at what time were the client-to-server TCP SYN segment and the server-to-client TCP ACK segment corresponding to the segments in question 5 above captured? What are the source and destination IP addresses and source and destination ports for these two segments? Which of these fields are the same, and which are different than your answer to question 5 above?

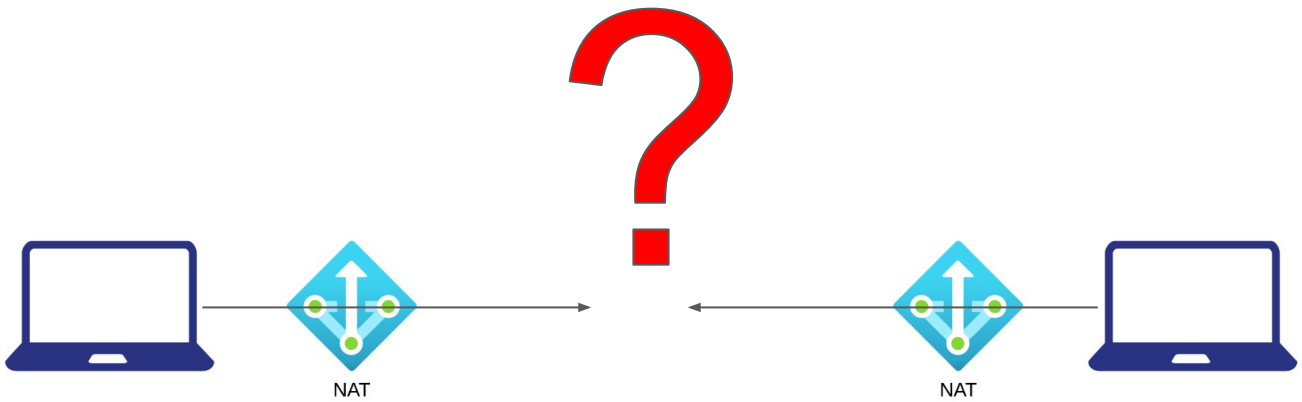
NAT

Open the NAT_ISP_side file.

1. Find the HTTP GET message was sent from the client to the Google server at time 7.109267 (where t=7.109267 is time at which this was sent as recorded in the NAT_home_side trace file). At what time does this message appear in the NAT_ISP_side trace file? What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP GET (as recording in the NAT_ISP_side trace file)? Which of these fields are the same, and which are different, than in your answer to question 3 above?
   6.069168, Src: 71.192.34.104, Dst: 64.233.169.104, Src Port: 4335, Dst Port: 80
2. Are any fields in the HTTP GET message changed? Which of the following fields in the IP datagram carrying the HTTP GET are changed: Version, Header Length, Flags, Checksum. If any of these fields have changed, give a reason (in one sentence) stating why this field needed to change.
   Checksum is different because it is calculated over the fields of the header, including a different source IP address
3. In the NAT_ISP_side trace file, at what time is the first 200 OK HTTP message received from the Google server? What are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP 200 OK message? Which of these fields are the same, and which are different than your answer to question 4 above?
   6.117570, Src: 64.233.169.104, Dst: 71.192.34.104, Src Port: 80, Dst Port: 4335
4. In the NAT_ISP_side trace file, at what time were the client-to-server TCP SYN segment and the server-to-client TCP ACK segment corresponding to the segments in question 5 above captured? What are the source and destination IP addresses and source and destination ports for these two segments? Which of these fields are the same, and which are different than your answer to question 5 above?
   6.035475 and 6.067775, Src: 71.192.34.104, Dst: 64.233.169.104 and reverse, Src Port: 4335, Dst Port: 80 and reverse

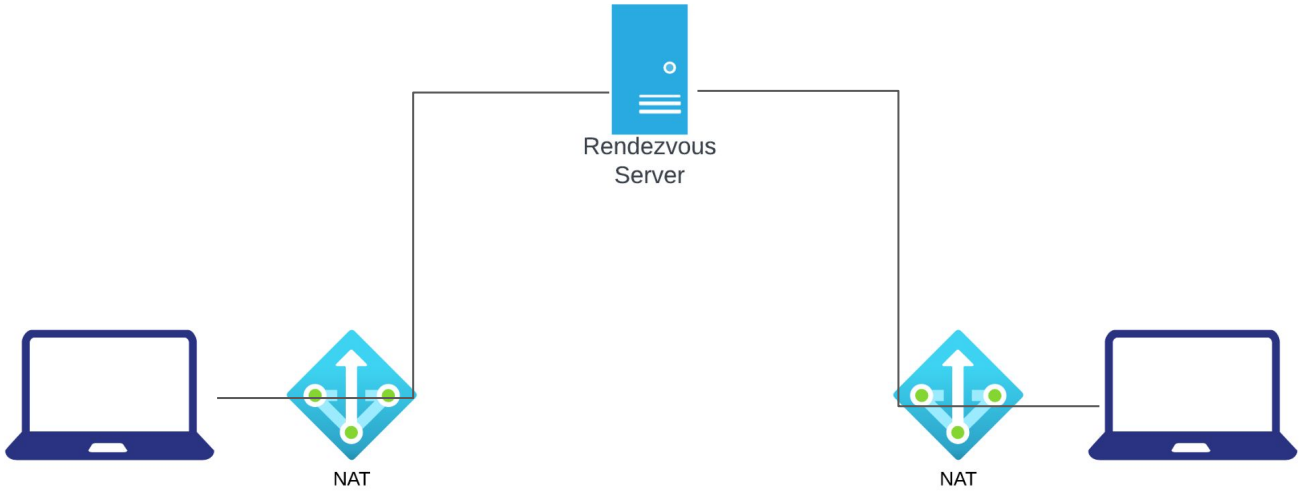
Connectivity Between NAT Machines

If you have two machines behind NATs that wish to communicate, how do they know the correct source and destination IPs / ports to use?



Hole Punching

Each machine sets up a connection to a publicly reachable rendezvous server, which then relays the streams for the clients. This is known as hole punching



IPv4 and IPv6 - Many OSes and Applications Use a Dual Stack