Chapter 9
The Transport Layer

- The Internet Protocol has three main protocols that run on top of IP: two are for data, one for control
- The data protocols are complementary
  - one is fast, unreliable, connectionless: UDP
  - the other is more sophisticated, reliable and connection-oriented: TCP
- The control protocol, ICMP, we have already seen as part of the network layer

The Transport Layer

Ports

- Both UDP and TCP use the concept of *ports*
- On a single host there can be many services running, web, email, and so on: how does a client indicate which service it wants?
- This is done by ports
- A port is just a 16 bit integer: 1-65535

The Transport Layer

Ports

- Every TCP and UDP connection has a source port and destination port
- When a service starts (i.e., a program that will deal with the service starts) it *listens* on a port (i.e., it informs the operating system that it wishes to receive data directed to that port)

The Transport Layer

Ports

- The OS checks that port is not already being used, and arranges for UDP/TCP packets with that destination port to be sent to the service program
- An analogy: a host as a block of flats. To address a letter to a specific person you need both a main address (IP address) and a flat number (port)

The Transport Layer

Ports

- TCP and UDP ports are entirely separate: one service can be listening for for a TCP connection on a port and another for UDP on the same port
- The OS can distinguish the two as they are different protocols
- TCP and UDP are completely separate and do not interact at all (at the transport level)

The Transport Layer

Ports

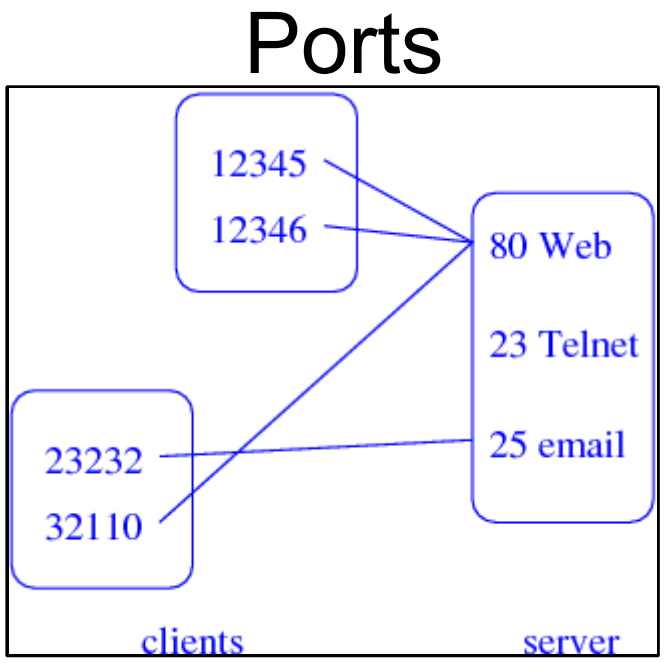
- Certain *well-known* ports are reserved for certain services
  - web server on port 80
  - email server on port 25
  - lots of others
- Some ports are reserved for privileged programs; most are available to any program that wants to use them

The Transport Layer

Ports

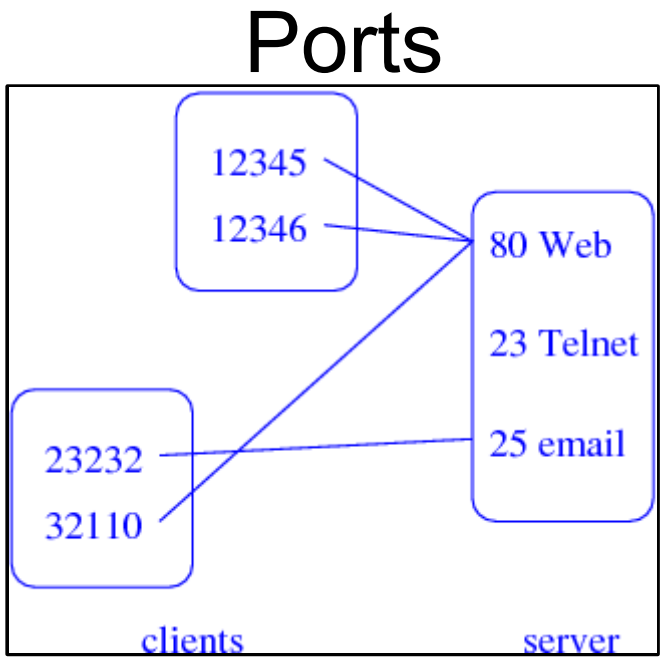
- This allocation is purely convention and for convenience only: no port is special and you can run any service on any port
- It just means you don't have the extra problem of determining the port for, say, the web server: it is almost always 80
- You can run a web server on port 25 if you wish: you will just confuse anyone who tries to send you email

