# The TCP/IP Reference Model

- TCP is Transmission Control Protocol.
- IP is Internet Protocol.
- Only 4 layers:

| | |
|---|---|
| 1 | **Application Layer** |
| 2 | **Transport Layer** |
| 3 | **Internet Layer** |
| 4 | **Link Layer (network)** |

# The Tanenbaum Reference Model

- The model used in Tanenbaum book adds a Physical layer (page 48). Also used by others.
- But we will stick to the official TCP/IP model since the physical layer is out of the course scope

| | |
|---|---|
| 1 | **Application Layer** |
| 2 | **Transport Layer** |
| 3 | **Internet Layer** |
| 4 | **Link Layer (network)** |
| 5 | **Physical Layer** |

# Layer 4: The Application Layer

- Higher-level protocols such as: TELNET, FTP, SMTP, DNS, HTTP, POP2, POP3.
- These are the protocols that are used by applications like MS internet explorer, Google Chrome, MS outlook, Skype, Waze, etc.
- This layer is essentially the same as the **OSI Model** layer 7

# Layer 3: The Transport Layer (TCP / UDP)

- This layer implements layers 4, 5, and 6 of the **OSI model** (session, presentation, and transport)
- Handles full messages (long documents, multimedia, etc.)
- Nevertheless, in many cases **OSI layer 6** makes sense (encryption, compression, data representation) and used in analysis
- The most used protocols are: TCP, UDP (but there are additional 15 new ones)
- Usually implemented at the operating system kernel (Unix and Windows) (why?)

# Layer 2: The Internet Layer (IP)

- Connectionless internetwork layer  (IP Protocol)
- Packet-switching: blocks of data constrained to a fixed size
- permitting hosts to send packets into any network and have them travel independently to the destination, potentially on a different network.
- Implemented at the operating system, at routers hardware, gateways, bridges, etc.
- A computer can act sometimes as a router or a gateway, so the operating system includes special modules to handle network operations
- Major interface: `SEND_IP_PACKET, RECEIVE_IP_PACKET`

# Layer 1: Link/network Layer (Ethernet/wireless)

- Almost everything below the internet layer is not defined in the TCP/IP reference model
- The network layer essentially performs the functions of the OSI physical and data link layers
- Usually implemented by network device drivers: Ethernet, Ring or Star card drivers (with the help of the device drivers of course)

# OSI and TCP/IP Reference Models

| | OSI | TCP/IP |
|---|---|---|
| 7 | Application | Application |
| 6 | Presentation | |
| 5 | Session | |
| 4 | Transport | Transport |
| 3 | Network | Internet |
| 2 | Link | Link/Network |
| 1 | Physical | |

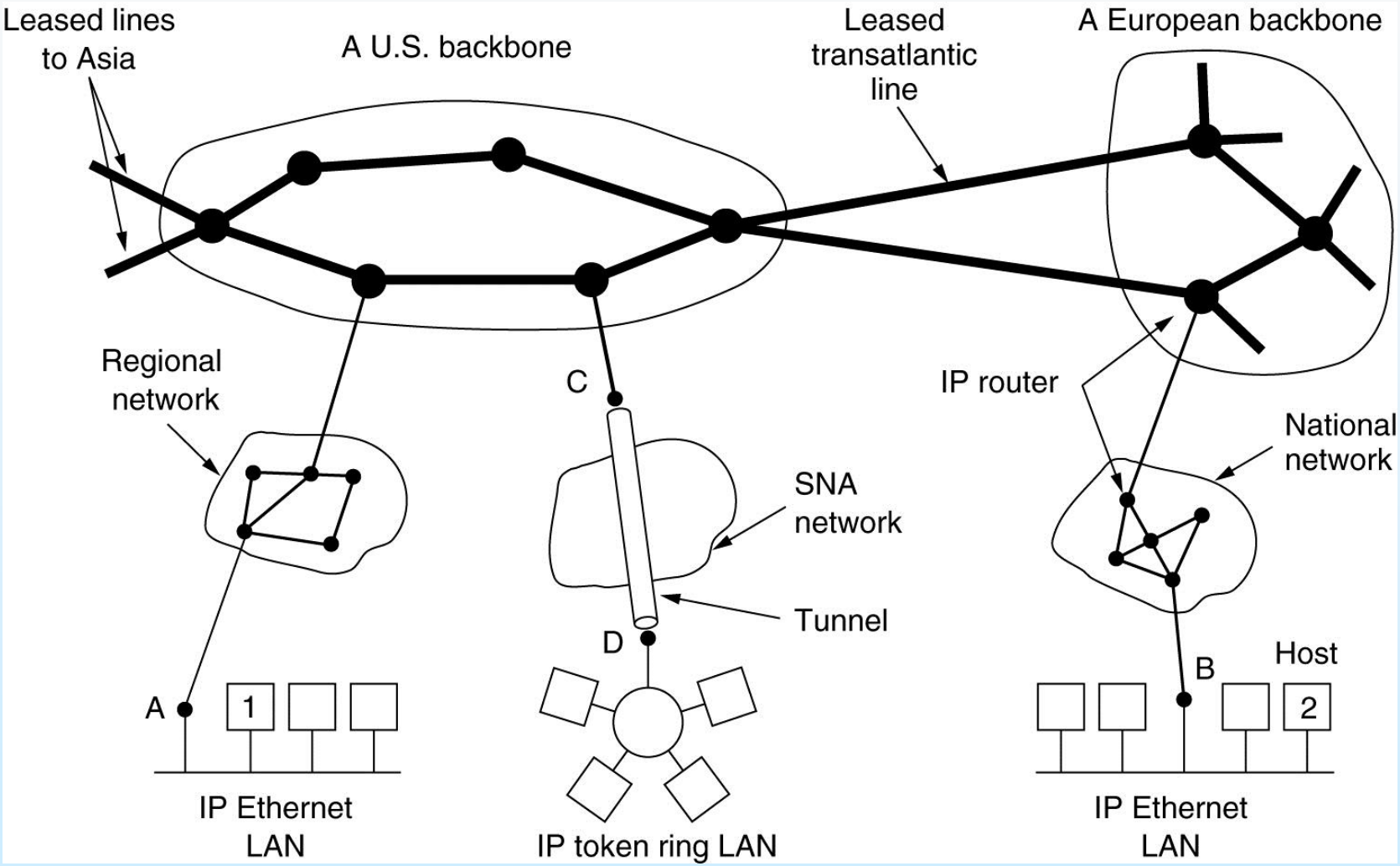
# TCP/IP Family

- TCP/IP refers to an entire communication protocol family based on the
  - Transmission Control Protocol (TCP)
  - The Internet Protocol (IP)
- It defines protocols at the network layer and the transport layer
- The TCP/IP suite has six basic elements:
  - Applications
  - The Transmission Control Protocol (TCP)
  - The User Datagram Protocol (UDP)
  - The Internet Protocol (IP)
  - Auxiliary protocols like the Internet Control, Message Protocol (ICMP), and the Address Resolution Protocol (ARP).

# TCP/IP Family: IP

- IP major role is to route packets from a process in one machine to another process at another machine (possibly the same machine)
- For that IP uses an IP address and a Port number
  - The port number determines the specific process to which the packet belongs
- When an application sends a data packet to another machine
  - IP determines to which network the packet should go
  - if necessary, IP routes the packet from one network to another
- IP figures out where to send a packet based on the IP address of the recipient
- At some hops, IP may fragment a large packet to smaller packets (“fragments”) if that network cannot handle large packets (link with a smaller MTU - maximum transmission unit)

# Internet: Collection of Subnetworks



# The IP Protocol

- Packet delivery service (host-to-host).
- IP provides connectionless, unreliable delivery of IP datagrams.
- Connectionless: each datagram is independent of all others.
- Unreliable: there is no guarantee that datagrams are delivered correctly or at all.

# IP Addressing (v4)

**Every host on the internet is assigned a unique IP address which consists of 32 bits.**

Example:

```
                |<-------  32 bits   ------->|
      address = 11000111111001011100110000001010
```

The IP address consists of two parts: Network ID + Host ID

```
 1-8      9-16     17-24    25-32
-----------------------------------------
Class A:  0nnnnnnn.hhhhhhhh.hhhhhhhh.hhhhhhhh       0-127
Class B:  10nnnnnn.nnnnnnnn.hhhhhhhh.hhhhhhhh     128-191
Class C:  110nnnnn.nnnnnnnn.nnnnnnnn.hhhhhhhh     192-223
Class D:  1110mmmm.mmmmmmmm.mmmmmmmm.mmmmmmmm     224-239
Class E:  11110rrr.rrrrrrrr.rrrrrrrr.rrrrrrrr     240-247
```

n = network bit
h = host bit
m = multicast
r = reserved for future use

# IP Addressing (v4)

**Example:**

```
                |<-------  32 bits   ------->|
      address = 11000111110010111001100000001010
```

Every IP address belongs to a network class and consists of two parts:

```
            [Network ID] + [Host ID]

   |<---------      24 bit     -------->|<- 8 bit ->|
   |<---------    Network ID   -------->|<-Host ID->|
   11000111110010111001100000001010
```

Subnet mask:

```
   11111111111111111111111100001010
   255.255.255.0
```

# IP Addressing (v4)

The algorithm to determine the address class is as follows:

```
Class A:  0nnnnnnn.hhhhhhhh.hhhhhhhh.hhhhhhhh       0-127
Class B:  10nnnnnn.nnnnnnnn.hhhhhhhh.hhhhhhhh     128-191
Class C:  110nnnnn.nnnnnnnn.nnnnnnnn.hhhhhhhh     192-223
Class D:  1110mmmm.mmmmmmmm.mmmmmmmm.mmmmmmmm     224-239
Class E:  11110rrr.rrrrrrrr.rrrrrrrr.rrrrrrrr     240-247
```

```
Class A:    0.0.0.0 -- 127.255.255.255   127 Networks of size=16M
Class B:  128.0.0.0 -- 191.255.255.255   16K Networks of size=64K
Class C:  192.0.0.0 -- 223.255.255.255    2M Networks of size=256
Class D:  224.0.0.0 -- 239.255.255.255
Class E:  240.0.0.0 -- 247.255.255.255   Reserved
```

# IP Addressing (v4)

- A Network ID is assigned to an organization by a global authority (**ICANN** - Internet Corporation for Assigned Names and Numbers)
- Host IDs are assigned locally by a system administrator or automatically by a DHCP server
- Both the Network ID and the Host ID are used for routing
- Very few organizations are assigned Class A addresses (USA military, government, Boing, large banks, ...)
  - But they do not use all possible host ids
- Many universities and companies were assigned class B addresses, but most of them do not use more than 1000 or 2000 host ids (out of the 64K possible host ids).

# IP Addresses

- An IP address is assigned per network interface, not host!
- So a host that belongs to two networks must have two network interfaces and thus two IP addresses!

