Mini Project 2 Report

Group Members:	Amy Jia Ying Tan Timothy Tan
Group #:	62
Course:	CMPT 371 D100 [Fall 2024]
Date:	December 2nd, 2024

Table of Contents

1) Design & Implement your protocol

To implement our own flavor of connection-oriented, reliable and pipelined protocol with flow and congestion control, multiple mechanisms are specified. An overview of these mechanisms are listed in the below table.

a) Socket Type:	UDP
b) Feedback Mechanism:	Acknowledgements (ACKs)
c) Error and Loss Detection:	Checksum and Countdown Timers
d) Pipelining Protocol:	Go-Back-N (GBN)
e) Connection-Oriented Mechanism:	Two-way handshake
f) Flow Control Mechanism:	Receiver-side buffering
g) Congestion Control Mechanism:	Additive Increase Multiplicative Decrease (AIMD)

a) Socket Type:

The socket type has to be one that does not provide reliability, pipelining or flow and congestion control by default so UDP is chosen to implement our own protocol.

b) Feedback Mechanism:

For our feedback mechanism to indicate errors and loss, we plan on just having acknowledgements sent to the sender from the receiver. This will be indicated by an ACK number in the payload of the UDP segment.

c) Error and Loss Detection:

Our mechanism for error and loss detection are checksums and countdown timers for acknowledgements. We plan on having timers in our code to indicate whether loss has happened between the sender and receiver. UDP implements checksum by default so this does not need to be considered since our program is built to only run on the same local machine.

d) Pipelining Protocol:

The Go-Back-N pipelining protocol will be used to send multiple packets with a sender window. This window will have a default value of 3 packets and will grow or be limited by the congestion and flow control mechanisms.

The amount of data sent in each packet will be limited by an arbitrary max segment size set to 12 bytes to see the effects of pipelining. This means long strings will be split into multiple packets if it exceeds a length of 12 bytes.

e) Connection-Oriented Mechanism:

A two-way handshake will be used to establish a connection between client and server. We will use SYN and FIN flags in the payload to indicate connection establishments and closes.

To open a connection, the udp_server.py then the udp_client.py simply needs to be run and they will automatically handshake before establishing a connection.

To close the connection, a reserved keyword “quit” is typed by the client to initiate a closure between client and server. The client will initiate a termination and it will close once it receives an ACK from the server. The server will immediately terminate its connection with the client after receiving data with the FIN flag set.

f) Flow Control Mechanism:

Our flow control mechanism will be receiver-side buffering where the receiver sends the size of their window through a flag to the sender upon connection. The sender will then limit the amount of unACKed data sent to the receiver based on this window size to prevent the receiver from being overwhelmed.

g) Congestion Control Mechanism:

Our congestion control mechanism will be Additive Increase Multiplicative Decrease (AIMD) where we increase the sender’s window size by 1 maximum segment size on every ACK received until loss is detected where we cut the sender window in half. Note that our flow control mechanism will limit the growth of the sender’s window to the receiver’s window.

2) Analyze Traffic

Testing Procedure

In order to test our implemented protocol and congestion control, we ran two tests and captured the packets that were sent and received. For each test, we sent the same four messages from the client to the server.

Note that to simulate loss, we dropped packets randomly with a set probability of 20% on both the client and server for sent client data, handshake messages and acknowledgements. They are dropped right after our sockets receive data. These dropped packets are managed and detected as seen in our analysis.

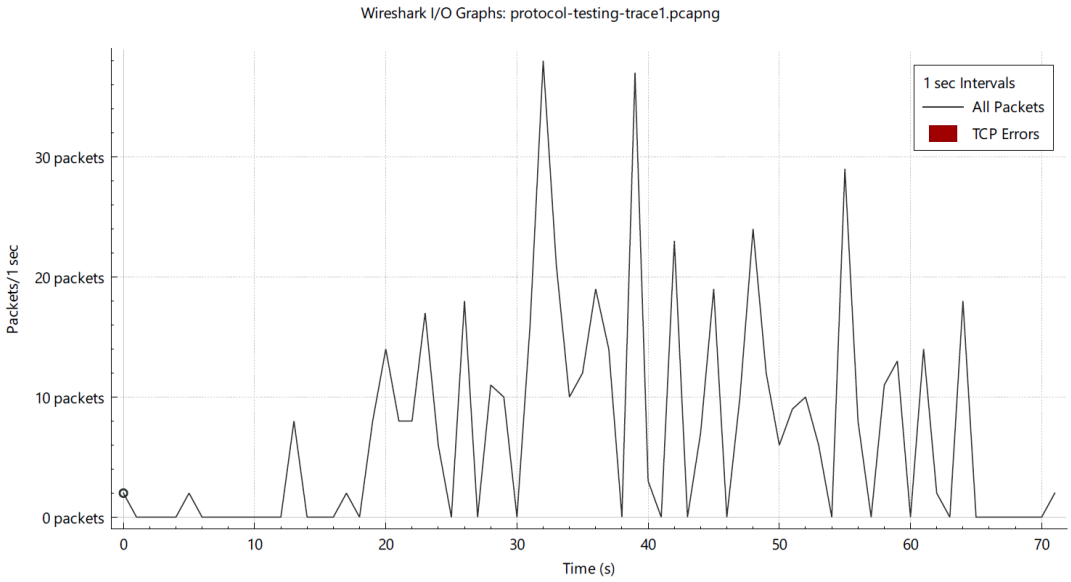
Below is a table with the messages we sent as we captured packets:

Message Order	Message Sent
1	Hello world!
2	Now we will send a very short message.
3	Hi!
4	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et libero nibh. Aliquam sit amet dui dictum, eleifend nisl sed, feugiat elit. Curabitur quis dui sed odio mollis ultricies. Quisque fermentum mollis quam. Phasellus ut consequat risus, quis gravida lacus. Duis fringilla congue porta. Suspendisse vitae magna et dolor scelerisque vulputate a id arcu. Sed vitae dolor mauris. Nunc vel orci nec diam aliquet tempus at convallis nunc. Nulla quis pharetra lectus. Fusce ut eleifend erat. Duis quis iaculis turpis. Nullam rhoncus lacinia neque et rutrum. Praesent lacinia, eros a consequat semper, purus nisl bibendum mauris, in pharetra leo lacus eget augue. In hac habitasse platea dictumst. Ut quis consectetur nibh. Quisque ligula metus, sagittis vitae augue eu, ultrices ultricies nulla. In pretium magna ac turpis semper feugiat. In hac habitasse platea dictumst. Suspendisse id convallis urna, at mollis massa. Phasellus fringilla volutpat auctor.

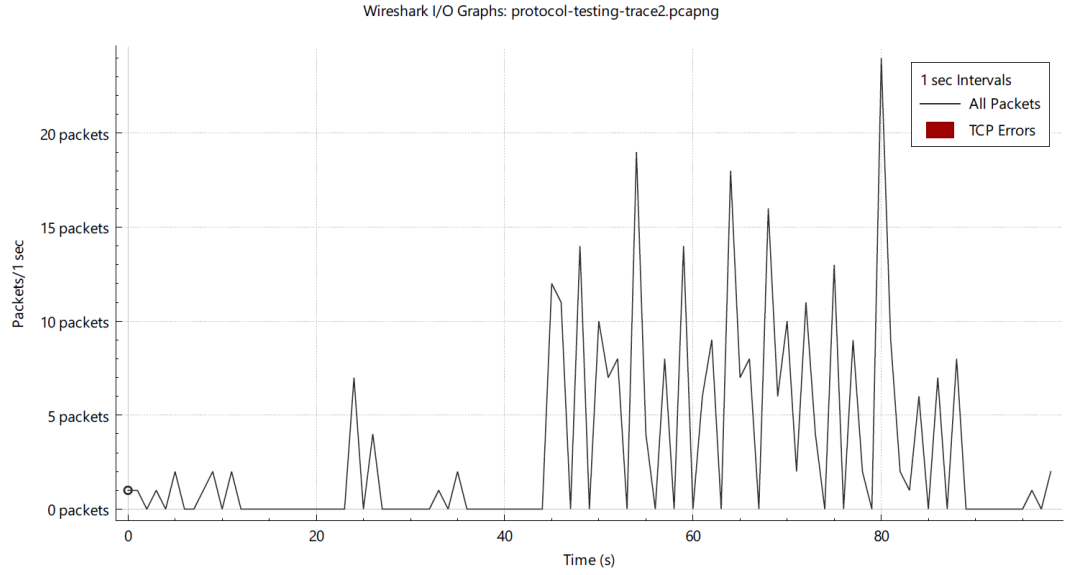
Results Analysis

To showcase our implementation of flow control, we tested our code with different receiver window sizes.

In our first test, we set the **receiver window** in the server to be equal to **10**. The results from this test can be seen in the trace file called protocol-testing-trace1.pcapng, in the I/O graph called io-graph1.pdf, and in the flow graph called flow-graph1.pdf.



In our second test, we set the **receiver window** in the server to be equal to **5**. The results from this test can be seen in the trace file called protocol-testing-trace2.pcapng, in the I/O graph called io-graph2.pd*f*, and in the flow graph called flow-graph2.pdf.



As you can see in our graphs, we have implemented congestion control as the amount of packets sent at a time increases over time until it detects loss in which the packets sent decreases drastically as the window size gets halved. A higher amount of packets are also sent when the receiver window is higher which indicates that our flow control works to prevent the receiver from being overwhelmed.

Our flow graphs (the flow graph for the first test is shown below) shows that our protocol is connection-oriented as we establish a connection between two hosts when they exchange data. Additionally, the connection clearly initiates, starts and terminates in our IO graphs as indicated by the traffic sent between them.