

CS 164 Project Definition

Winter 2007

Submit by Friday March 9, 11:10am

Demo date: Friday, 9 March, 11:10am *

Please read this page and the class mailing list periodically for any updates

The objective of a transport layer protocol is to provide service over the unreliable channel. In the Internet Protocol (IP), the network layer provides best effort, yet no guaranteed reliability of service. In your project, you are asked to implement transport layer functions, using an application network programmers interface (the sockets) and we will use UDP sockets this time, as opposed to the TCP sockets we mainly implemented in the labs.

This project involves the implementation of a derivative of Selective Repeat, one of the simplistic approaches for transport layer protocol. For ease of comprehension of the programming here, think of SR as running on top of UDP.

Recall that UDP does not provide a reliable service, i.e. it does not guarantee packet delivery. However, it does not often lose packets in the local network. To test and prove the functionality of SR (the transport scheme you will implement) in this project we wish UDP to drop packets. Hence, this project requires you to emulate packet losses and have the SR protocol, running on top of UDP, help in recovery of lost packets.

About Selective Repeat:

The principal tasks of a transport layer protocol are:

- Application multiplexing and demultiplexing
- Reliable data transfer
- Flow control
- Congestion control

Reliable data transfer should take care of the following situations:

- Packet loss
- Packet corruption
- Packet duplication

One of the approaches for building reliable data transfer is the Selective Repeat protocol. Here the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgement, but is constrained to have no more than some maximum allowable number, N , of unacknowledged packets at any given time.

Please read a detailed description of SR from the resource provided [1] - or any other resource - for a better understanding of the protocol.

*Making a demo of your program is mandatory. No delays to the demo date are available. The students who would like to demo earlier than March 12th should arrange an appointment with the TA in advance

For your project you have to write a code to create a sender and a receiver, which communicate using SR. Sender will communicate window size N to the receiver. We will be emulating half duplex communication, i.e. data is sent only from sender to receiver. (As you did in the labs as well.) The communicating entities will exchange character strings in a specific format. Sender will read a specified file and construct packets of fixed length.

The format of a packet will be:

Field Name	Exact Length (in characters)	Comments
Packet Number	2	Interpreted in decimal, should be something among "00", "01", "02", "99".
Control number (CN)	1	0 → Not a control packet 1 → Window length declaration packet 2 → Acknowledgement packet 3 → Connection close packet Valid CN's for a packet from sender are 0, 1, and 3. For receiver, only 2.
Control data (CD)	2	Interpreted in decimal, should be something among 00, 01, 02, 99. (CN == 0) → The packet contains valid payload. CD has no significance (CN == 1) → CD is the value of window, N , proposed by sender. (CN == 2) → CD is the packet number that is acknowledged by the receiver. (CN == 3) → Sender wants to close connection implying data transmission is complete.
Data or Payload	20	Content characters from the file-to-send file. The last packet may contain less than actual 20 characters; the unused parts will be filled up by the '#' character. (For simplicity make sure that the file to be sent does not contain any '#' character).

The sender (which is the client) should be started as
`<sr_sender> <rcvr-hostname> <rcvr-port#> <file-to-send> <N>`

where

- `rcvr-host` and `rcvr-port#` are the hostname and port number of the SR receiver,
- `file-to-send` is the name of the ASCII file whose contents will be sent,
- `N` is the size of window to be used

Sender starts by sending a packet that contains the proposed window size and waiting for receiver to acknowledge. If the proposed window size is permissible (in this case, say, $0 < N < 31$) then receiver acknowledges that packet. Otherwise receiver does not acknowledge and the sender dies after timeout.

Example of senders first packet: 00120##### [sender is proposing a window size of 20].

Example of receivers first packet: ##200##### [receiver acknowledges packet number 00 i.e. accepts window size of 20].