The Transport Layer



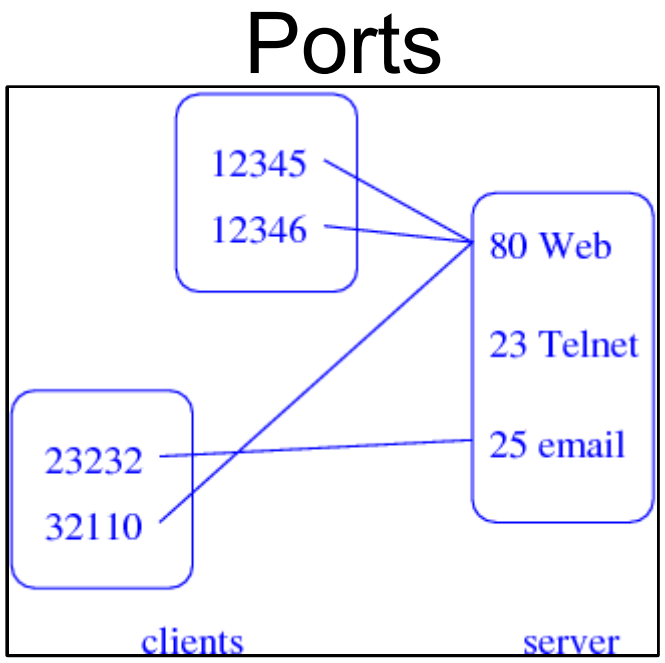
- Ports also multiple simultaneous connections between two machines, e.g., fetching several web pages

The Transport Layer



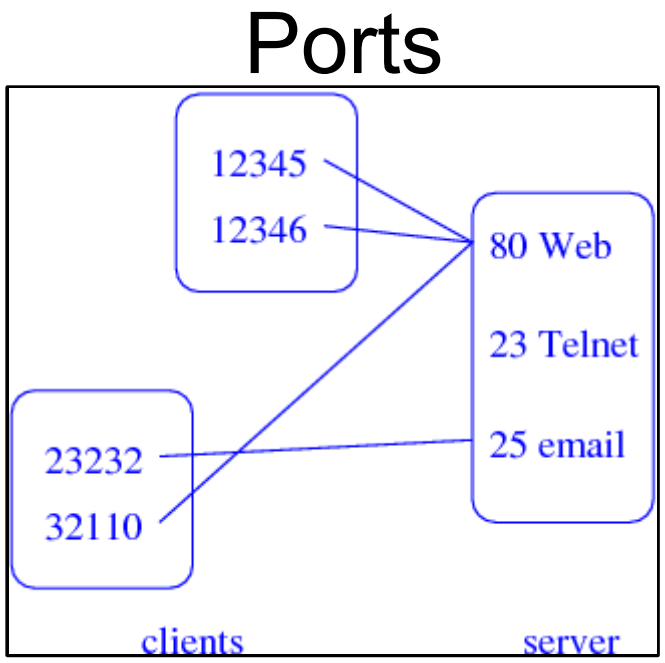
- The source port allows the OS to identify which packet belongs to which connection

The Transport Layer



- Source ports are usually chosen afresh for each new connection and are called *ephemeral* ports as they only live for the duration of the connection

The Transport Layer



- There is no technical difference between ephemeral and well-known ports, just the way they are used

