Computer Networks 159.334 Assignment 2 due 27 Sep 2013

A Reliable Transport Protocol (15 marks)

Introduction

Your task is to write a simple **reliable transport protocol** using sockets and UDP. You don't need to create your solution from scratch, all you need to do is to modify the two sources provided with this assignment.

Two source codes are available: *server_Unreliable_2013.c* and *client_Unreliable_2013.c*. Two auxiliary programs are also available, *UDP_supporting_functions_2013.c* and *CRC_simple.c*.

The client will try to transfer a single text file using a (simulated) unreliable channel. Packets can get lost and/or damaged randomly.

The client: - reads "**file1.txt**"
- builds a packet, adding a header with a sequence number (e.g., "PACKET #")
- sends the line via UDP

The server: - reads the stream via UDP
- processes the packet, extracting the line
- finds the sequence number by separating it from the header (i.e., filters "PACKET #")
- saves the lines on a file called "**file1_saved**.txt"

If no packets are damaged or lost, than file1_saved.txt is identical to file1.txt. However, as we want to simulate a unreliable channel, we are going to force packets to be transmitted with damaged bytes or to be discarded. In this case, the file received by the server will be different than the original one.

General Guidelines

You need to decide about some options you have to develop this protocol. For example:

- What do you include in the header? (e.g., sequence number, CRC value etc)
- Use Ack or Nack or both? Remember that these could get lost or damaged too!
- How to establish a timeout? This can be fixed for all packets within the sliding window, e.g., use `sleep()`
- Are acknowledgments issued for every packet or are they cumulative?

Reliable protocols need to use some form of error check. For this assignment you should use a simple CRC implementation, available inside CRC_simple.c.

In UDP there is no need to connect or accept connections. A socket is created for UDP:

```
s = socket(PF_INET, SOCK_DGRAM, 0);
```

And bind is used to tell UDP what is the local port:

```
bind(s, (struct sockaddr *)&localaddr, sizeof(localaddr));
```

Using the destination IP address and port number, one can send and receive messages by using:

```
int sendto(int socket, char *buffer, int length, int flags, struct sockaddr *destination_address, int address_size);
```

In order to simulate the unreliable channel, we wrapped the `sendto()` function with a customised

`send_unreliably()` function. The receive function `recvfrom()` can be used as:

```
int recvfrom(int socket, char *buffer, int length, int flags, struct sockaddr *sender_address, int *address_size);
```

Running the code

In order to run the prototypes you need to download the correspondent files from the 159334 website. You **are not allowed** to change any of the *UDP_supporting_functions_2013.c* functions.

Initially, try to run the prototypes without any losses. The randomization can be done separately for the client and the server. Start the server with:

```
server_Unreliable_2013 1235 0 0
```

Then the client with:

```
client_Unreliable_2013 127.0.0.1 1235 0 0
```

You should see messages from both programs. The messages coming or going are represented by arrows. For example:

```
<-- SEND PACKET 9 line 10 jjjjjjjjjjjjjjjjjjj
RECEIVED --> ACKNOW 9
```

The client will proceed reading and sending N lines of text. The server should be able to read those messages, split the message (filter the header off) and save the file. Compare the two *.txt files. Are they identical?

Now try to introduce some random problems in the channel by changing the options. For example, introduce losses in the client's side:

```
client_Unreliable_2013 127.0.0.1 1235 1 0
```

After running it again, are the two *.txt files identical? What are the differences?

Submitting the assignment

You need to submit the **server** and the **client** separately.

You should **not** submit the file *UDP_supporting_functions_2013.c* with your server and client because the assignment is going to be marked using the original one. Make sure to include the file (using `#include` statement).

You are allowed to work in groups, maximum of 3 students. In this case, please submit only once with the *names and number of all students in the header* of both *.c files.

Marking the assignment

The assignment is going to be marked based on functionality (i.e., checking the contents of file1_saved.txt). The marks are distributed as follows:
- 3 marks: correct use of the CRC function.
- 3 marks: correct use of acknowledgments.
- 3 marks: correct re-transmissions of packets that are damaged or lost.
- 3 marks: correct treatment of packet duplications.
- 3 marks: correct implementation of sliding windows (e.g., pipeline with multiple packets, avoiding out-of-order packets, a Go-back-N approach suffices)

Notes

1 - Submit your files electronically via Stream.
2 - This assignment is worth **15 marks**.
3 - Marks will be subtracted for obvious copying and/or for delays without justification.