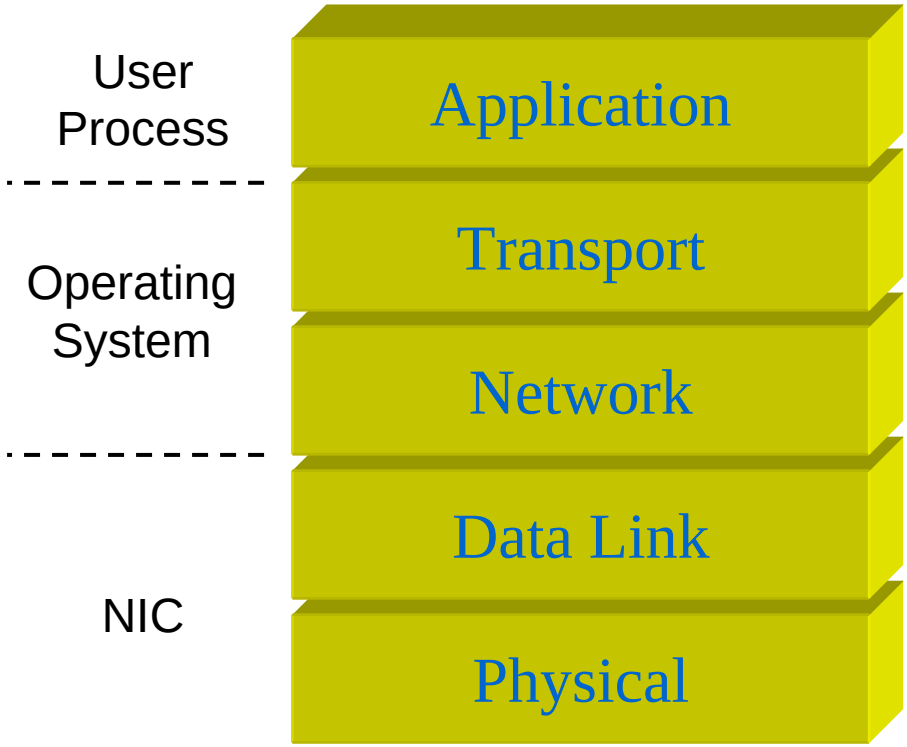
Transport Protocols

Dr Steve Gordon
ICT, SIIT

Contents

- Overview of Transport Protocols
- Addressing and Multiplexing
  - Common to TCP and UDP
- UDP
- TCP
  - Segments and Sequence Numbers
  - Connection Management
  - Reliability
  - Flow Control
  - Congestion Control

Where Are We?



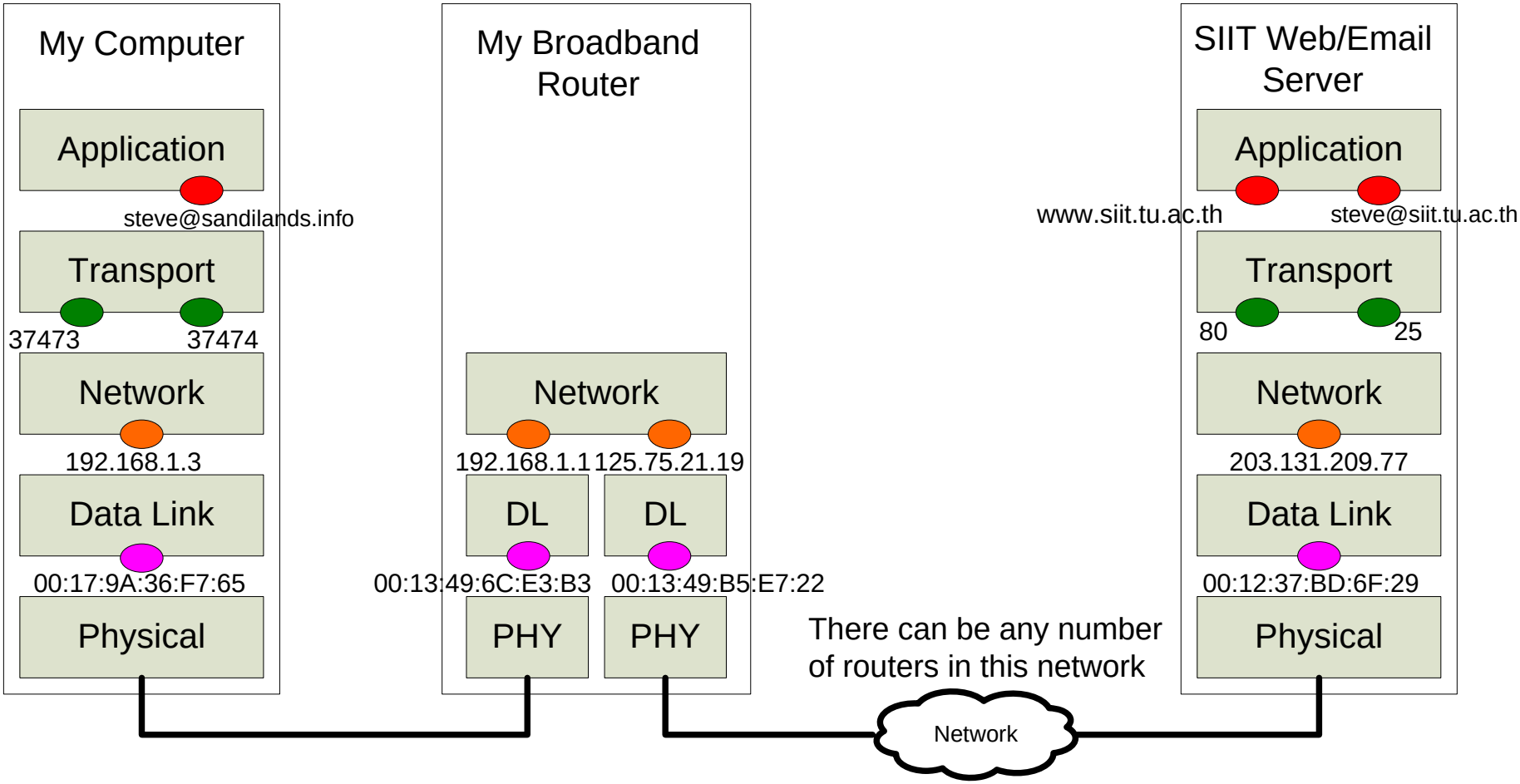
Send data between source and destination processes/applications

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 Control Transmission Protocol (SCTP)
  - Internet Control Message Protocol (ICMP) (not a 'normal' transport protocol used by applications, but is at transport layer)
- 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)