The Transport Layer

Ports

- The quad

source address

source port

destination address

destination port

specifies a connection uniquely

The Transport Layer

Ports

- The pair (source address, source port) is often called a *socket*
- A full quad is often called a *socket pair*
- Both TCP and UDP have port fields early in their headers: this is so that the port number are included in the “IP header plus 8 bytes of data” that an ICMP error contains
- Thus the OS can identify which process an ICMP belongs to

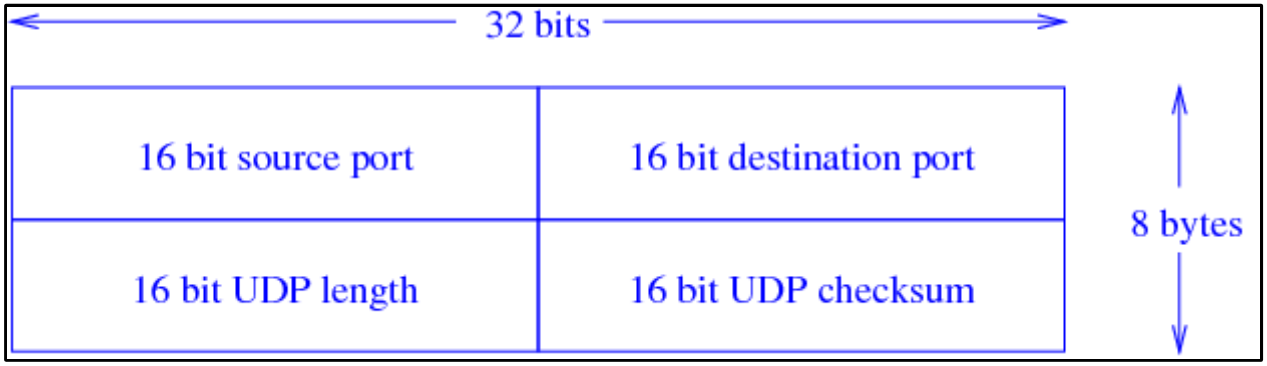
The Transport Layer

UDP

- The *User Datagram Protocol* is the transport layer for an unreliable, connectionless protocol
- Recall that “unreliable” means “not guaranteed reliable”
- UDP is not much more than IP with ports

The Transport Layer

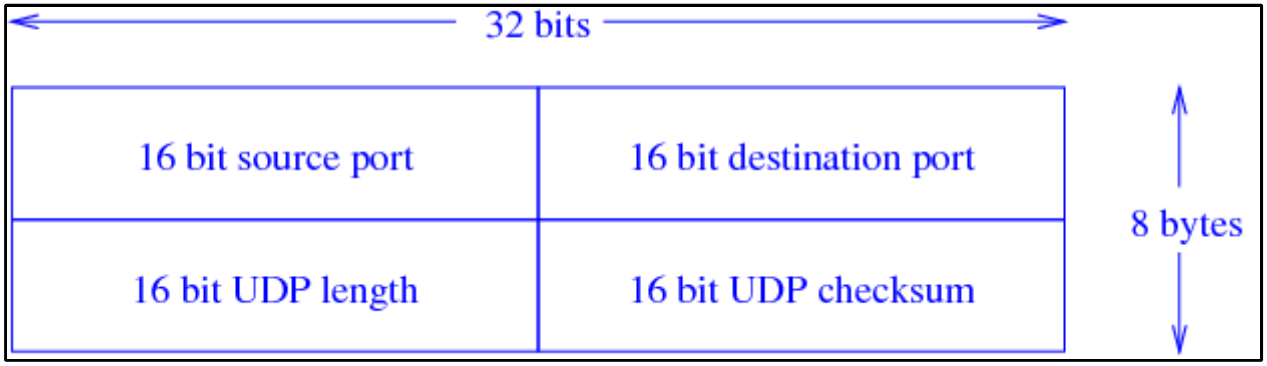
UDP



- Ports: as described
- Length: of the entire packet, including the 8 bytes of the header: could be deduced from the IP layer, but this keeps layer independence

