ITS 323 –TRANSPORT PROTOCOLS

1 Overview

Transport protocols such as TCP are very complex. Lets try to summarise the main points about transport protocols.

We will focus on the main transport protocol used today, the Transmission Control Protocol (TCP). We will also mention a very simple alternative, User Datagram Protocol (UDP). Although there are other transport protocols, none are as widely used as TCP and UDP.

2 Feature Comparison: TCP vs UDP

Feature	**TCP**	**UDP**
Ports for multiplexing	✓	✓
Connection Establishment	✓	-
Sequence numbers for bytes	✓	-
Reliability with ACKs and Retransmissions	✓	-
Flow Control	✓	-
Congestion Control	✓	-
Connection Close	✓	-

3 Ports

Since multiple applications may execute on a single computer at the same time, and all applications may use a single transport protocol (e.g. TCP), there needs to be a way to identify which application data belongs to. For example, if the transport layer receives Data, which application does it send the Data to: email, web browser, instant message client, …?

Both TCP and UDP use *port* numbers to identify applications. The application that sends the Data is identified by a port number, and the destination application is also identified by a port number. The transport layer then keeps track of which application is associated with which port number. The source and destination port numbers are sent in the Transport layer headers, so when the Transport layer receives Data, it looks at the destination port number to determine which application should receive the Data.

Some details of port numbers:

- Server applications use well-known or reserved port numbers. As a result, the corresponding client application will know the port number of the server it is to communicate with. For example, a web server uses port number 80. A web client (e.g. browser) will automatically set the destination port number to 80.
- Client applications use private ports. The client will typically pick an unused port number (not used by any other application on the clients computer), and set that as the source port when sending the first message to the server. Hence, when the server receives the message, the server will know the client port number to respond to.

Other details about port numbers are in the Lecture Slides.

4 Transmission Control Protocol (TCP)

4.1 TCP Segment

A TCP packet is often called a segment. It contains a Header and Data and can be shown as follows:

<table>
<tr><td colspan="3">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">Acknowledgement Number</td></tr>
<tr><td>Offset</td><td>Reserv</td><td>Flags</td><td>Window</td></tr>
<tr><td colspan="3">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="4">Options</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

The fields are described in the Lecture Slides. Some important points:

- The Options field is optional
- The Flags fields contain a set of 1-bit flags. If a flag is set (bit 1), then this TCP segment is used for a special purpose. Some common flags are:
  - SYN: This TCP segment is a SYNchronize segment used for connection establishment
  - ACK: This TCP segment carries an ACKnowledgement – the value in the Acknowledgement Number field is to be used.
- Note that with the use of flags, a single TCP segment can carry both Data and an ACK

Although there is only one type of TCP segment, we will often refer to them by name based on the flag set, such as:

- DATA: the segment carries information in the Data field
- SYN: the SYN flag is set
- ACK: the ACK flag is set
- DATA, ACK: the segment carries Data and the ACK flag is set
- SYN, ACK: both the SYN and ACK flags are set

4.2 Sequence Numbers

TCP uses sequence numbers to keep track of the data sent. A sequence number refers to a byte of data. For every new connection, TCP will typically use a different Initial Sequence Number (so there is no confusion between data in a new connection and data from an old connection).

Consider an example.

TCP establishes a connection between A and B (see Section 4.4) and A selects the Initial Sequence Number of 1053. If A has 10,000 bytes to send to B within this connection, then:

- The first byte will have sequence number 1054
- The second byte sequence number 1055
- The third byte sequence number 1056
- …
- The last byte ($10{,}000^{th}$) will have sequence number 11,053

