Design Document for Assignment 1

Distributed System
600.337/437

Sihao Lu, Yixiao Wu
Slu21@jhu.edu, ywu66@jhu.edu

Second Draft

We use sliding window and selective repeat techniques. In the previous draft attached at the end, we used fixed window – the next block of data doesn't start transmitting until current block is completed, which is inefficient and may stutter. The improved strategy slides the window as data transfer goes, and send negative acknowledgements along with the index till which all the packets are received.

- ncp:
  1. Read file from source file path, prompt error if path is illegal.
  2. Determine an appropriate transferrable packet size.
  3. Specify a custom time-window length, e.g. 128. A time-window represents the synchronization group of numerous packets.
  4. Packet construction: every packet uses its first 8 bits as its type, one following bit as the completion flag (which will be 0 at all times unless it's the last packet), the following 32 bits as its index and the rest of packet space as file content. As every packet is ~1KB, this index system can support file size up to 2^32 * 1K bytes = 4TB. Thus, our packet structure is going to look like this.

     ```
     typedef packet_type char;
     packet_type packet_type_normal = 0;
     packet_type packet_type_metadata = 1;
     …
     typedef struct Packet {
       packet_type type;
       bool completed;
       unsigned int index;
       unsigned char data[0]; // It's actually a variable length array because we can allocated more memory than sizeof(Packet) and fill the data into the extra space.
     };
     ```
  5. Before sending the file content, it sends a metadata type packet including the destination file name. It keeps resending it until it receives the feedback that the recipient has acknowledged the information.
  6. The ncp maintains three records:
     - ARU: The packet index until which all the packet has been successfully sent and received by the recipient side.
     - nacks: An queue of negative acknowledgement packet indices – the indices of lost packets.
     - current_sending_index: The index of current sending packet.

The ncp starts with ARU = 0 and nacks equals to empty array. Then it starts transmitting packets of normal type constructed by the rule above using modified sendto function sendto_dbg. Here's the detailed description of its strategy to send packets:

1. If nacks is empty, send the packet at index current_sending_index and increment this value.
2. If nacks is not empty, send the packets in FIFO order from nacks queue and set it empty. Go to rule 1.
3. In some rare cases, the sender sends too fast and current_sending_index – ARU > time window length. In this case, the sender stops to wait for the receiver to respond with new ARU and nacks.

Upon receiving feedbacks from rcv, it refreshes these two information by changing to the latest ARU and appending new nacks from the feedback.

7. The ncp counts the time as it sends packets. Every time ncp sends a distinct packet, it adds its size to a counter. If the size counter exceeds 50MB, it reports the total amount of data successfully sent and calculates the transfer rate.

8. rcv:

1. Listens to the host specified by input.
2. The feedback is formatted below:

```
typedef feedback_type char;
feedback_type feedback_type_normal = 0;
feedback_type feedback_type_busy = 1;
feedback_type feedback_type_metadata_received = 2;
…
typedef struct Feedback {
  feedback_type type;
  unsigned int ARU;
  int nacks_length;
  int nacks[0]; // A variable length array
};
```

3. When receiving packets,
    a. When receiving the first packet, it records the host address and receives subsequent packets only from that host. If another host makes a request, rcv sends a feedback of type "busy", indicating it is busy.
    b. When receiving the metadata packet, it returns a "metadata received" type feedback and extracts the destination file name.
    c. The rcv maintains a buffer of received packets implemented with a circular array with size equal to window size. Whenever it receives a packet, it puts it to the corresponding buffer address according to its index (using mod to calculate its index in the array rapidly). Whenever the first n consecutive packets are all received, it writes the data in these packets to the file and moves the head of the circular array forward.
    d. The rcv maintains these records:

- ARU: The packet index until which all the packet has been successfully sent and received by the rcv.
- nacks: It scans through the buffer from the head to the latest_index and find packet indices that are not received and construct an array of indices, naming it nacks.
- latest_index: The largest index of the packet received. It is used to mark the end of the actual buffer.

Every time it receives a packet, after adding the packet to buffer and doing the misc. procedures as depicted in step c, it sends back a normal type feedback with ARU and nacks defined above. It also keeps tracking of the completion flag. When it receives the last chunk of file and nacks are empty, the rcv flushes its last piece of data to file and completes the transfer.

e. Whenever it receives 50MB of data, it spits out statistics about the total data received and the average transfer rate.