The Transport Layer

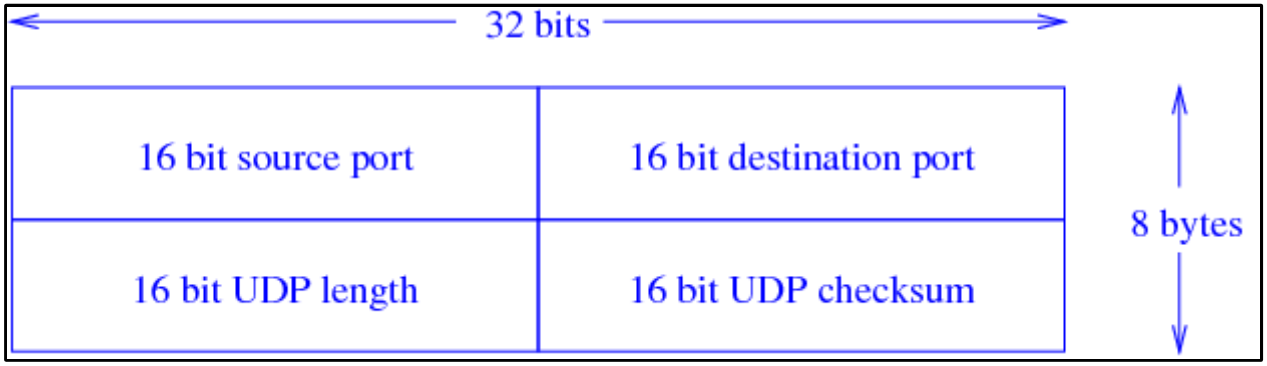
UDP



- Checksum: of the UDP header, the data and *some fields from the IP header*
- Incorporating fields from the IP header is poor design, as it ties UDP to IPv4. Changing the Network layer (e.g., to IPv6) involves changing the way this checksum is computed

The Transport Layer

UDP



- The checksum is optional: put 0 in this field if you want to save a little time

The Transport Layer

UDP

- UDP is a very thin layer on top of IP
- It is as reliable or unreliable as the IP it runs on
- It is just about as fast and efficient as IP, with only a small overhead

The Transport Layer

UDP

- UDP is widely used as it is good in two areas:
  1. One shot applications. Where we have a single request and reply. For example, DNS
  2. Where speed is more important than accuracy. For example, audio streaming ,where the occasional lost packet is not a problem, but a slow packet is

The Transport Layer

UDP

- No provision is made for lost or duplicated packets in UDP. Any application that uses UDP must deal with these issues itself
  - DNS over UDP sets a timer when a request is sent. If the reply takes too long in coming, assume the request or the reply was lost and resend. Duplicates are not a problem with DNS

The Transport Layer

TCP

- The *Transmission Control Protocol* is the transport layer for a reliable, connection-oriented protocol
- Often called “TCP/IP”
- It is hugely more complicated than UDP as it must create a reliable transport from the unreliable IP is runs on

The Transport Layer

TCP

- There is a lot of complication to deal with the error cases
- And more to improve performance and flow control

The Transport Layer

TCP ACKs

- The basis of the reliability is the use of acknowledgements (ACKs) for every packet sent
- If A sends B a packet, B must send a packet back to A to acknowledge arrival of that packet
- If A does not get an ACK, it resends its packet

The Transport Layer

TCP ACKs

- This raises the *Two Army Problem*: two armies A and B wish to coordinate an attack on C
- A sends a message to B: “attack at dawn”
- How does A know that B got the message? A cannot safely attack until it knows B is ready
- So B sends an acknowledgement to A: “OK”
- But the ACK might be intercepted and A might not get the ACK

The Transport Layer

TCP ACKs

- B can't attack until it knows A got the ACK
- So A has to send and ACK for the ACK back to B
- But this might not get through...