# EXAMPLE

- The IP number of Netanya College Linux mail server, moon.netanya.ac.il is a 32 bits binary integer:

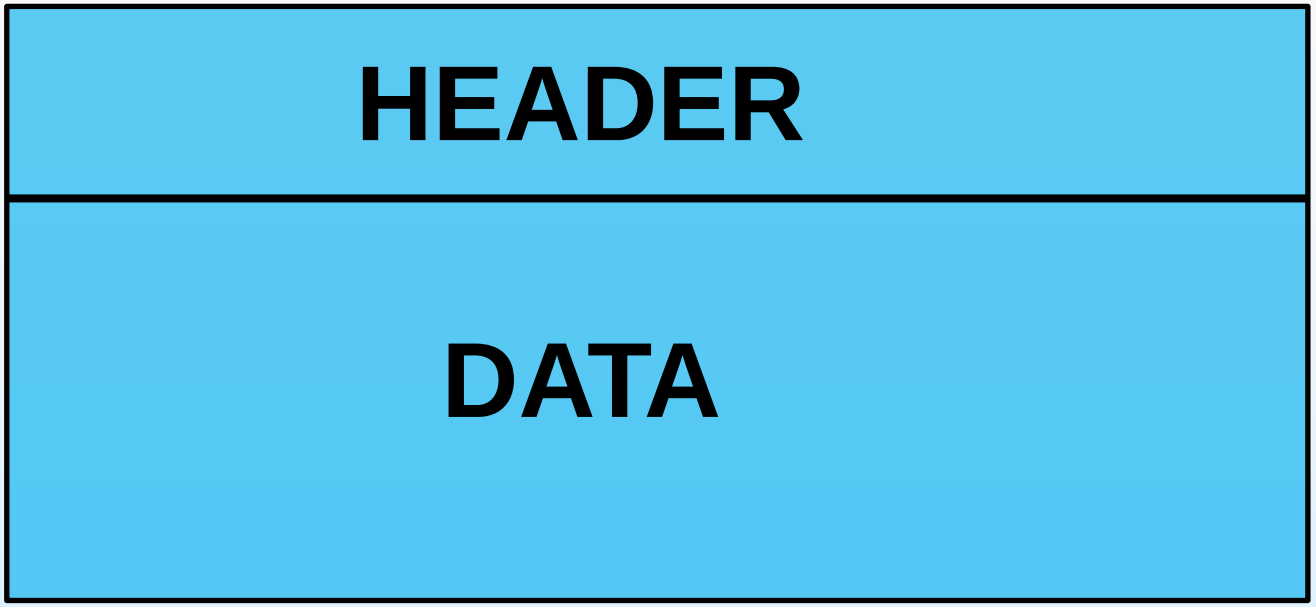
  11000111111001011100110000001010

- It is better viewed 4 bytes:

  **110**00111.11001011.10011000.00001010

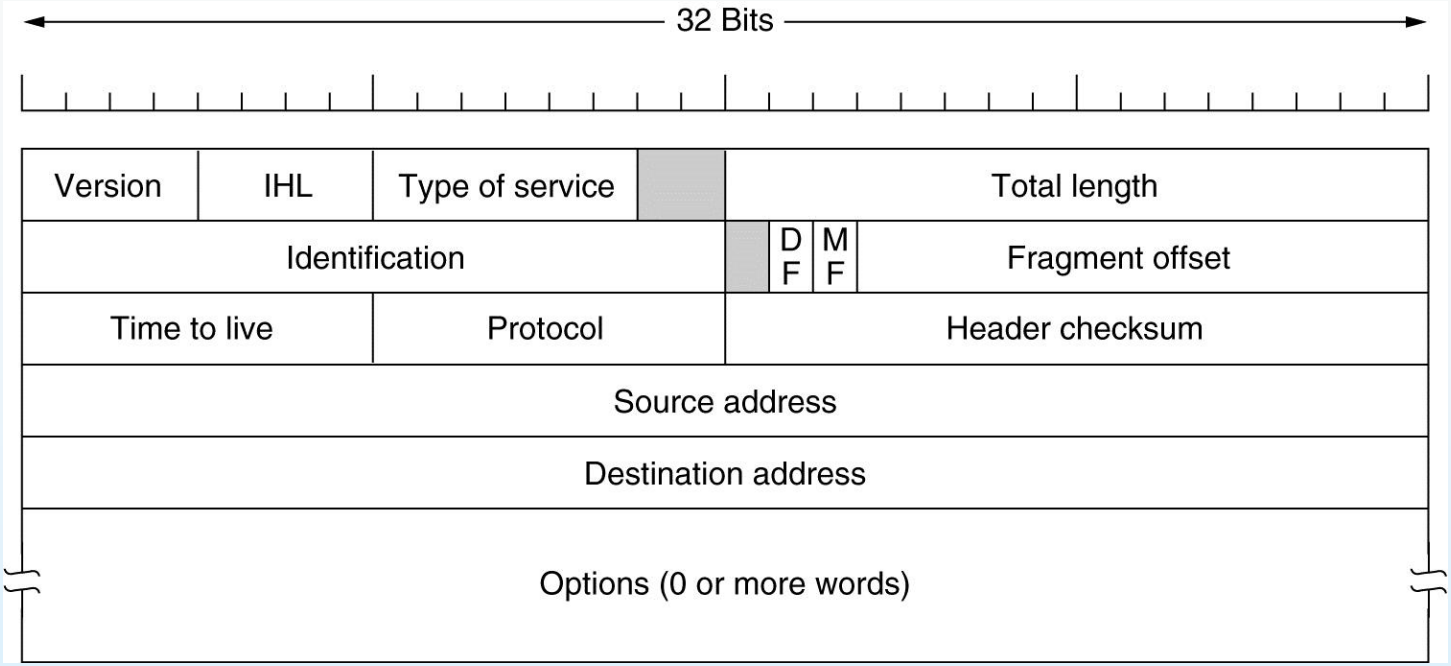
- Even better as: **199.203.152.6**
- Since it starts in "110" it is a class C address, and therefore its network mask is: 11111111.11111111.11111111.00000000
- The network number is 199.203.152.0.
- Broadcast address is 199.203.152.255.
- Broadcast mask is 255.255.255.255.

# The IP Datagram Structure



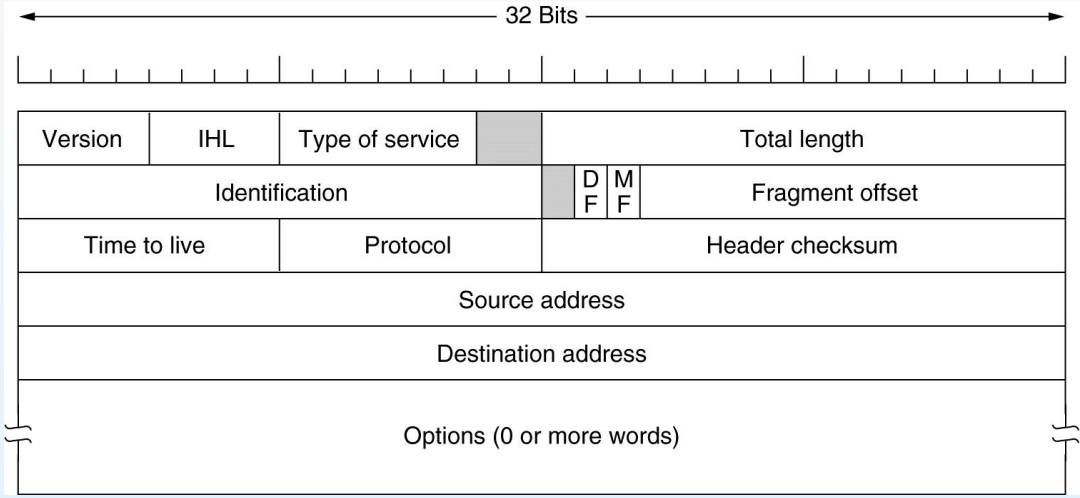

- Header length is 20 bytes minimum and 60 bytes maximum
- Packet size can range from 40 bytes to 64K bytes depending on networking software and networking hardware
- The data part is usually a small fragment of the total message which the TCP (or UDP) protocol is trying to transmit
- TCP and UDP are the drivers of the IP protocol

# The IP Datagram Header (v4)



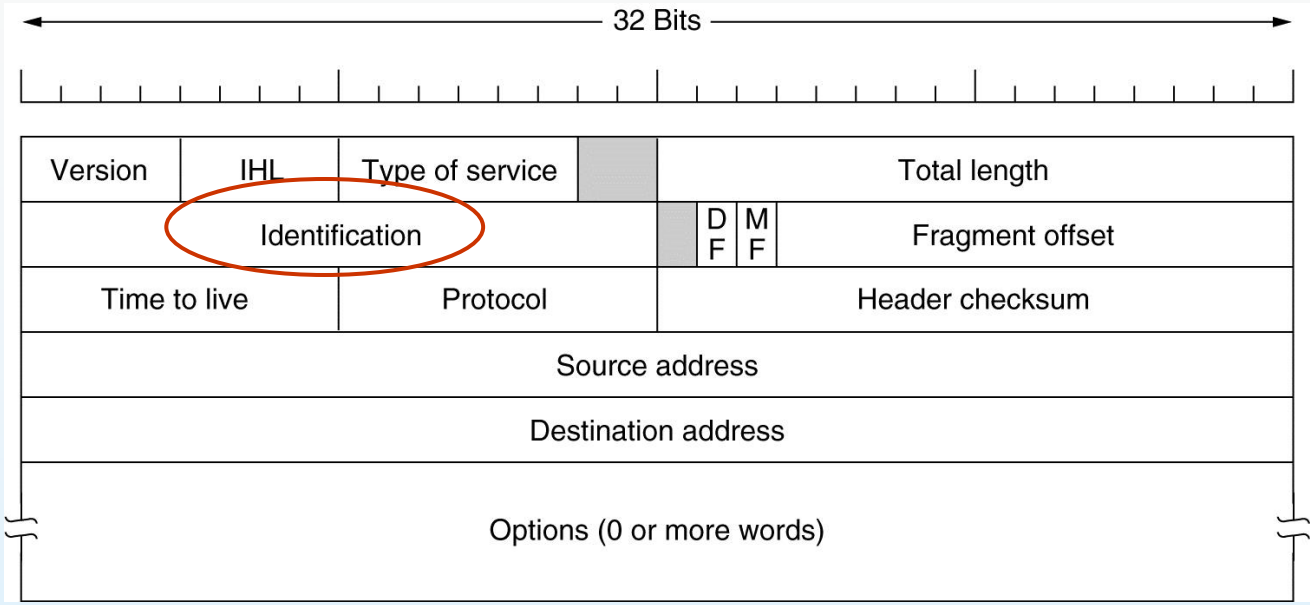

- Has a 20 bytes fixed part and a variable length optional part
- Version – IP Protocol Version (v4, v5, v6)
- IHL – (4 bits) The number of 32-bit words in the header (min=5W, max=15W). That is, the header can be at most 60 bytes!
- Total Length - total length of the datagram in bytes
  - size of the data  = total length - header length"

