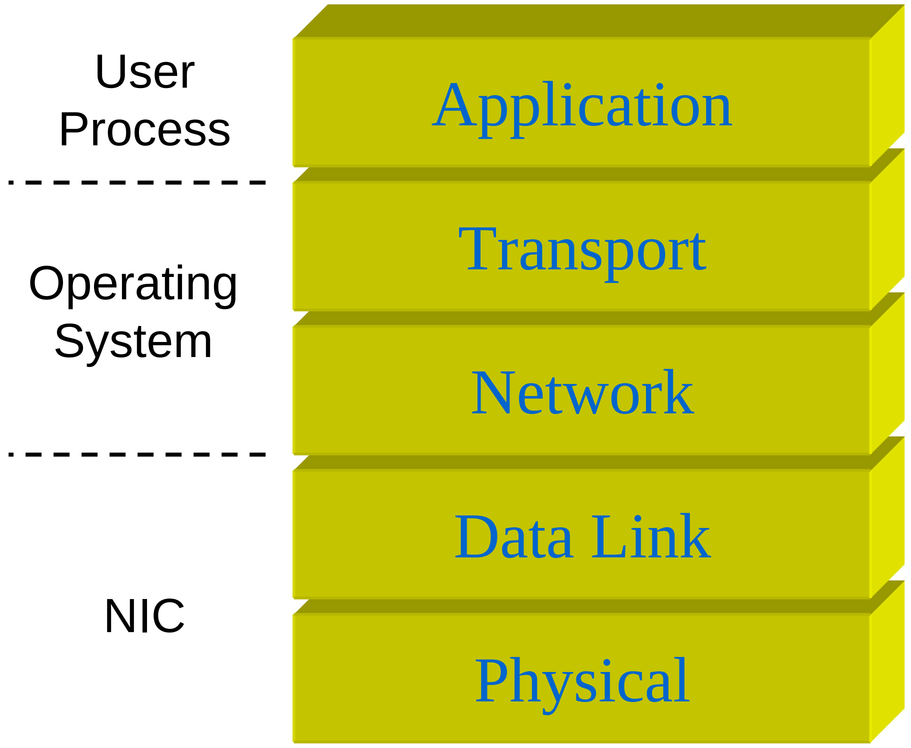
Transport Protocols

Dr Steve Gordon

ICT, SIIT

Contents

Where Are We?



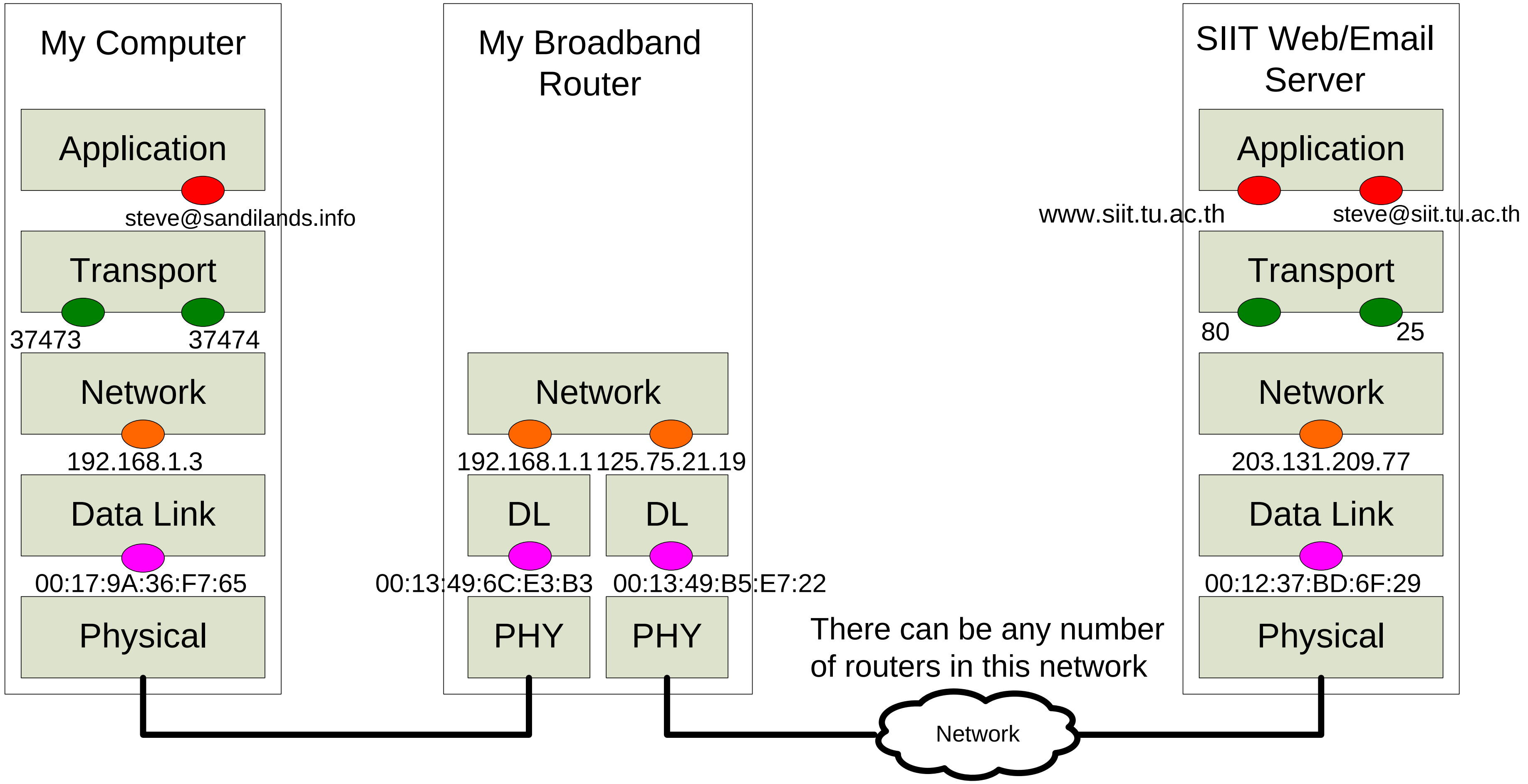
Send data between source and destination processes/applications

IP: joins the networks together, and finds paths from source to destination interfaces

Many different types of networks
LANs/WANs: Ethernet, ADSL, ATM, Frame Relay, SDH/SONET, PDH, Telephone network, Satellite, Wireless LAN, Mobile Phone, USB, Firewire, …

Host-to-Host Communications

Transport layer deals with sending data between processes/applications on source host and destination host. Often referred to as End-to-End communications.



Transport Protocols

- Transport protocols can be connection-oriented or connection-less
  - Connection-less: send data from a source process to a destination process usually with no concern about reliability, flow control. Simple.
    - IP is a connection-less network layer protocol
    - UDP is a connection-less transport layer protocol. Very simple.
      - If your application wants to send data fast and can cope with errors, then use UDP
  - Connection-oriented
    - Most transport protocols are connection-oriented
      - TCP is a connection-oriented transport layer protocol.
    - Include functions for setting up connection, error control, flow control, addressing. Very complex.
    - Since IP is connection-less, connection-oriented transport protocols are very important:
      - Provide the reliability that IP does not provide
      - Most applications need reliability, and hence use TCP
  - Our focus is on Internet technologies, so:
    - We will describe TCP and then quickly describe UDP

Transport Protocols

- Internet transport protocols (standardised by the IETF)
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
  - Stream Transmission Control Protocol (SCTP)
- Other transport protocols (standardised and experimental)
  - OSI: TP-0 to TP-4
  - Transaction-based: T/TCP, WAP (transaction layer)
  - IBM SNA: NetBEUI
  - AppleTalk: ATP
  - Secure transport protocols
  - Transport protocols for high-speed networks
  - Transport protocols for wireless networks

Transport Protocol Services

- The main functions of a transport protocol
- Common to both TCP and UDP
  - Addressing
  - Multiplexing
- Specific to TCP (and other connection-oriented transport protocols)
  - Connection Management
  - Error control
  - Flow control
  - Congestion control
- We will assume that the network layer (IP) is unreliable:
  - Datagrams can be lost (sent, but not arrive at destination)
  - Datagrams can arrive, but with errors
  - Datagrams can arrive in a different order than they were sent
  - Datagrams can be duplicated (one datagram sent, two copies arrive)

