CN HW2 report

B00902031 Kevin Tsai, B00902107 Suhorng You

December 5, 2012

1 Introduction

In this homework, we have implemented a modification of Go-Back-N protocol with partially realized connection management. The sender transmits 8 packets without waiting for acknowledgements of each packet. To simplify the design, in each connection, data to be sent is given to the sender as a stream, and `rdt_send` is automatically invoked by the sender when the window is empty to refill the window by data from the stream. Similar changes is made to the receiver. When called, the receiver blocks and wait until a transmission came, and it automatically allocates memory to store data, returns when all data has arrived.

We also add connection termination control to both sender and receiver to handle when to end a transmission. The connection termination phase is drawn from that of the TCP's, except that we use a three-way handshake since the connection is unidirectional.

"make" compiles two binary files: "sender" and "receiver." Start receiver at the target computer, and then execute sender to send the file. The file sent will be saved at the working directory of sender, which is normally the directory sender is in. Filename and file permissions are preserved during the transmission. If there already exists a file with the same name, the transmission will be terminated. Sender closes after the transmission, but the receiver continues to wait for the next file until it is aborted (maybe by `^C`).

Command line arguments is as follows:

```
$ make                                                          build targets
$ make clean                                          clean targets and object files
$ ./receiver LISTEN_PORT                    receive files, waiting at port LISTEN_PORT
$ ./sender TARGET_HOSTNAME TARGET_IP FILENAME                          send file
```

Transferring multiple files simultaneously is not supported, however.

2 Unreliable Data Transfer Layer

This layer simply uses UDP to transfer data. This layer provides functions to open UDP sockets, to send packages to and receives packages from the socket, and to close the socket. The receive and send functions are both blocking, but the receive function only waits for a limited time which can be set by the function caller and is 0.5 seconds by default. This layer also provides an additional function that clears the socket buffer in case that there are any redundant packages left in the socket.

3 Reliable Data Transfer Layer

3.1 Packet Format

The size of packets varies from 9 bytes to 256 bytes. The first byte of the packet specifies the size of the following data, that is, 8 bytes plus the length of the content. It is followed by a 4-byte unsigned integer sequence number, and a 4-byte unsigned int CRC-32 checksum. The rest are the actual data that we want to transfer, whose length could be from 1 byte to 247 bytes.

Entry Name	Size (bytes)	Interpretation
packet size	1	size of the packet (exclusive of this entry)
sequence number	4	sequence number of the packet
checksum	4	the CRC-32 checksum of the whole package, with this entry treated as zero
data	1-247	the data being transferred

The constant used for CRC-32 checksum is `0xEDB88320`.

3.2 Receiver and Sender FSM

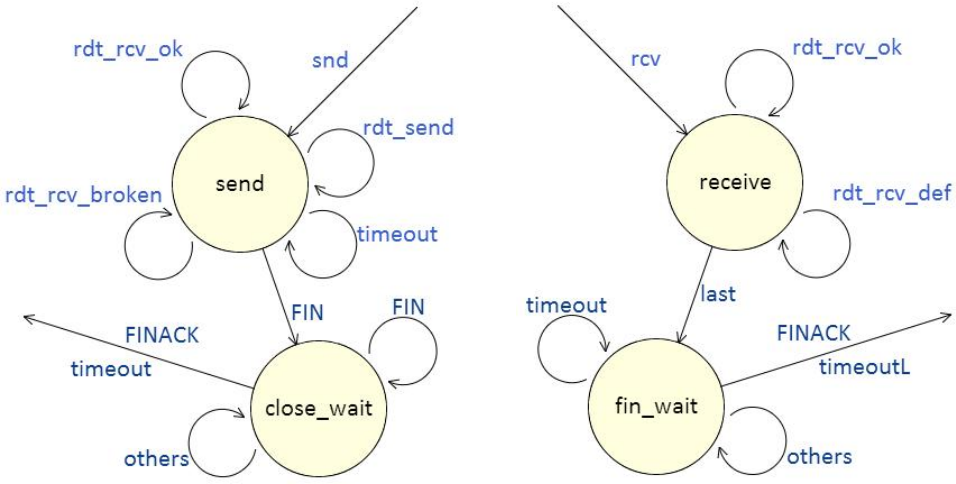


Fig.1 The state diagram of the sender and receiver; Go-Back-N with extended FSM

Figure 1 is the diagram of the sender and the receiver, respectively. The details are described below.

