Department of
Computer Science and Engineering

University of Dhaka

Title: Implementation of TCP Flow Control and Reliable Data Transfer through Management of Timeout, Fast Retransmit, and Cumulative Acknowledgment

CSE 3111: Computer Networking Lab

Batch: 28/3rd Year 1st Semester 2024

Course Instructors

Dr. Ismat Rahman (IsR)
Mr. Jargis Ahmed (JA)
Mr. Palash Roy (PR)

1 Objective(s)

- To gather knowledge about how TCP controls the flow of data between a sender and a receiver
- To learn how TCP implements reliable data transfer and cumulative acknowledgment Implement reliable data transfer (RDT) with TCP features in Java.
- To learn how TCP uses EWMA (Exponential Weighted Moving Average) to calculate Timeout dynamically after sending packets.
- To learn how TCP uses Cumulative ACKs at the server side and Fast Retransmit at the client on three duplicate ACKs.

2 Background Theory

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. 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 add 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 and reliable data transfer mechanisms of the TCP protocol. Flow control ensures that a sender does not overwhelm a receiver with data.

TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the receive window field the amount of additional data (in bytes) that it is willing to buffer for the connection. The receiver acknowledges the highest sequence number received in order. It means if segments 1, 2, and 3 are received, an ACK for segment 3 is sent, implying that 1 and 2 were also received. This feature is known as Cumulative Acknowledgment. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host.

Normally, if three duplicate ACKs are received, the sender immediately retransmits the missing segment without waiting for a timeout.

TCP uses the Exponential Weighted Moving Average (EWMA) concept to estimate Round Trip Time (RTT) for during sending packets. It can be determined as follows:

$$EstimatedRTT = (1 - \alpha) \times EstimatedRTT + \alpha \times SampleRTT$$

$$devRTT = (1 - \beta) \times devRTT + \beta \times |sampleRTT - estimatedRTT|;$$

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$$

3 Lab Task 1: Implement TCP Flow Control (Please implement yourself and show the output to the instructor)

- Configure the server to use TCP flow control by setting the receive window size. The receive window size determines how much data the receiver is willing to accept before sending an acknowledgment.
- Configure the clients to use Cumulative Acknowledgment. Cumulative Acknowledgment means that the receiver sends an acknowledgment for the highest sequence number it has received in order, and it assumes that all packets up to that sequence number have been received.
- Test the TCP flow control by sending data from the clients to the server.

3.1 Problem analysis

The detailed algorithms of the server-side and client-side connections are given in Algorithm 1 and 2.

Algorithm 1: Algorithm of Server-Side Socket Connection for TCP Flow Control.

1: Create a ServerSocket object, namely handshaking socket, which takes the port number as input
2: Create a plain Socket, namely a communication socket object that accepts client requests
3: Set the Receive Buffer Size to simulate the Receive Window.
4: When a connection is received, create a new instance of a custom Thread class (e.g., ClientHandler) with the client Socket.
5: Call start() method on the thread to begin execution
6: Inside ClientHandler.run()– Setting up Streams, which are used for binary data transmission (e.g., chunks of files).
7: Receiving Data with Flow Control. In this step, the server uses a receive window size to limit how much data it reads per interaction.
8: Sending ACK – The server acknowledges up to the highest in-order packet (simulated cumulative ACK).
9: Close connections properly.

Algorithm 2: Algorithm of Client Side Socket Connection for TCP Flow Control

1: Create a Socket object which takes the IP address and port number as input.
2: Receive the server prompt asking for the file name.
3: Input the desired file name and divide the file into packets with sequence numbers.
4: **while** all packets are not sent. **do**
5: Send packets one-by-one (chunk by chunk) to the server.
6: After sending each chunk, wait for acknowledgment.
7: Upon acknowledgment, continue sending the next chunks.
8: **end while**
9: Close the connection.

3.2 Sample Output

The sample output of the server and client-side programming for TCP Flow control is as follows
Server Side Output

```
The server started on port 5000
Client connected: /127.0.0.1
Received 1024 bytes, Total: 1024 bytes
Received 1024 bytes, Total: 2048 bytes
...
Client disconnected.
```

Client Side Output

```
Server is connected at port no: 5000
Server is connecting
Waiting for the client
Client request is accepted at port no: 56190
Server's Communication Port: 5000
Connected to server!
Received Acknowledgment: ACK for 1024 bytes
Received Acknowledgment: ACK for 2048 bytes
...
File sent successfully!
```

4 Lab Task 2: Implement Reliable Data Transfer (Please implement yourself and show the output to the instructor)

- Start a timer after sending data packets and calculate SampleRTT values for each of the packets.
- Configure the clients to use the EWMA equation to calculate the TimeOut value. The EWMA equation is used to estimate the round-trip time (RTT) and calculate the retransmission timeout (RTO) value.
- Implement the Cumulative ACK
- Implement the fast retransmit algorithm on receiving three duplicate acknowledgments for the same packet.
- Test the reliable data transfer control mechanisms by sending data from the clients to the server