Addressing and Multiplexing

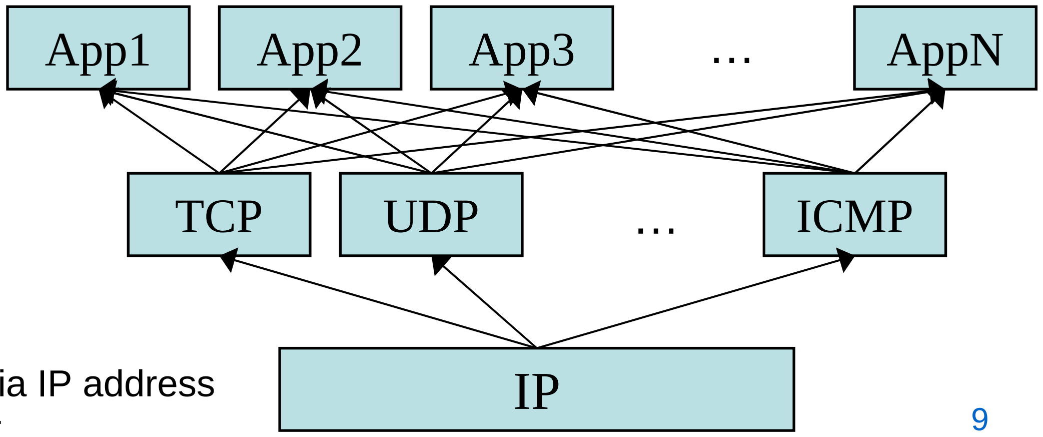
Ports and Multiplexing

- A datagram is sent to a computer on the Internet based on the destination IP address
  - IP addresses identify network interfaces on computers
    - E.g. your PC may have one IP address for its Ethernet interface; your Laptop may have an IP address for Ethernet and another for wireless LAN; a router will have one IP address for each LAN/WAN interface
- The IP datagram also contains a Protocol Number to determine which protocol will handle the data
- Once the transport protocol has the data, how does it know which application it is destined to?
  - Port numbers

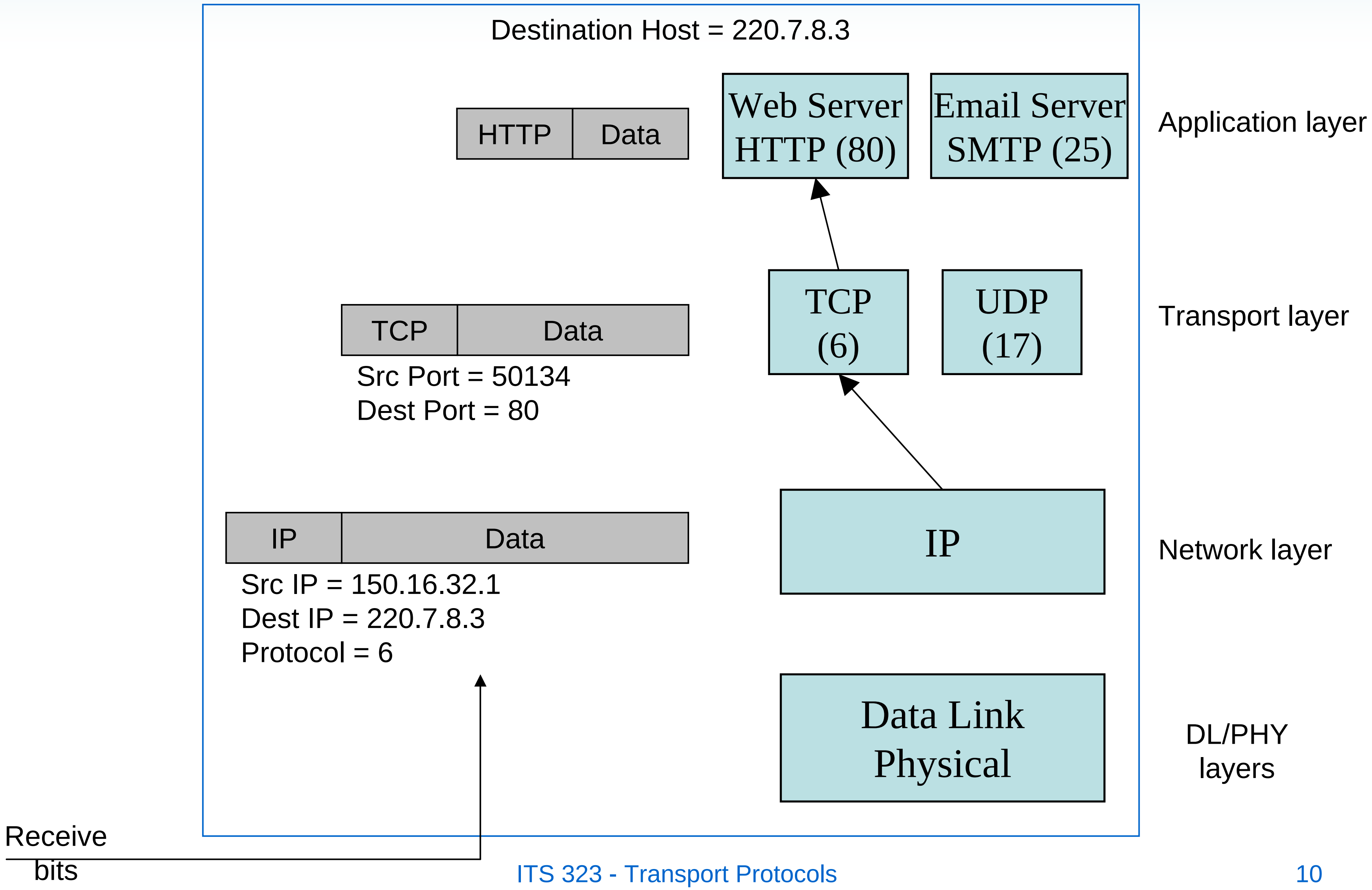
Identify the Application by Port Number in Transport header

Identify the Transport protocol by Protocol number in IP header

Identify the IP interface via IP address in IP header



Ports and Multiplexing



Ports and Multiplexing

- Multiplexing
  - Use of port numbers to identify applications allows traffic from multiple applications to be sent over the one interface
  - At the sender: When an application sends data to the Transport Protocol, the destination port number is identified and included in the Transport Protocol header
  - At the receiver: When the Transport Protocol receives data, it determines which application to send it to based on the destination port number in the Transport Protocol header
  - The Transport Protocol header also contains the source port number (so a response can be sent)
- Port numbers
  - IANA defines Port numbers for Internet applications
    - Well Known Ports: 0 to 1023
    - Registered Ports: 1024 to 49151
    - Private Ports: 49152 to 65535
  - Examples: Web server (80), Email server (25), FTP Server (21), Telnet (23)
    - Client applications generally have an unused port in the Private Port range chosen by the operating system
- Hence, a Transport Connection is uniquely identified by:
  - source port, source IP address, destination port, destination IP address

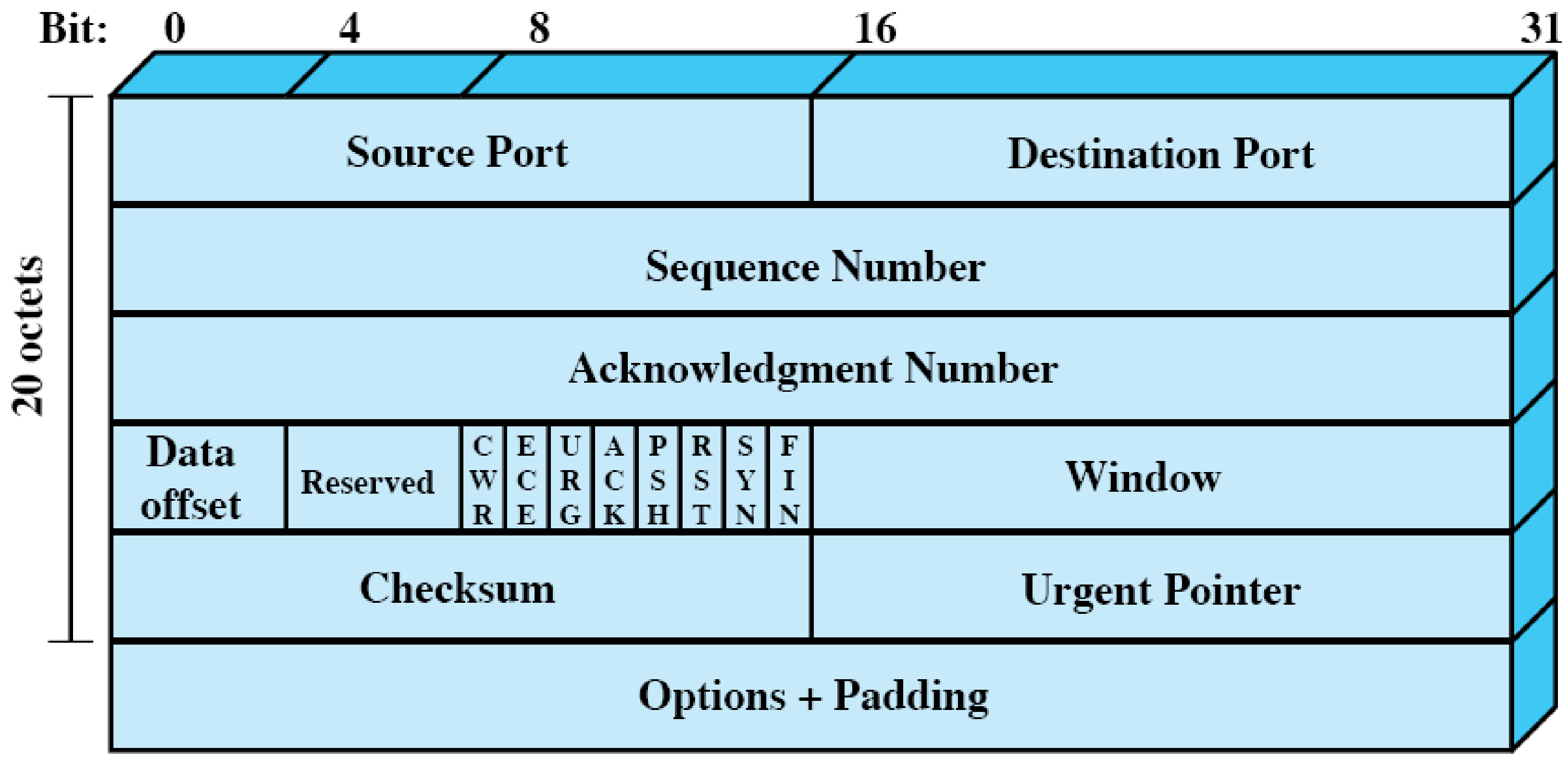
Transmission Control Protocol (TCP)

