INDIAN INSTITUTE OF TECHNOLOGY

KHARAGPUR

COMPUTER COMMUNICATION LABORATORY
A REPORT ON

"Simulation of Stop-and-Wait protocol using Sockets"

November 13, 2020

Y SAI SANJEET
(16EC35025)

DEPT OF ELECTRONICS AND ELECTRICAL COMMUNICATION ENGINEERING

Contents

OBJECTIVE

The aim is to simulate the stop-and-wait flow control protocol between processes using socket programming.

DISCUSSION

- The socket server and client are realized using the same socket library in the previous experiment.
- The propagation delays across the channel are modeled by introducing a delay between receiving a data packet and sending an acknowledgment.
- Loss of data packets due to errors in the channel is modeled by sending a NACK instead of an ACK with a small probability, which is the probability of error of the channel.
- A large probability of error results in a higher number of NACKs being sent, which in turn reduces the data rate.
- The receiver, which is the server, has to be run first followed by the sender, which is the client. Else the client would terminate as the server is not reachable. The code continues to run until the sender transmits all the data.
- End of data transmission is indicated by sending a newline character `\n`.

CODE EXPLANATION

This section gives a brief overview of the code present in `sender.cpp` and `receiver.cpp`.

Sender:

- The `sender` has a vector containing the data to be sent, whose last element is `\n`.
- An instantiation of the `SocketClient` object is used to send the data.
- The sender sends data and waits for an `ACK`. The `dataIndex` variable keeps track of what data is to be sent. The `dataIndex` is incremented if an `ACK` is received.

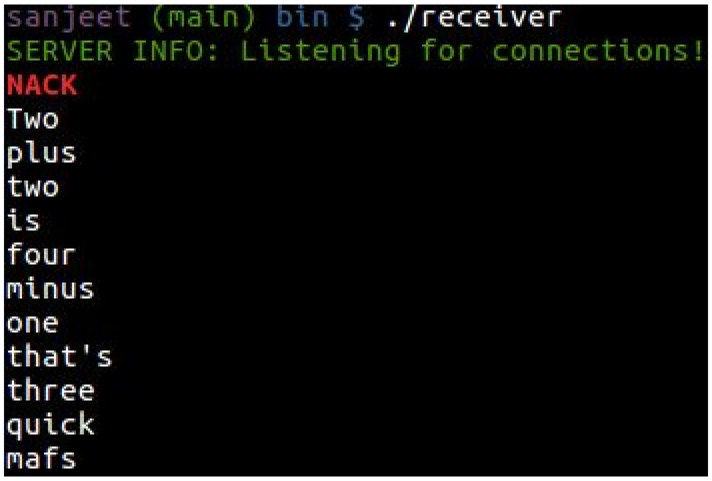
Receiver:

- An instantiation of the `SocketServer` object is used to receive the data.
- The receiver receives the data and sleeps for 500 milliseconds. After 500 milliseconds, it generates a random number and decides to send an `ACK` or a `NACK`.

- It stores the received data in a vector if an ACK is sent.
- The receiver stops receiving if it successfully receives a \n.

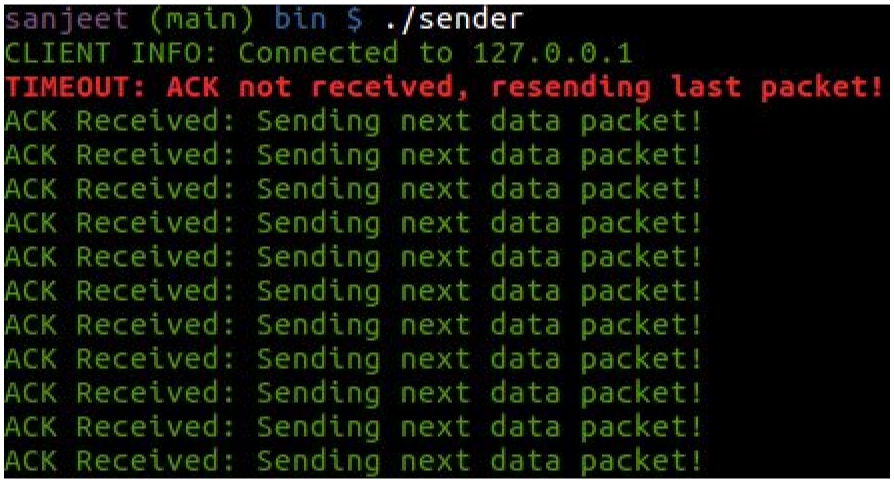
RESULTS

The sender-side:



```
sanjeet (main) bin $ ./receiver
SERVER INFO: Listening for connections!
NACK
Two
plus
two
is
four
minus
one
that's
three
quick
mafs
```

The receiver-side:



```
sanjeet (main) bin $ ./sender
CLIENT INFO: Connected to 127.0.0.1
TIMEOUT: ACK not received, resending last packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
ACK Received: Sending next data packet!
```

CONCLUSION

- The stop-and-Wait flow control protocol is implemented.
- The Socket library written in the previous experiment is used to create the socket server and client.

REFERENCES

https://www.geeksforgeeks.org/stop-and-wait-arq/