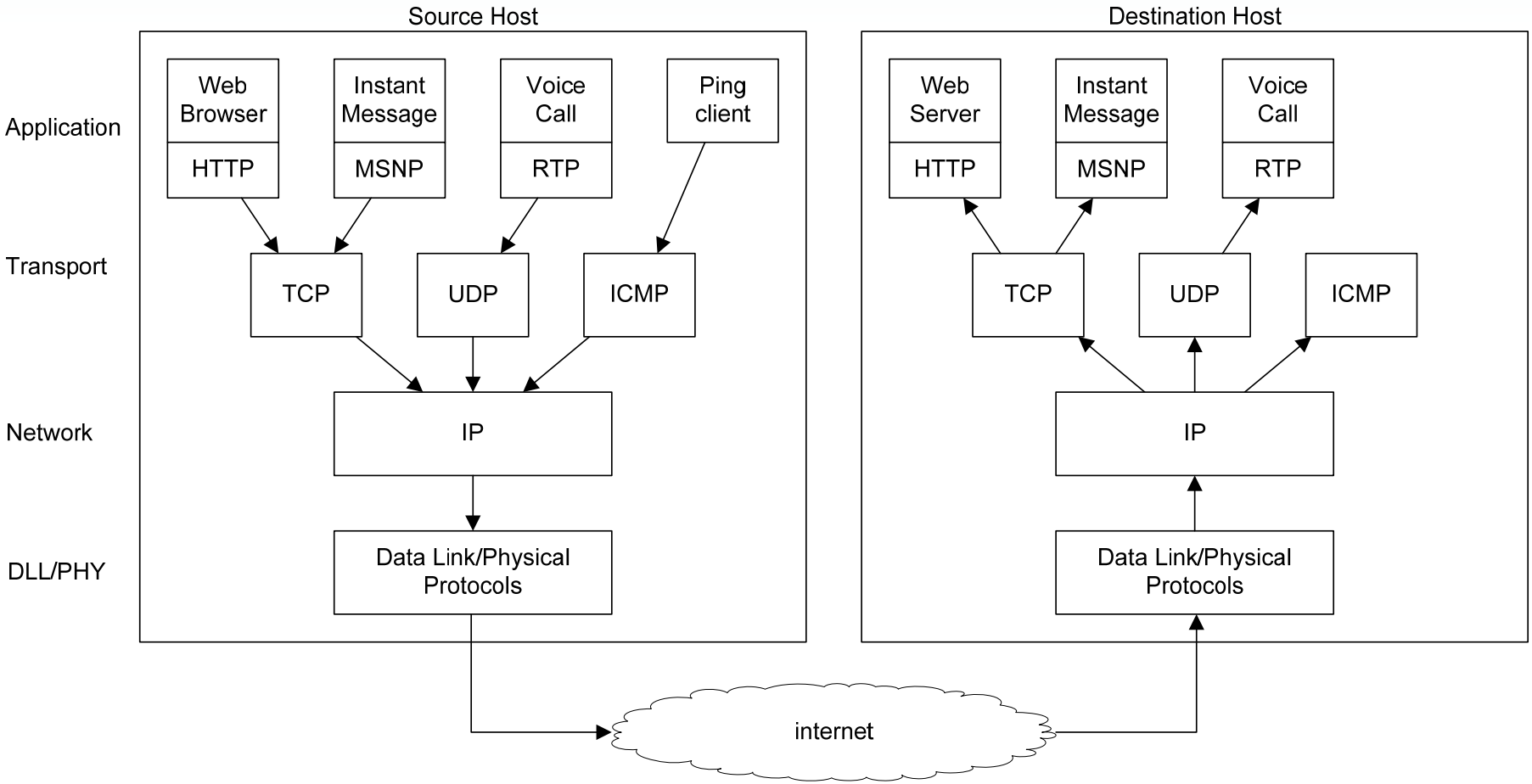
Addressing and Multiplexing

Internet Applications

- Most Internet applications follow a *client/server model* of initiating communication:
  - Client initiates communication
  - Server waits for client to initiate communication
  - Once the communication is initiated, data can flow in both directions (client to server and server to client)

- Addressing
  - For client to initiate communication to server, the client needs to know address of server:
    - IP address is used to identify network interface of host
    - Protocol number is used to identify transport protocol on a host
    - Port number is used to identify application protocol on a host

Multiple Applications, Multiple Transport Protocols



A host may have multiple Internet applications running at the same time, and be using multiple transport protocols

At the Destination Host …

- The IP software at the Destination Host receives an IP datagram

  - How does IP know which transport protocol the Data is destined to?
    - (Remember, IP does not "look" at the Data)
    - Protocol field of the IP header
      - 6 = TCP; 17 = UDP; 1 = ICMP, … http://www.iana.org/assignments/protocol-numbers/
    - IP sends to Data, as a "segment", to the transport protocol

- The Transport protocol software receives the segment from IP
  - How does the transport protocol know which application protocol the Data is destined to?
    - Destination port field in Transport Protocol header
      - 80 = Web server; 25 = Email server; … http://www.iana.org/assignments/port-numbers
    - Transport protocol sends Data, as a "message", to application protocol

Multiplexing

- Multiplexing at the Network layer:
  - IP uses the Protocol field to identify the transport protocol being used
  - IP at source host sets the Protocol field, allowing IP at destination host to deliver to correct transport protocol
  - A host only needs one instance of a particular transport protocol
    - E.g. only one instance of TCP running on host
    - Hence, Protocol field sufficient to identify transport protocol at source and destination

- Multiplexing at Transport layer:
  - TCP/UDP use Port number fields to identify the application protocol being used
  - TCP/UDP at source host sets the Port number field based on the application that sent the Data, as well as the intended destination application
  - A host may have multiple instances of application protocols
    - E.g. a host running multiple web browsers; host running an email server and an email client
    - May have multiple instances on source host communicating with one instance on destination host
    - Hence, require Source Port and Destination Port

Port Numbers

- Ports and port numbers represent the end-points of the transport protocol communications
  - Processes are the actual end-points of the communication; ports provide abstract view of processes
- IANA maintains list of port numbers, dividing them based on types of applications:
  - Well Known Ports: 0 – 1023
    - Typically used for common Internet servers
    - E.g. Web server (80), Email server (25), FTP Server (21), Telnet (23)
  - Registered Ports: 1024 – 49151
    - Often used for Internet servers (some operating systems also use for clients)
  - Dynamic (private, ephemeral) Ports: 49152 – 65535
    - Not to be assigned to specific applications/clients
    - Typically randomly chosen by operating system for client applications
- Communication using particular transport protocol can be uniquely identified by:
  - Source IP address, Source Port number, Destination IP address, Destination Port number

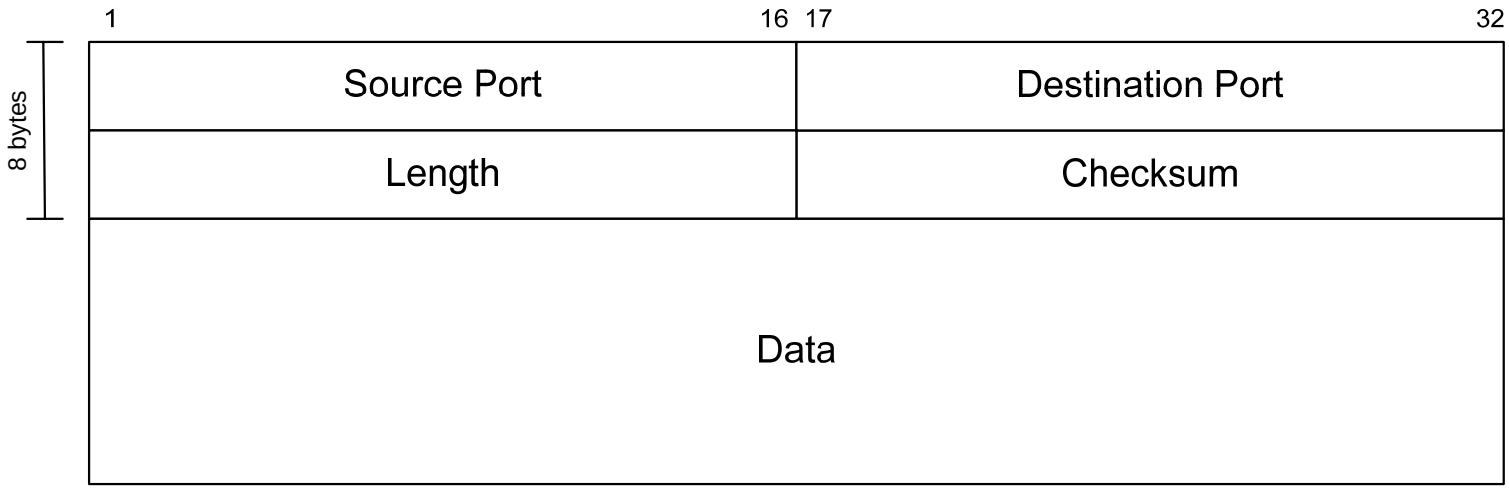
User Datagram Protocol (UDP)