TCP

- The most commonly used transport protocol today
  - Almost all Internet applications that require reliability use TCP
    - Web browsing, email, file sharing, instant messaging, file transfer, database access, proprietary business applications, some multimedia applications (at least for control purposes), …
- TCP provides a reliable, stream-oriented transport service:
  - Stream of bits (or bytes) flow between end-points
    - Stream is unstructured
  - Virtual circuit connection
    - Set up a connection before sending data
  - Buffered transfer
    - Applications generate any sized messages
    - TCP may buffer messages until large datagram is formed
    - Option to force (push) the transmission
  - Full duplex connection
    - Once the connection is setup, data can be sent in both directions
  - Reliability
    - Positive acknowledgement with retransmission

TCP Header Format

- Header contains 20 bytes
  - Additional options are possible, must be padded out to multiple of 4 bytes
- TCP Segment = Header + Data



TCP Header Format

- **Source/Destination port**: 16 bit port number of the source/destination
- **Sequence number** of the first data byte in this segment
  - Unless the SYN flag is set, in which case the sequence number is the Initial Sequence Number (ISN)
- **Acknowledgement number**: sequence number of the next data byte TCP expects to receive
- **Data offset**: Size of header (measured in 4 bytes)
- **Reserved** for future use
- **Window** contains the number of bytes the receiver is willing to accept (for flow control)
- **Checksum** for detecting errors in the TCP segment
- **Urgent pointer** points to the sequence number of the last byte of urgent data in the segment
- **Options**: such as maximum segment size, window scaling, selective acknowledgement, …

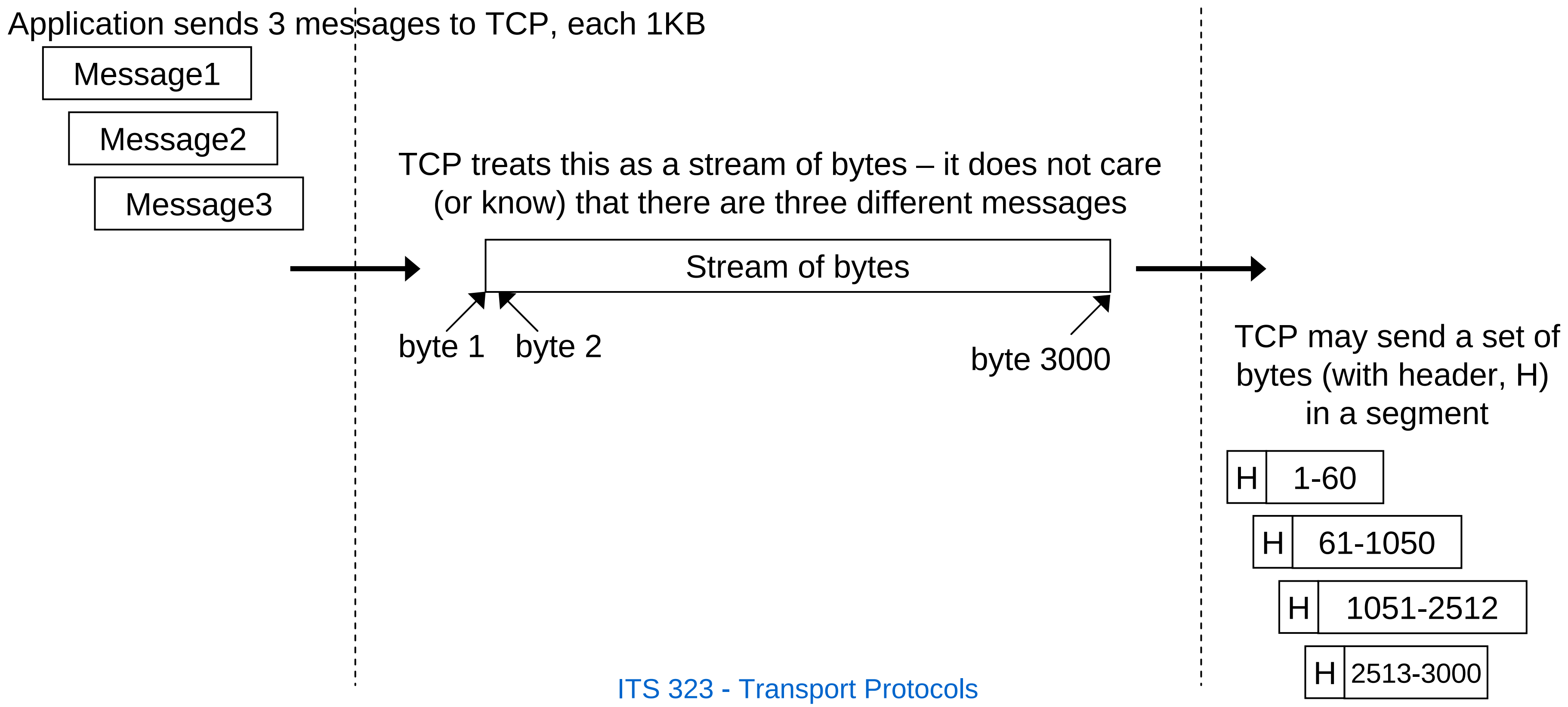
TCP Header Format

- Flags (1 bit each, if 1 the flag is true or on):
  - CWR: Congestion Window Reduced
  - ECE: Explicit Congestion Notification Echo
    - CWR and ECE are used on a special congestion control mechanism – we do not cover this in ITS 323
  - URG: segment carries urgent data, use the urgent pointer field
  - ACK: segment carries ACK, use the ACK field
  - PSH: push function
  - RST: reset the connection
  - SYN: synchronise the sequence numbers
  - FIN: no more data from sender

- Note
  - There is only one type of TCP packet
    - However the purpose of that packet may differ depending on the flags set
    - If SYN flag is set, we may call it a “SYN packet or TCP SYN”
    - If the ACK flag is set, we may call it a “ACK packet”
    - If the packet carries data, we may call it a “DATA packet”
    - If the packet carries data and the ACK flag is set, it is both a DATA and ACK packet