# Type of service
## (also called: Differentiated Services)



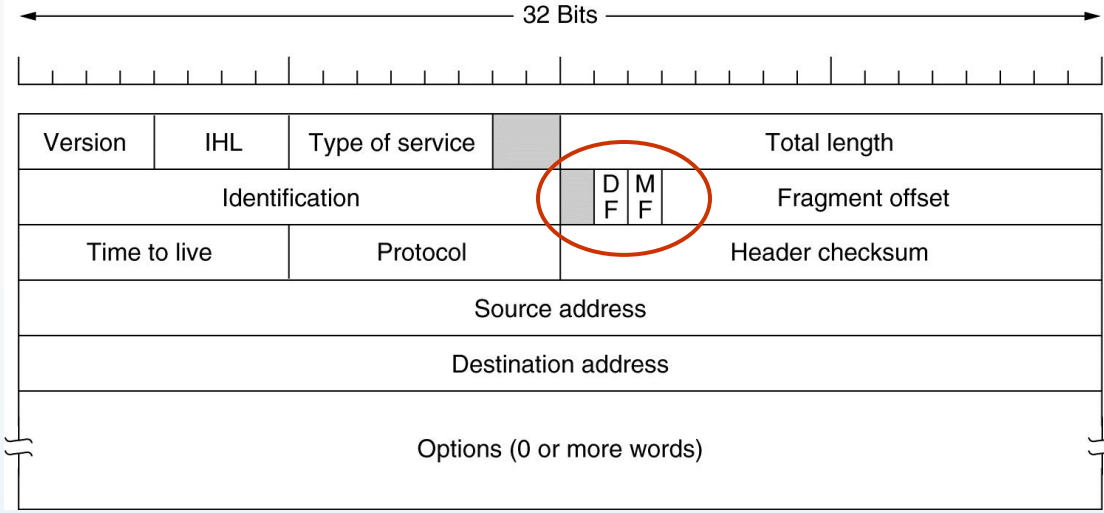

- Consists of 6 bits:
  - 1000 - minimize delay
  - 0100 - maximize throughput
  - 0010 - maximize reliability
  - 0001 - minimize monetary cost
  - The other two bits used to record congestion history but now used for VOIP
- This is a "hint" to the physical layer to which path to use
- Not supported in most implementations. Some implementations have extra fields in the routing table to indicate delay, throughput, reliability, and monetary cost.

# Identification



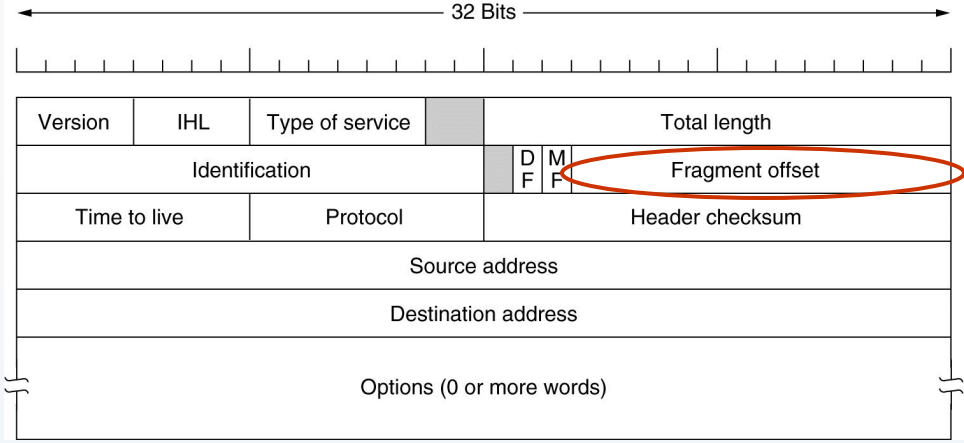

- Uniquely identifies the datagram
- Usually incremented by 1 each time a new datagram is sent
  - Puts a max limit on packet sequence: 2^16 * (packet_length) ~ 4G
- All fragments of a datagram contain the same identification value
- This allows the destination host to determine which fragment belongs to which datagram

# FLAGS



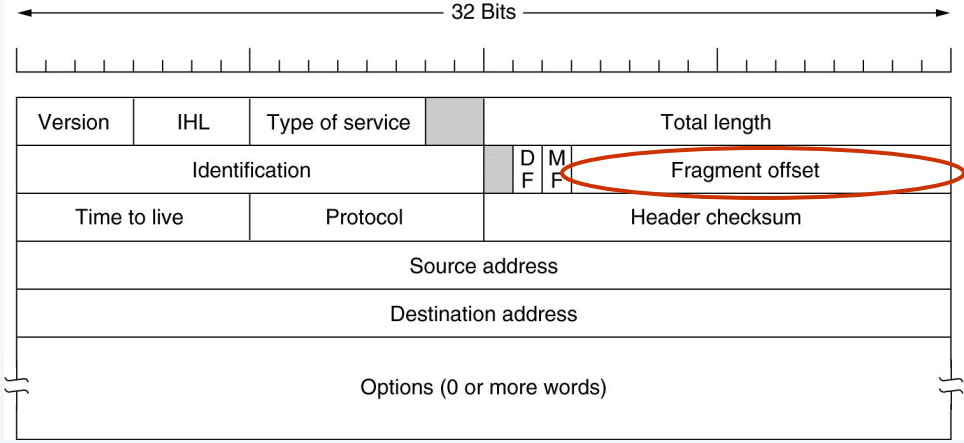

- Used for fragmentation
- DF means “do not fragment”
  - It is a request to routers not to fragment the datagram since the destination is incapable of putting the pieces back together
  - Can be use for MTU detection
- MF means “more fragments to follow”
  - All fragments except the last one have this bit set!
  - It is needed to know if all fragments of a datagram have arrived
- The bit to the left of DF is still unused … (electrical waste …)
  - Required to be 0

# Fragment Offset

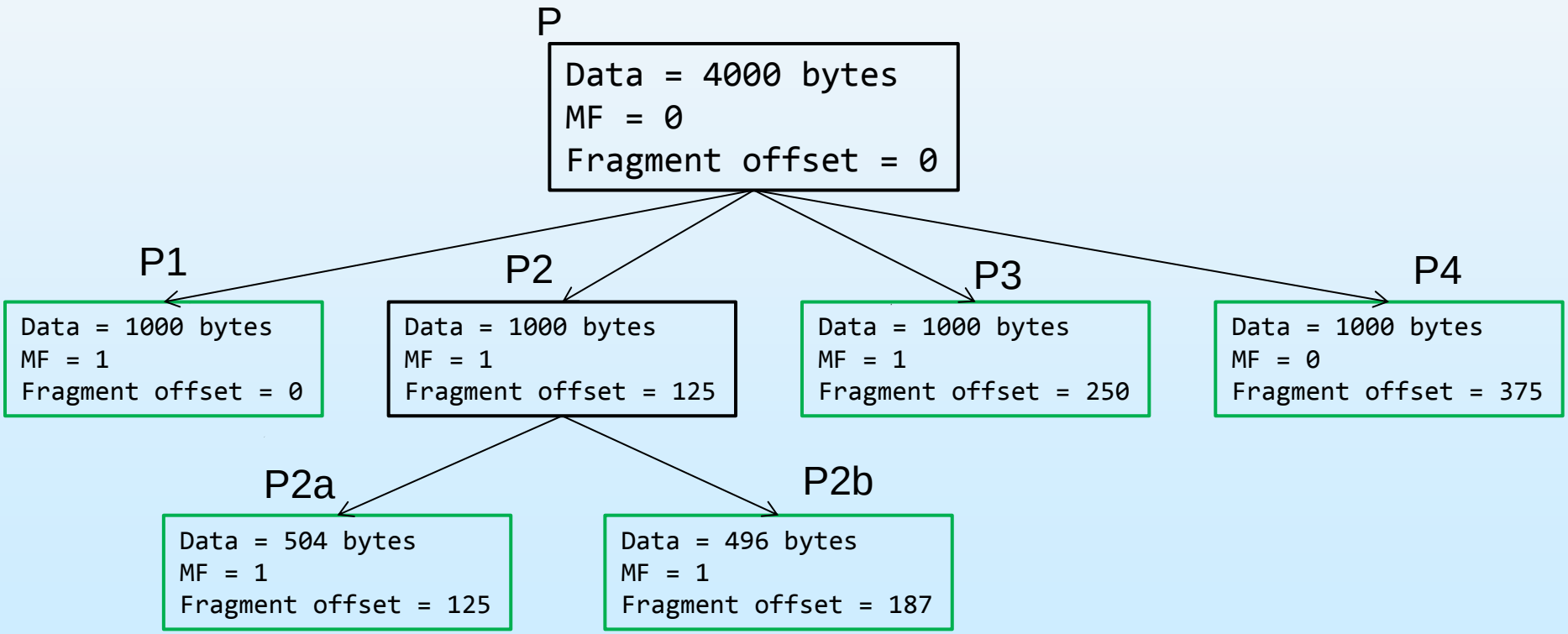



- Initial state: **fragment offset=0, MF=0**
- A router may divide a packet to small fragments, if next hop MTU is small
- Each fragmented packet will have to change these fields:
  - The **total length** field = fragment size
  - The **more fragments** (MF) flag is set for all fragments except the last one
  - The **fragment offset** field is set to the offset of the fragment in the original data payload (measured in units of eight-byte blocks)
- The **header checksum** field is re-calculated
- **fragment offset** **=** number of **eight-byte blocks** relative to the start of the original data payload
- Maximum **fragment offset** = $(2^{13} - 1) \times 8 = 65{,}528$ bytes

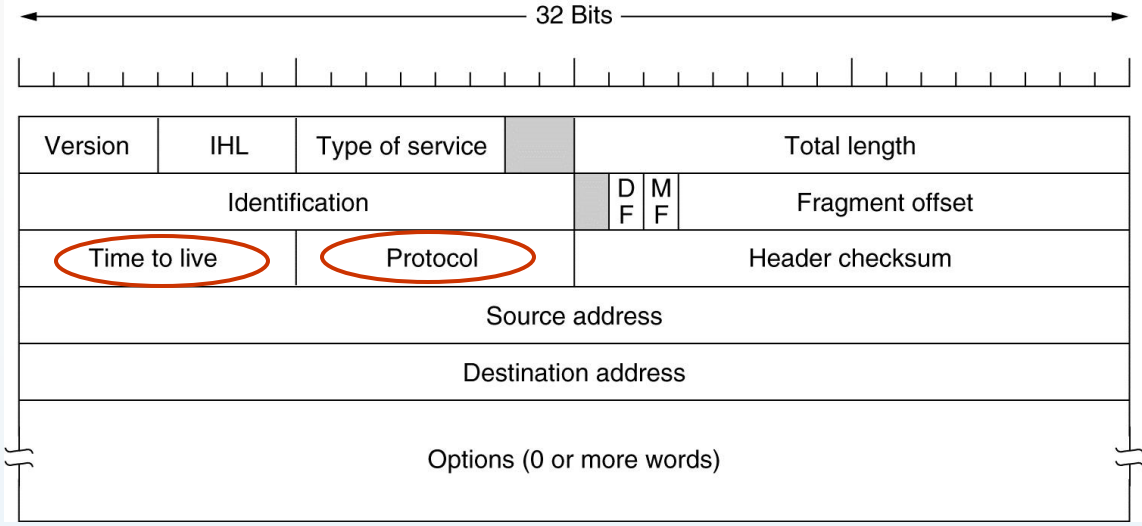
# Fragment Offset




Packet P has reached a router at Albania and got fragmented to 4 Fragments: P1, P2, P3, P4
Packet P2 has reached a router at Micronesia and got fragmented To: P2a, P2b



