Department of
Computer Science and Engineering

Title: Implementation of TCP Flow control mechanism

Computer Networking Lab

CSE 312

Green University of Bangladesh

1 Objective(s)

- To gather knowledge about how the TCP transport protocol controls the flow of data between a sender and receiver.
- To implement this flow control mechanism using JAVA.

2 Problem analysis

TCP is one of the protocols of the transport layer for network communication. TCP provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer all rely on TCP. TCP is connection-oriented, and a connection between client and server is established before data can be sent. The server must be listening (passive open) for connection requests from clients before a connection is established. Three-way handshake (active open), retransmission, and error-detection adds to reliability. Thus TCP can maintain various operations to establish perfect communications between a pair of hosts, e.g connection management, error detection, error recovery, congestion control, connection termination, flow control, etc. In this lab, we will have a look at the flow control mechanism of the TCP protocol.

2.1 TCP Flow Control

2.1.1 Introduction

In data communications, TCP uses a flow control mechanism which is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver. It provides a mechanism for the receiver to control the transmission speed, so that the receiving node is not overwhelmed with data from transmitting node. Flow control is important because it is possible for a sending computer to transmit information at a faster rate than the destination computer can receive and process it. This can happen if the receiving computers have a heavy traffic load in comparison to the sending computer, or if the receiving computer has less processing power than the sending computer.

2.1.2 General Mechanism

When we need to send data over a network, this is normally what happens. The general process of communication through all the layer is illustrated in Figure 1a. The sender application writes data to a socket, the transport layer protocol, TCP will wrap this data in a segment and hand it to the network layer (e.g. IP), that will somehow route this packet to the receiving node. On the other side of this communication, the network layer will deliver this piece of data to TCP, that will make it available to the receiver application as an exact copy of the data sent, meaning if will not deliver packets out of order, and will wait for a retransmission in case it notices a gap in the byte stream.

Meanwhile, for the control of accurate flow using TCP few entities exist inside the data packets along with some operations. Zooming in the general flow among layers, if we consider only transport layer we get the summary in Figure 1b. TCP stores the data it needs to send in the send buffer, and the data it receives in the receive buffer. When the application is ready, it will then read data from the receive buffer. Flow Control is all about making sure we don't send more packets when the receive buffer is already full, as the receiver wouldn't be able to handle them and would need to drop these packets. To control the amount of data that TCP can send, the receiver will advertise its Receive Window ($rwnd$), that is, the spare room in the receive buffer. This is shown in Figure 2a. Every time TCP receives a packet, it needs to send an ack message to the sender, acknowledging it received that packet correctly, and with this ack message it sends the value of the current receive window, so the sender knows if it can keep sending data.

Suppose host A sends a data to host B. Then we need to know the following two variables:

- LastByteRead: the number of the last byte in the data stream read from the buffer by the application process in B
- LastByteRcvd: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B

Now since TCP is not permitted to overflow the allocated buffer, we must have the following equation to be true:

$$LastByteRcvd - LastByteRead \leq RcvBuffer \tag{1}$$

The receive window, denoted *rwnd* is set to the amount of spare room in the buffer:

$$rwnd = RcvBuffer - [LastByteRcvd - LastByteRead] \tag{2}$$

Because the spare room changes with time, *rwnd* is dynamic. Host B keeps the value of this *rwnd* in the receive window filed of the TCP segment going from B to A. Initially, Host B sets *rwnd* = RcvBuffer. Host A in turn keeps track of two variables, **LastByteSent** and **LastByteAcked**. The difference between these two variables, **LastByteSent - LastByteAcked**, is the amount of unacknowledged data that A has sent into the connection. Simpley put, By keeping the amount of unacknowledged data less than the value of *rwnd*, Host A is assured that it is not overflowing the receive buffer at Host B. Thus, Host A makes sure throughout the connection's life that:

$$LastByteSent - LastByteAcked \leq rwnd \tag{3}$$

This is the fundamental and general equation maintained by any sender throughout the data communication to help in the flow control of TCP. The following method is the most widely used to maintain the above equation for flow control mechanism using TCP.