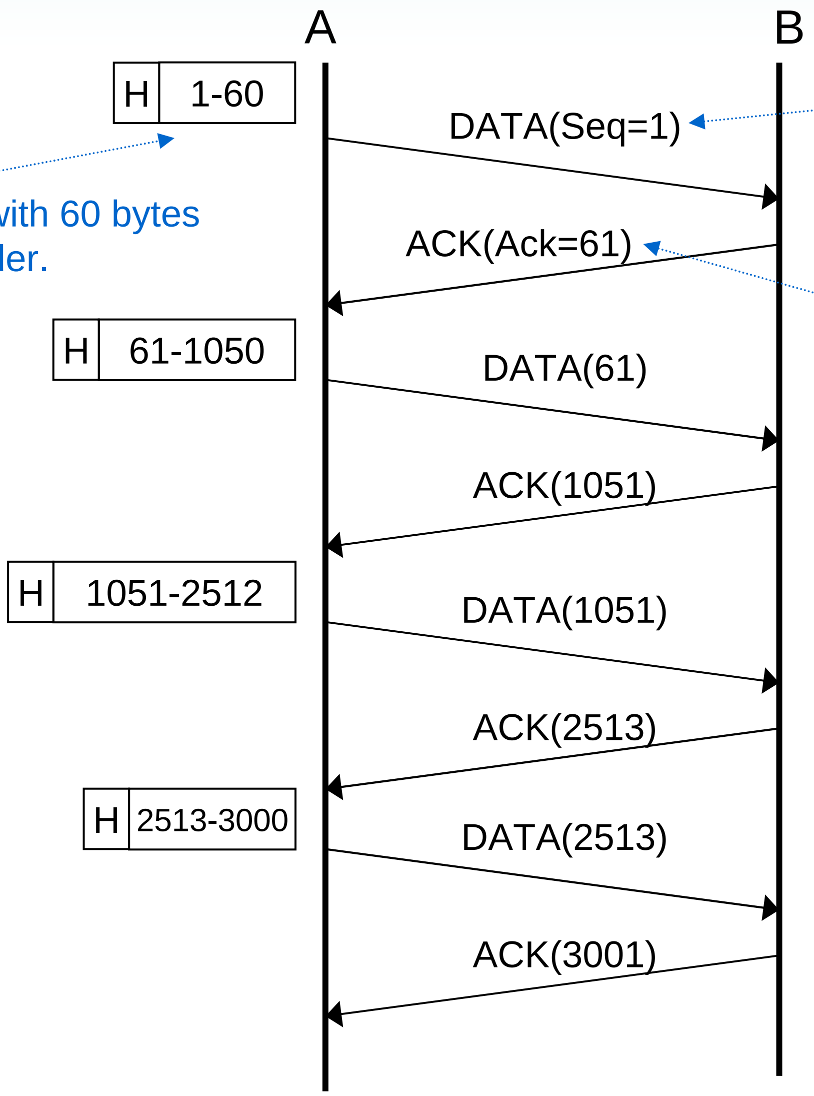
Segments, Bytes and Sequence Numbers

- TCP messages are called *segments*
- But TCP operates on a stream of bytes
  - Sliding windows and sequence/ACK numbers refer to bytes (not segments or messages)



Application sends 3 messages to TCP, each 1KB
Message1
Message2
Message3
TCP treats this as a stream of bytes – it does not care (or know) that there are three different messages
Stream of bytes
byte 1
byte 2
byte 3000
TCP may send a set of bytes (with header, H) in a segment
H 1-60
H 61-1050
H 1051-2512
H 2513-3000

Segments, Bytes and Sequence Numbers



A TCP segment with 60 bytes of data, plus header.

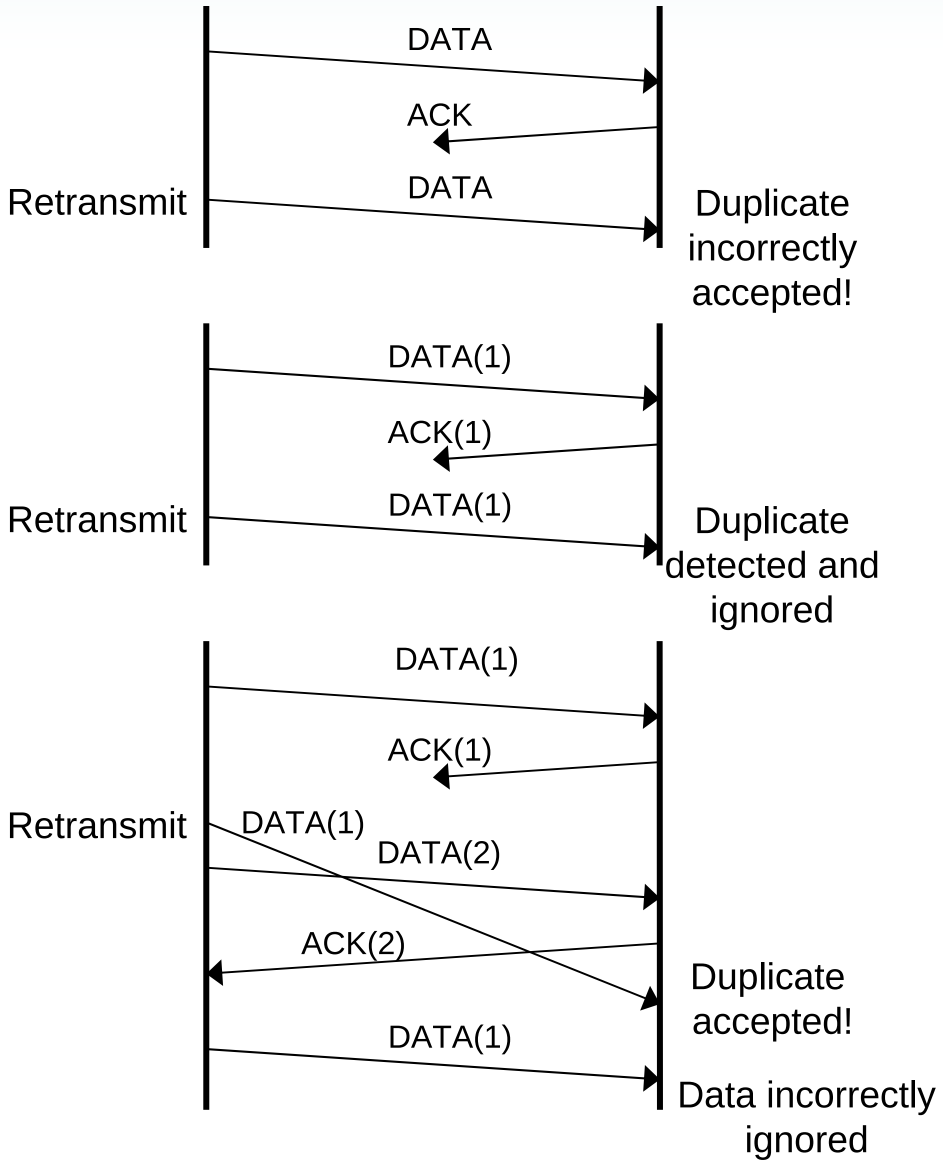
This is a TCP segment carrying Data, and Sequence Number 1. This assumes the Initial Sequence Number was 0.

This is a TCP segment carrying no Data, but an ACK number of 61, meaning the next byte expected will have sequence number 61.

Although its not shown here, we can "piggyback" an ACK on a DATA. That is, a segment may carry DATA from B to A, as well as an ACK which acknowledges DATA received from A to B.

Sequence Numbers and Duplicates

- If a segment is lost, then the sender will timeout and retransmit that segment – only one segment received at receiver
- But if an ACK is lost (the receiver has received a segment), then the sender will timeout and retransmit that segment – two duplicate segments may be received at the receiver
- Need a way to detect duplicate segments:
  - Use increasing sequence numbers
  - If two segments are received with the same sequence numbers, then duplicates (discard the second copy)
- But sequence numbers "wrap"
  - 0, 1, 2, 3, …127, 0, 1, 2, …, 127, 0, ...
- To avoid incorrect detection of duplicates (shown on right), sequence numbers must be large enough so they do not wrap within maximum segment lifetime



Connection Establishment

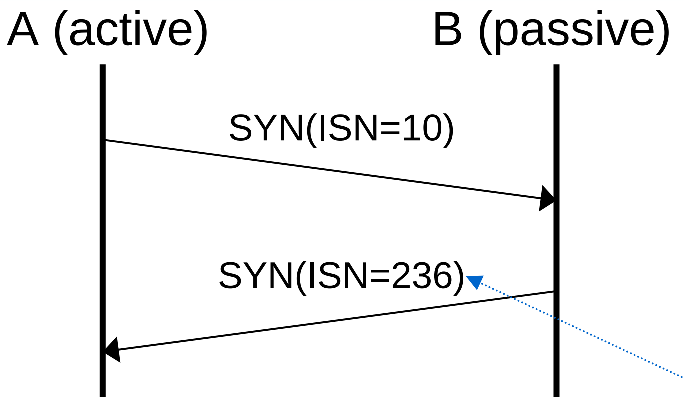
- Before sending data, TCP needs to establish a “connection” between source and destination
  - TCP uses a client/server model for connection establishment
    - Server waits and listens for TCP entity to connect (Passive)
    - Client initiates the connection to a TCP entity (Active)
    - Note: it is possible for two “Clients” to connect to each other (initiate connection at same time)
    - Note: Once the connection is setup, full duplex data transfer is possible; there is no longer a concept of client/server (or master/slave) – each peer is the same
  - A TCP connection is identified by:
    - Source port, source IP address, destination port, destination IP address
      - Allows many different computers (or even applications on one computer) to connect to the same application on one computer
      - E.g. a web server on port 80, IP address of 201.17.32.4 can have simultaneous connections with multiple browsers:
        - » Browser on 130.160.25.5, port 49010
        - » Browser on 130.160.25.5, port 49011
        - » Browser on 120.52.16.24, port 49010
  - How does the client know the IP address/port of server to initiate connection?
    - IP address is either entered by user, or obtained from naming server (e.g. Domain Name Service)
    - Port number is either well known (by default web servers use port 80), entered by user, or obtained from a naming server