# Time to Live & Protocol

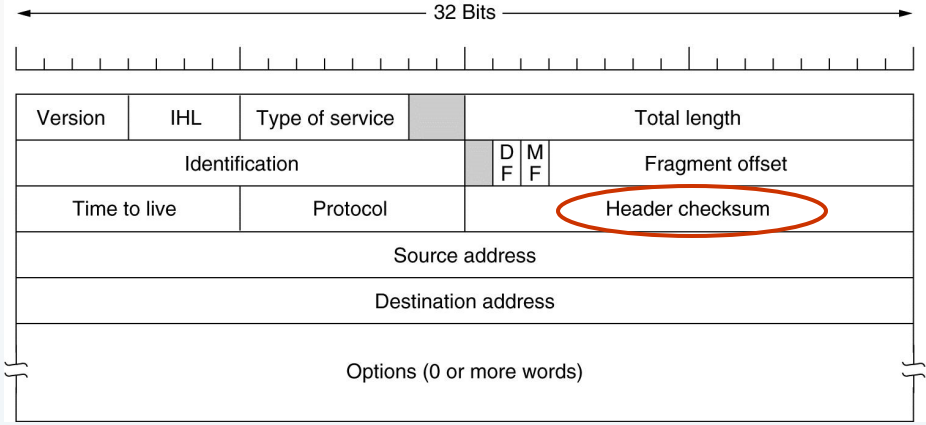



- Upper limit of routers to pass
- Usually set to 32 or 64
- Decremented by each router that processes the packet
- Router discards the datagram when TTL = 0

**Protocol**

- Tells IP where to send the datagram up to
  - 6 means TCP
  - 17 means UDP

# Header checksum




- Only covers the header, not the data!
- How the checksum is computed?
  - Put a 0 in the checksum field
  - Add each 16-bit value together
  - Add in any carry
  - Inverse the bits and put that in the checksum field
- To check the checksum:
  - Add each 16-bit value together (including the checksum)
  - Add in carry
  - Inverse the bits
  - The result must be 0
- The ttl field changes at each hop so this needs to be recomputed on each hop
- Probability for error?

# Example of IP Header

- What is IP Version? IHL? Type of Service
  Start Python console and run:

```
>>> bin(0x45) = 0100,0101
>>> bin(0x6c) = 0110,1100
>>> bin(0x92) = 1001,0010
>>> bin(0xcc) = 1100,1100
```

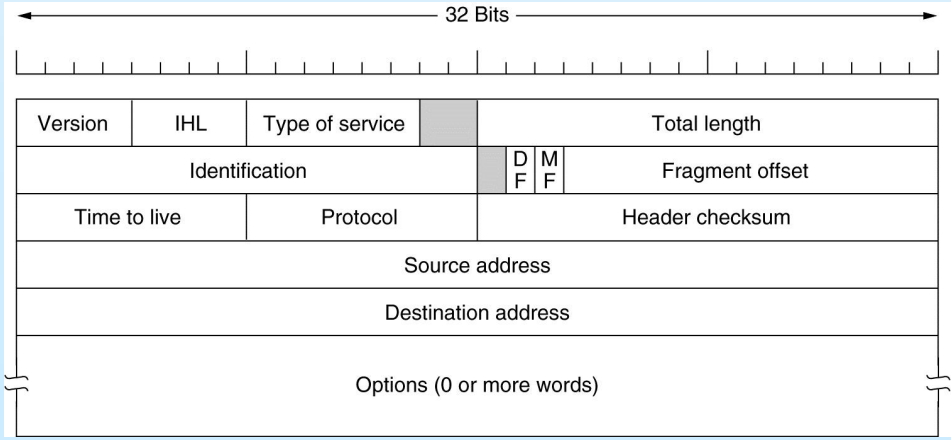
- Convert binary to decimal:

```
45:       Version = v4
          IHL = 5
00:       Type of Service = 0000
00 6c:    Total Length = 108
92 cc:    Identification = 1001001011001100
92 cc:    Checksum = 0x00
...
```

**IP HEADER**

```
45 00 00 6c
92 cc 00 00
38 06 00 00
92 95 ba 14
a9 7c 15 95
```

**Note:** When we build the IP header
We start with checksum=0x00 (RED)
And then calculate the checksum and
Write it back in that place



# Checksum Calculation

first add all 16-bit values together,
adding in the carry each time:

```
   4500
 + 006c
   ----
   456c
 + 92cc
   ----
   d838
 + 0000
   ----
   d838
 + 3806
   ----
  1103e   ← We have a carry here !
   103e      Remove the leading 1 and add back
 +    1
   ----
   103f
```

**IP HEADER**

| | | | |
|---|---|---|---|
| 45 | 00 | 00 | 6c |
| 92 | cc | 00 | 00 |
| 38 | 06 | 00 | 00 |
| 92 | 95 | ba | 14 |
| a9 | 7c | 15 | 95 |

# Checksum Calculation

```
  103f
+ 0000
  ----
  103f
+ 9295
  ----
  a2d4
+ ba14
  ----
 15ce8    ← Again we have a carry here !
  5ce9    ← Remove the leading 1 and add back
+ a97c
  ----
 10665    ← Again we have a carry here !
  0666    ← Remove the leading 1 and add back
+ 1595
  ----
  1bfb    ← Now we have to inverse the bits:
  1bfb = 0001 1011 1111 1011
  e404 = 1110 0100 0000 0100
  e404    ← This is the Checksum !
```

**IP HEADER**

| | | | |
|---|---|---|---|
| 45 | 00 | 00 | 6c |
| 92 | cc | 00 | 00 |
| 38 | 06 | 00 | 00 |
| 92 | 95 | ba | 14 |
| a9 | 7c | 15 | 95 |

**IP HEADER**

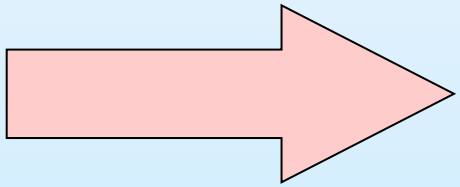
| | | | |
|---|---|---|---|
| 45 | 00 | 00 | 6c |
| 92 | cc | 00 | 00 |
| 38 | 06 | e4 | 04 |
| 92 | 95 | ba | 14 |
| a9 | 7c | 15 | 95 |

# Checksum Validation

- The receiver must validate the checksum
- It uses exactly the same algorithm, but this time it starts with “e404” and must end with “0000”
- If the computation does not end with “0000”, the receiver does not accept the packet

**IP HEADER**

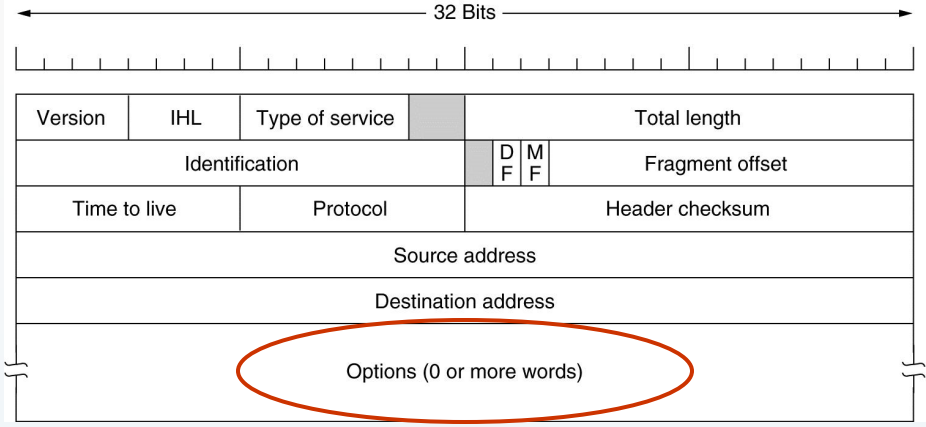
| | | | |
|---|---|---|---|
| 45 | 00 | 00 | 6c |
| 92 | cc | 00 | 00 |
| 38 | 06 | e4 | 04 |
| 92 | 95 | ba | 14 |
| a9 | 7c | 15 | 95 |



**IP HEADER**

| | | | |
|---|---|---|---|
| 45 | 00 | 00 | 6c |
| 92 | cc | 00 | 00 |
| 38 | 06 | 00 | 00 |
| 92 | 95 | ba | 14 |
| a9 | 7c | 15 | 95 |

# Options




- Each option consists of 4 bytes
- The first byte is the option control block

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| copy flag | option class | | option number | | | | |

- Copy flag: if 1, then copy option to fragments
- Option classes are
  - 0 - control
  - 1 - reserved
  - 2 - debugging and measurement
  - 3 – reserved
- The second byte designates the size of the entire option in bytes (including the control fields) and the other bytes are the option data.
- A padding to fill out the 32 bit words may be needed after all options
- There is room for at most 40 bytes for options (IP header max words = 15 words)

# Our First Project !

Design a Python class IpDatagram with the following Interface:

```
hexstr ="4500006c92cc00003806e4049295ba14a97c1595217a6f2c"

# Class constructor
p = IpDatagram(hexstr)

# Class members
p.version = 4
p.ihl = 5
p.length = 40  (bytes)

# Class methods
p.source() = 192.68.25.7
p.destination() = 157.29.41.2
p.protocol() = 17
p.ttl() = 32
p.header() = The hex string of header part
p.data() = the hex string of the data part
p.checksum() = 0xe404
p.option(n) = Hex string of option n
>>>>> MORE TO COME SOON ...  (at the course web site)
```

# TCP = Transport Control Protocol

- A reliable end-to-end byte stream over an unreliable internetwork
- Independent of network architecture, topology, speed
- Robust in the face of many kinds of failures
- Defined in RFC's 793, 1122, 1323
- A machine that supports TCP must have a single "TCP entity" as part of the operating system on top of the IP layer
- TCP sometimes mean a protocol, and sometimes it means a running computer process (operating system service)
- A bidirectional Protocol!
  - The peers (sender and receiver) exchange data in the same TCP segment format in both directions

# TCP Connections

- Two machines establish a TCP connection by creating (or using) connection end-points that are called sockets
- A **socket** is fully identified by **network IP** and a **Port number**
  - But it has more structure and operations
- Port numbers are assigned by the OS as 16-bit number
- Each machine can have up to 65535 (2**16-1) open ports
- So it is possible to have many connections between two machines (how many in principle?)
- One port can be involved in many connections
  - with different ports on the other host
  - with the same port on different hosts
  - Several browsers on the same host connected to ynet http server

# TCP Segments (1)

- TCP receives data from the Application layer (explorer, gmail, etc.)
- It may send it immediately or buffer it until it collects a large amount to send at once
- If urgent, it is possible to force TCP to flush its buffers
  - Socket flush method (sender side)
  - special bit in the TCP packet (receiver side)
- TCP breaks the data into segments (TCP packets)
- Each segment is shipped separately from the others
- may even take a different route than others
- may arrive to their destination out of order
- some of them may be lost
- It's up to the TCP entity at the other end to reassemble, report missing segments, etc, and deliver the data to the receiving process.