User Datagram Protocol

- UDP is a unreliable connection-less transport protocol
  - Takes Data from the application layer, attaches a UDP header, and delivers to IP
  - UDP provides multiplexing and checksum over the packet
  - UDP segments may be: lost, arrive out of order, duplicated, arrive in error
    - If you write an application that uses UDP and you want reliable data transfer, then you must ensure the application provides reliability!
- UDP is simple (standard describes it in 4 pages)
- UDP is used by applications which:
  - Require simplicity
    - Trivial File Transfer Protocol (TFTP), network management protocols (must be implemented in embedded devices)
  - Don’t require reliability
    - Voice and video applications (can tolerate data loss), network management (e.g. provide status update every second)
  - Require low overheads
    - Voice and video applications (require low delay)

UDP Segment

- UDP segment (or datagram) contains 8 byte header plus Data
- Length is count of bytes in header plus Data
  - Minimum: 8 bytes Maximum: 65535 bytes
- 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)



Example UDP Datagram

```
Internet Protocol, Src: 10.10.1.237 (10.10.1.237), Dst: 10.10.1.1 (10.10.1.1)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 330
    Identification: 0xc62b (50731)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 128
    Protocol: UDP (0x11)
    Header checksum: 0x5c76 [correct]
        [Good: True]
        [Bad : False]
    Source: 10.10.1.237 (10.10.1.237)
    Destination: 10.10.1.1 (10.10.1.1)
User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)
    Source port: bootpc (68)
    Destination port: bootps (67)
    Length: 310
    Checksum: 0xf08e [correct]
        [Good Checksum: True]
        [Bad Checksum: False]
Bootstrap Protocol
```

IP Header

IP Data

UDP Header

UDP Data

Example UDP Datagram (Hex Bytes)

```
0000  00 50 ba 4c 6b 45 00 17 31 5a e5 89 08 00 45 00   .P.LkE..1Z....E.
0010  01 4a c6 2b 00 00 80 11 5c 76 0a 0a 01 ed 0a 0a   .J.+....\v......
0020  01 01 00 44 00 43 01 36 f0 8e 01 01 06 00 44 17   ...D.C.6......D.
0030  f7 05 00 00 00 00 0a 0a 01 ed 00 00 00 00 00 00   ................
0040  00 00 00 00 00 00 00 17 31 5a e5 89 00 00 00 00   ........1Z......
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0110  00 00 00 00 00 00 63 82 53 63 35 01 03 3d 07 01   ......c.Sc5..=..
0120  00 17 31 5a e5 89 0c 09 44 72 2d 47 6f 72 64 6f   ..1Z....Dr-Gordo
0130  6e 51 0d 00 00 00 44 72 2d 47 6f 72 64 6f 6e 2e   nQ....Dr-Gordon.
0140  3c 08 4d 53 46 54 20 35 2e 30 37 0b 01 0f 03 06   <.MSFT 5.07.....
0150  2c 2e 2f 1f 21 f9 2b ff                           ,./.!.+.
```

Example: UDP-based Application

- Source Host: 63.14.102.5
- Simple Client Application

```
For (i=1; i<=5; i++) {
    data = “hello” + i;
    send(data, 203.16.3.4, 40123);
}
Exit;
```

- Send 5 pieces of data using UDP to destination

- Destination Host: 203.16.3.4
- Simple Server Application

```
While (true) {
    data = receive(40123);
    printf(“$s”,data);
}
Exit;
```

- Loop forever, when a UDP datagram is received, print the data

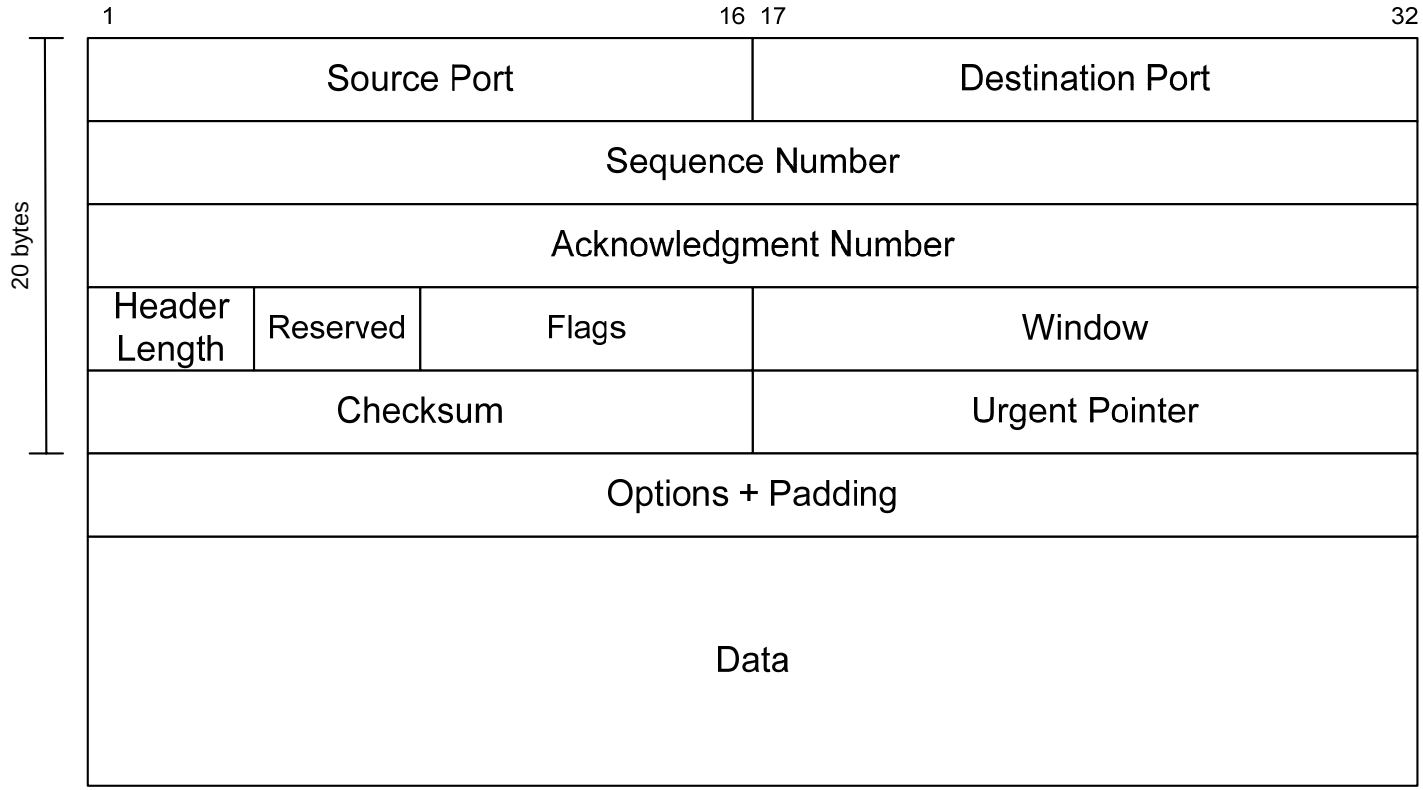
Transmission Control Protocol (TCP)