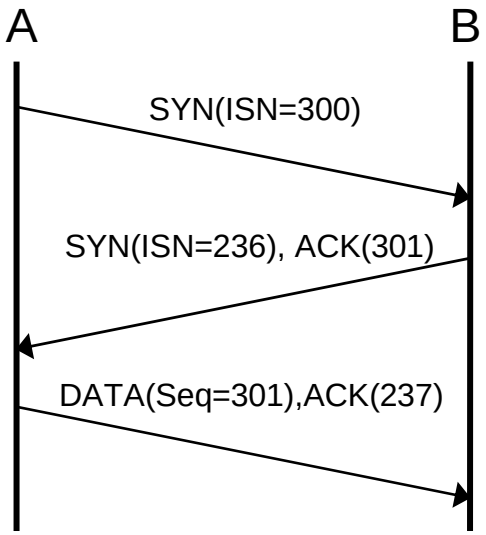
The sequence numbers are used in the Error Control (Section 4.6), Flow Control (Section 4.7) and Congestion Control (Section 4.8).

4.3 Full Duplex Connection

Once a connection is established between two nodes, data can be transfer in either direction. So if A initiates the connection to B, A can send Data to B and B can send Data to A. A and B keep maintain independent sequence numbers, that is, they both choose there own Initial Sequence Numbers.

4.4 Connection Establishment

TCP sets up a connection using a process called a *three-way handshake*. The main purpose of connection establishment is for A and B to agree upon Initial Sequence Numbers. SYN segments are used to establish the connection.



The above example illustrates:

1. A choosing the Initial Sequence Number 300, and sending it in a TCP segment with SYN flag set
2. B accepting A's ISN and Acknowledging with the ACK flag set. B also chooses its own Initial Sequence Number 236 and sends it to B.
3. A ACKnowledges B's Initial Sequence Number, and also sends the first piece of Data to B.

4.5 Data Transfer

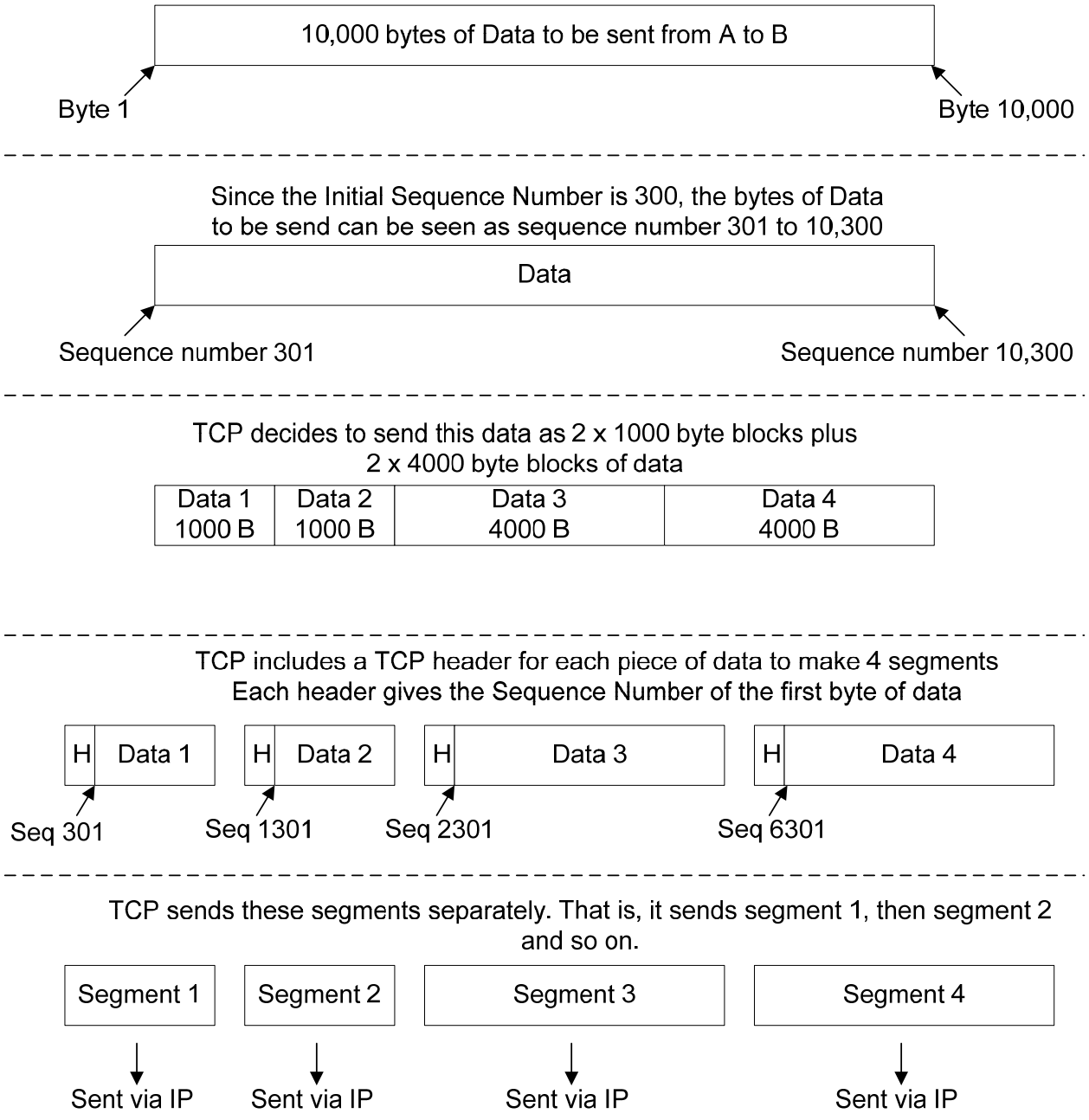
Once a connection is established, A can send data to B and B can send data to A. TCP will choose how many bytes of data to send in one segment (as well as when to send the segment). Many things impact this decision, including Error Control, Flow Control, Congestion Control and implementation specific details.