# TCP Segments (2)

- TCP breaks the data into segments (“TCP packets”)
- Each segment is shipped separately from the others
- Each segment may take a different route than the others
- Segments may arrive to their destination out of order
- Some segments may get lost and not reach their destination
- It is up to the TCP entity at the other end to
  - Acknowledge received segments
  - Ignore corrupt segments (no ack is required)
  - Reassemble segments to full message
  - Deliver the data to the receiving process.

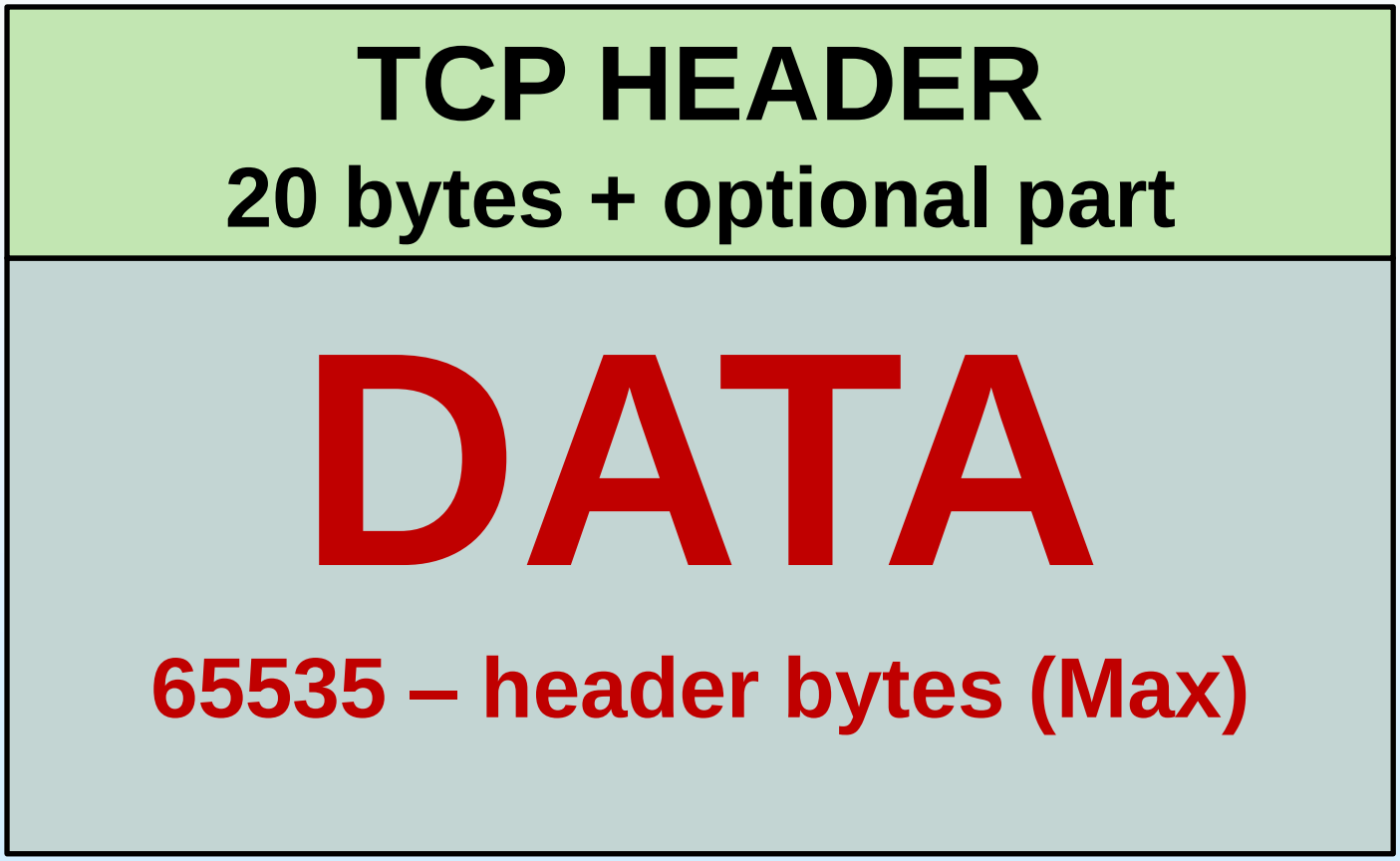
# TCP Connection Control

- After TCP sends a segment it maintains a timer for receipt of an acknowledgment from the other end
  - Every received segment is acknowledged
  - Timeout/retransmission is adaptive
  - Checksum on TCP pseudo-header
  - A bad segment is discarded without a NAK
- Duplicate segments are discarded by the receiving TCP
  - IP may deliver duplicate datagrams
- Sender times out and retransmits (if no ack. received)
- Flow control (sliding windows algorithm) Ensures that a fast sender does not swamp a slow receiver

# TCP Congestion Control

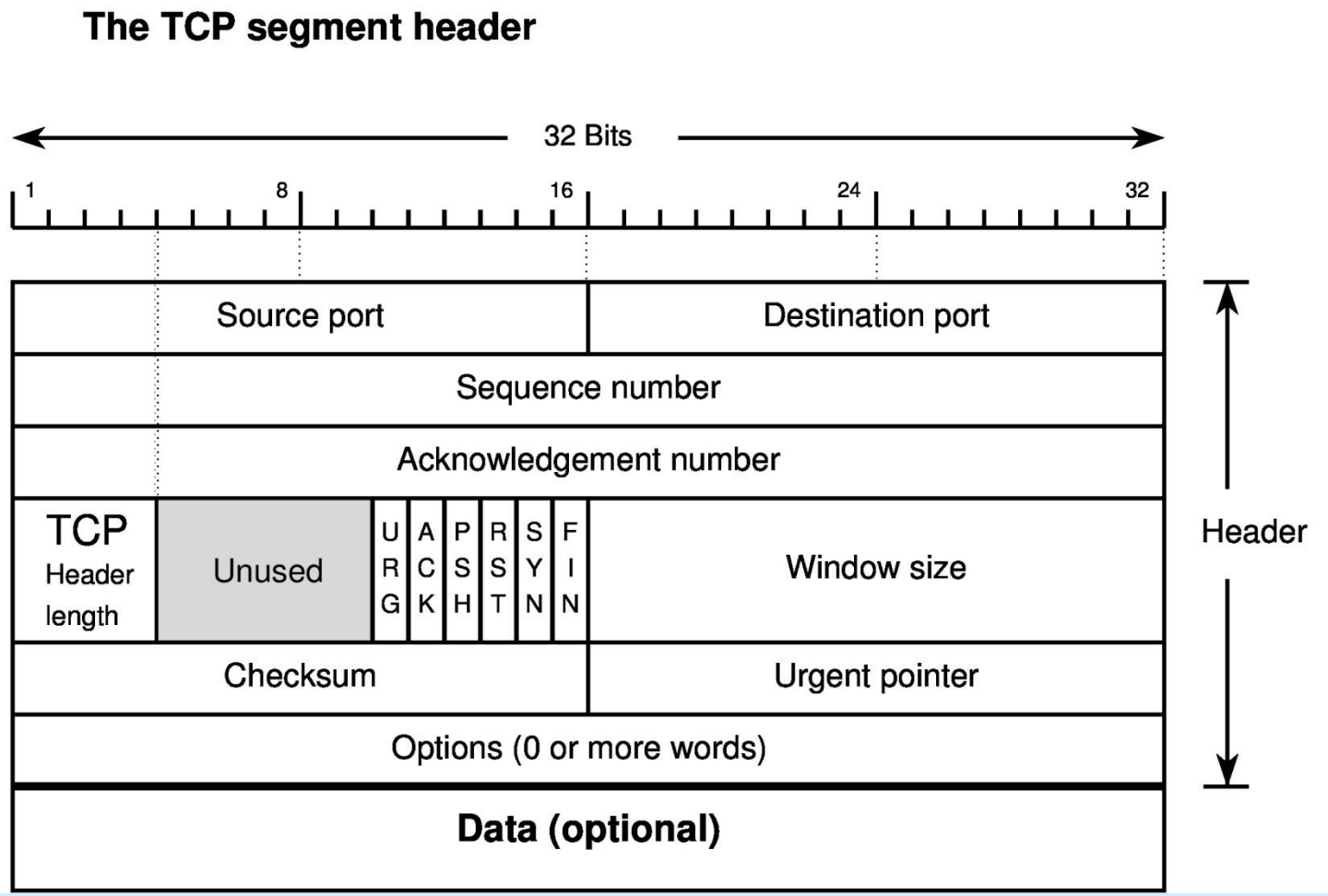
- Congestion control (host-network interaction) Prevents too much data from being injected into the network
- TCP avoids sending small packets by accumulating octets until a buffer is full or until a timer expires (default 2 ms).
- Each data byte has a sequence number!
  - Used to reassemble segments in order
- Each sequence number must be acknowledged
  - This is done by acknowledging the id of the first byte of the next TCP packet (it is indicated at its header ack. 16 bits number)
- Initial sequence numbers should be assigned randomly to minimize problems with duplicate numbers from different connections

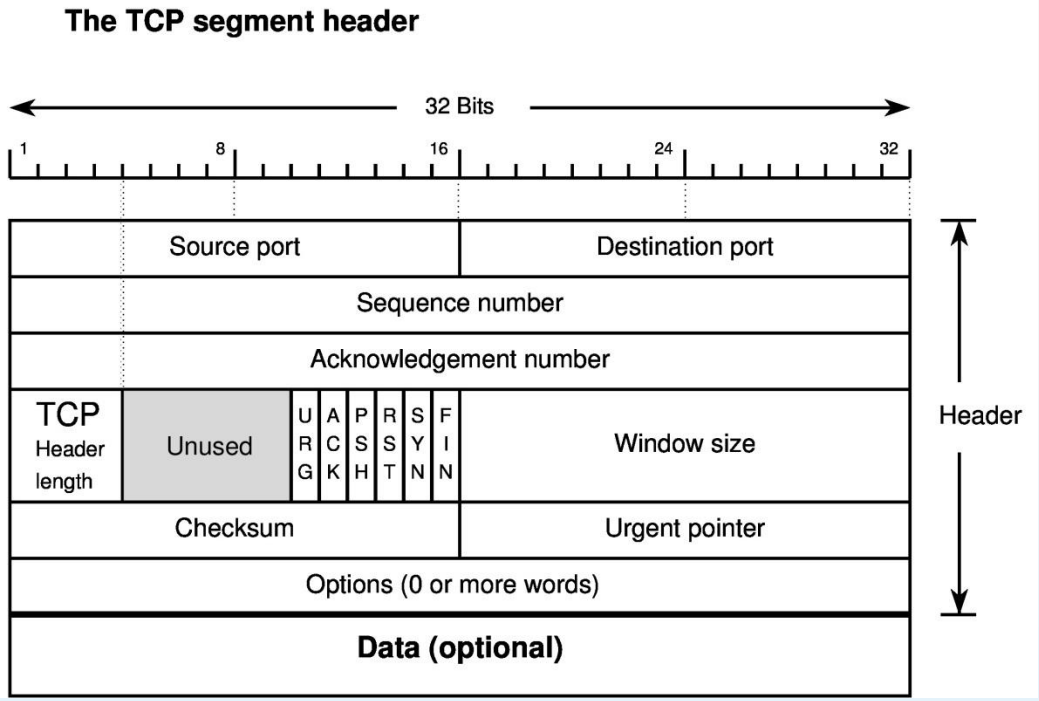
# TCP Segment Structure




In real life TCP packets are much smaller 500 bytes to 4K, and often
Just header with no data at all!

