



UNIVERSITY OF DHAKA

CSE:3111 COMPUTER NETWORKING LAB

Lab Report 5

Implementation of TCP Flow Control

Shariful Islam Rayhan

Roll: 41

Md Sakib Ur Rahman

Roll: 37

Submitted to:

Dr. Md. Abdur Razzaque

Dr. Muhammad Ibrahim

Dr. Md. Redwan Ahmed Rizvee

Dr. Md. Mamun Or Rashid

Submitted on: 2024-02-21

1 Introduction

When computers communicate over the internet, they use a protocol called TCP to ensure messages get where they need to go reliably. TCP flow control is like a traffic cop for this data, making sure it doesn't overwhelm the receiving computer.

Imagine sending a large file to a friend. TCP flow control ensures that your friend's computer can keep up with the data you're sending, preventing it from getting swamped. It's like a conversation between your computer and your friend's, making sure they're both on the same page about how much data can be sent and when.

In simpler terms, TCP flow control is all about keeping the data flow smooth and steady, so everyone stays happy and nothing gets lost along the way.

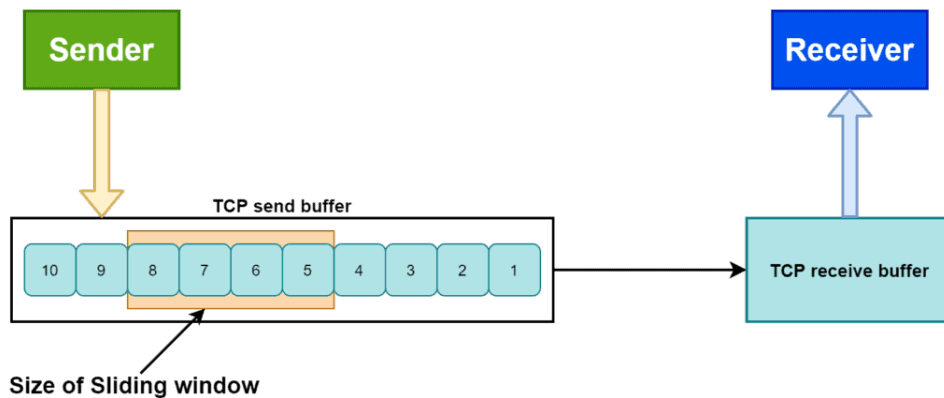


Figure 1: Flow Control in TCP/IP Protocol

1.1 Objectives

- Gaining a clear understanding of the sliding window flow control mechanism used in TCP communication.
- Investigating the impact of varying receive window sizes on TCP performance
- Gaining practical experience by implementing TCP flow control algorithms

2 Theory

TCP flow control is a mechanism used in the TCP protocol to manage the rate of data transmission between two devices over a network connection. It is designed to prevent the receiver from being overwhelmed by a flood of incoming data, and to ensure that data is transmitted smoothly and without errors.

TCP flow control works by using a sliding window protocol, where the receiver advertises the amount of data it is able to receive by sending a message containing the number of bytes of available buffer space in its receive window. The sender then limits the amount of data it sends to the receiver based on this advertised window size. For example, if the receiver advertises a receive window size of 10,000 bytes, the sender will transmit up to 10,000 bytes of data and then wait for an acknowledgement from the receiver before sending additional data. This process is repeated until all data has been transmitted.

TCP flow control helps prevent network congestion and reduces the likelihood of packet loss or corruption. It is an essential component of the TCP protocol, ensuring that data is transmitted efficiently and reliably across a network connection.

3 Methodology

We have followed the lab instruction to implement flow control in TCP

3.1 Preliminary Work

We have declared a custom header size of 12 bytes which contain

- Sequence number of 4 bytes
- Acknowledgement number of 4 bytes
- Ack bit represented by one byte
- sf bit represent by one byte
- Receive window size information of 2 bytes

Parsing and packaging of these data are done by the makeheader and fromheader function. We have explicitly declared a sender and receiver using python TCP sockets and we simulated TCP on it. We have a fixed MSS (Maximum Segment Size) among both the hosts so that the packet reading stays consistent. This is the abstraction of the TCP packet structure that we have used to simulate TCP.

