**Programs to assist debugging your Java programs**

The programs in this folder are provided to help you debug your Java programs.

The main idea is to debug your sender and receiver implementations separately by using the provided binary programs.

First, take your sender and the binary receiver (e.g., Receiver1b). Here the assumption is that both your implementation and the binary implementation at least agree on the header format (e.g., the location of sequence number and EoF field, their length and the values used for each field). **See below more details**. Otherwise, you cannot use the binary programs for debugging your implementation.

Run an experiment with your sender and the receiver and see if the file is correctly delivered to the receiver even in the presence of packet losses and premature timeouts. If there is no problem, you may conclude that your sender implementation is correct. If not, you need to debug your sender and fix problems.

Similarly, now take your receiver and the binary sender. Run the same experiment and see if your receiver correctly receives the image. If so, you may conclude that your receiver has no problem.

Finally, using both your sender and receiver, run the same experiment. See if there is no problem in transmitting the image file.

**How to run programs**

**<Part 1b>**

./Receiver1b <port number> <filename to save>

./Sender1b localhost <port number> <filename> <retryTimeout>

**<Part 2a>**

./Receiver2a <port number> <filename to save>

./Sender2a localhost <port number> <filename> <retryTimeout> <window size>

**<Part 2b>**

./Receiver2a <port number> <filename to save> <window size>

./Sender2a localhost <port number> <filename> <retryTimeout> <window size>

Note: In Part 2b, the <window size> on both sender and receiver should be set to the same value.

**Note on the header formats**

**<Header format for Sender -> Receiver>**

| 2 byte seq no | 1 byte EoF flag | data (up to 1024 bytes) |

Let's say the two byte seq no in the header consist of seq[0] and seq[1]. It follows the big-endian format.

For part 1b, seq[0] is always 0 (0x00) and seq[1] is either 0 (0x00) or 1 (0x01).

For part 2a, let’s take seq number 273 as an example. In the big-endian format, seq[0] is 1 (0x01) and seq[1] is 17 (0x11).

When the sender specifies a sequence number in the packet, it needs to convert the sequence number into a big-endian format. The following code would be useful:

seq[0] = (byte) ((seqno >> 8) & 0xff);

seq[1] = (byte) (seqno & 0xff);

The EoF flag is 1 (0x01) if the packet is a last one; otherwise, 0 (0x00).

**<Header format for Receiver -> Sender>**

| 2 byte seq no |

The 2 byte seq no contains an ACK number. The format of the seq no number is exactly the same as the above.

When the sender receives an ACK, it requires to convert the ACK number in a big-endian format into one in little-endian format. The following code would be useful:

int ackno = (seq[0] & 0xff) << 8 | (seq[1] & 0xff);

**Final Note**

The binary may only work in the dummynet VM because the sources are compiled in it. Unlike Java, C/C++ programs are OS-dependent.

The programs do produce some messages useful for debugging, but do not print the final throughput and # of retransmissions on purpose. The receiver definitely stores the received file; therefore, you should be able to check if the programs work correctly.

When you use the programs, first run receiver and sender in the VM and check whether they work correctly. Afterwards, you can debug your implementation using either the binary sender or receiver.

If the binary programs are only used, they should work correctly. **IF NOT, DO NOT USE THEM FOR DEBUGGING**. Report the problems to me.

However, during the debugging (e.g., your sender and the binary receiver are in use), the binary programs (e.g., the binary receiver) may crash or work incorrectly due to some incompatibility issue between your (buggy) programs and the binary programs. This is normal; when all bugs in your programs are fixed, the binary programs should work correctly.