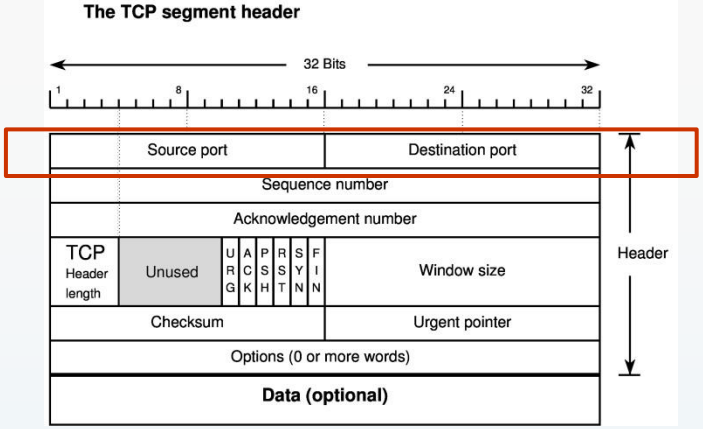
# TCP Segment Structure





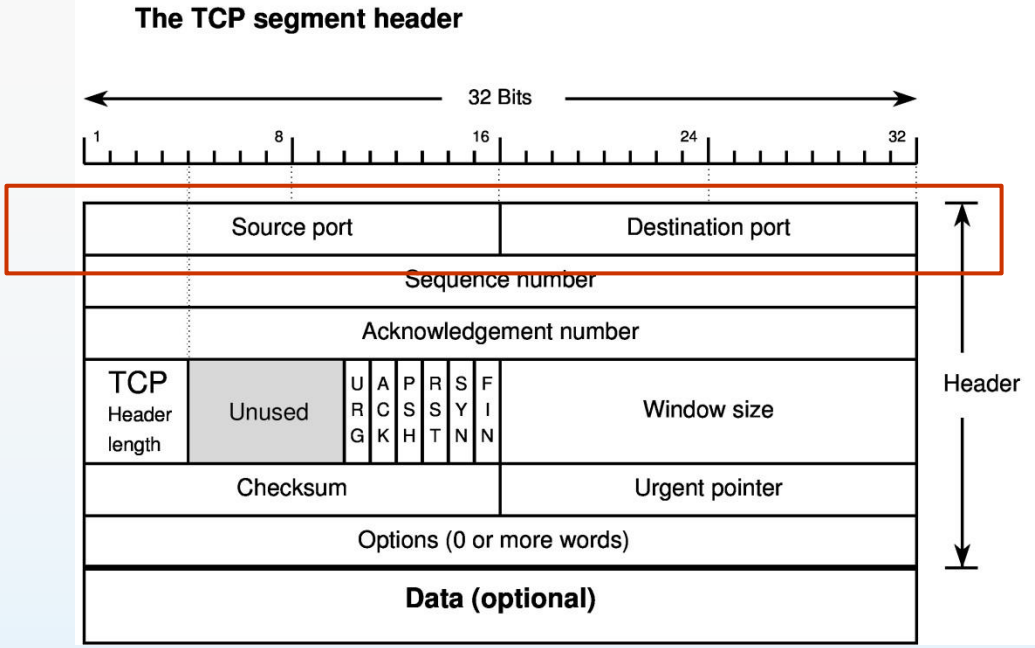

- The TCP header consists of:
  - Minimum 20-byte (5 words) of fixed-format info
  - Optional part (always an integer multiple of 4-bytes)
- The TCP Data has at most 65535-20-20 minus the options length (bytes)
  - The second -20 comes from IP header
- Thus any TCP segment can have at most 65535-20 (2^16-21) bytes in total
- However this number is usually severely limited by the network MTU (maximum transfer unit) which is usually 1500 bytes

# TCP PORTS



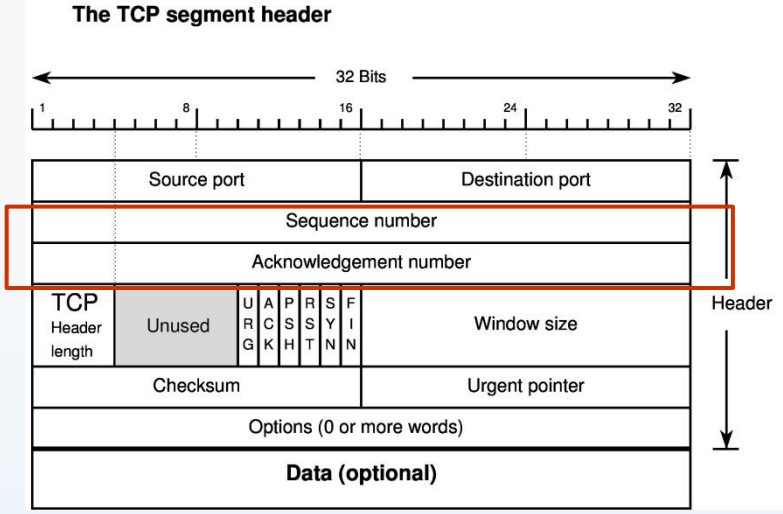

- A port is a logical address for intercrosses communication node
- Ports provide multiple destinations within one host computer, and even within the same process!
- port numbers below 256 are "well-known" ports like:
  - 21 for FTP
  - 23 for TELNET
  - 25 for SMTP
  - 80 for HTTP
  - 110 for POP3
- port numbers below 1024 are reserved for system services
  - Only the administrator (like root in Unix) is allowed to allocate them
- Port numbers from 1024 to 65535 (2**16-1) can be used by user processes without any special permission

# TCP SOCKETS



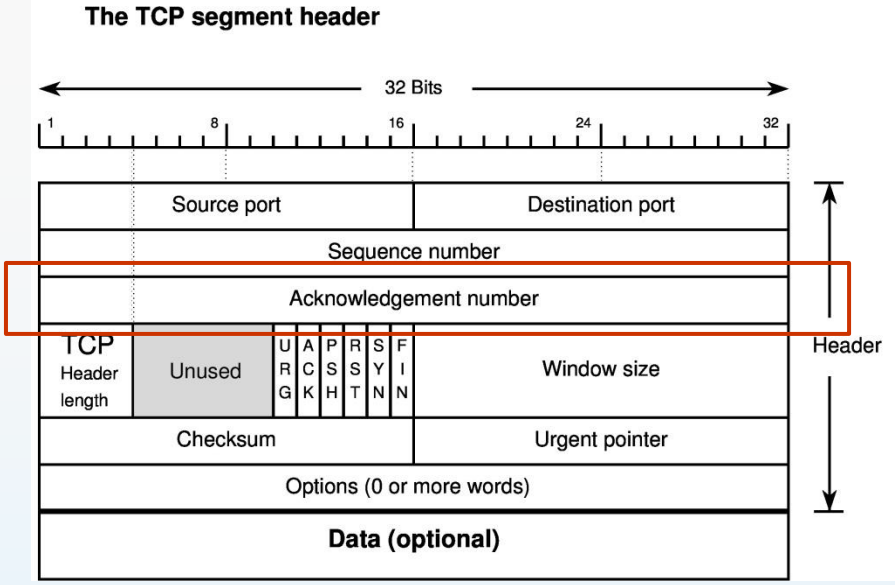

- A socket is a software object which represents a point of inter-process communication (node)
- Sometimes called: Berkeley sockets
- Sometimes called: TSAP - Transport Service Access Point
- A socket is sometimes characterized by its IP number and port number, but it has more than that (as a software unit with methods and data fields)
- Sockets provide multiple connection points within one host computer, and even within the same process!
- More on sockets in the next lecture unit

# Sequence and Acknowledgement numbers



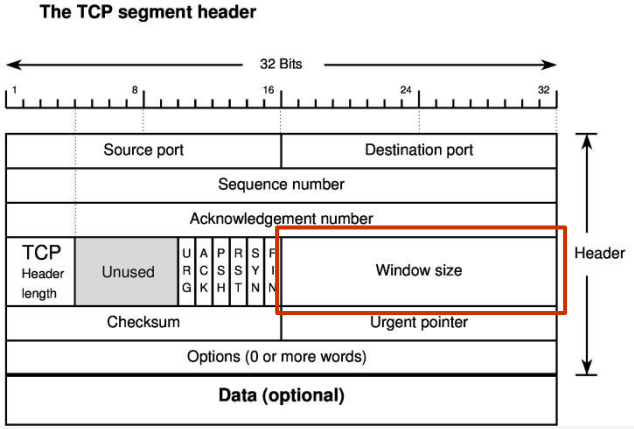

- A 32-bit number
- Every byte of the data is numbered
- The sequence number for a TCP segment is the id number of the first data byte in the segment
- It does not need to start with 1!
  - for good reasons – it better be random (after each reset)
- The range of valid sequence numbers is:
  - 0 to 4,294,967,295
  - Or: 0x0000,0000 to 0xFFFF,FFFF

# Acknowledgement Number



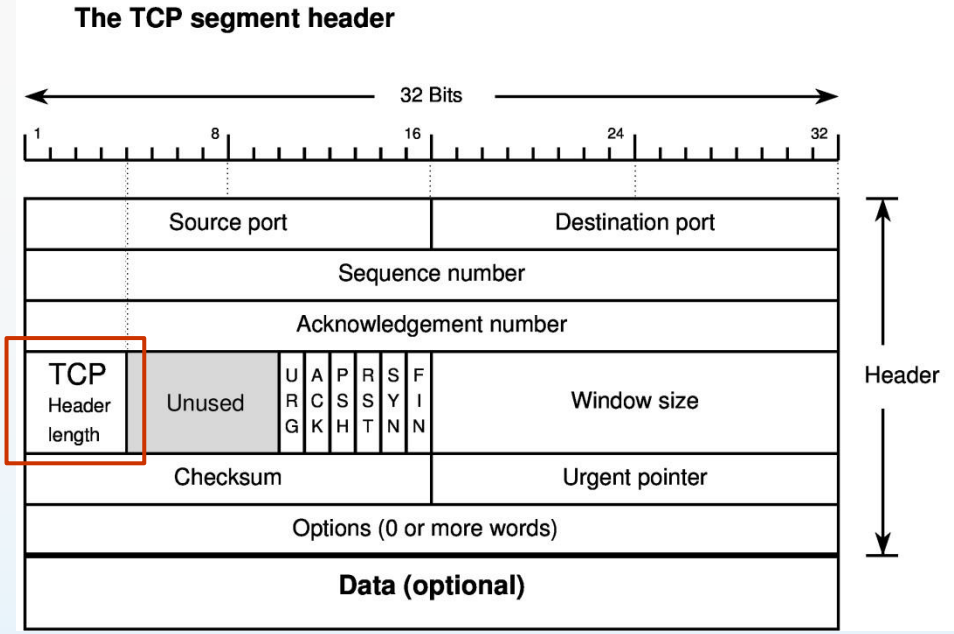

- A 32-bit number. Valid only if the ACK bit is turned on.
- Specifies the number of the next byte expected from the sender
  - **This the last byte correctly received + 1**
- Sent with data from the receiver to the sender
- By this, the receiver confirms to the sender that it has received all bytes below this number (ack. number)
- If this ack. segment does not arrive in certain time, the sender re-transmits the previous segment (timer timeout)