3.2 Flow Control

The idea of flow control is that the sender should not send so much data that it overwhelms the receiver. It is fully dependent on the behaviour of the network. In an actual web application the TCP buffer is cleared by the application layer randomly when needed thus this capacity to accept data changes wildly. Here to abstract this we have implemented a scheme for the cumulative ack. The receiver sends an ack if one these two events occur

- If we have filled the buffer.
- Timer on the receiver to runs out.

In both cases we have sent an ack, acknowledging the latest in order data. Along with this ack we calculate the empty space in the buffer. this was calculated by subtracting the received data in the buffer from the received buffer size. This value is then used to restrict the flow of data dynamically on the sender size. The sender window is restricted to the maximum the receiver can handle.

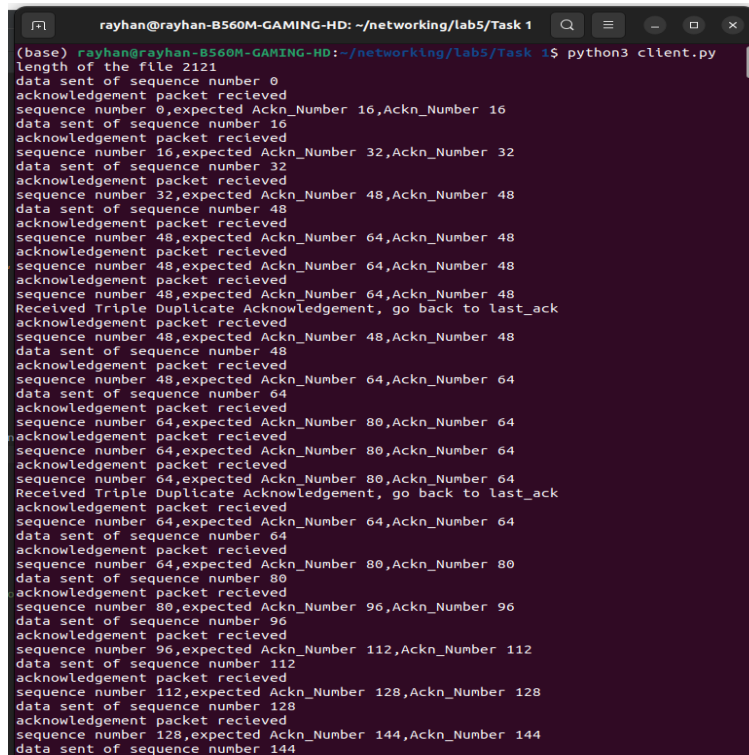
4 Experimental result

4.1 Client

This is the snapshot from the client side after executing client. After calculating the size of file, client sends the read byte packet by packet. Every packet has a sequence number and client takes the acknowledgement number from the server and matches with the expected sequence number. If it gets three times same acknowledgement, it transmits that packet again.

To see actually what is happening, we have printed some necessary values of variables like sequence number of sent data, expected seq from server and acknowledgement from server.

After sending the packet of sequence number 48, the server is sending three times 48 acknowledgement. As a result, we have sent it again. The same thing happened to packet with sequence number 64 also.