Roughly speaking, when calling the sender, it first clears the buffer, sends data, and wait. If timeout, the packets in the window are resent. If received ACK, the sender slides the window to the right until `base` touches received sequence number, i.e. only those packets that haven't been ACKed are left. Both corrupted packets and out-of-range ACKs are simply ignored. If FIN, which is the same as the ACK of the last packet, is received, the sender enters termination phase to close the connection.

When calling the receiver, it waits for data to come, and accepts only the packet with sequence number equals `expseq`. If the received packet is not accepted, it simply throws it away and resend previous ACK (or ACK-INIT, is no packet ever accepted). When regarded data in the packets as a stream, the first 4 bytes specify the total length, and hence the receiver can judge when should the connection be closed. Upon the last packet is accepted, the receiver sends FIN to the sender to require the termination of the connection.

The three-way termination phase is initiated by the receiver. The receiver first send FIN to the sender and waits for FINACK. If the sender sends too many incorrect replies or the timer time-out, the receiver will resend FIN. As soon as the receiver received FINACK, it sends the last FINACK to sender and returns to the caller, regardless the FINACK is correctly sent to the sender or not. However, after 15 seconds of wait, the receiver still exits no matter FINACK has ever been received or not. If the sender receives FINACK, the connection is the closed. If FINACK is lost, the connection still terminates after timeout (30s).

```
snd            rcv
|    <---FIN---  |    FIN == last packet's ACK
| \              |
|  --FINACK-->  -|
|                /
|    --FINACK----
|   /
| <-
```

3.2.1 Sender

1. `snd`: initial data stream, `base`, `nxtseq` (the window); empty socket buffer; invoke `rdt_send`
2. `rdt_send`: fill window by data until full; send packets
3. `rdt_rcv_ok`: slide the window if `seq` is correct, otherwise do nothing
4. `rdt_rcv_broken`: pretending that nothing happens
5. `timeout`: resend packets in the window

6. `send::FIN`: send FINACK
7. `others`: ignore
8. `close_wait::FIN`: send FINACK
9. `FINACK`, `close_wait::timeout`:

3.2.2 Receiver

1. `rcv`: empty socket buffer; initial default reply (ACK-INIT)
2. `rdt_rcv_ok`: make new ACK reply; send reply, increase `expseq` by 1. If this is the very first packet the receiver ever received, read 4 bytes stream length and initial the data stream. If this is the last packet to be received, send FIN instead and go to `last`.
3. `rdt_rcv_def`: send previously made reply
4. `last`: (should send FIN)
5. `timeout`: resend FIN
6. `others`: ignore; but if too many unexpected packet arrived, resend FIN.
7. `FINACK`, `timeoutL`: for `FINACK`, send FINACK and exit. for `timeoutL`, exit.

3.3 Protocol

Besides the packet format, our modified Go-Back-N has its own protocol. Firstly, for every session, the first 4 bytes of the first packet (so that it should be at least 4 bytes long) specifies the length of data to be sent in this session (not including itself). Secondly, the reply, either ACK, ACK-INIT, FIN, FINACK, should be sent by a packet whose content is exactly 1 byte long, i.e. it should only contains the flag and nothing else. For ACK-INIT the sequence number should be -1, for FIN the sequence number should be that of the last accepted packet's, and for FINACK the sequence number should be that of the FIN's plus one.

Since data packets and reply packets are undistinguishable, that the receiver sends FIN request first is necessarily. It is possible that the FIN sent by the receiver will lost, so that the receiver may receive data packet from the sender due to timeout. The receiver identifies the FINACK reply by checking the sequence number.

4 Loss and error solving

4.1 Packet loss solving

Packet loss, though technically impossible to be perfectly detected, could be partially bypassed through an ACK-based protocol, such as the Go-Back-N protocol. After sending packets, we wait for a predefined time (500ms for example), and if no acknowledgement from the receiver is received, we seek for resending, assuming the packets are lost.