Sender will start from packet number "00" and it can go up to maximum of "99". Then packet numbers will wrap. Sender can have maximum of N sent-but-not-acknowledged packets at a time. A packet from receiver with ack number "n" means receiver has received packet with number "n". Let "snd_base" be the minimum packet number that is not yet acknowledged. Receiver individually acknowledges all correctly received packets. Buffers packets, as needed, for eventual in-order delivery to upper layer. Sender only resends packets for which ack not received. For each unACKed packet it keeps a separate timer.

When the full file has been sent, sender sends a packet indicating end of transmission. E.g. 963##### [this itself is packet number 96]. However sender should exit only after getting acknowledgement of this packet.

There is another part in the receiver which is not a part of the standard SR. We are doing this to facilitate testing. You will have two versions of the scheme such that each version is capable of performing the functionalities described in 1 and 2 below:

1) *Instance 1:* Assume that the receiver part consists of two independent blocks; one block receives the data from UDP and the other block sends the ack packets. The part that accepts packets will probabilistically discard packets. On receipt of each packet it will generate a random number between 0 and 1. Let the probability of discard be 0.1. If the random number is less than 0.1 then the packet will be discarded otherwise it will be passed on to the next block.

This receiver, which is the server in program instance 1, should be started as

<sr_receiver> rcvr-port-no <flag> where

- *rcvr-port-no* is the port number to listen to and
- *flag* is the instance that indicates which type of packet dropping is implemented.

2) *Instance 2:* The part that accepts packets wont probabilistically discards packets, but it will be able to receive an input from keyboard, which will correspond to a packet drop. In order to do that you should probably need to implement threading in your design. The receiver will work normally and once there is an input from the user it will drop the next packet it receives from the sender.

This receiver (server in program instance 2) should be started as

<sr_receiver> rcvr-port-no <flag>

where

- *rcvr-port-no* is the port number to listen to and
- *flag* is the instance that indicates which type of packet dropping is implemented.

Output:

The sender and receiver both maintains three log files:

- Sent.log
- Received.log

The format of these two files is

<TIMESTAMP> <The whole string received or sent>
e.g.: Tue Apr 23 2002 14:11:10 PDT 130##abcdefghijklm

- Discard.log

The format of this file is

<TIMESTAMP> <The whole string received or sent> <Reason>

For sender, the values that the Reason attribute can have are: ACK_OUT_OF_WINDOW

For receiver, values that the Reason attribute can have are: PKT_NOT_EXPECTED, PROBABILITY_DISCARD

Receiver maintains another file, FileReceived that contains the contents of the file received so far.

Deliverables:

You must submit the following:

All source code. There should be NO object files, binaries or core dumps as part of the submission! Do not tar/gzip the solution either. Your source code should compile without ANY warnings. (Use the -Wall option.) You will lose one point per warning. You should write the program in C. (Please contact the TAs if you are interested in implementing the project in another programming language.)

Documentation. There should be a file called README. It should include a general outline of how your program works, and any special comments or notes that you need to share. Please keep it straight forward and easy to read. The README file should be straight ASCII text.

The grading ¹ for this project is as follows:

- 70% for program functionality/correctness
- 20% for documentation
- 10% for programming style

Following you can find a compact description of the receiver and the sender.

Sender	Receiver
Data from above: <ul style="list-style-type: none"> • If next available seq # in window, send packet Timeout(n): <ul style="list-style-type: none"> • Resend packet n, restart timer n ACK(n) in [snd_base,snd_base+N-1]: <ul style="list-style-type: none"> • Mark packet n as received • If n smallest unACKed packet, advance window base to next unACKed seq # 	Receive packet n in [rcv_base,rcv_base+N-1]: <ul style="list-style-type: none"> • Send ACK(n) • Out-of-order: buffer • In-order: deliver, advance rcv_base to next not-yet-received packet, deliver all buffered, in-order packets Packet n in [rcv_base-N,rcv_base-1]: <ul style="list-style-type: none"> • ACK(n) Otherwise: <ul style="list-style-type: none"> • Ignore

References

[1] James F. Kurose, Keith W. Ross, "Computer Networking: A Top - Down Approach Featuring the Internet", 3rd edition.

¹Reminding that the project is strictly personal, and same codes in two or more different projects will be graded accordingly.