Connection Establishment

- What is the purpose of connection establishment?
  - Allows each end to assure that the other exists
  - Allows exchange or negotiation of optional parameters
    - Specifically, for TCP, synchronise sequence numbers
      - Both sides need an Initial Sequence Number, this is agreed upon during connection establishment
      - E.g. A chooses an ISN=10 and needs to inform B of this value; similarly, B chooses an ISN=236 and needs to inform A of this value; if either A or B disagree with the ISN chosen by the other, the connection is not established
      - A and B choose there ISN independently
  - Triggers the allocation of resources for the connection (e.g. allocate buffer space in memory)
- TCP Connection Establishment
  - Send segments with the SYN flag set
  - Include the Initial Sequence Number in Sequence number field
- Connection establishment often called *handshake*
  - Two Way Handshake: has problems!
  - Three Way Handshake: used by TCP

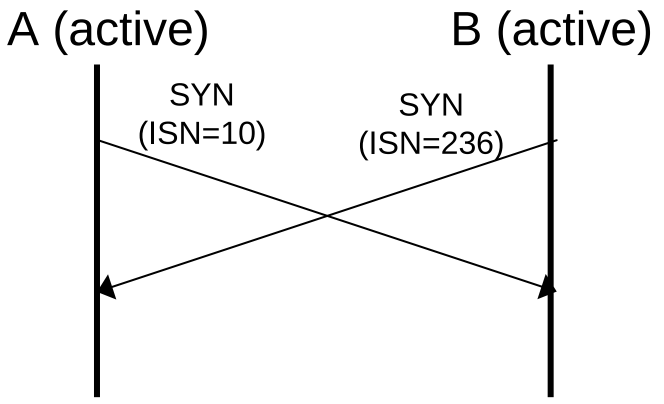
Two Way Handshake

This shows a typical scenario where a server (B) listens for connections, and a client (A) initiates the connection.



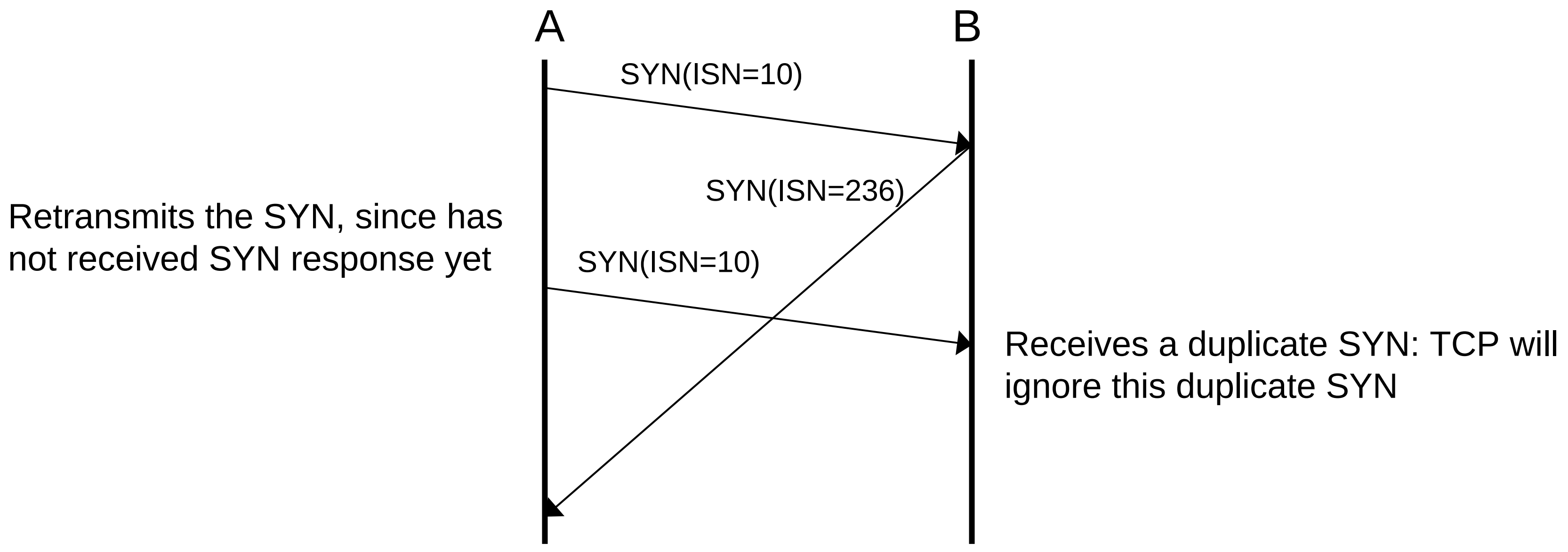
These are TCP segments with the SYN flag set and Sequence Number set to the value of ISN. The segments carry no data.

This scenario is not so common, but possible. Both A and B initiate the connection at the same time.



Problems with Two Way Handshake

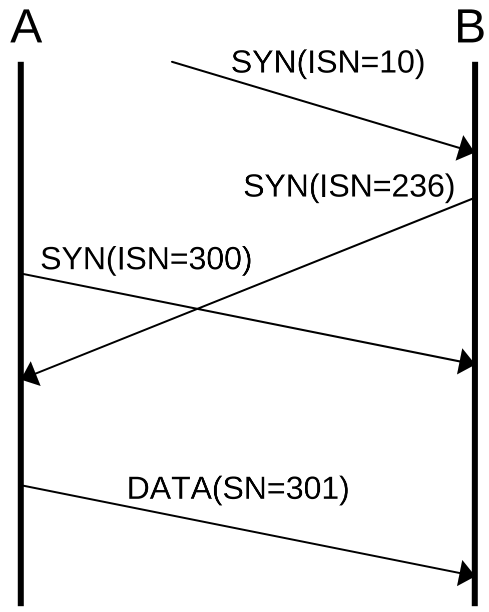
- Since we have unreliable network, use retransmissions if no response is received
  - If A sends a SYN, but receives no response from B after a timeout, then A will send the SYN again
  - May be possible for duplicate SYN's to be received


A
B
SYN(ISN=10)
SYN(ISN=236)
SYN(ISN=10)
Retransmits the SYN, since has not received SYN response yet
Receives a duplicate SYN: TCP will ignore this duplicate SYN

- Easy to fix: ignore duplicate SYNs once a connection is open (Established)

Problems with Two Way Handshake

- But may have an old SYN (from previous connection) arrive



This is an old SYN from a previous connection. But B must respond to it because it has no connection open

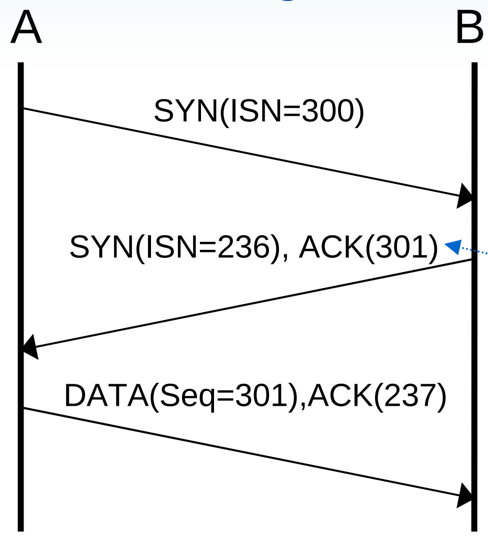
A sent a SYN with ISN 300, and received a response. So A believes the response acknowledged its SYN, and so sends DATA with SN of 301.

Receives a duplicate SYN: TCP will ignore this duplicate SYN. So B is Expected next sequence number of 10+1 = 11 (not 301)

B expects 11, but receives 301 and so discards the segment. This is a problem!