However, the receiver side doesn't have such measures. When an ACK it lost, it just disappeared forever. But when the sender side timeout, it will resend packets, and the ACK will also be resent. Such protocols could still go wrong, but at least it is adequate for most uses.

4.2 Packet error solving

There is a checksum in every packet sent. When received, the program checks if the checksum in the package is correct. Corrupted packets are ignored. The use of CRC-32 which is based on cyclic error-checking codes guarantees low error rate.

5 Application Layer

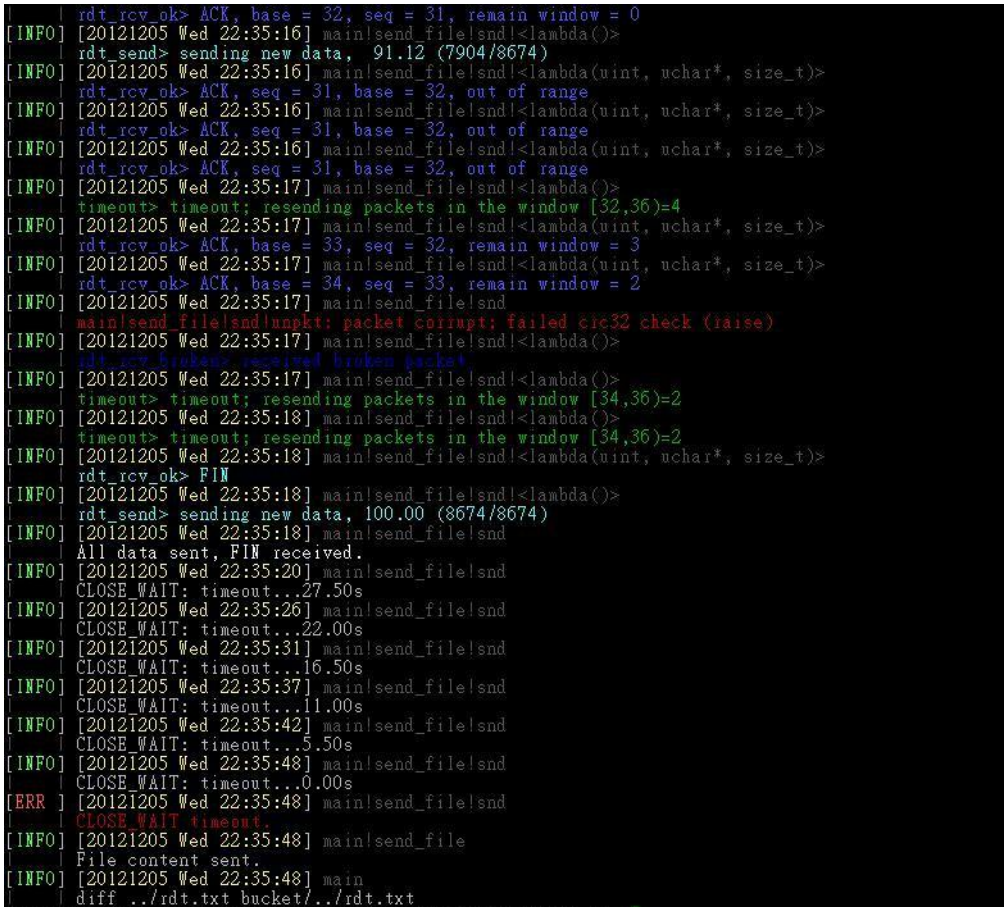
Sender and receiver calls the RDT (reliable data transfer) layer multiple times to send a file. More precisely, the sender sends four things in order: the file's name, the file's permission, the file's size, and at last the file's content. The first three items are sent in one call, and the last one may be sent in many calls. The receiver receives the four things in order, and creates the file after it gets the file's size.