Transmission Control Protocol

- 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
  - Connection-oriented data transfer
    - 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 Segment

- Header contains 20 bytes, plus optional fields
  - Optional fields must be padded out to multiple of 4 bytes



TCP Segment Fields

- **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
- **Header Length**: Size of header (measured in 4 bytes)
- **Reserved** for future use
- **Flags** see next slide
- **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 Segment Flags

- 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; receiver should notify application program of urgent data as soon as possible
  - 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

Example TCP Segment

```
Ethernet II, Src: 00:17:31:5a:e5:89, Dst: 00:50:ba:4c:6b:45
Internet Protocol, Src: 10.10.1.237, Dst: 10.10.6.11
Transmission Control Protocol
    Source port: 1710
    Destination port: 80
    Sequence number: 1887125694
    Acknowledgement number: 3127238573
    Header length: 20 bytes
    Flags: 0x18 (PSH, ACK)
    Window size: 62597
    Checksum: 0x1e00
Hypertext Transfer Protocol
```

Example TCP Segment

```
Ethernet II, Src: 00:50:ba:4c:6b:45, Dst: 00:17:31:5a:e5:89
Internet Protocol, Src: 10.10.6.11, Dst: 10.10.1.237
Transmission Control Protocol
    Source port: 80
    Destination port: 1710
    Sequence number: 3127238572
    Acknowledgement number: 1887125694
    Header length: 32 bytes
    Flags: 0x12 (SYN, ACK)
    Window size: 5840
    Checksum: 0xe9db
    Options: (12 bytes)
         Maximum Segment Size: 1460
         SACK permitted
         Window Scale: 7
```

Sequence Numbers

- TCP uses sequence numbers to keep track of the data sent and received
  - For flow, error and congestion control
    - Example: sliding window mechanisms such as Go-Back-N
- Once a TCP connection is setup, data can be transferred in either direction
  - The sequence numbers used in each direction are independent
  - Upon connection setup, an Initial Sequence Number (ISN) is chosen for each direction
  - Each byte has a sequence number relative to the ISN
- Example:

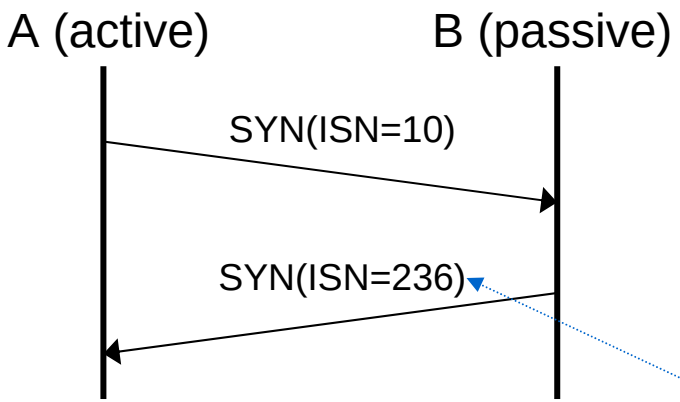
	A to B	B to A
ISN	1530	36
First byte of data	1531	37
100th byte of data	1630	136

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
  - 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

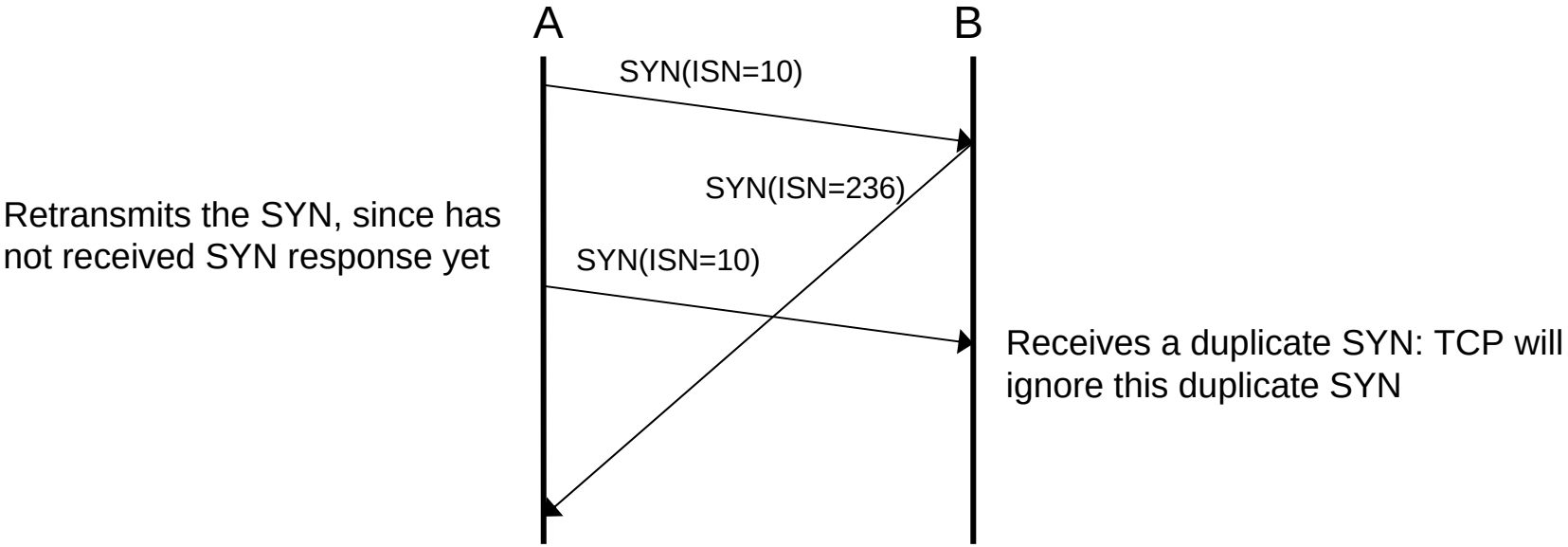
This shows a typical scenario where a server (B) listens for connections, and a client (A) initiates the connection.