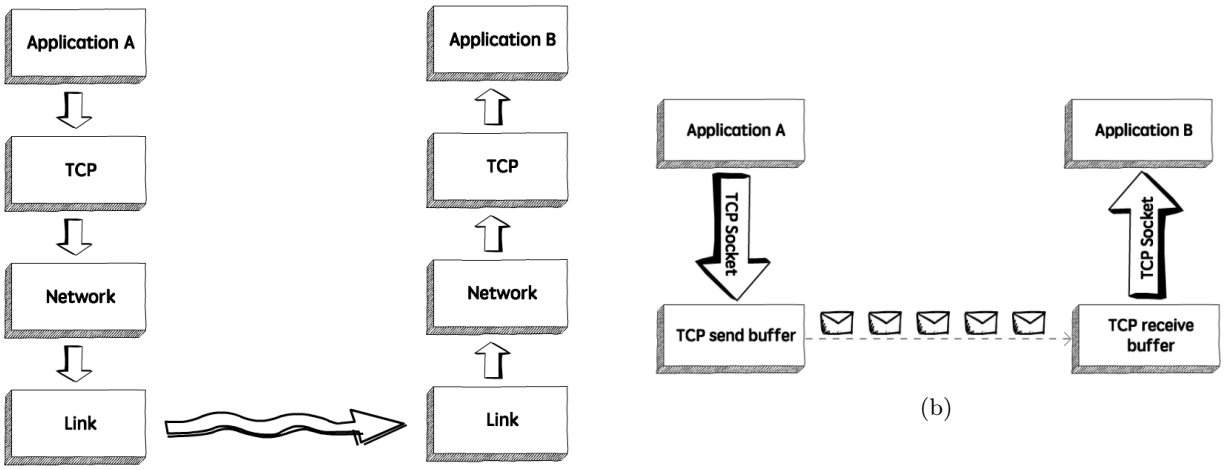

Figure 1: (a) General flow of data through different layers between two hosts (b) Flow of data for the hosts only in case of transport layer

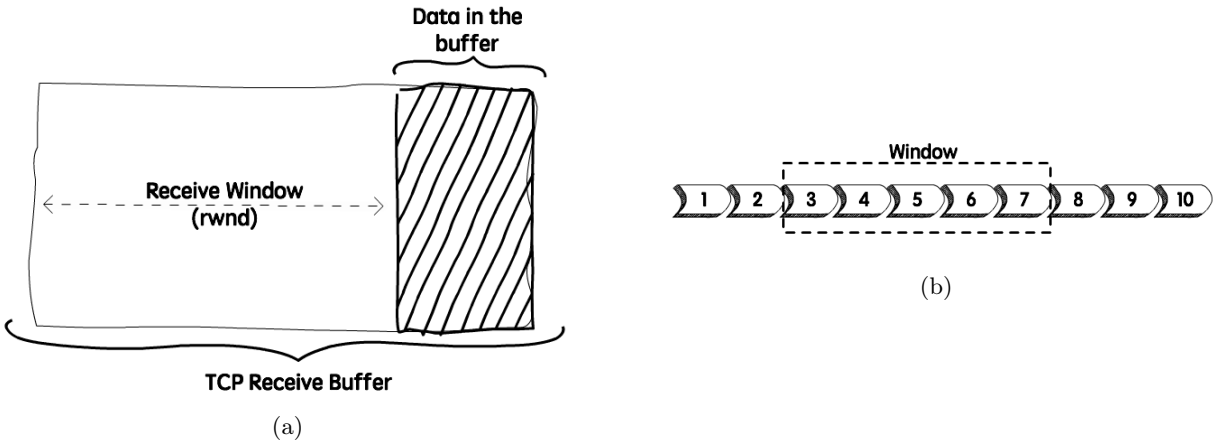

Figure 2: (a) Receive Window, *rwnd* for help in flow control for the sender (b) The Sliding Window protocol example. As soon as packet 3 is acked, the window is slided to the right and send the packet 8 for a window size of 5.

2.1.3 Sliding Window protocol for Flow Control

TCP uses a sliding window protocol to control the number of bytes in flight it can have. In other words, the number of bytes that were sent but not yet acked.

Let's say we want to send a 150000 bytes file from node A to node B. TCP could break this file down into 100 packets, 1500 bytes each. Now let's say that when the connection between node A and B is established, node B advertises a receive window of 45000 bytes, because it really wants to help us with our math here.

Seeing that, TCP knows it can send the first 30 packets (1500 * 30 = 45000) before it receives an acknowledgment. If it gets an ack message for the first 10 packets (meaning we now have only 20 packets in flight), and the receive window present in these ack messages is still 45000, it can send the next 10 packets, bringing the number of packets in flight back to 30, that is the limit defined by the receive window. In other words, at any given point in time it can have 30 packets in flight, that were sent but not yet acked.