Assume during a connection between A and B:

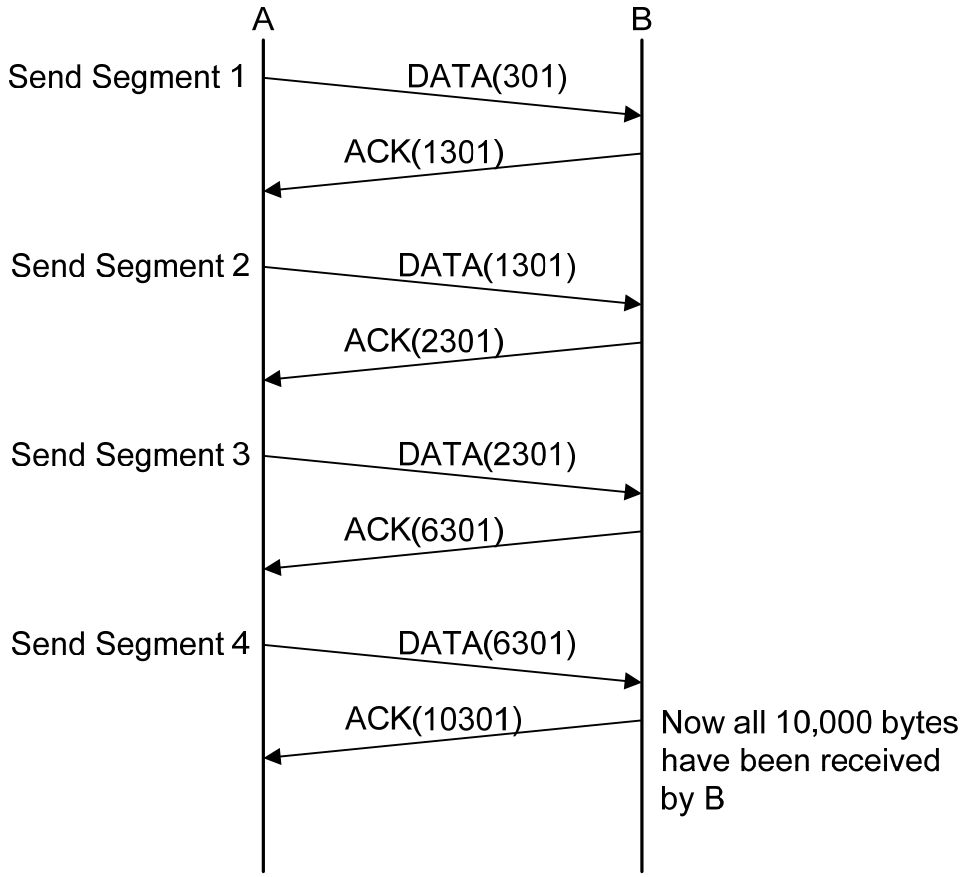
- A wants to send a 10,000 byte file to B
- B wants to send a 2,000 byte file to A
- A has chosen Initial Sequence Number 300
- B has chosen Initial Sequence Number 236