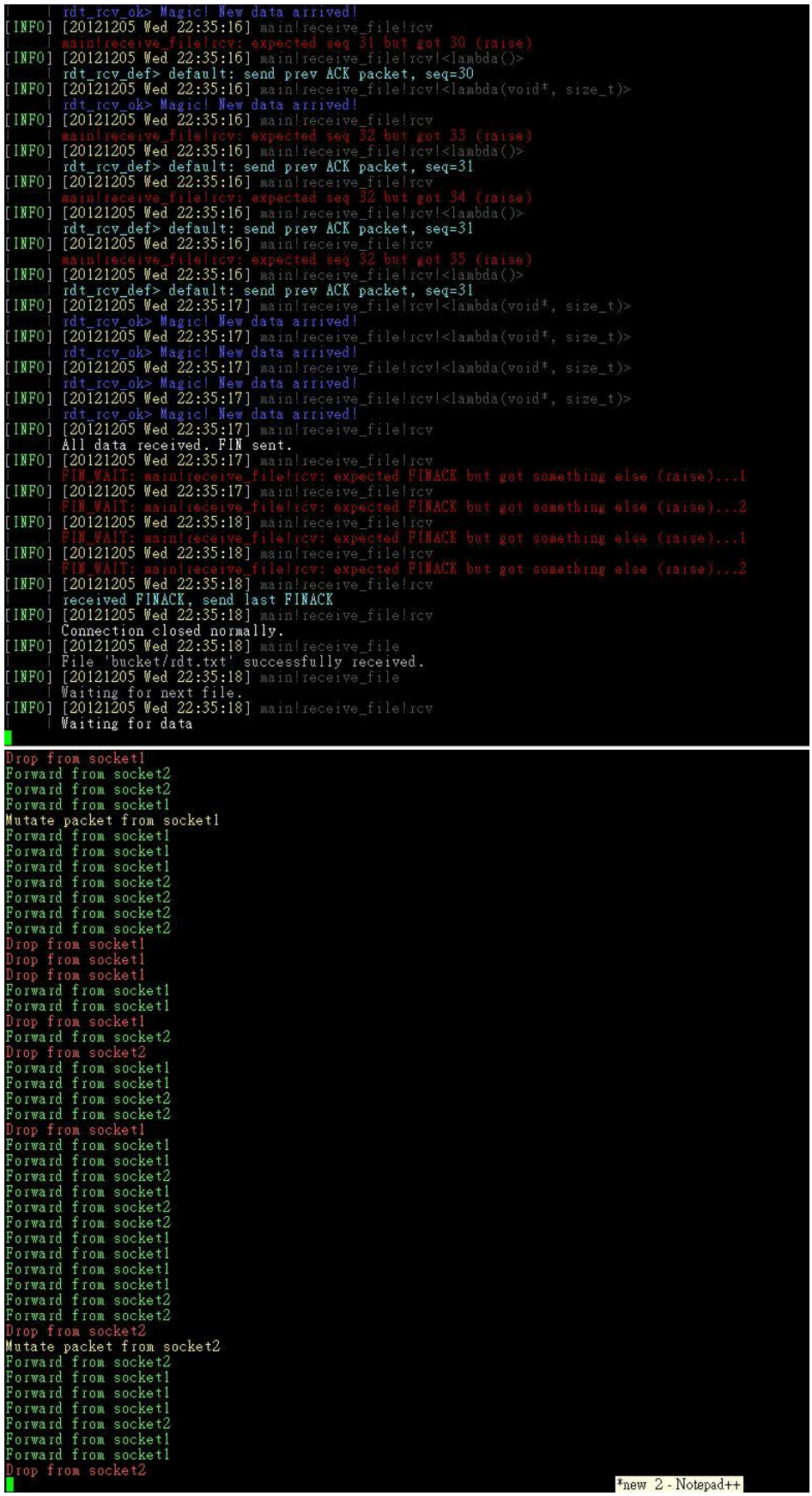
6 Extra work

Our reliable data transfer functions enable us to freely send data without worrying data loss or corruption. The connection termination control realized makes it possible to send data more then once, and the sending process is much stabler. Connection termination is implemented by sending FIN packets using three-way handshake, see section "Reliable Data Transfer Layer" for details. Though neither full connection management nor simultaneously transferring data is supported, the design has made it easier to use.

Besides the additional connection termination control, when transferring files, the name and permissions are preserved under transmission. Also, we have implemented a colourful logging functions which provides (handmade) stack trace to make debugging more tolerable. See snapshots.

7 Snapshots





The sender, receiver, and UDPProxy