Algorithm 3: Algorithm of Server-Side Socket Connection for Reliable Data Transfer

1: Create a ServerSocket object, namely handshaking socket, which takes the port number as input
2: Create a plain Socket, namely a communication socket object that accepts client requests
3: Set the Receive Buffer Size to simulate the Receive Window.
4: When a connection is received, create a new instance of a custom Thread class (e.g., ClientHandler) with the client Socket.
5: Call start() method on the thread to begin execution
6: Inside ClientHandler.run()– Setting up Streams, which are used for binary data transmission (e.g., chunks of files).
7: **for** each received packet **do**
8: Randomly simulate packet loss with probability p
9: **if** the packet is "lost" **then**
10: drop it (do not send ACK).
11: **else if** the packet is received normally **then**
12: Check if the packet sequence number is expected.
13: Update expected sequence number.
14: Sending ACK – The server acknowledges up to the highest in-order packet (simulated cumulative ACK).
15: **end if**
16: **end for**
17: Close connections properly.

Algorithm 4: Algorithm of Client Side Socket Connection for Reliable Data Transfer

Create a Socket object which takes the IP address and port number as input.
Receive the server prompt asking for the file name.
Input the desired file name and divide the file into packets with sequence numbers.
while all packets are acknowledged. **do**
    Send end packets one-by-one to the server.
    Records the send time (Start a timer) for the SampleRTT measurement.
    Save sent packets in a buffer for potential retransmission.
    Implements Exponential Weighted Moving Average (EWMA) for timeout.
    On receiving ACK:
    if ACK matches the expected one **then**
        Update timers and sequence progress.
        Measure SampleRTT and update timeout.
    else if duplicate ACKs received 3 times **then**
        Fast Retransmit immediately.
    end if
    if timeout happens: **then**
        Retransmit the next expected packet.
    end if
end while
Close the connection and exit.

4.1 Sample Output

The sample output of the server and client-side programming for TCP Reliable Data Transfer is as follows
Server Side Output

```
Received Packet 1. Sending ACK 1
Received Packet 2. Sending ACK 2
Received Packet 3. Sending ACK 3
— Packet 4 not received —
Received Packet 5. Sending Duplicate ACK 3
Received Packet 6. Sending Duplicate ACK 3
Received Packet 7. Sending Duplicate ACK 3
Received Packet 4 (retransmitted). Sending ACK 4
Received Packet 8. Sending ACK 5
Received Packet 9. Sending ACK 6
Received Packet 10. Sending ACK 7
```

Client Side Output

```
Server is connected at port no: 5000
Server is connecting
Waiting for the client
Client request is accepted at port no: 56190
Server's Communication Port: 5000
Connected to server!
Sending Packet 1 with Seq# 1
ACK 1 received. RTT = 105ms. EstimatedRTT = 105ms, DevRTT = 0ms, Timeout = 105ms
Sending Packet 2 with Seq# 2
ACK 2 received. RTT = 98ms. EstimatedRTT = 104ms, DevRTT = 1.75ms, Timeout = 111ms
Sending Packet 3 with Seq# 3
ACK 3 received. RTT = 100ms. EstimatedRTT = 103.5ms, DevRTT = 1.31ms, Timeout = 108.75ms
Sending Packet 4 with Seq# 4
— Packet 4 lost during transmission —
Sending Packet 5 with Seq 5
Received Duplicate ACK for Seq 3
Received Duplicate ACK for Seq 3
Received Duplicate ACK for Seq 3
Fast Retransmit Triggered for Packet 4
Resending Packet 4
ACK 4 received. RTT = 102ms. EstimatedRTT = 103.3ms, Timeout = 108.1ms
Sending Packet 6 with Seq# 6
ACK 5 received. RTT = 97ms. EstimatedRTT = 102.6ms, Timeout = 107.5ms
Sending Packet 7 with Seq# 7
ACK 6 received. RTT = 101ms. EstimatedRTT = 102.2ms, Timeout = 106.6ms
Sending Packet 8 with Seq# 8
ACK 7 received. RTT = 99ms. EstimatedRTT = 101.8ms, Timeout = 106.2ms
Sending Packet 9 with Seq# 9
ACK 8 received. RTT = 103ms. EstimatedRTT = 102ms, Timeout = 106.5ms
Sending Packet 10 with Seq# 10
ACK 9 received. RTT = 100ms. EstimatedRTT = 101.75ms, Timeout = 106ms
ACK 10 received. All packets delivered successfully!
```

5 Policy

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