The Transport Layer

TCP ACKs

- TCP avoids the Two Army Problem by using packet retransmissions
- A starts a *retransmission timer* when it first sends to B
- If the timer runs out before it gets an ACK, it resends
- Repeat until A gets an ACK

The Transport Layer

TCP

- Problems to solve include:
  - how long to wait before a resend? This might be a slow but otherwise reliable link and resending will just clog the system with extra packets
  - how many times to resend before giving up? It might be the destination has gone away entirely

The Transport Layer

TCP

- Problems to solve include:
  - how long B should wait before sending the ACK? You can piggyback an ACK on an ordinary data packet, so it may be better to wait until some data is ready to be returned rather than sending an empty ACK. This saves on packets sent
  - IP packets can arrive out of order, so we need some way to recognise which ACK goes with which packet

The Transport Layer

TCP

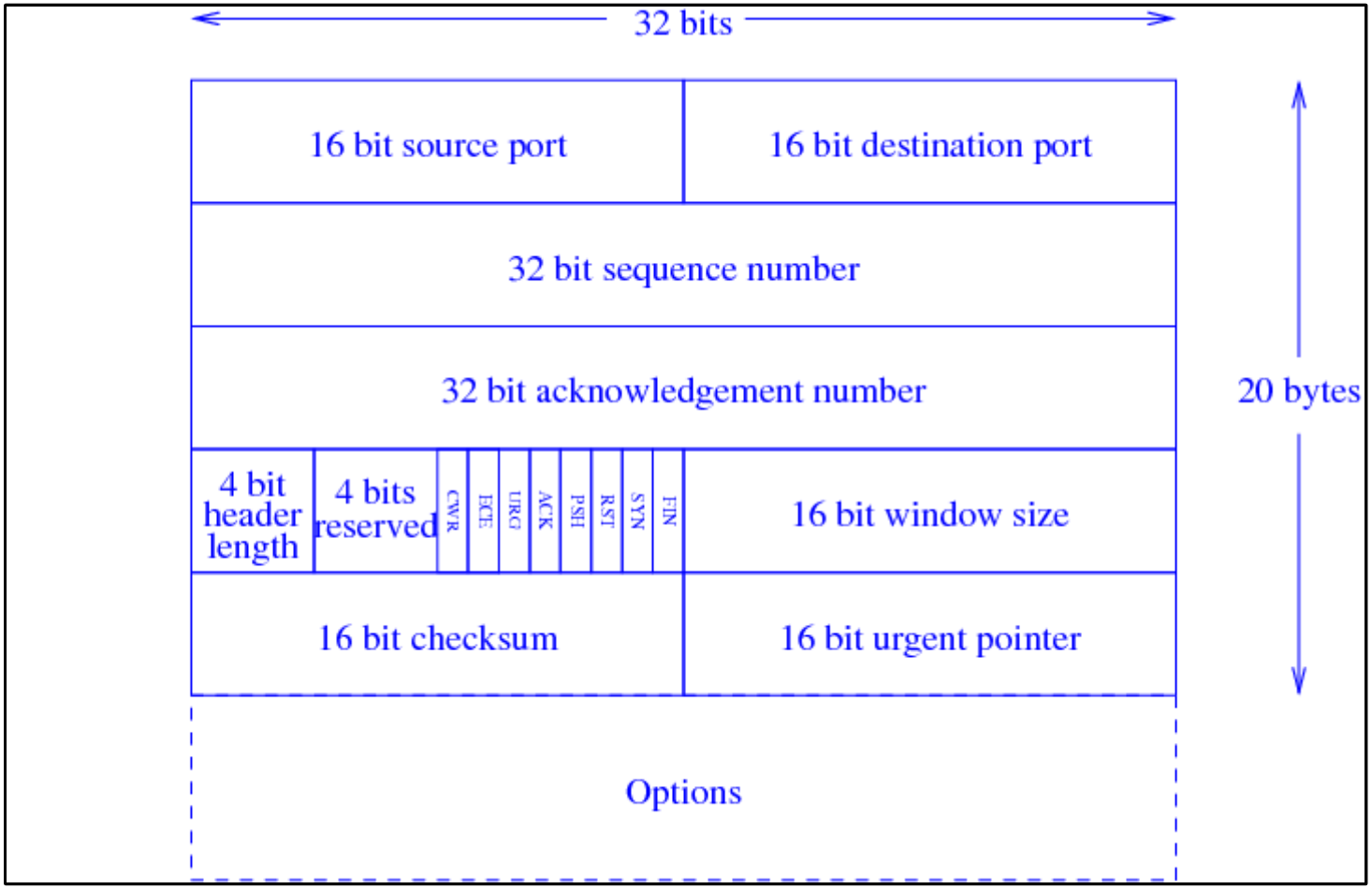
- Problems to solve include:
  - how to maintain order in the data? IP packets can arrive out of order, so we need some way of reassembling the original data stream
  - how to manage duplicates? Resends can duplicate packets so we need some way to recognise and discard extra copies
  - Flow control: how to increase the rate of sending packets when things are going well, and decrease the rate when they are not