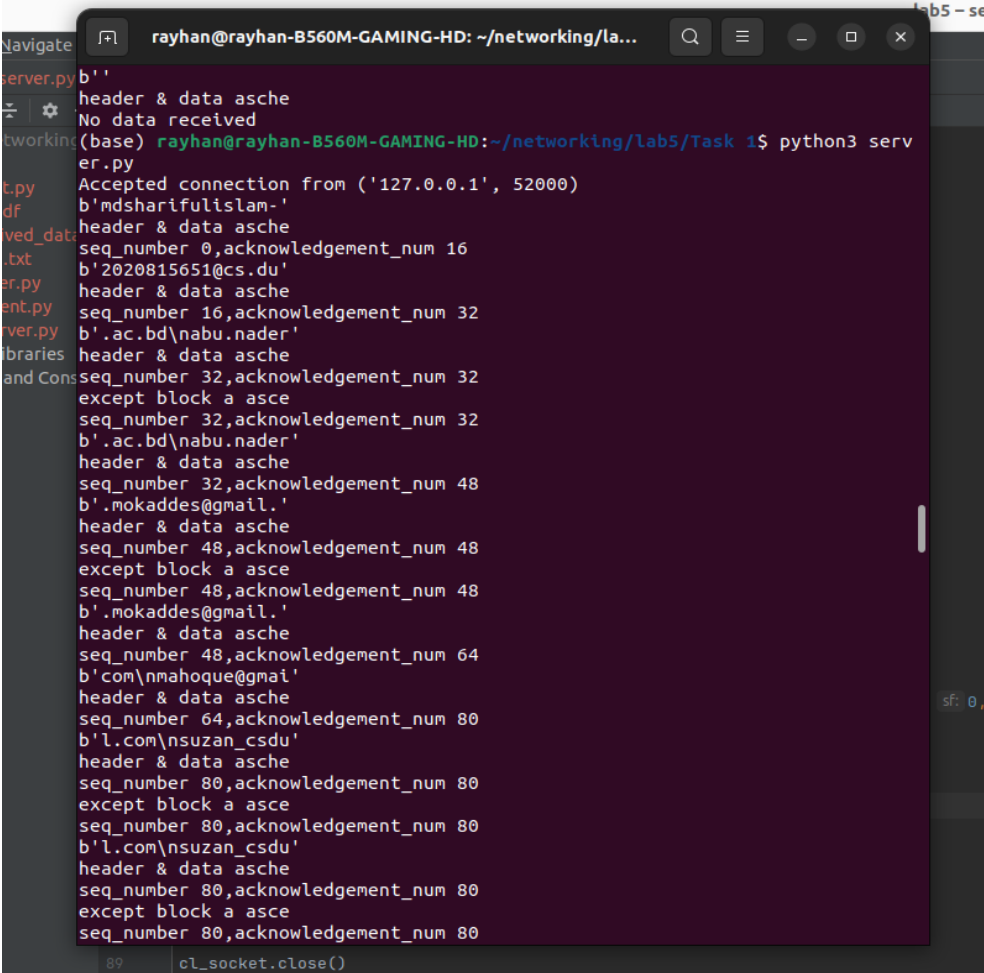
A terminal window titled 'rayhan@rayhan-B560M-GAMING-HD: ~/networking/lab5/Task 1' displays the output of a Python script named 'client.py'. The script simulates a client's flow control logic. It starts by printing the file length (2121) and then enters a loop where it sends data packets and receives acknowledgements. The logs show that for sequence numbers 48 and 64, the client receives three duplicate acknowledgements from the server, which triggers it to retransmit the packet. The terminal output is as follows:

```
(base) rayhan@rayhan-B560M-GAMING-HD:~/networking/lab5/Task 1$ python3 client.py
length of the file 2121
data sent of sequence number 0
acknowledgement packet recieved
sequence number 0,expected Ackn_Number 16,Ackn_Number 16
data sent of sequence number 16
acknowledgement packet recieved
sequence number 16,expected Ackn_Number 32,Ackn_Number 32
data sent of sequence number 32
acknowledgement packet recieved
sequence number 32,expected Ackn_Number 48,Ackn_Number 48
data sent of sequence number 48
acknowledgement packet recieved
sequence number 48,expected Ackn_Number 64,Ackn_Number 48
acknowledgement packet recieved
sequence number 48,expected Ackn_Number 64,Ackn_Number 48
acknowledgement packet recieved
sequence number 48,expected Ackn_Number 64,Ackn_Number 48
Received Triple Duplicate Acknowledgement, go back to last_ack
acknowledgement packet recieved
sequence number 48,expected Ackn_Number 48,Ackn_Number 48
data sent of sequence number 48
acknowledgement packet recieved
sequence number 48,expected Ackn_Number 64,Ackn_Number 64
data sent of sequence number 64
acknowledgement packet recieved
sequence number 64,expected Ackn_Number 80,Ackn_Number 64
acknowledgement packet recieved
sequence number 64,expected Ackn_Number 80,Ackn_Number 64
acknowledgement packet recieved
sequence number 64,expected Ackn_Number 80,Ackn_Number 64
Received Triple Duplicate Acknowledgement, go back to last_ack
acknowledgement packet recieved
sequence number 64,expected Ackn_Number 64,Ackn_Number 64
data sent of sequence number 64
acknowledgement packet recieved
sequence number 64,expected Ackn_Number 80,Ackn_Number 80
data sent of sequence number 80
acknowledgement packet recieved
sequence number 80,expected Ackn_Number 96,Ackn_Number 96
data sent of sequence number 96
acknowledgement packet recieved
sequence number 96,expected Ackn_Number 112,Ackn_Number 112
data sent of sequence number 112
acknowledgement packet recieved
sequence number 112,expected Ackn_Number 128,Ackn_Number 128
data sent of sequence number 128
acknowledgement packet recieved
sequence number 128,expected Ackn_Number 144,Ackn_Number 144
data sent of sequence number 144
```