Now, if for some reason the application reading these packets in node B slows down, TCP will still ack the packets that were correctly received, but as these packets need to be stored in the receive buffer until the application decides to read them, the receive window will be smaller, so even if TCP receives the acknowledgment for the next 10 packets (meaning there are currently 20 packets, or 30000 bytes, in flight), but the receive window value received in this ack is now 30000 (instead of 45000), it will not send more packets, as the number of bytes in flight is already equal to the latest receive window advertised.

So all in all, the receiver maintains a sliding window method to move the window position along the data bytes received and acknowledged. This information is sent to the sender using the *rwnd* value usign the filed in the TCP segment header. Thus overall throughout the entire data connection, the sender must maintain the aforementioned equation to maintain flow control:

$$LastByteSent - LastByteAcked \leq rwnd \tag{4}$$

The Sliding Window can be displayed as a clocking mechanism with the help of the sequence numbers of the TCP data segments. This is illustrated in Figure 3.

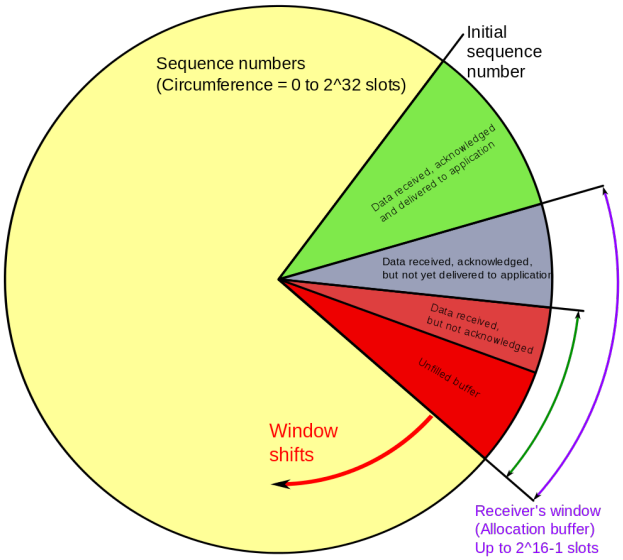

Figure 3: TCP sequence numbers and receive windows behave very much like a clock. The receive window shifts each time the receiver receives and acknowledges a new segment of data. Once it runs out of sequence numbers, the sequence number loops back to 0.

3 Algorithm

In this section, we observe the algorithm of the sliding window protocol using some formal steps, to visualize this flow control technique.

Algorithm 1: TCP flow control using Sliding Window method

1. Step 1: Begin
2. Step 2: Generate a random variable that gives the total number of frames to be transmitted.
3. Step 3: Set the size of the window.
4. Step 4: Generate a random number less than or equal to the size of the current window and identify the number of frames to be transmitted at a given time.
5. Step 5: Transmit the frames and receive the acknowledgement for the frames sent.
6. Step 6: Find the remaining frames to be sent.
7. Step 7: Find the current window size.
8. Step 8: If an acknowledgement is not received for a particular frame retransmit the frames from that frame again.
9. Step 9: Repeat the steps 4 to 8 till the number of remaining frames to be send becomes zero.
10. Step 10: End