- What went wrong? There is no explicit acknowledgement of each others SYN and ISN
- Solution: Three way handshake, include explicit acknowledgements

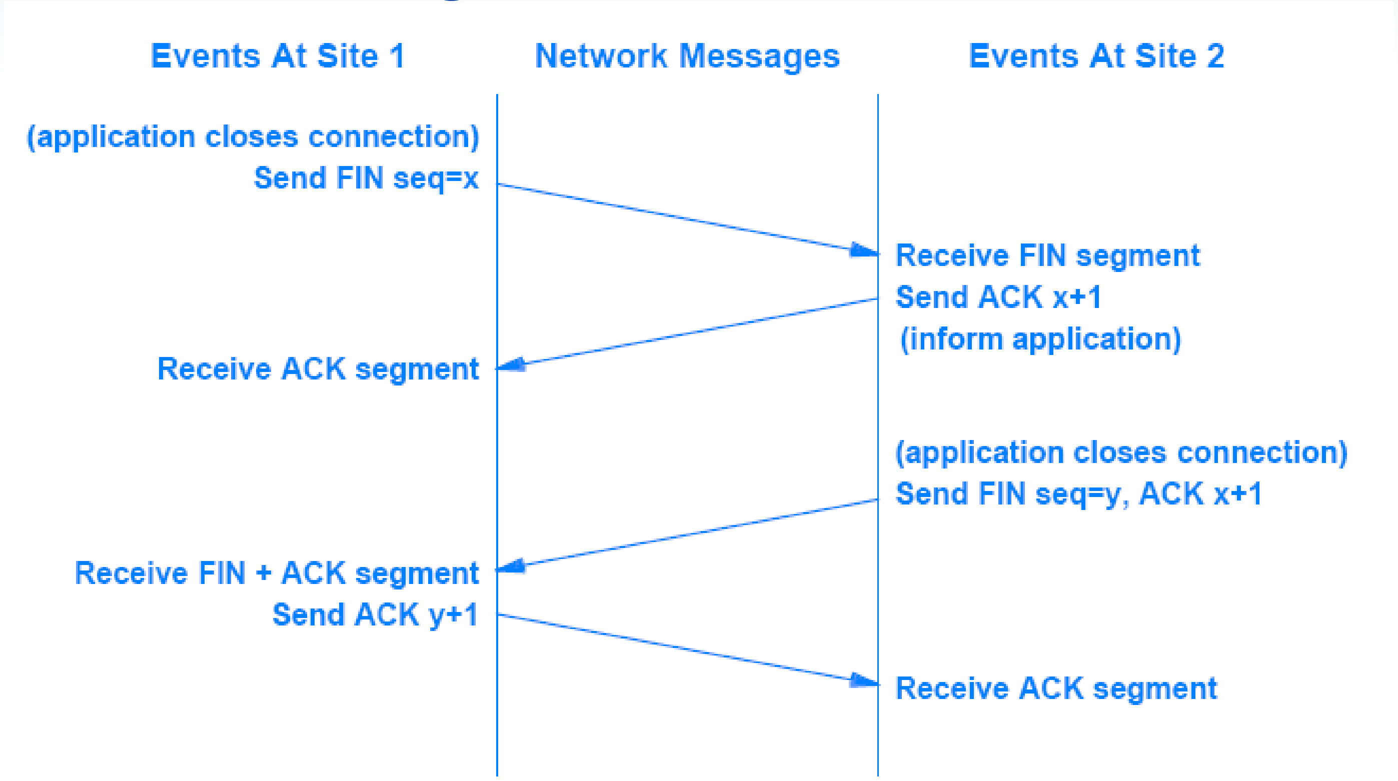
TCP Three-way Handshake



This is a single TCP packet with the SYN flag set, the ACK flag set, but not carrying any data

- A sends segment with SYN flag set
  - Sequence number = ISNA = i
- B sends segment with SYN and ACK flag set
  - Sequence number = ISNB = j
  - Ack number = i + 1
- A sends segment with ACK flag set, and includes the first DATA
  - Sequence number = i+1
  - Ack number = j+1
- Handles the loss of messages and receiving duplicates from old connections
- Note that the ACK sent by A can contain DATA

Closing TCP Connection



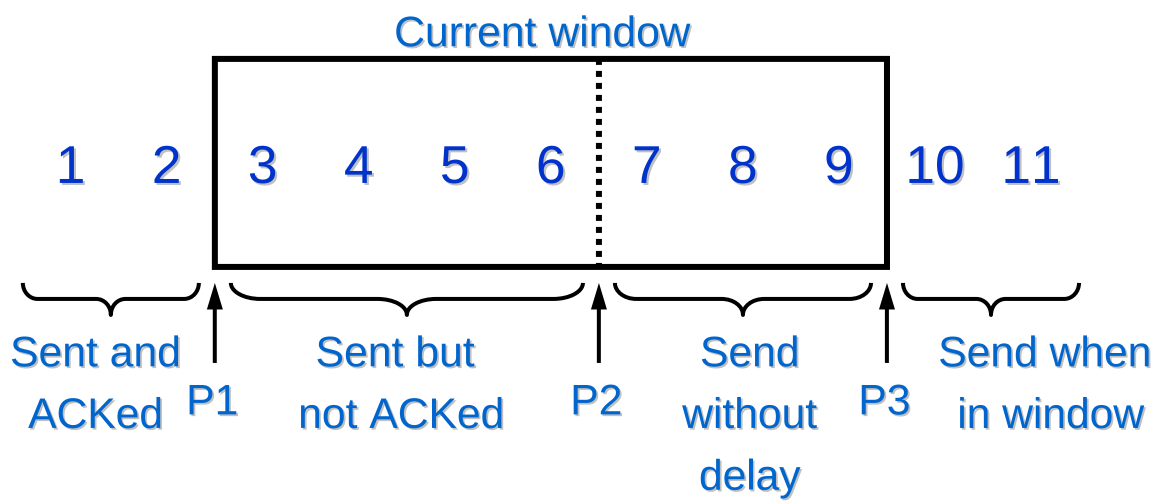
- Closing a connection is similar to opening a connection: need an Acknowledgement of the close (or FIN segment)
- It is possible for connection to be closed from A to B (so A cannot send more data to B), but open in other direction (B can send data to B)
- Also possible for connection reset (abort) – no attempts are made to send any outstanding data, and a RESET (RST) segment is sent.

Data Transfer in TCP

- Once a connection is opened:
  - Need reliable delivery of data
    - Acknowledgements and retransmissions
  - Do not overflow the receivers
    - Flow control
  - Do not overflow the network (e.g. routers along the path)
    - Congestion control
- TCP using a sliding window mechanism
  - For efficient transmissions
  - To avoid overflow of receivers and network
- Although the detailed algorithms are specified for TCP, implementations can still choose certain options, e.g.
  - When does the TCP receiver pass the data to the receiving application?
  - Does the TCP sender send a segment immediately when it has data from an application, or does the TCP sender wait until it has a certain number of bytes to send?
  - In this lecture, we will cover some selected options – in fact, TCP implementations may choose different options
    - Usually it is a tradeoff between performance and implementation complexity

TCP Sliding Window

- Operates on the byte level, not segment
  - Three pointers (P1, P2, P3) to bytes in the data stream
- Sender and receiver maintain windows for each direction



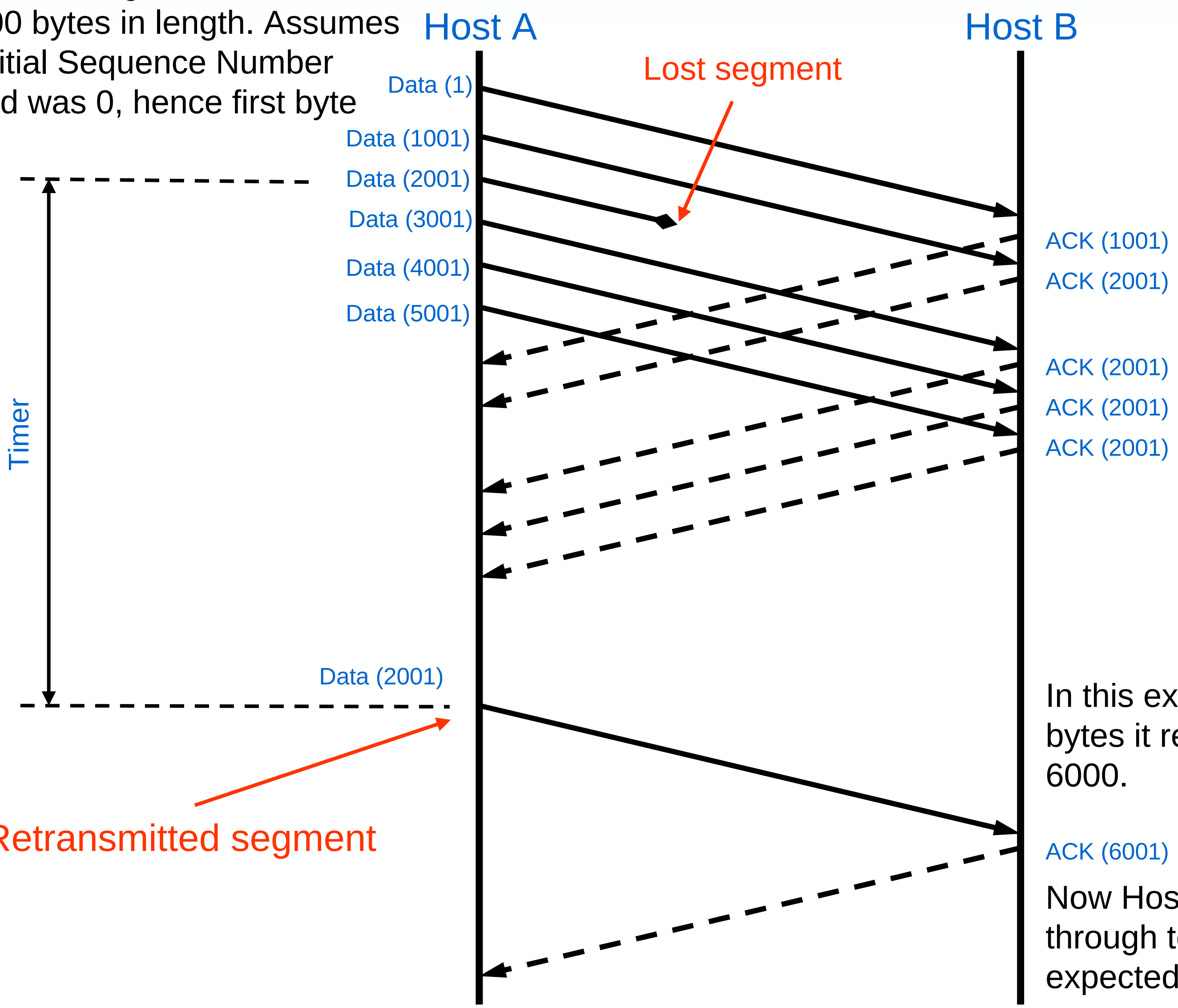
- Variable sized window, based on advertised window