There are many ways this could be done, but lets consider a single example.

When sending Data, the Sequence number field in the TCP header is set to the sequence number of the first byte of data. Lets assume A breaks the 10,000 bytes into 4 TCP segments, the first two are 1000 bytes and the last 2 are 4,000 bytes in length. This can be shown as follows:



With Acknowledgments, the receiver indicates the *next byte expected to be received*. That is, when B acknowledges the first segment (which contain sequence number 301 to 1300 of data), then B includes an Acknowledgement Number of 1301.



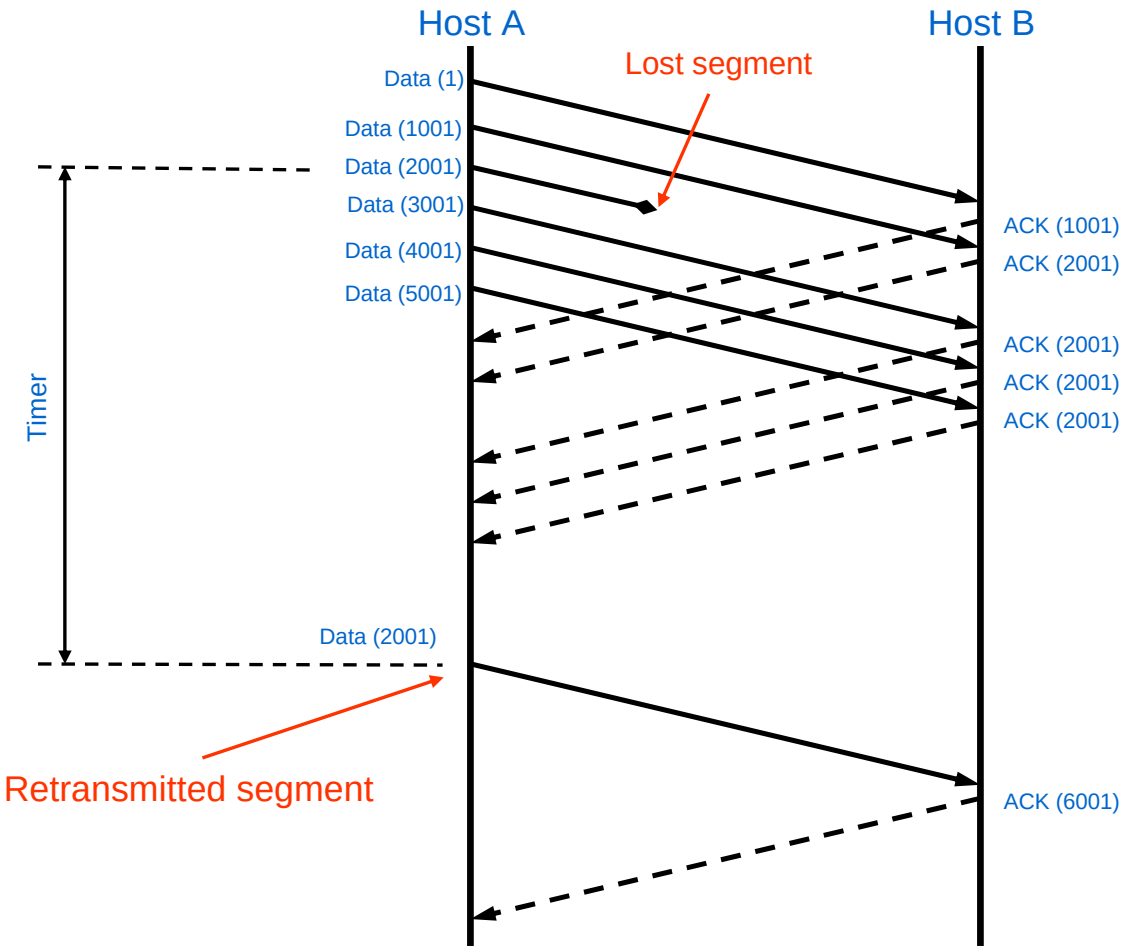
A similar process will be follow for B sending Data to A. In fact, B could send the Data with the ACKs.