The Transport Layer

TCP

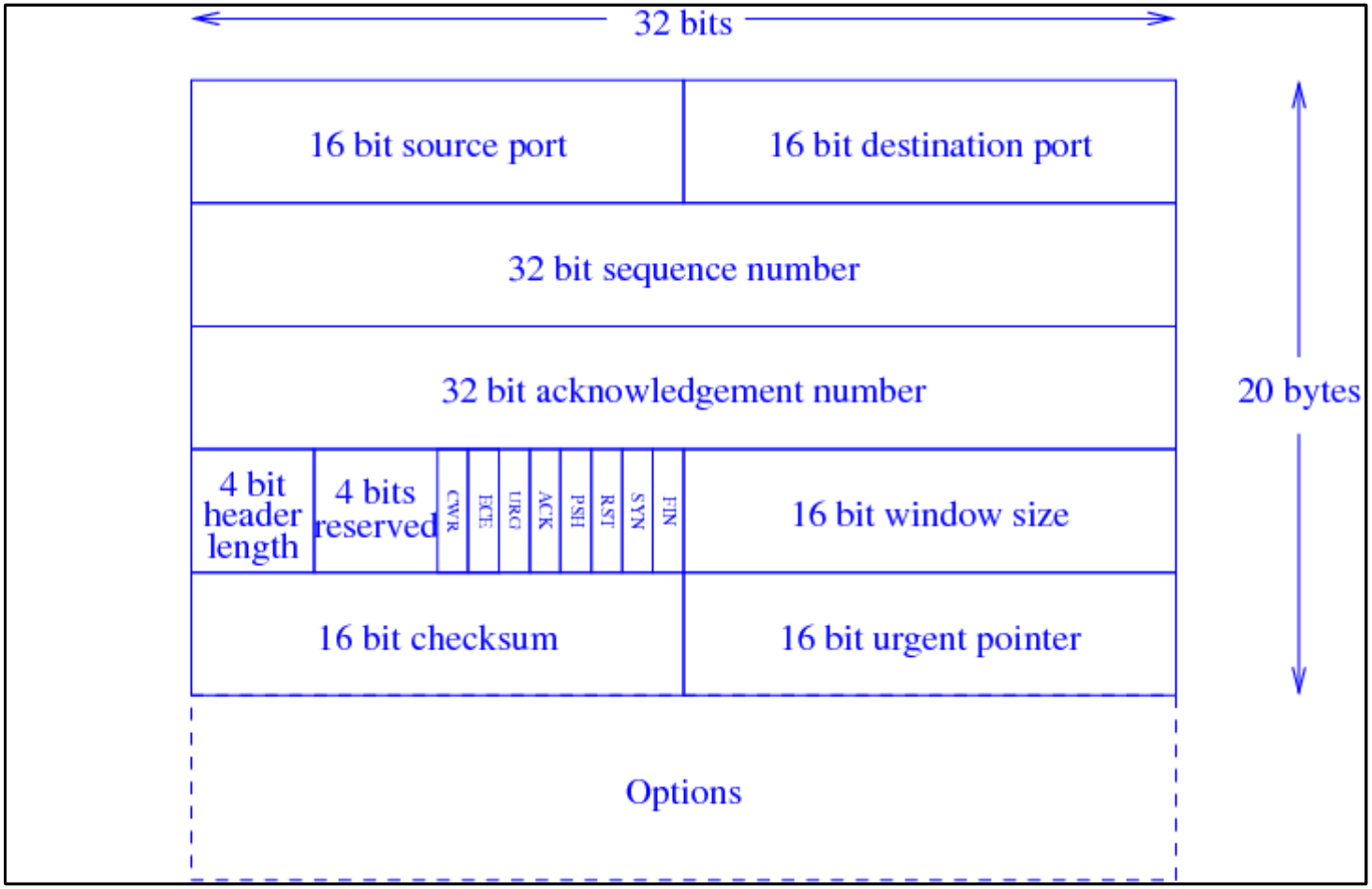
- TCP packets are often called *segments*
- TCP headers are complicated as they must address many complex issues