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

Problems with Two Way Handshake

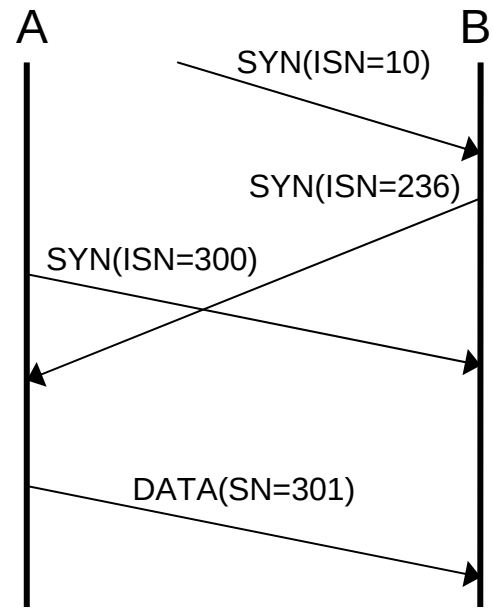
- Since we have unreliable network, TCP will use retransmissions if no response is received
  - If A sends a SYN, but receives no response from B after a timeout, then A will send the SYN again
  - May be possible for duplicate SYN's to be received



- Easy to fix: ignore duplicate SYNs once a connection is 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

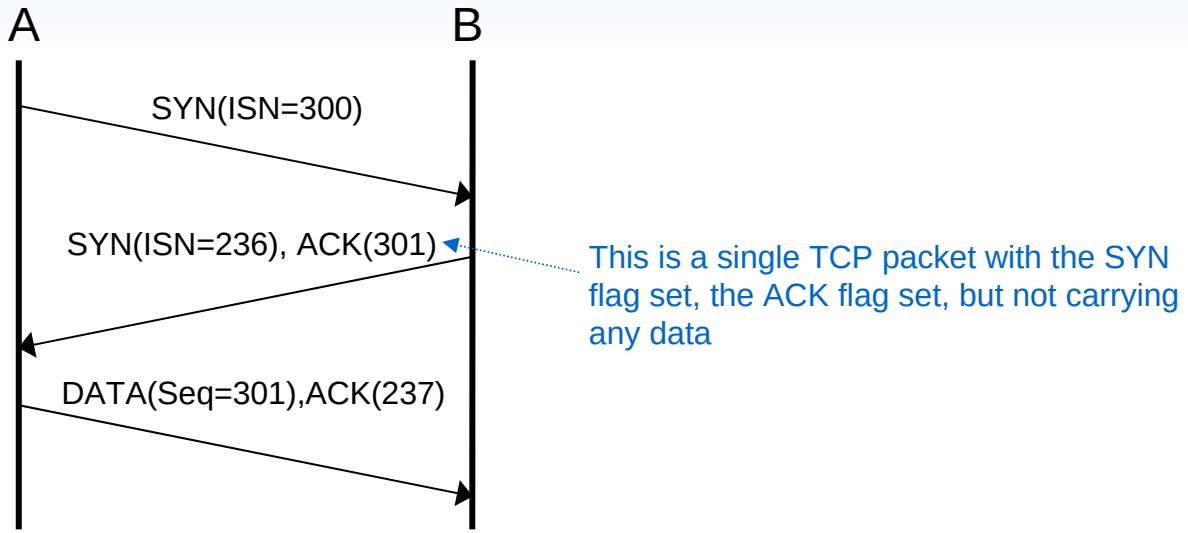
A sent a SYN with ISN 300, and received a response. So A believes the response acknowledged its SYN, and so sends DATA with SN of 301.

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!

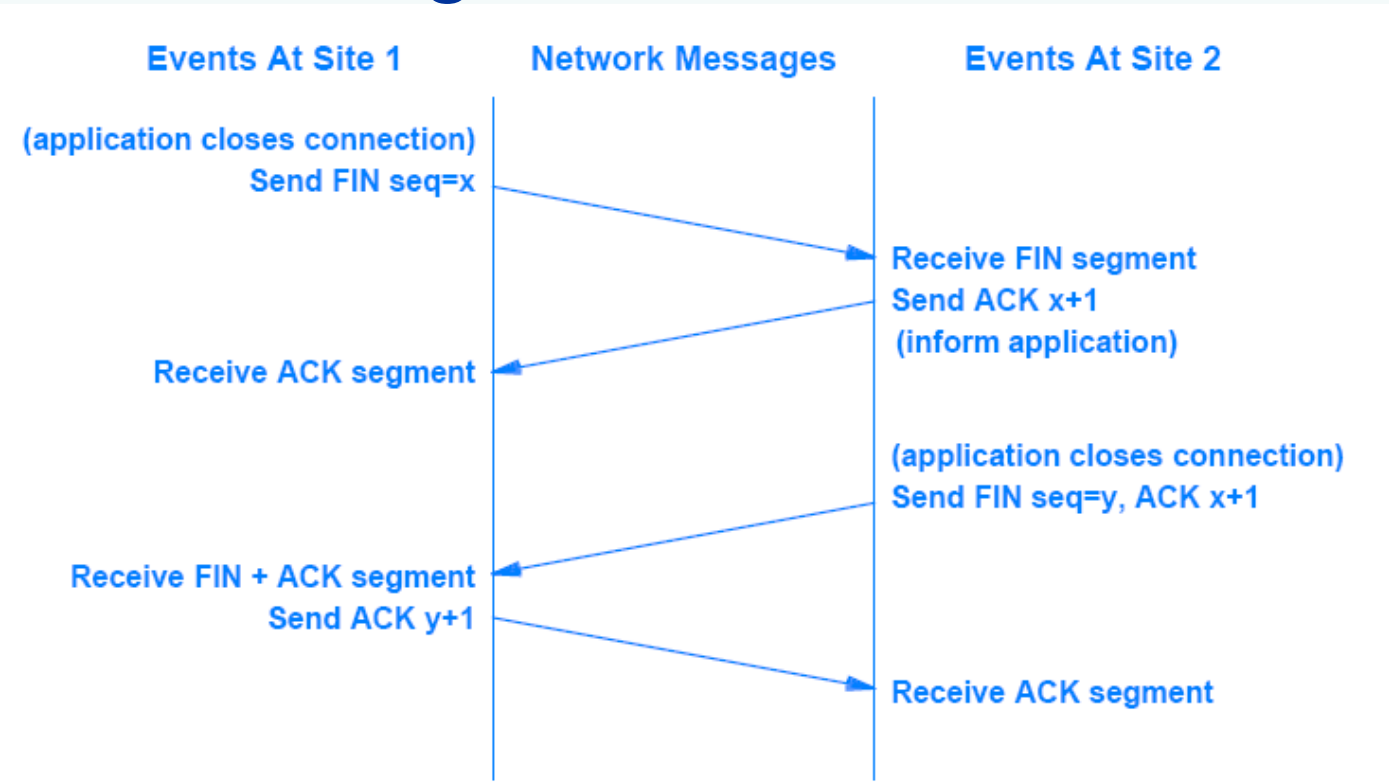
- What went wrong? There is no explicit acknowledgement of each others SYN and ISN
- Solution: Three way handshake, include explicit acknowledgements

TCP Three-way Handshake



- 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.