Reliability: ACK with Retransmit

- When TCP sender sends data segment, a timer is started for that segment
  - If the timer expires (that is, timeout), the segment is retransmitted
  - Retransmission occurs up to a maximum number of times, after which TCP gives up and indicates an error to the application
- When TCP receiver receives data segment, it *may send* a cumulative acknowledgement for the segment (and previous segments)
  - The ACK indicates the sequence number of the next byte received
  - E.g. If byte with sequence number 1000 received, ACK will indicate 1001 as next expected byte
  - The ACK number X can be interpreted as: “I have received all bytes up to and including (X-1), and now expect to receive byte X”
  - (We say *may send:* depending on the TCP implementation, it may choose to send an ACK for every segment it receives, or to send an ACK to acknowledge a group of segments)
- Improvement - Fast Retransmit:
  - If 3 duplicate ACKs received, retransmit
  - No need to wait for timeout

TCP Retransmission (Basic)

Host A sending DATA which is 1000 bytes in length. Assumes the Initial Sequence Number agreed was 0, hence first byte is 1.

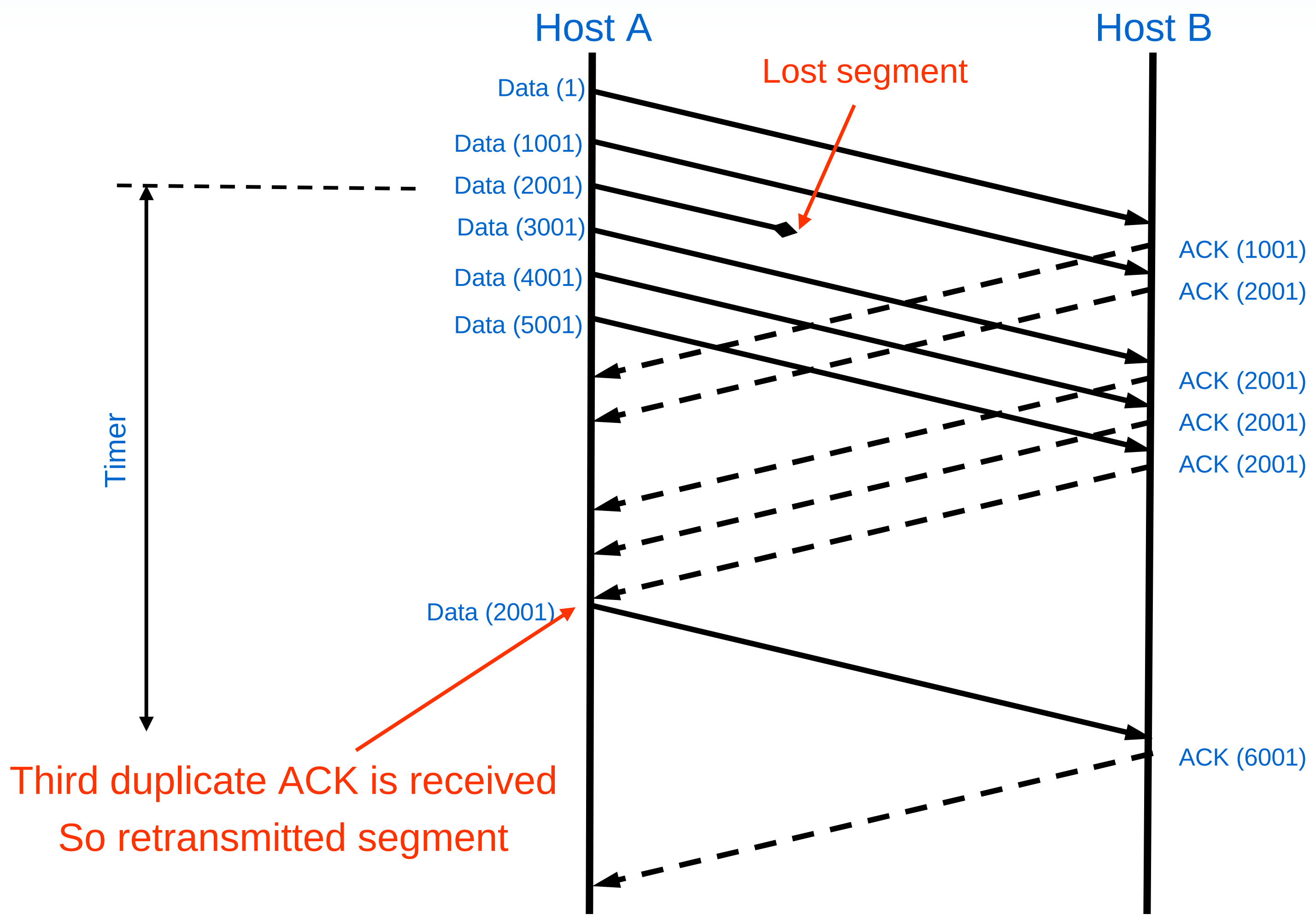


Host B receives DATA with bytes from 3001 to 4000. But it has not received bytes 2001 to 3000, and so sends an ACK indicating it is *still* expected to receive byte 2001.

In this example, Host B buffers all bytes it receives, including 3001 to 6000.

Now Host B has received byte 1 through to byte 6000, and so next expected is 6001.

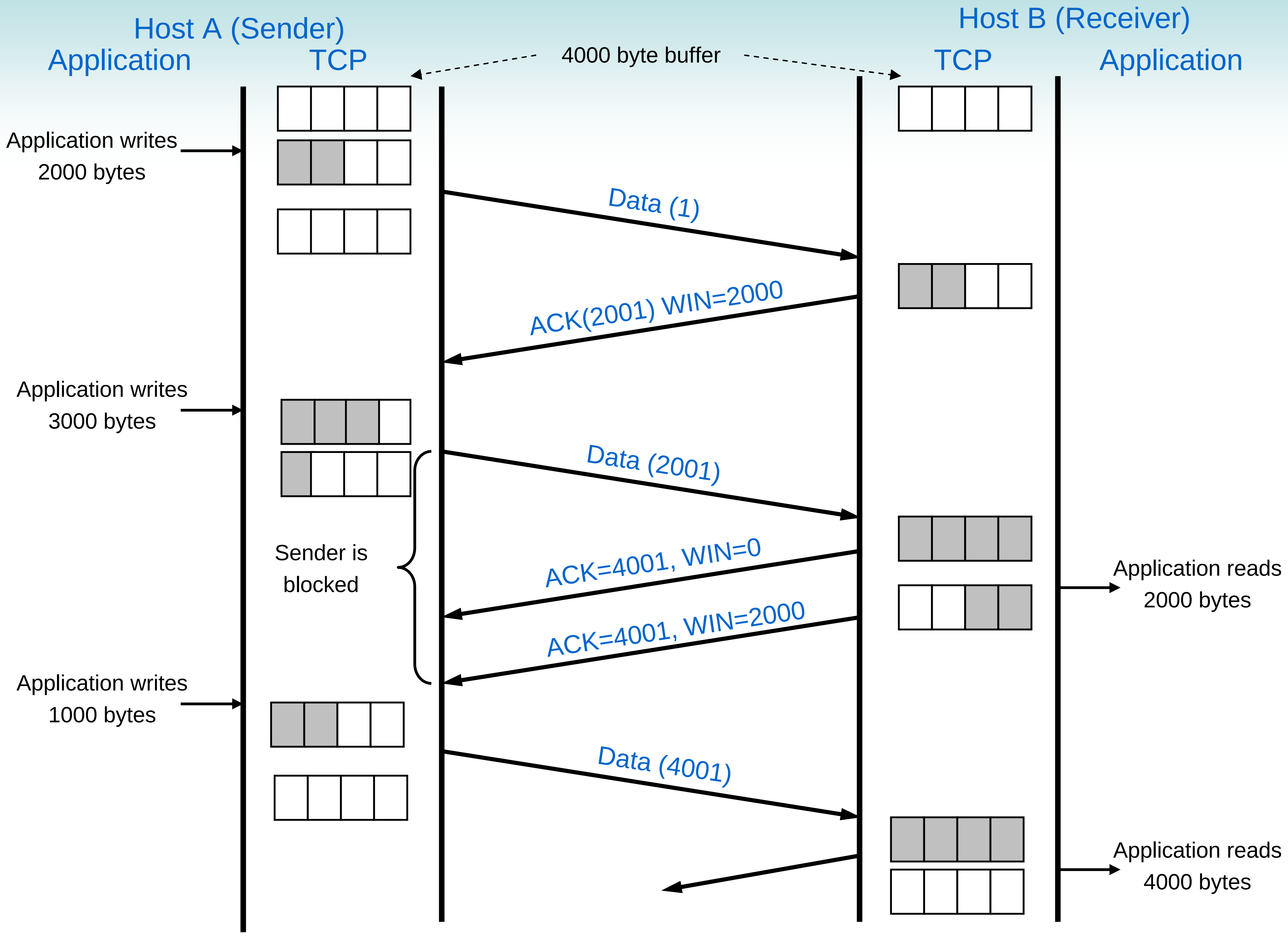
TCP Retransmission (Fast Retransmit)