# 16-bit window



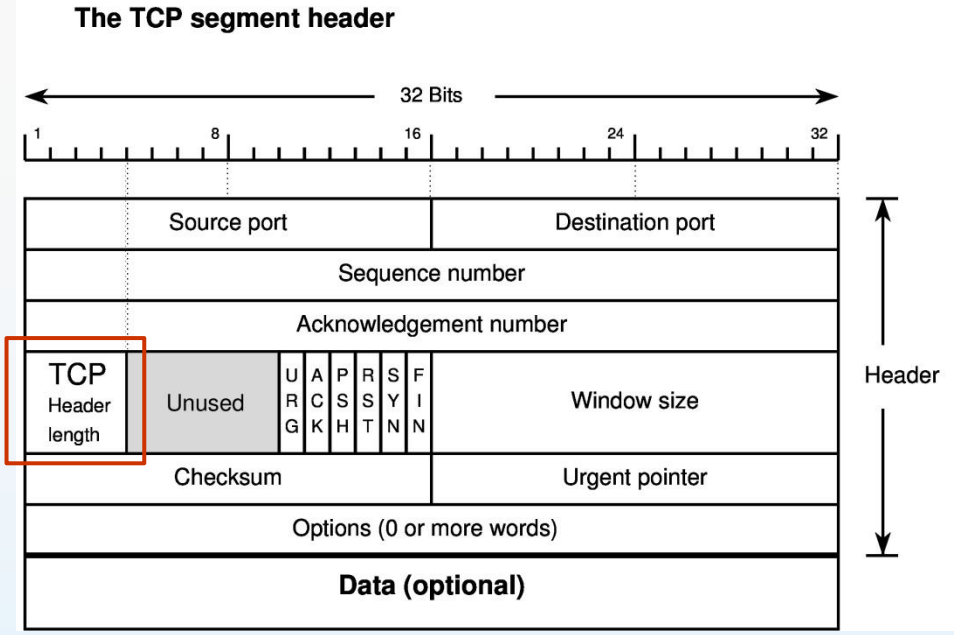

- The number of data bytes in the segment beginning with the one indicated in the acknowledgment field, which the sender of this segment is willing to receive next
- The “Acknowledgement number” field is the remaining receiver buffer size (bytes)
- Ack=0 signals that that the bytes up to acknowledgment number-1 have been received, but the receiver is incapable to accept more data at this moment
- Later, if the receiver is ready to receive more data, it sends a segment with the same acknowledgment number and a non-zero window size
- If this segment is lost, the sender re-transmits after timeout

# Header Length



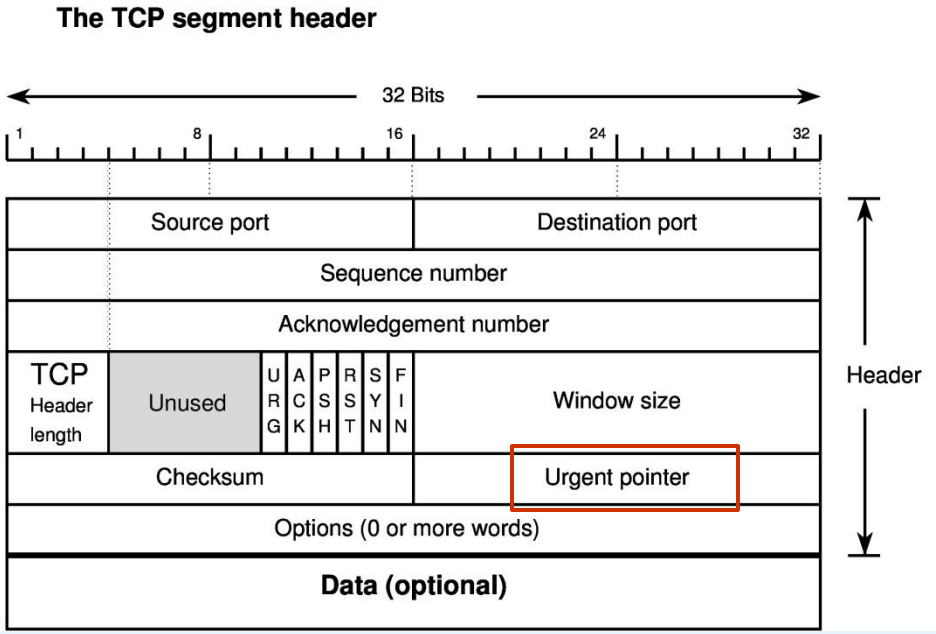

- This is the number of 32-bit in the TCP header
- This info is required since the header sometimes can be longer than 4 words
- Only 4 bits are allocated to the TCP header length field
- So it can be at most 15 words long (60 bytes)

# Checksum



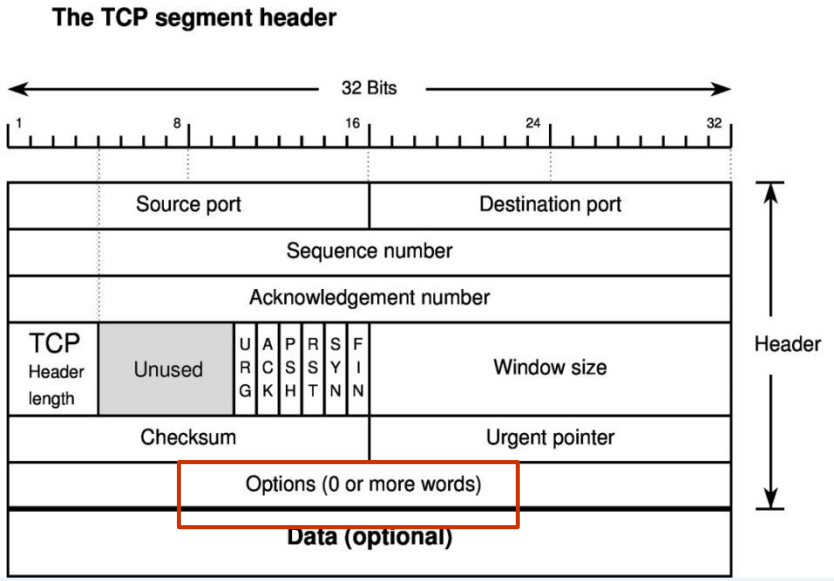

- Unlike the case of the IP datagram, checksum for TCP segment covers the whole segment including data and header
- Before computing the checksum, the algorithm zeros the checksum field and also includes a dummy IP header

# Urgent Pointer




- Points to an urgent data a byte offset from the current sequence number
- Used to signal the receiver to abort broken FTP or TELNET sessions
- seldom used

# Options

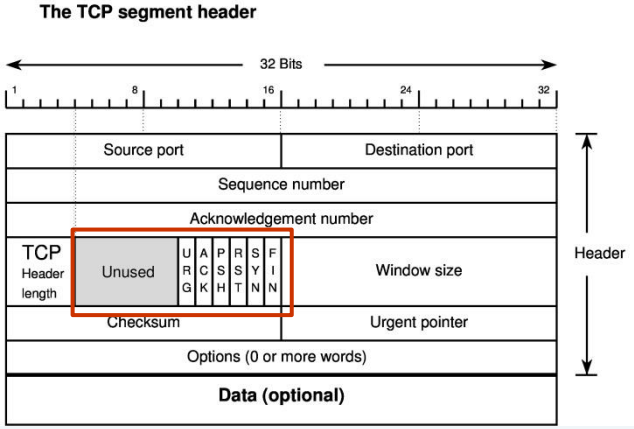



- Simpler than IP options
- TCP option format:
  - A single byte for the option type
  - A length byte
  - data bytes
- If the type requires it.
- Currently implemented options are:

| kind | length | meaning |
|---|---|---|
| 0 | - | end of option list |
| 1 | - | no operation |
| 2 | 4 | maximum segment size |

- End of option list indicates the end of the options, in case the end of the option bytes does not coincide with the end of the TCP headers
- Maximum segment size specifies the maximum segment size the sending TCP would like to receive

# BITS




- **URG=1** means the urgent pointer is a valid byte offset from the current sequence number at which urgent data are to be found (interrupt message)
  - Urgent mode is used when aborting rlogin or telnet connections, or ftp data transfers
- **ACK=1** means the acknowledgment number is valid
- **ACK=0** means there is no acknowledgment in this segment (usually no data)
- **PSH=1** then receiver should pass this data to the application ASAP
  - The receiver is requested to deliver the data to the application upon arrival and not buffer it
- **RST** - Reset (close) the connection
  - after a crash or errors (such as ack to a packet you never sent)
- **SYN** - Synchronize sequence numbers to begin a connection (see next slide)
- **FIN** - The sender has finished sending data (close)
- **Unused 6 bits** – too bad! (lots of electricity waste …)
  - at some point used to debug the protocol
  - Lately used to pass performance info between hosts

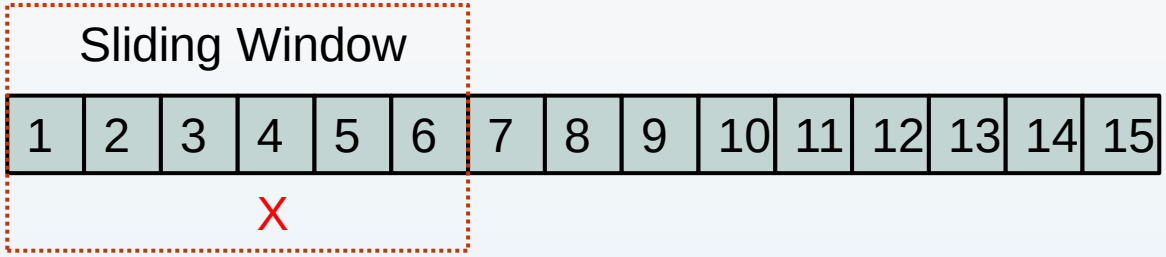
# SYN: handshake (by example)

- **Step 1:** Sender sends a TCP segment with **SYN = 1**, **ACK = 0**, and **ISN=7000** (Initial Sequence Number example)
  - **SYN** is short for **Synchronize**
  - The **ISN=7000** is the beginning of the sequence numbers for data that the sender will transmit
  - **SYN** flag announces an attempt to open a connection
- If connection established then the first byte transmitted to the receiver will have the sequence number **ISN+1**
- **Step 2.** After receiving this TCP segment, the receiver returns a TCP segment with **SYN = 1**, **ACK = 1**, **ISN = 5000** (the receiver starting sequence number), and **Acknowledgment Number = 7001**
- **Step 3.** the sender sends a TCP segment to the receiver that acknowledges the receiver's **ISN**, With flags set as **SYN = 0**, **ACK = 1**, **Sequence number = 7001**, **Acknowledgment number = 5001**
- This handshaking technique is referred to as the Three-way handshake or SYN, SYN-ACK, ACK.