4.6 Error Control

Error control in TCP is similar to some of the error control mechanisms we saw at the Data Link layer: if the sender does not receive an ACK, it will retransmit the DATA. (If the sender has to retransmit too many times, it will give up and tell the application there was an error).

4.6.1 Basic Retransmission Scheme

The example used in the Lecture Slides is a good explanation.



In this example, A has 6 segments to send to B, each of 1000 bytes. The Initial Sequence Number chosen by A was 0 (just for simplicity), and so the first sequence number is 1. The six segments are sent to B. Note that A does not wait for an ACK before sending all segments, we assume flow control allows A to send at least 6000 bytes at once.

When B receives a segment, it sends an ACK indicating the next sequence number it expects to receive. But B must receive the data in order. Since the third segment (which contains bytes 2001 to 3000) is lost, when B receives the 4th segment (containing bytes 3001 to 4000), B sends an ACK back saying *it still expects to receive byte 2001*. So although B has received the 4th, 5th and 6th segment (bytes 3001 to 6000), it has not received bytes 2001 to 3000.

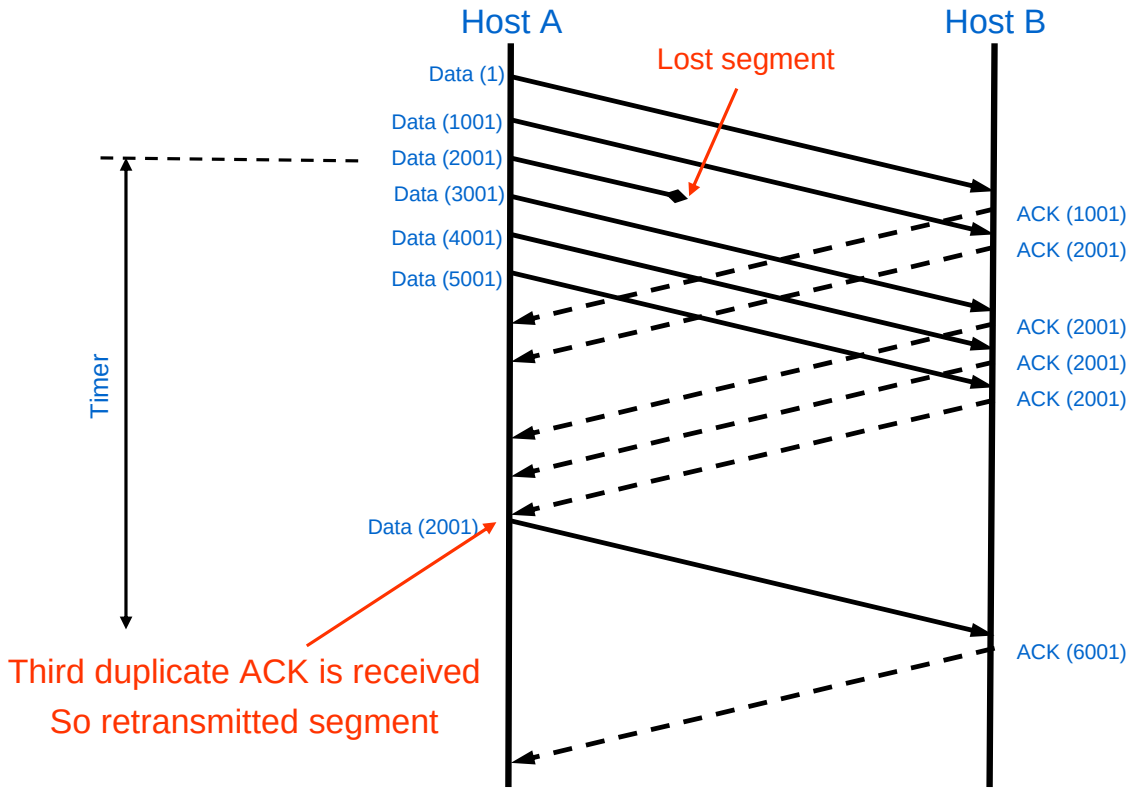
When A transmits a segment, it starts a timer. If no ACK has been received for that Data within a timeout interval, the timer expires and A retransmits. So retransmits the third segment (bytes 2001 to 3000).