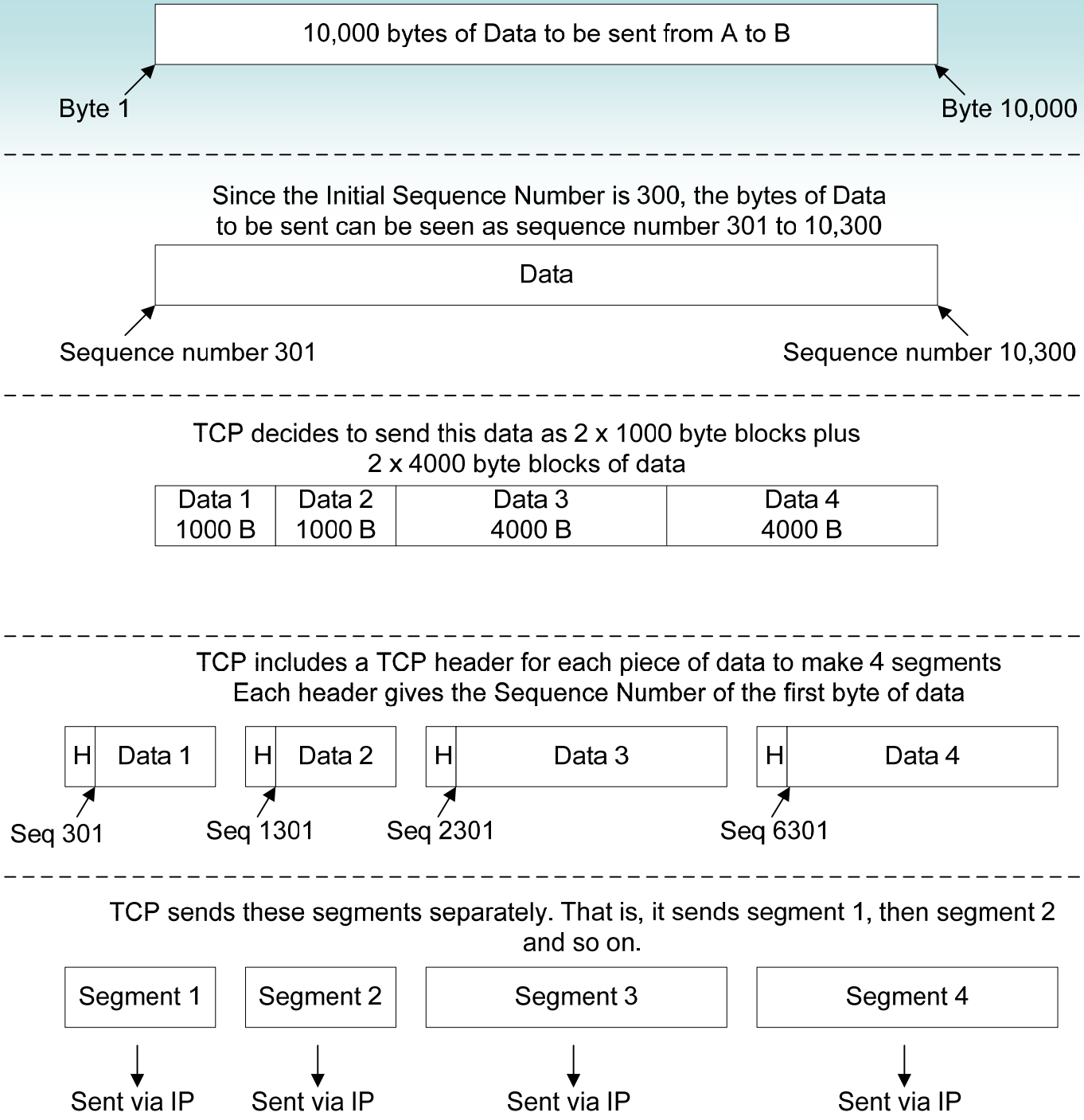
Data Transfer in TCP

- 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 uses 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?
  - Is an ACK sent for each received segment, or a cumulative ACK sent for a set of received segments?
  - Different implementations of TCP may choose different approaches
    - Trade-off between performance (efficiency) and complexity

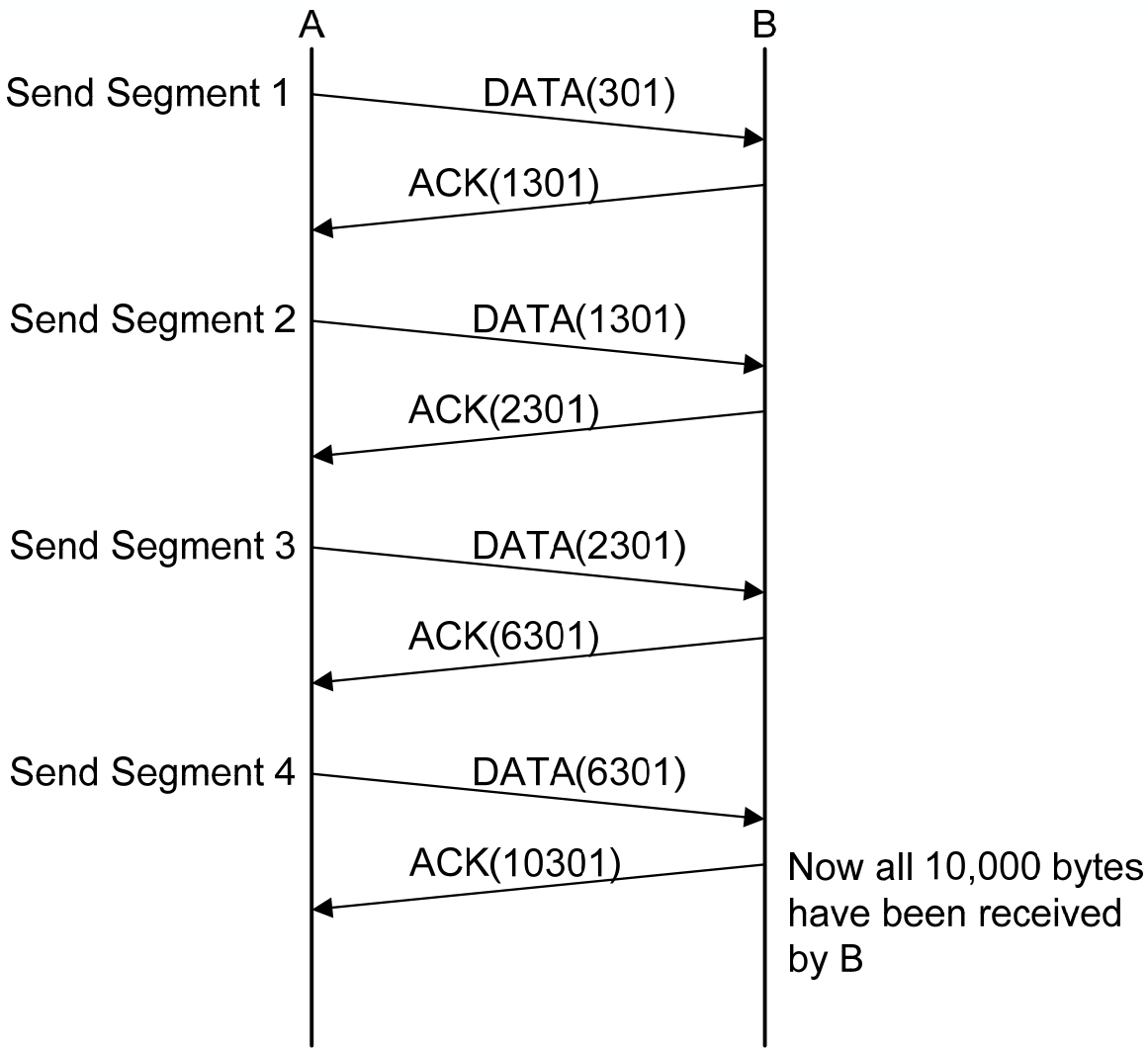
Example: Sequence Numbers and Segments

- Assume Application at A wants to send 10,000 byte message (file) to Application at B
  - 10,000 bytes are in the buffer at A
- TCP connection between A and B has been established
  - A has chosen an ISN = 300
- The TCP implementation at A decides to break message into 4 segments:
  - First two segments are 1000 bytes (plus header)
  - Last two segments are 4000 bytes (plus header)

- Why would TCP do this?
  - An implementation must follow the flow control, error control and congestion control algorithms, while trying to maximise efficiency