Figure 2: Client View of Flow Control

4.2 Server

This is the snapshot from the server side after executing server.py .Server recieve packet of data consisting of header and payload and spilt it into two parts.recieved data is added to data buffer and header is extracted to find the sequence number of the packet.Then send the acknowledgement number to the client.If it doesn recieve a consecutive sequence numebr data it sends that acknowledgment again and again.



```
server.py b' '
header & data asche
No data received
(base) rayhan@rayhan-B560M-GAMING-HD: ~/networking/lab5/Task 1$ python3 serv
er.py
Accepted connection from ('127.0.0.1', 52000)
b'mdsharifulislam-'
header & data asche
seq_number 0,acknowledgement_num 16
b'2020815651@cs.du'
er.py header & data asche
ent.py seq_number 16,acknowledgement_num 32
rver.py b'.ac.bd\nabu.nader'
libraries header & data asche
and Cons seq_number 32,acknowledgement_num 32
except block a asche
seq_number 32,acknowledgement_num 32
b'.ac.bd\nabu.nader'
header & data asche
seq_number 32,acknowledgement_num 48
b'.mokaddes@gmail.'
header & data asche
seq_number 48,acknowledgement_num 48
except block a asche
seq_number 48,acknowledgement_num 48
b'.mokaddes@gmail.'
header & data asche
seq_number 48,acknowledgement_num 64
b'com\nmahoque@gmai'
header & data asche
seq_number 64,acknowledgement_num 80
b'l.com\nsuzan_csdu'
header & data asche
seq_number 80,acknowledgement_num 80
except block a asche
seq_number 80,acknowledgement_num 80
b'l.com\nsuzan_csdu'
header & data asche
seq_number 80,acknowledgement_num 80
except block a asche
seq_number 80,acknowledgement_num 80
89 cl_socket.close()
```

Figure 3: Server View of Flow Control

5 Experience

1. We have seen how flow control in tcp works.
2. Had an opportunity to see the working principle sliding windows algorithm.
3. It was nice to implement server and client for tcp flow control

References

- [1] Flow Control VS Congestion Control : <https://www.javatpoint.com/flow-control-vs-congestion-control>
- [2] Flow Control and Window Size : https://www.youtube.com/watch?v=412_BCr-bhw&t=177s