The Transport Layer



- Two 16 bit ports, identical to UDP (on purpose)

The Transport Layer



- Two 32 bit values: *sequence* and *acknowledgement*

The Transport Layer

Sequence numbers

- These numbers are the heart of TCP's reliability
- Every byte in a TCP connection is numbered
- The 32 bit sequence number starts at some random value and increases by 1 for each byte sent
- So if a segment contains 10 bytes of data, the sequence on the next segment will be 10 greater

The Transport Layer

Sequence numbers

- The sequence number in the header is the number of the first byte of data in the segment
- The destination acknowledges those bytes it has received by setting the ACK field
- The ACK field is only active if the ACK flag (see below) is set
- The reverse connection from destination to source has its own sequence number as TCP is fully duplex

The Transport Layer

Sequence numbers

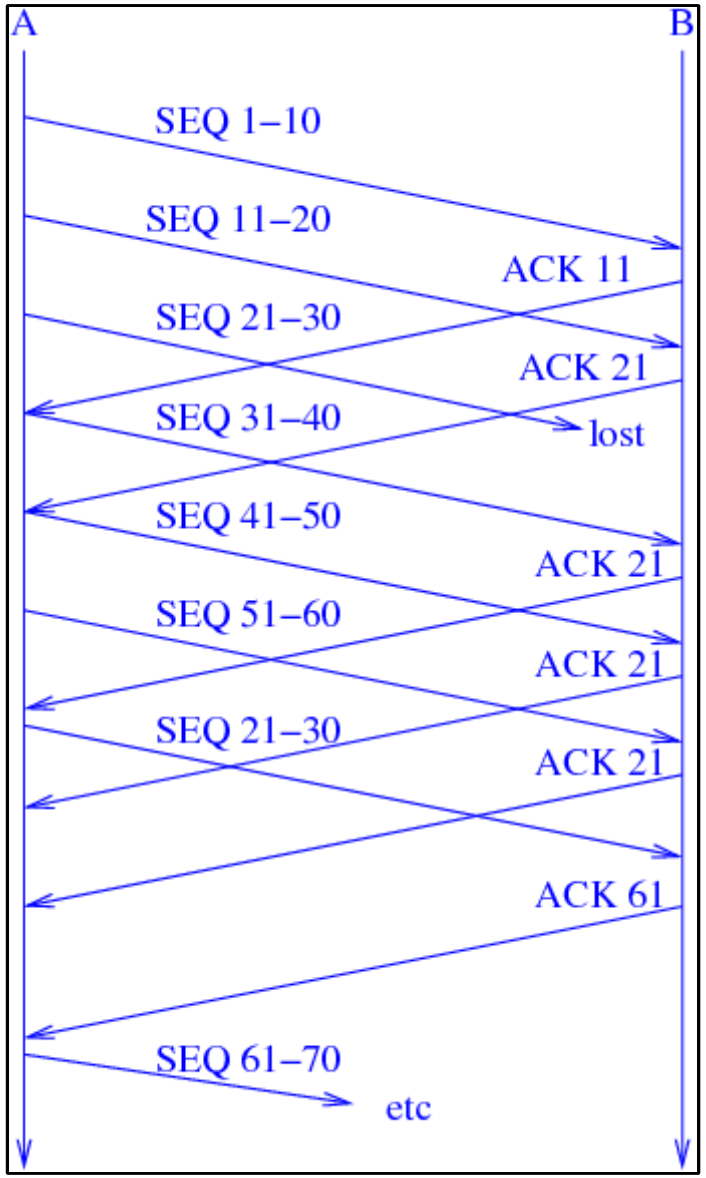
- Note that a destination might not immediately get the whole segment that was sent due to fragmentation in the IP layer
- In this case TCP must reconstruct the segment before it can send the ACK

