Cooper Weaver (cpweaver)
Connor O'Kelly (cokelly)

Compile: make
Run: ./vodserver <frontend port> <backend port>

The Cooper & Connor Protocol (CCP)

Checkpoint progress:

For our project we implemented a variation of the TCP protocol that was a bit lighter and simpler than standard TCP. We've decided to call this protocol CCP for "Cooper and Connor Protocol" because we are original and creative. We eliminated all the flags except for SYN, ACK, and FIN, and we included a flow ID to simplify multiplexing and demultiplexing flows. To keep track of flows we implemented a global list of flow structs (struct definition below), which holds all the data about a particular flow which allows us to identify either the file we are supposed to be reading or the client we are supposed to be forwarding to based upon which side of the connection we're on, as well as other important information (sequence numbers and ports). We capped the number of concurrent flows at 10, but this number was arbitrary and could be raised or lowered appropriately.

We split our packet sending into 5 different helper functions:

send_CCP_request()	The send_CCP_request function is the first part of the three-way handshake. This function sends a SYN request to the backend server containing the file. The filename is sent in the data.
send_CCP_accept()	send_CCP_accept is called when the backend server has received a syn request with the name of a file in the data. It sends a SYN/ACK with the length of the file in the data.
send_CCP_ack()	The send_CCP_ack function is the final step of the three-way handshake, as well as what will be sent by the receiver for the remainder of the flow. It is sent with no data.
send_CCP_data()	The send_CCP_data function forwards the header along with the file data (up to 1484 bytes per packet - 16 used for the header) to the client requesting the file. If it is sending the end of the file then it also sets its FIN flag.
send_CCP_ackfin()	The send_CCP_ackfin function is called when

	the server receives a data packet with a FIN flag -- this is the final ACK and the flow will be closed following.

In order to implement peer add, we created a global dictionary of content structs to be stored on the backend server. The implementation of the dictionary was simply an array of the content structs. Each content struct contained the file name, its host, its port, and its bit rate. So, when we received a peer view request, we simply searched through our dictionary to extract all of the relevant information regarding the requested file. From there, we built the corresponding CCP header (also using the active_flow struct information) and sent the file back to the front end server.

For Final Submission:

Since the checkpoint, we have implemented windowing, peer_config, peer_status, packet drop recovery, and updated our active_flow struct.

Since peer_config sets the overall bit rate for our backend server, we had the function set a global bit rate variable. To configure the rate, we added some elements to our active flow struct (which is shown below), one of which includes the maximum window size. We set this field using the following equation:

- Current_flow → max_window_size = (Round Trip Time) * (Max_bandwidth) / (Packet Size)

We calculated the Round Trip Time by adding start and end fields to the active flow struct. We clocked start when the first SYN is sent and clocked end when the SYN_ACK comes back. Thus, each connection will have its own max_window_size to ensure the requested bitrate is met.

To implement peer_status, we looped through our set of active flows and printing the flow id, the file being transferred, and the completion percentage on the http page.

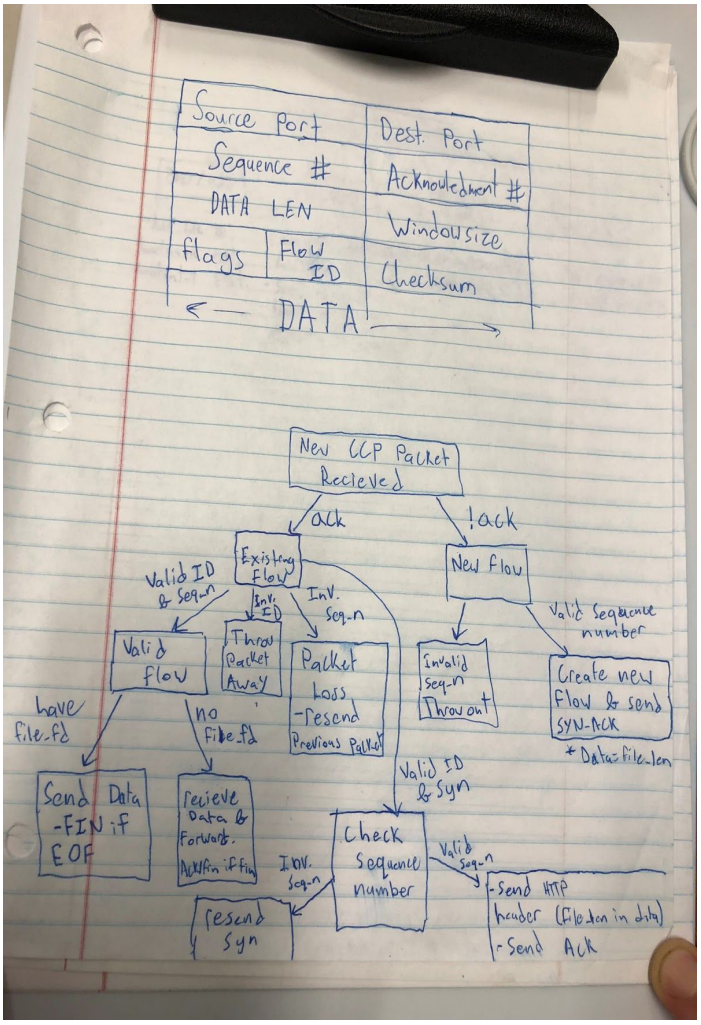
In order to detect a dropped/missed packets we compared received and expected sequence numbers and if they did not match (i.e. the expected packet was not received) we would set an internal flag marking an error, decrement it's own sequence number and resend the previous acknowledgment. The internal error flag tells the receiver to throw out all other queued packets in the flow until it receives the missed packet. When the sender receives the repeated acknowledgement it would then move back in the file, adjust its sequence number, and continue sending from where the packet was dropped. This does not implement caching -- if Packet 4 is lost and Packets 5 and 6 were delivered to the receiver, the sender will resend all three packets. This is not ideal -- but it was much simpler.