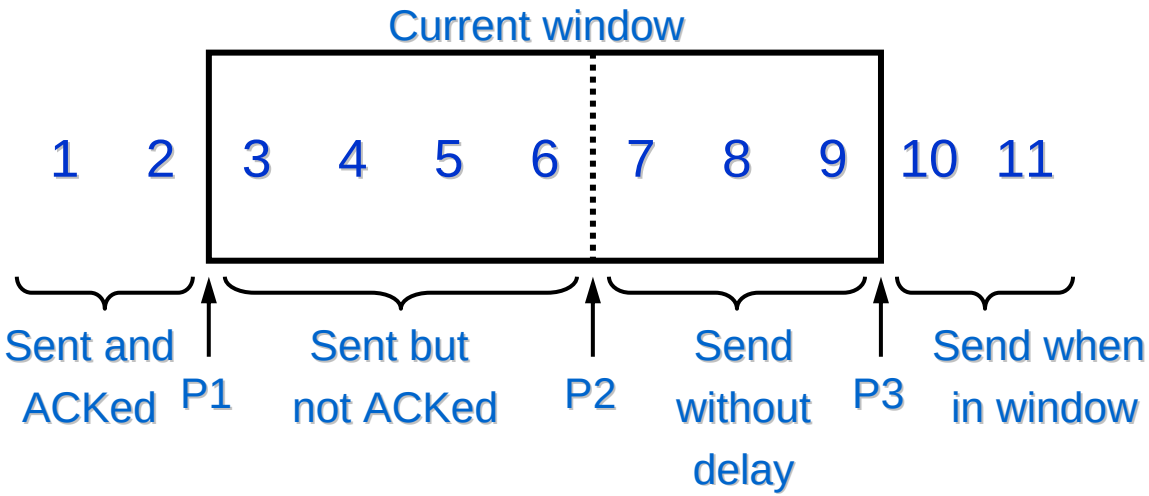

10,000 bytes of Data to be sent from A to B
Byte 1
Byte 10,000
Since the Initial Sequence Number is 300, the bytes of Data
to be sent can be seen as sequence number 301 to 10,300
Data
Sequence number 301
Sequence number 10,300
TCP decides to send this data as 2 x 1000 byte blocks plus
2 x 4000 byte blocks of data
Data 1
1000 B
Data 2
1000 B
Data 3
4000 B
Data 4
4000 B
TCP includes a TCP header for each piece of data to make 4 segments
Each header gives the Sequence Number of the first byte of data
H
Data 1
H
Data 2
H
Data 3
H
Data 4
Seq 301
Seq 1301
Seq 2301
Seq 6301
TCP sends these segments separately. That is, it sends segment 1, then segment 2
and so on.
Segment 1
Segment 2
Segment 3
Segment 4
Sent via IP
Sent via IP
Sent via IP
Sent via IP

Example: Sequence Numbers and Segments



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



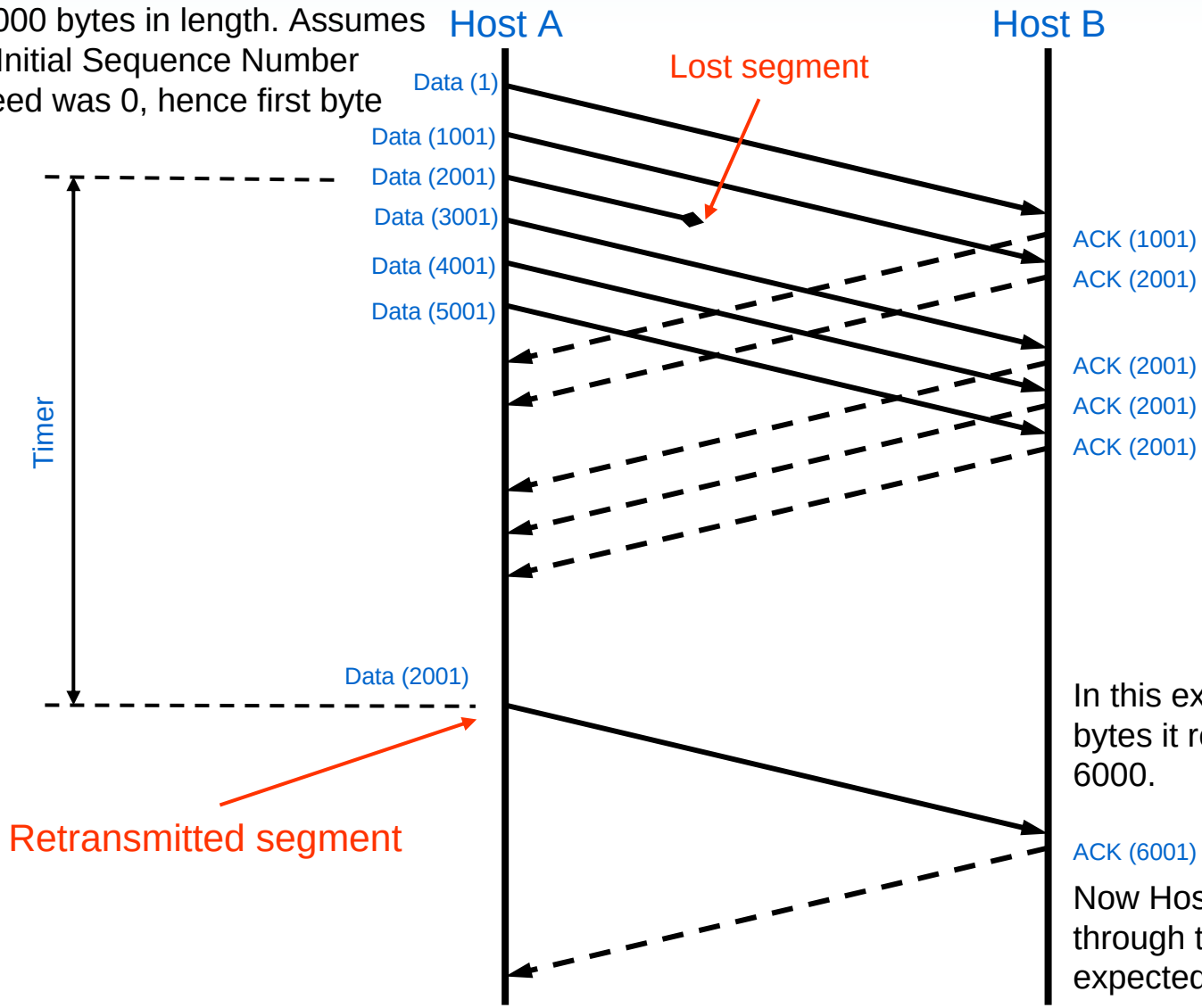
- Variable sized window, based on advertised window

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)

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.