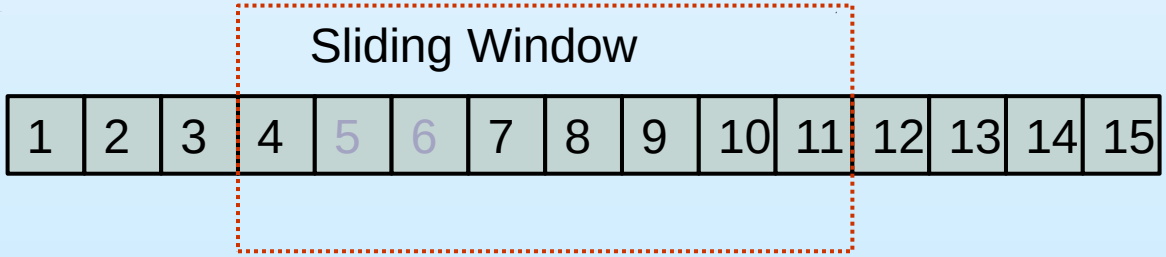
# TCP Sliding Window Algorithm

- YouTube Visualization movie
- The idea: allow sender to send multiple packets without waiting for acknowledgement
- But how many packets?
- Step 1: Send 1 packet and wait for ack.
- After getting ack. 1 from the receiver, inspect the advertised “window size”: this is the size of buffer that the receiver has for buffering packets
- Sender calculates how many packets can fit in window size and send all of them without waiting for ack. After that the sender waits for acks.
- This process repeats after getting each ack.
- Sender usually buffers window packets, since it may need to re-transmit some of them
- Receiver also need to buffer them in order to acknowledge early packets
- If the receiver’s buffer is squeezed or finished, it may advertise a very low window size which will force the sender to slow down or stop

# TCP Sliding Window Algorithm: Example

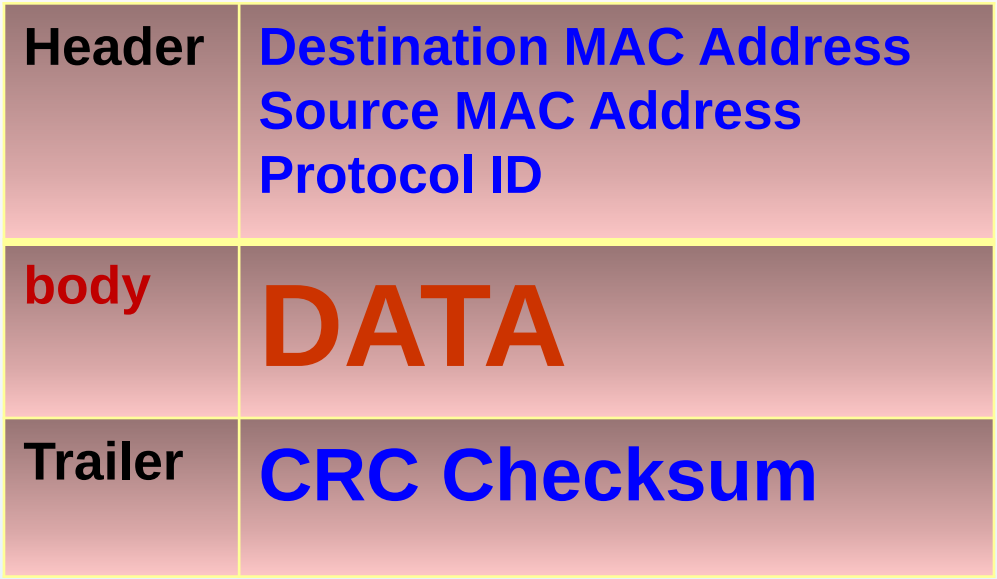



- Sender shoots 6 packets in a row with no ack. And then waits for ack. (window size is large enough to allow 6 packets )
- Receiver gets all packets except for packet 4
- Receiver sends ack. to packets 1,2,3 but cannot ack. packets 5, 6 (packet 4 was lost)
- After timeout, sender re-transmits packet 4, waits for ack. to packets 4, 5, and 6
- Receiver gets packet 4 and sends ack. for packets 4, 5, and 6
- Receiver may decrease window size of TCP header, and thus “slide” the window down



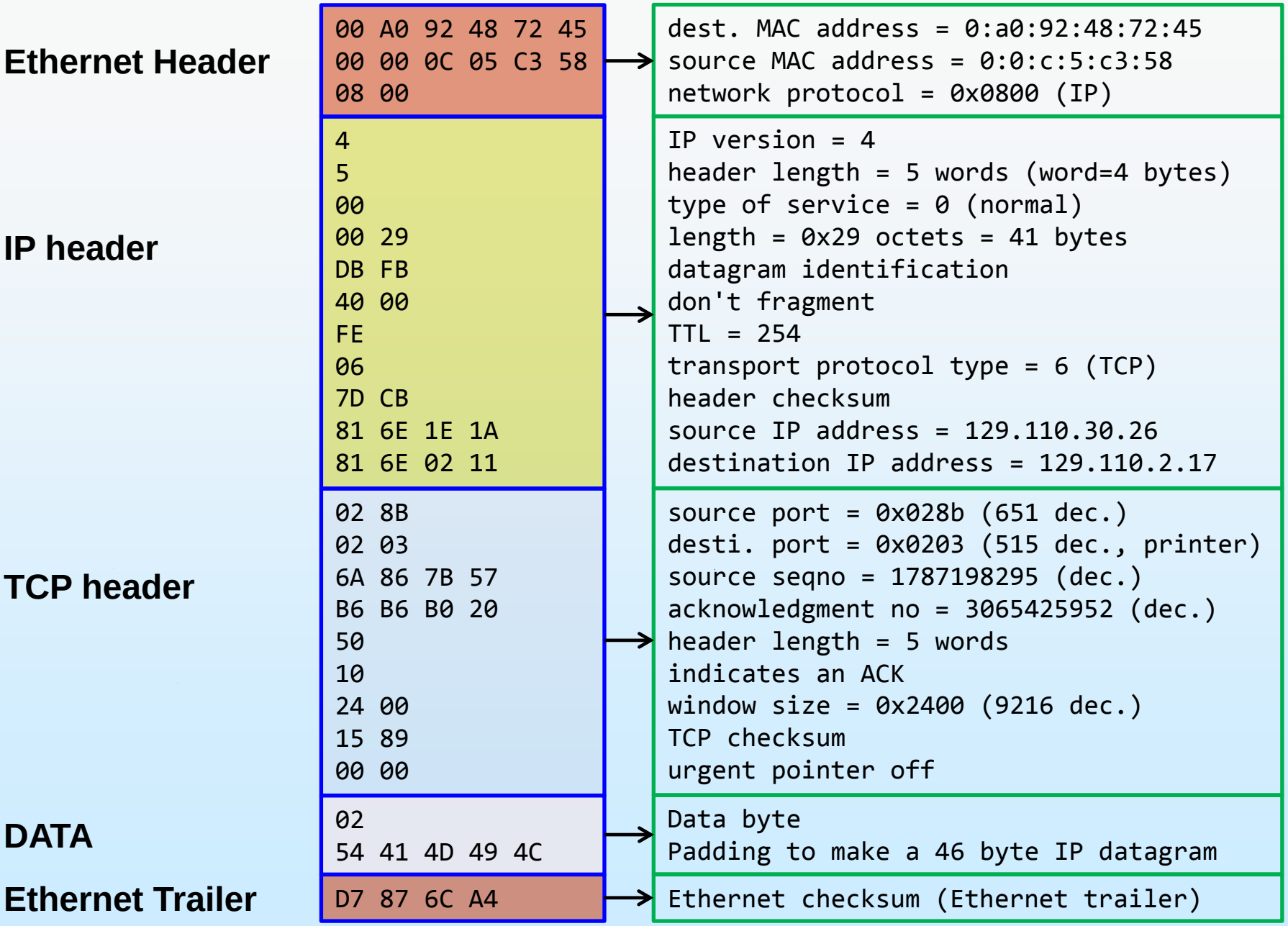

- Advanced protocols dynamically tune the window size to be suitable for both sides
- This sliding window is usually noticed when transmitting big files from one Windows machine to another, initially the time remaining calculation will show a large value and will come down later

# Ethernet Frame

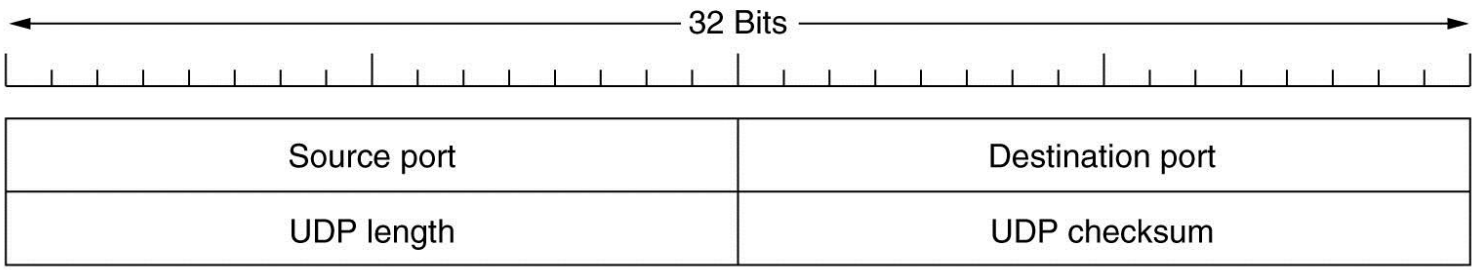



- MAC Address = Ethernet card 12 bytes id
- MAC = Media Access Control
- Example: b0:a0:92:48:72:45
- placing the CRC at the end of a frame reduces packet latency and reduces hardware buffering requirements

# A DECODED ETHERNET FRAME



# UDP Header




- Protocol number = 17
- Always 8 bytes length header
- UDP Length = Header + Data length in bytes
- Maximum length = 65515 (due to IP size limit)
- Checksum cover the full packet (header+data)
- Checksum usage is optional (usually=0)
- No flow control!
- No congestion control!
- Unreliable! (up to user processes)
- Packet order, timing, and error control are usually done at the data level
- DNS I using UDP for name resolution

# ICMP Protocol

- ICMP - Internet Control Message Protocol
- used by the operating systems to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached
- Another example: if a router receives a packet larger than the next hop MTU, it may drop the packet and send an ICMP message which indicates the condition “Packet too Big”, or it may fragment the packet and send it over the link with a smaller MTU
- ICMP can also be used to relay query messages
- It is assigned protocol number 1
- We skip the header and other details in this course (read Tanenbaum for more details)