Now when B receives the retransmitted third segment, B now has all bytes from 1 up to 6000. So it responds with an ACK saying it expects to receive byte number 6001.

4.6.2 Improved Retransmission Scheme for TCP – Fast Retransmit

The problem with waiting for a timeout, is that A may have to wait a long time (without sending any data) and so the transmission becomes inefficient. As an efficiency improvement, TCP includes the Fast Retransmit feature, which adds the extra condition:

- If three duplicate Acknowledgements are received, then the sender can retransmit.



So using the same example as the Basic Retransmission scheme, when Host A receives the third duplicate ACK with Acknowledgement number 2001 (that is the fourth ACK(2001)), it assumes the segment with byte 2001 was lost and immediately retransmits. Host A will not wait for the timer to expire before retransmitting.

There are many other details and options for TCP retransmission – you may cover them in other courses.

4.7 Flow Control

We already know the purpose of flow control – stop the sender from sending too fast for the receiver – and we know of mechanisms for performing flow control (sliding windows). TCP uses similar mechanisms as we covered in Data Link layer flow control.

TCP flow control uses a sliding window mechanism:

- The sender can only send the number of bytes as given by the Window size (without waiting for an ACK).
- The receiver tells the Sender the current Window size by including it in the Window field of the TCP header.
- The receiver chooses the Window size based on the available space in its buffer.
    - Example: if the receivers buffer is full, then the receiver cannot receive any more data at this time. So the receiver sets the Window value to 0, so the sender cannot any more data.

The example in the Lecture Slides shows how the receiver controls how much the sender can send. Note how as the Receivers TCP buffer changes, the receiver indicates the available space in the Window field of the ACKs.

4.8 Congestion Control

We know the congestion control is aimed at avoiding overloading a network. We don't cover congestion control mechanisms in this course, instead the will be covered in detail in later courses.

4.9 Connection Close

Once there is no more data to send, the connection should be closed. If Host A has no more data to send to B, then A sends a TCP segment with FIN flag set (FINished). B responds with an ACKnowledgement. However, B can still send data to A – the connection is only closed in one direction (from A to B). For B to close the connection, it then sends a FIN to A, and A responds with an ACK. Now the connection is closed in both directions. Neither A nor B can send data.

The Lecture Slides show an example of the connection close procedure.

5 User Datagram Protocol (UDP)

UDP provides the same functionality as IP: the ability to send a packet (or datagram) from source to destination, without guarantee of delivery. This makes UDP very simple to implement and efficient – as long as the application does not need guaranteed delivery!

A typical application that can use UDP is voice over Internet. With these applications, if a packet is lost, the application still works, except the voice quality degrades slightly. That is, the receiver can still understand what the other person is saying.

The Lecture Slides show the format of the UDP datagram and list some example applications.