9. t_ncp and t_rcv:

1. Because TCP doesn't experience packet loss, the t_ncp simply send all the packets in order and t_rcv receives it and append the data to file. t_ncp can send special packet to indicate EOF.

First Draft (NOTE: this is an obsolete draft)

We are going to implement selective repeat ARQ strategy in our ncp and rcv.

10. ncp:

1. Read file from source file path, prompt error if path is illegal.
2. Determine maximum transferrable packet size.
3. Specify a custom time-window length, e.g. 128. A time-window represents the synchronization group of numerous packets.
4. Every packet uses its first several bits as the index in the time window and the rest as file content to be transferred. Since the arrival of packets may be out of order, we include an index in every packet to ensure that the recipient inserts that part of file in the correct place.
   For example if the time-window length is 128, the index length should be log(128) = 7.
   Because the packet arrives later than it was sent and under some circumstances, the last packet may arrive when the rcv finishes last time-window and starts to receive packets in the next time-window. A mechanism is introduced to distinguish packets belonged to different time-windows to avoid contamination. For even time-windows we set the time-window to 0 and for odd time-windows we set the time-window bit to 1.

TF #	0	1	2	3	4	5	…

TF bit	0	1	0	1	0	1	…

A visualized packet structure is shown below:

Packet #	Valid bit	Time-window bit	Index	File content
Packet 0	1	0 or 1, explained above	0000000	
Packet 1	1		0000001	
Packet 2	1		0000010	
…	1		…	
Packet 127	1		1111111	

The packet also includes a valid bit that is used to solve the following problem. Normally the valid bit is 1. However the transfer ends when reaching the end of file, the last time-window size may be less than usual. We fill up all the empty spaces in the time-window with dummy packets with valid bit 0 to make recipient complete receiving the whole time-window and know it has reached the end of file.

Here's an example of the packets in the last time window (suppose there are only two packets left to be transferred):

Packet #	Valid bit	Time-window bit	Index	File content
Packet 0	1	0 or 1, explained above	0000000	
Packet 1	1		0000001	
Packet 2	0		0000010	X
…	0		…	X
Packet 127	0		1111111	X

5. (Part of sendto_dbg) Generates a random number within [0, 1), discard the packet if the number is smaller than user-specified drop rate. Send the packet one by one if it is not dropped. If the rcv replies "busy" because it's handling another sender, ncp will sleep for some amount of time and retry again.
6. After the transfer of all packets in a time window, the ncp stops and send previous lost packets again to the recipient according to its reply (please refer to rcv behavior described below). It will not start sending packets in the next time-window unless it receives the signal that all packets in current time-window are received.
7. Ncp will count the time as it sends packets. Every time ncp sends a distinct packet, it adds its size to a counter. If the size counter exceeds 50MB, it reports the total amount of data successfully sent and calculates the transfer rate.

11. rcv:
4. Listens to the host specified by input.
5. When receiving packets,

a. When receiving the first packet, it records the host address and receives subsequent packets only from that host. If another host makes a request, rcv sends a single bit response packet, indicating it is busy.
b. It discards packets that are not from the current time-window.
c. It reserves a buffer for all packets in a time window, filling up the cache with newly received valid packet according to its index.
d. Discards the packets randomly. For every packet it receives, it marks that packet received and reports a feedback specified below:
   The feedback length is the length of time-window.
   The feedback contains a binary number with 1 representing packet received and 0 representing packet missing.
   For example the time-window length is 8:

Status	Response packet content								
None received		0	0	0	0	0	0	0	0
Missing 1,3,5		1	1	0	1	0	1	0	1
All received		1	1	1	1	1	1	1	1

e. The ncp counterpart will continue sending missing packets in the current time-window until it receives all-received response, then it will turn to send packets in a new time-window with different time-window bits. At this time, the rcv writes the buffer containing data from last time-window to the file and completes a time-window iteration.
f. It reads invalid bits and leaves a mark. Because an invalid package indicates end of file, it completes the file transfer after successfully receiving the last time-window data. However it still marks invalid packets successfully received and reports back so that the sender knows that it completes file transfer.
g. Whenever it receives 50MB of data, it spits out statistics about the total data received and the average transfer rate.

12. t_ncp and t_rcv:
2. Because TCP doesn't experience packet loss, the t_ncp simply send all the packets in order and t_rcv receives it and append the data to file. t_ncp can send a single bit special packet to indicate EOF.