4 Implementation in Java

```
1  /* TCP Flow Control using Sliding Window technique*/
2  package tcpflowcontrol;
3
4  import static java.lang.Math.random;
5  import java.util.Scanner;
6  import java.util.Random;
7
8  /**
9   *
10  * @author Md. Ehsan Shahmi Chowdhury, Lecturer, CSE, Green University of
        Bangladesh
11  */
12 public class TcpFlowControl {
13
14     public static int generateFrame(int winSize) {
15         //this function returns a randomly generated number of generated frames
16         //that must be within the size of declared window of the receiver
17
18         Random random = new Random();
19         int i, noOfGeneratedFrame;
20
21         noOfGeneratedFrame = random.nextInt(500) % winSize;
22         //generated random value within 500 is mod within the window size
23         //since number of sent frames cannot be more than the provided window
               size
24
25         if(noOfGeneratedFrame == 0) return winSize;
26         //if this value is zero, it means it is the window size and not more
27         //else return this value as the number of generated frame in this
               iteration of the main while loop
28         else return noOfGeneratedFrame;
29     }
30
31     public static int generateAck(int noOfSent) {
32         //this function returns a randomly generated number of acks
33         //that must be within the total number of frames to be sent by sender
```

```
34
35          Random random = new Random();
36          int i, noOfAckFrame;
37
38          noOfAckFrame = (int) (random.nextInt(500) % noOfSent);
39          //number of ack-ed frames is mod within the total number of frames to-be
                sent
40          //since number of ack-ed frames can never be more than total number of
               frames to-be sent
41
42          return noOfAckFrame;
43      }
44
45      public static void main(String[] args) {
46          int noOfFrame, winSize, startByte = 0, endByte = 0, noOfAck = 0,
               noOfSent = 0;
47
48          //The scanner class to take in any user inputs
49          Scanner scn = new Scanner(System.in);
50
51          System.out.println("Enter the total no of frame: ");
52          //The number of frames to be sent by sender
53          noOfFrame = scn.nextInt();
54
55
56          System.out.println("Enter the window size: ");
57          //The entire window size that will slide over the to-be sent frames
58          winSize = scn.nextInt();
59
60          int dueFrame = noOfFrame;
61          //At this stage, the number of due frames is equal to total number of
               frames
62
63
64          while(dueFrame>=0){
65              //Following loop iterates until all frames have been sent
66
67              noOfSent = generateFrame(winSize);
68              //number of sent frames are randomly generated using this function;
                   not more than window size
69
70              endByte += noOfSent;
71              //the last byte upto now is previous byte added by the already sent
                   frames
72              if(endByte > noOfFrame)
73                  endByte = noOfFrame;
74              //if last byte exceeds total frame to be sent, (which is not
                   possible practically)
75              //then reduce to the highest frames that could be sent,
76              //which is number of frames to be send by sender
77
78              for(int i = startByte+1; i <= endByte; i++){
79                System.out.println("Sending frame " + i);
80              }
81              //all the sent frames up to the ack-ed frames are printed out
82
83              noOfAck = generateAck(noOfSent);
84              //number of ack-ed frames are generated randomly using function
```

```
85
86              startByte += noOfAck;
87              //the first byte of frame that is already acknowledged by receiver
88
89              if(startByte > noOfFrame)
90                  startByte = noOfFrame;
91              //if first byte exceeds total frame to be sent, (which is not
                    possible practically)
92              //then reduce to the highest frames that could be sent,
93              //which is number of frames to be send by sender
94
95              System.out.println("Acknowledgement for the frame upto " + startByte
                    );
96              //printing out upto how many frames have been ack-ed by receiver
97
98              dueFrame -= noOfAck;
99              //number of frames left will be reduced from previous value
100             //by the number of frames ack-ed by receiver
101
102             endByte = startByte;
103             //the last byte is updated to the value last ack-ed by receiver
104         }
105         System.out.println("\nThe Sliding Window Protocol concludes here.");
106     }
107 }
```

5 Input/Output

Output of the above program is given below.

```
Enter the total no of frame:
15
Enter the window size:
4
Sending frame 1
Sending frame 2
Sending frame 3
Acknowledgement for the frame upto 1
Sending frame 2
Sending frame 3
Sending frame 4
Acknowledgement for the frame upto 3
Sending frame 4
Sending frame 5
Sending frame 6
Acknowledgement for the frame upto 4
Sending frame 5
Sending frame 6
Acknowledgement for the frame upto 5
Sending frame 6
Sending frame 7
Sending frame 8
Sending frame 9
Acknowledgement for the frame upto 6
```

```
Sending frame 7
Sending frame 8
Sending frame 9
Acknowledgement for the frame upto 6
Sending frame 7
Sending frame 8
Sending frame 9
Sending frame 10
Acknowledgement for the frame upto 8
Sending frame 9
Sending frame 10
Sending frame 11
Sending frame 12
Acknowledgement for the frame upto 10
Sending frame 11
Sending frame 12
Acknowledgement for the frame upto 11
Sending frame 12
Sending frame 13
Sending frame 14
Sending frame 15
Acknowledgement for the frame upto 14
Sending frame 15
Acknowledgement for the frame upto 14
Sending frame 15
Acknowledgement for the frame upto 15
Acknowledgement for the frame upto 15

The Sliding Window Protocol concludes here.
```

6 Discussion & Conclusion

In summary, TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the receive window field the amount of additionally received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgement and window update from the receiving host.

Based on the focused objectives to understand about TCP and its flow control mechanism, the additional thinking exercise made the student more confident towards the fulfilment of the objectives.

7 Lab Task (Please implement yourself and show the output to the instructor)

1. What will happen the receive window, *rwnd*, value reaches zero and is advertised to the sender?
2. How will the sender resume its sending of data again, once it gets a *rwnd* value to be equal to zero?
3. What do you understand by *Silly Window Syndrome*? Why does this problem occur?

8 Lab Exercise (Submit as a report)

Lab report of this experiment is connected with the next experiment, hence will be provided there.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.