The Transport Layer

Sequence number

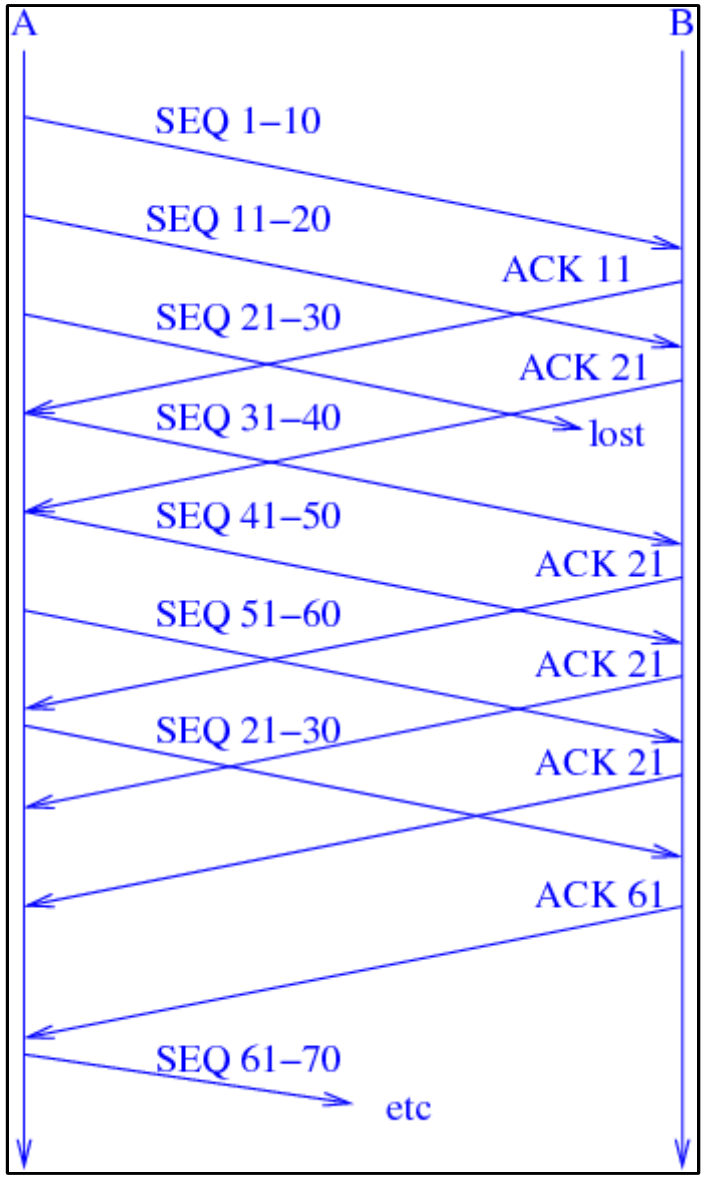
- If the sequence number is 10000 and 10 bytes are received, the ACK is the sequence number of the next bytes the destination expect to receive, i.e., 10010
- ACKs can be *piggybacked* on normal returning data packets
- This helps reduce the amount of network traffic

The Transport Layer



- A is sending 10 byte segments to B, and B is ACKing them
- The segment containing bytes 21-30 is lost
- When B next gets a segment it still ACKS with 21: that's the byte it wants next

The Transport Layer



- While the ACK travels back to A, A is still sending new data
- Eventually A gets *duplicate ACKs* from B: this is a sign of a problem
- A resends bytes 21-30
- When B gets these bytes it can ACK all the way up to 60

The Transport Layer

Seqs and ACKs

- There is much more on ACKs later
- The sequence number wraps around after

$$2^{32}-1 = 4294967295 \text{ bytes}$$

- This is under 10 seconds for a 10Gb/s Ethernet
- Additional mechanisms have had to be devised in the light of modern fast networks: see PAWS later