With Fast Retransmit, in many cases is a segment is lost, a retransmission will be triggered by receiving three duplicate ACKs. So Host A does not have to wait for a timeout to retransmit. This is more efficient than the basic scheme for retransmission.

TCP Flow Control

- Aim: Prevent sender from overrunning capacity of receivers
- Needed for TCP because:
  - Application cannot keep up with incoming data
  - TCP cannot keep up with incoming segments
- Must take into account:
  - Variable end-to-end round trip times (RTT)
  - Interactions between TCP and IP and application protocols
- TCP flow control:
  - Receiver notifies sender of amount of buffer space is left (that is, receiver says how much it can receive)
    - This is carried in the Window field of the TCP header, and called the Advertised Window
  - The Sender cannot send more than the Advertised Window
- Example (next slide)
  - Assumes TCP receiver has a 4000 byte buffer which is initially empty, and that the TCP sender knows that the buffer is empty (that is, the previous advertised window was 4000)
  - Shows how the receivers buffer (and hence advertised window) control the flow of data from the TCP sender
  - Also shows the independence between the Application writing/reading data and TCP sending segments. If the Application writes data, it doesn't mean that TCP will immediately send that data.


Host A (Sender)
Application
TCP
4000 byte buffer
Host B (Receiver)
TCP
Application
Application writes 2000 bytes
Data (1)
ACK(2001) WIN=2000
Application writes 3000 bytes
Data (2001)
Sender is blocked
ACK=4001, WIN=0
Application reads 2000 bytes
ACK=4001, WIN=2000
Application writes 1000 bytes
Data (4001)
Application reads 4000 bytes

TCP Congestion Control

- Flow control protects slow receiver from a fast sender
  - Congestion control protects the network from a fast sender
- Without congestion control:
  - To transport protocol, congestion is seen as increased delay
  - Increased delay results in more retransmissions
  - More retransmissions results in more congestion
  - Leads to network collapse, no data can be successfully received
- TCP Congestion Control
  - Uses implicit congestion detection
    - Loss of segments imply congestion
      - TCP assumes segments are not lost due to errors on links (since most networks TCP was designed for had very few errors on links)
    - If a TCP sender does not receive an ACK within a timeout (or receives 3 duplicate ACKs), the TCP sender assumes there is congestion in the network and implements appropriate congestion control
  - Mechanisms used by TCP include:
    - Slow Start: start by sending at a slow rate, and if no segments are lost, increase that rate
    - Multiplicative Decrease: if congestion is detected, decrease the rate you send by half
  - Congestion control is implemented by:
    - Maintaining second window at sender, called congestion window
    - TCP sender determines how much it can send from:
      - minimum (advertised window, congestion window)

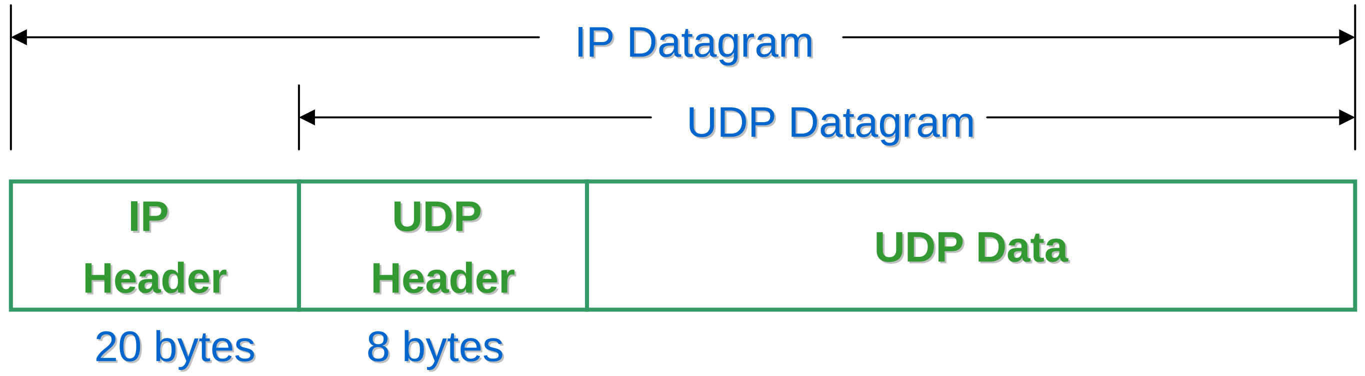
TCP in Details

- TCP is complex!
  - We have introduced some basics of connection setup, retransmissions and flow control
  - There are many more details of these concepts
    - Estimating timeouts, congestion control mechanisms, connection setup options, sending/receive policies, …
  - And there are different versions of TCP, each providing new features and improvements, e.g.
    - Original TCP - RFC 793 (1981)
    - TCP Tahoe (1988)
    - TCP Reno (1990)
    - TCP NewReno (1995)
- Some more details of TCP may be covered in:
  - ITS 327 and ITS 413
  - Practical use of TCP will be illustrated in ITS 332 (Lab class)

User Datagram Protocol (UDP)

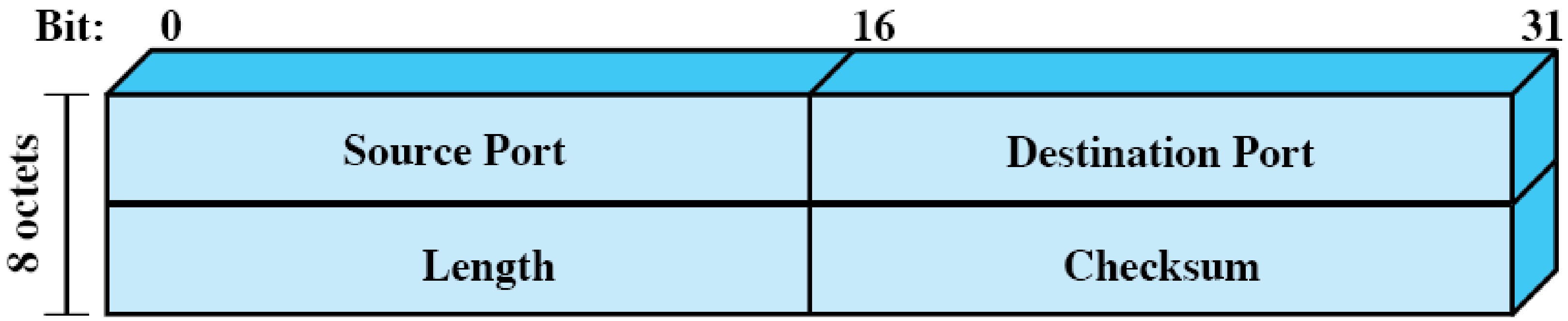
User Datagram Protocol

- IP provides unreliable, connection-less service at network layer
- UDP provides same service to applications
  - UDP messages can be lost, duplicated, or arrive out of order
  - Adds multiplexing to support multiple applications
    - That is, UDP supports port numbering the same way as TCP
- UDP is simple (standard describes it in 4 pages)



UDP Message Format

- 8 byte header + Data
- Length is count of bytes in header plus Data
- Checksum calculated over:
  - UDP Header
  - UDP Data
  - Some parts from IP header such as Source and Destination IP address, IP protocol type code (17 for UDP)



Applications Using UDP

- Network management
  - SNMP, DNS, BOOTP, DHCP
- Simple user protocols
  - TFTP
- Multimedia applications (voice, video)
  - Strict delay/jitter requirements make TCPs flow/congestion control and error detection unsuitable
  - Use UDP and own retransmission scheme if necessary
  - Voice over IP: RTP over UDP over IP