Computer Networks Course Project

Reliable Transport Protocol

The problem of reliable data transfer is of central importance in networking. A reliable data transfer protocol provides upper layer entities with the service abstraction of a reliable channel through which data can be transferred. The task is made difficult by the fact that the channel is unreliable, and it is the task of the reliable data transfer protocol to deal with unreliability problems: packet loss, packet corruption and out-of-order packets. We have studied two reliable data transfer protocols: Go-Back-N and Selective Repeat. In this project, our goal is to implement the Go-Back-N protocol.

In a Go-Back-N (GBN) protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline. *N* is referred to as the window size.

You should watch **lecture 7 and Socket Programming LAB and** may read **section 3.4.3** in the textbook for more details about the GBN protocol

Project description

In this project, we will implement a transport protocol that provides some reliability services on top of the unreliable UDP. This will be done by augmenting UDP with the GBN protocol. You are required to implement a special GBN sender and receiver whose details are specified in the following subsections.

Sender

The sender script should be called with three arguments: *filename, receiver IP address*, *receiver port*. Other variables like the *maximum segment size* (*MSS*), *window size N*, and *time-out interval* should be statically defined in your script.

The *filename* argument is the name of a file residing at the sender side, and it is required to send the file to the receiver. Other arguments are self-explanatory.

When the sender script is called, it proceeds with the following steps:

1. Read the file and divide it into K chunks of size less than or equal to the *MSS*. The structure of the sender packet is shown in figure 1. The packet ID is a unique identifier for each packet starting from 0. The file ID is a unique identifier for each file being sent. The value of the trailer bits should be set to the value of 0x0000 by the sender for all file chunks except the last one. The packet of the last file chunk should have a trailer value of 0xFFFF. This is done by the sender to indicate the end of the file being sent, so that the receiver knows when to stop waiting for more bytes of the same file.

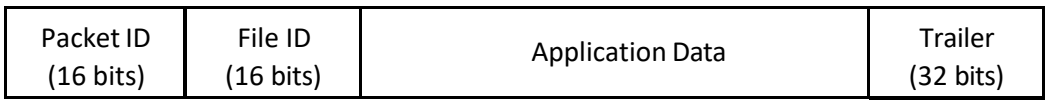

Figure 1: Sender packet structure

2. The structure of the acknowledgement message is shown in figure 2.

Figure 2: Acknowledgment structure

3. Transmit the first *N* packets to the receiver over UDP where *N* is the window size, then wait for acknowledgement receipts.
4. The sender will then respond to the following types of events:
    a. *Receipt of an ACK*: If the received ACK number is more than the window base, the sender slides its window forward. The sender then sends packets inside the window which have not been sent yet.

For example, suppose that the window size is 4 and packets 0, 1, 2, 3 have been transmitted and the received ACK is 2. Go-Back-N uses cumulative acknowledgements, so the received ACK number implies that all packets up to packet 2 have been correctly received. In this case, the sender slides its window forward such that it includes packets 3, 4, 5, 6 and transmits packets 4, 5, and 6.

b. *Timeout event*: If a timeout occurs, the sender resends all the packets that were transmitted but not yet acknowledged.

The sender terminates on receiving an acknowledgement for the last packet in the file.

Receiver

On running the receiver program, it deals with only one event: *Reception of a packet*. The receiver parses the received packet separating the header and trailer information from the application data and indicates via the user interface that a file is being received. If the packet ID is the expected packet to be received, the application data is stored. Otherwise, the data is discarded. Afterwards, the receiver sends an acknowledgment packet to the sender with the ID of the last correctly received packet. When the last packet has been received, the receiver then acknowledges the packet, writes the data to a new file, and indicates via the user interface that the file reception is complete.

Simulated loss

As the network is small in our experiments, there is almost no packet loss. So, to test the implemented protocol, packet loss will need to be simulated by randomly dropping some packets at the receiver and assuming that they were lost. Thus, for each received packet at the receiver, generate a random number which decides whether the packet is lost or not. The simulated loss rate should be in the range from 5% to 15%. Note that we will assume that there are no errors introduced by the channel to the received packets, so there is no checksum verification needed.

Deliverables and deadlines

Initial demo

In the initial demo, you should have a minimal working version of the project. The sender and receiver should be capable of exchanging data; however, the received file should not be necessarily correct. Reliability issues which ultimately result in corruption of the received file will be tolerated in this phase, but you should be able to discuss the perceived issues, their causes, and your action plan to fix them.

Demo day: Week 13

Final demo

Before the final demo, you should submit the code of the project and a report of the results. The report should include the following:

1. Screenshots of the captured packet listing window showing the packets exchanged at the sender and the receiver. The screenshots should verify that the test files are received correctly and their transfer time. In the packet capture screenshots, the first and last packet of each file should be highlighted with their time stamps.
2. For each file, include the plot of the received packet ID versus time. Use this figure to verify that retransmissions have indeed happened. Hint: plot a dot for each sent packet ID (y axis) versus time (x axis). Mark the retransmitted packets with a different color and display the number of retransmissions and the test parameters (window size, timeout interval, loss rate, etc.) as part of each figure.
3. Update the user interface of the sender and the receiver to display the file transfer information after each file transferred. This should include the transfer start time, end time, elapsed time, number of packets, number of bytes, number of retransmissions (sender), average transfer rate (bytes/sec and packets/sec), etc. Keep a log of screenshots throughout your test iterations.
4. Repeat your tests with three different combinations of *window size N* and *timeout interval.* Comment on your results.

5. Discuss how the protocol of this project can be modified for a better performance. The improvement of this protocol should be inspired by one or more features of the TCP protocol standard. Explain the improvement, reference its relevant sections, and page numbers in the TCP standard. **(BONUS)**
6. Modify the sender in a way that exploits any weakness in the protocol specifications to maliciously disrupt the receiver. Describe how the proposed attack method works and your suggested mitigation methods on the receiver side. Also, suggest a list (more than 2) of possible protocol updates for a relatively more secure version of this protocol. Further, identify the regulations and laws against performing such threat actions (local and international). Your legal discussion should get into the legal details and the specified penalties. Also, discuss the possible economic and societal impact of freely spreading tools that can disrupt network communication. **(BONUS)**

You will find the test files attached.

Submission deadline: 15 May 2024

Late submission policy: minus 20% per day (max 2 days).

Plagiarism Policy: Any student found plagiarizing others' work or helping others plagiarize his/her work, intentionally or unintentionally, will have the related class work credit removed.