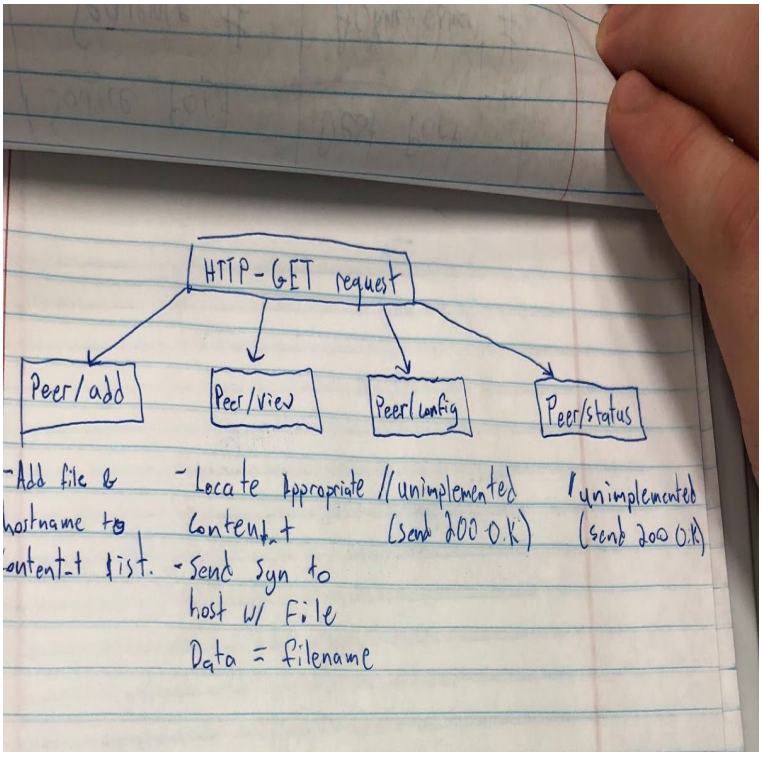
Additional Notes:

Regarding Videos, our implementation can support multiple large video files (400MB). However, it does not support .ogg files. We suspect that this is because we are not buffering any of the packets that we are sending and .ogg files are trying to do computations on a continuous stream of bits (which we are not providing). We did discover that at random occurrences .ogg files are supported - we do not know why this happens. When testing files, make sure to avoid .ogg files.

```
typedef struct active_flow {
  uint8_t id;
  struct sockaddr_in partneraddr;
  int addrlen;
  uint16_t destport;
  uint16_t sourceport;
  uint16_t my_seq_n;
  uint16_t your_seq_n;
  clock_t last_clock;
  int RTTEstimate;
  FILE *file_fd;
  int client_fd;
  char *version;
  char *file_name;
} active_flow;


typedef struct content *content_t;
struct content {
  char *file;
  char *host;
  char *port;
  char *brate;
};
```


Source Port
Dest. Port
Sequence #
Acknowledgment #
DATA LEN
Windowsize
Flags
Flow ID
Checksum
DATA
New UDP Packet Recieved
ack
!ack
Existing Flow
New Flow
Valid ID & Seq-n
Inv. ID
Inv. Seq-n
Valid Sequence number
Valid Flow
Throw Packet Away
Packet Loss -resend Previous Packet
Invalid Seq-n Throw out
Create new Flow & send SYN-ACK
* Data=File_len
have file-fd
no File fd
Send Data -FIN if EOF
Recieve Data & Forward. Ack/fin if Fin
Valid ID & Syn
Check Sequence number
Inv. Seq-n
Valid Seq-n
resend Syn
-Send HTTP header (File_len in data)
-Send Ack


HTTP-GET request
Peer/add
Peer/view
Peer/config
Peer/status
-Add file & hostname to content_t list.
- Locate appropriate content_t
(send 200 O.K)
/unimplemented (send 200 O.K)
- Send Syn to host w/ File
Data = filename