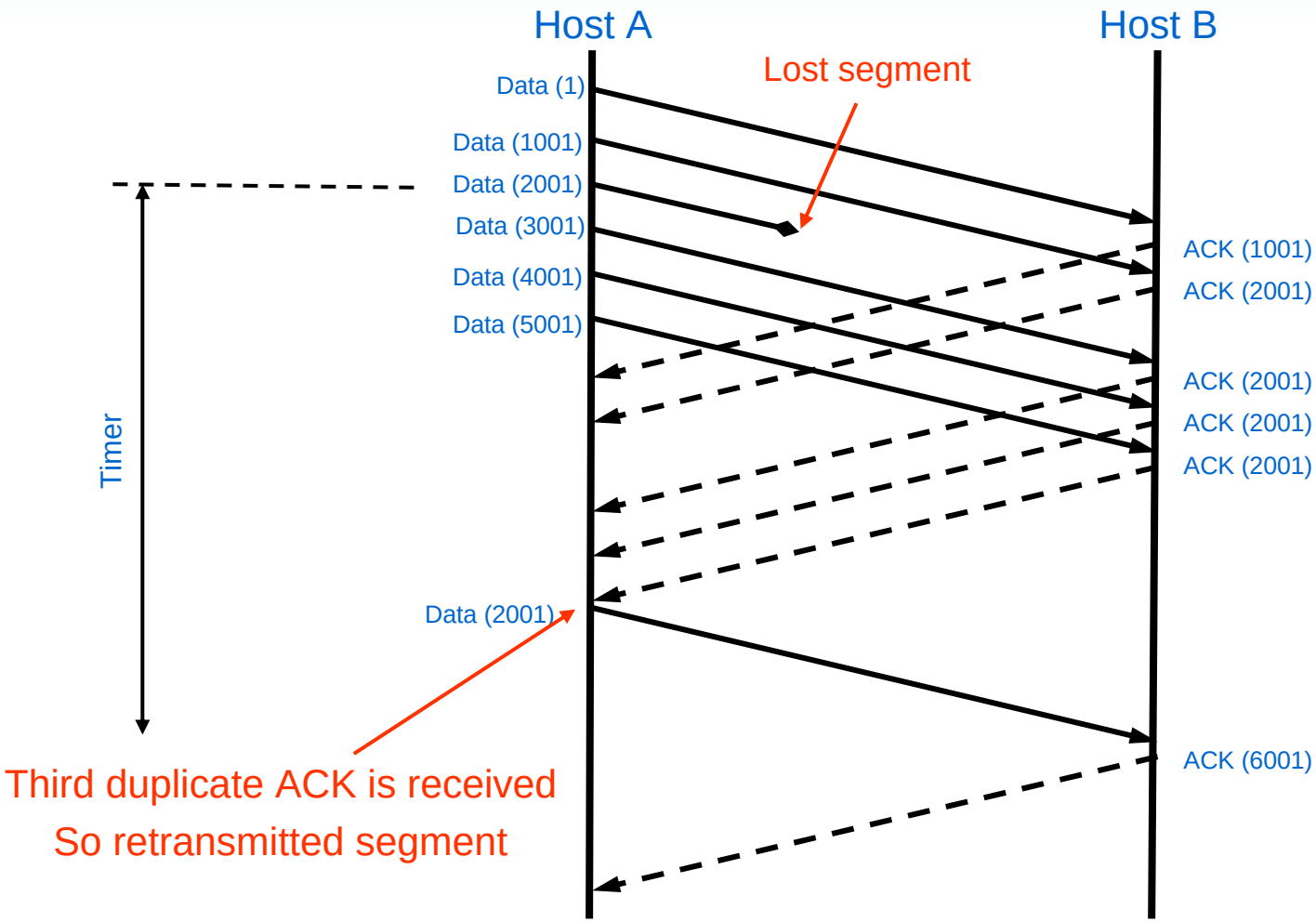
In this example, Host B buffers all bytes it receives, including 3001 to 6000.

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

Fast Retransmit

- The basic retransmission scheme can be improved
  - Waiting for a timeout to expire usually results in poor performance
- Fast Retransmit is available in TCP
  - If 3 duplicate ACKs received, retransmit
  - No need to wait for timeout
    - (However, timeout still occurs if 3 duplicate ACKs are not received)

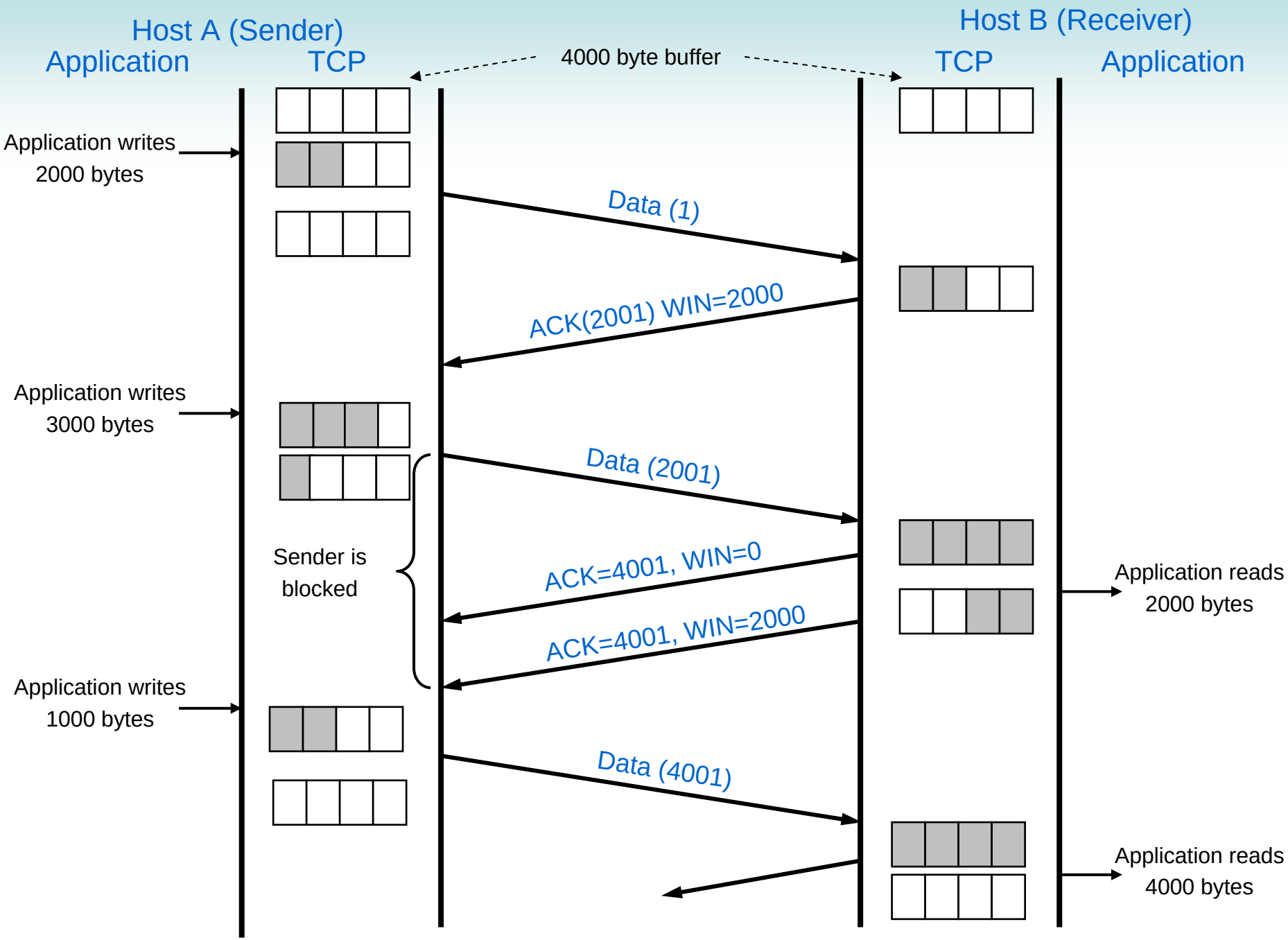
TCP Retransmission (Fast Retransmit)



With Fast Retransmit, in many cases if 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
TCP
Host B (Receiver)
Application
Application writes
2000 bytes
Data (1)
ACK(2001) WIN=2000
Application writes
3000 bytes
Data (2001)
Sender is
blocked
ACK=4001, WIN=0
ACK=4001, WIN=2000
Application reads
2000 bytes
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)