Project 2: Reliable File Transfer Protocol

Team Members

Hao Cui (hc42@rice.edu); Mo Tang (mt60@rice.edu); Siqi Huang (sh71@rice.edu)

Usage

```
make all
```

And you will get the executable files `recvfile` and `sendfile`.

recvfile

```
./recvfile -p <recv_port>
```

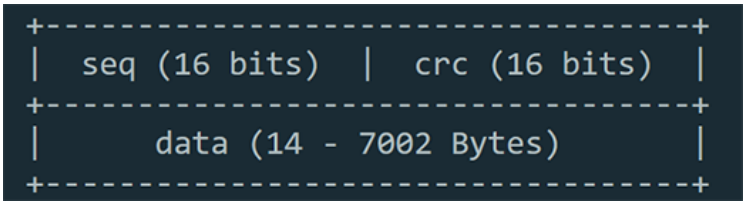
sendfile

```
./sendfile -r <receiver_host>:<receiver_port> -f <sub_dir>/<file_name>
```

Features

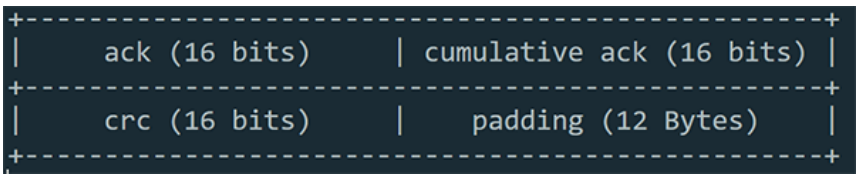
This implementation is a modification of `Sliding Window` and `Selective Repeat`.

Data Packet



- `seq`: The sequence number of the packet, which is a global count (i.e., 0, 1, 2, ...) rather than a file stream position, so there's no reuse of seq number and each packet has a distinct seq. Since the file size will not exceed 30M, a 16-bit number will be enough.
- `crc`: Calculated by seq + data.
- `data`: The file stream. By default the length is 7000 bytes, except the first and last packet (which will be discussed later). The min length is 14 bytes because a Ethernet packet should be larger than 46 bytes (46B - ip header 20B - UDP header 8B - self header 4B).

ACK Packet

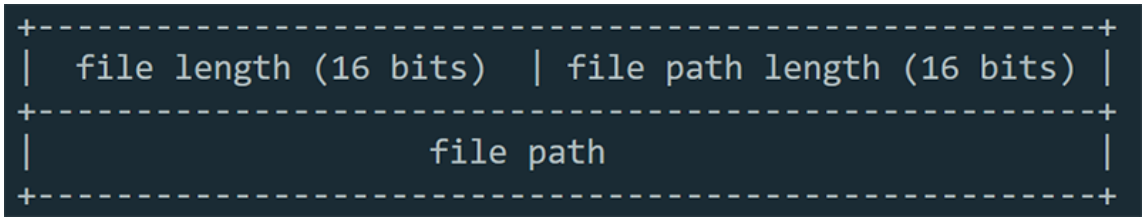


- `ack`: The seq number of the packet that is being acked. This field acks a single packet.
- `cumulative ack`: One plus the seq number of the biggest in-order packet that the receiver has. This field acks all packets with seq numbers smaller than it.

- `crc` : Calculated by seq + cumulative ack + padding
- `padding` : To reach the min length of a Ethernet packet (46B - ip header 20B - UDP header 8B - self header 6B).

The First Data Packet

The first packet will include the length of the file and the file path info in its data field:

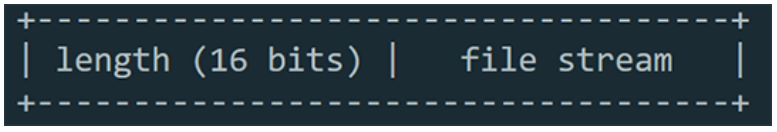


- `file length` : The length of the file (in packets). This indicates the sequence number of the last packet.
- `file path length` : The length of the file path (next field, in bytes)
- `file path` : The path that the file will be stored at the receiver.

Attention: To fulfill the min length of a Ethernet packet, this packet may have paddings.

The Last Data Packet

The last packet may not be 7000 bytes in length, so the actual length should also be transmitted:



- `length` : The length of data in this packet (in bytes). So the receiver will know how long to read.
- `file stream` : The actual data. Note that it is possible for the length of the last packet to be just 7000 bytes, so the total length of the data field can be as large as 7002 bytes.

Attention: To fulfill the min length of a Ethernet packet, this packet may have paddings.

Data Field Length and Window Size

Under this particular situation, that `packets` may be dropped/mangled at a percentage, the larger a packet is, the more efficient this protocol will be. However, we should also have enough windows to cache data in case of receiving out-of-order packets. Given 1M memory, after making some tests, we finally choose 7000 bytes for data field length and 100 for window size.

ACK and Cumulative ACK

Since there may be drops, reorders and delays that happen randomly, we adopt both ack and culumative ack.

A `selective ack` is helpful when out-of-order packets arrive, so the receiver may know exactly which packets should be retransmitted.

If there are many drops or delays, the `cumulative ack` will gain the chance for the receiver to know which packets are received and thus avoid unnecessary retransmisions. For instance, if the sender sends packet 1, 2 and 3, and the acks for the first two packets are lost, the senders will still know their arrivals if it receives a cumulative ack of 4.

For receiver, if it accepts a packet, it will immediately send a ack for this packet, with the seq number of this packet in its ack field, and also a cumulative ack for currently received in-order packets. If the packet corrupts or it is out of the receiving window, the receiver will send an ack packet with only cumulative ack.

Timeout and Retransmision

Because packets may be dropped at a percentage, the TCP implementation that only resends the packet at the beginning of the sending window when timeout happens won't work well. In our design, each packet has its `own timer`. However,

checking all packets in every loop is inefficient. To address this, we only check the timers `periodically`, say every 0.7ms or so. If there are any timeout packets, just resend them and reset their timers.

File Reading and Writing

With limited memory, file should be read and written in `stream`.

For sender, we calculate the length of the file before sending, and track the position we are at in the stream, so we will know whether next packet is the last one. If there are available windows, read file as packets into them and send.

For receiver, writing will only happen when a `in-order` packet arrives. It keeps writing until there's no cached data in the next window.

Graceful Exit

Both sender and receiver should exit gracefully.

For receiver, it knows the seq number of the last packet from the first packet (with seq number 0), so when it has written the last packet, it sends a special packet with a `FINISH_FLAG` in the ack field, and exits.

For sender, when it receives the ack of `FINISH_FLAG`, it exits. However, in case the final ack is lost, it will also exit if the resend counter reaches a certain threshold after it has sent the last packet.