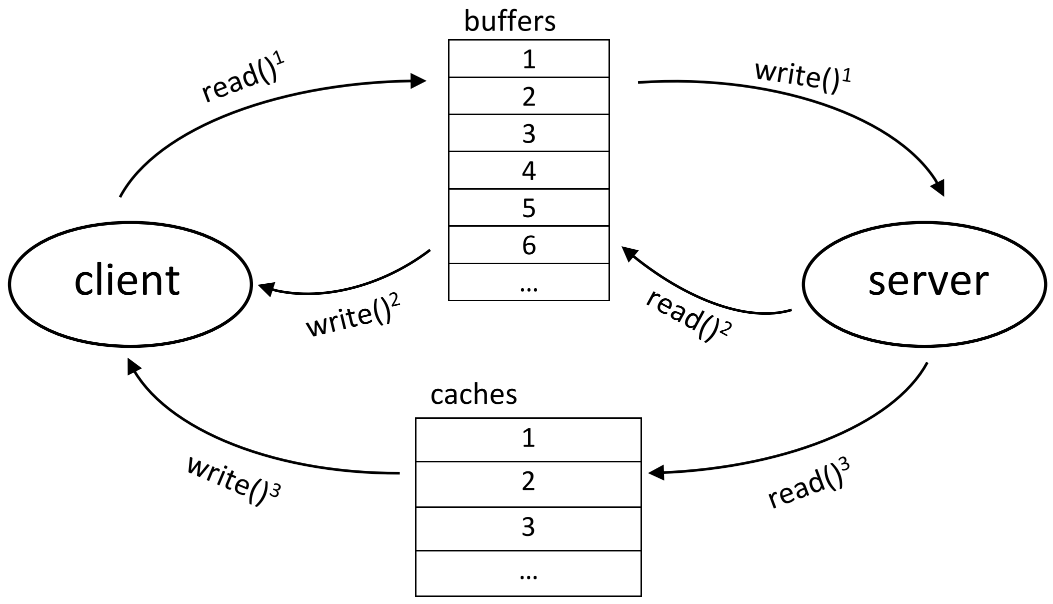
SUMMARY

For our current design, we use the “select” function to also select a read and a write file descriptor set. We implemented rate limiting by having a fixed-size read buffer assigned to each file descriptor when connected so that we can limit how many bytes are transferred during a successful selection. This is also our idea of bandwidth limiting for clients: when multiple clients send requests to the server, the server should limit or cap the bandwidth of each one so that no clients get more resource over time than others. Furthermore, we implemented a non-blocking search on our cache so that the cache can show all the message bodies that contain a search term.

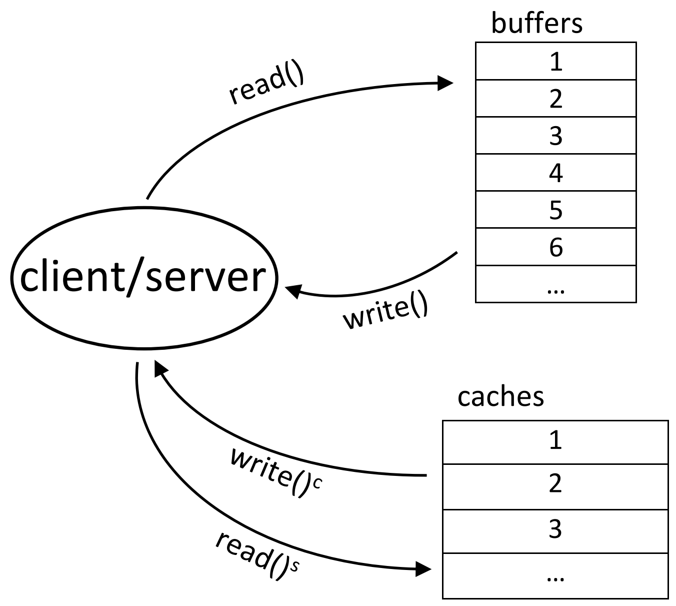
DESIGN



Pic 1. initial design structure

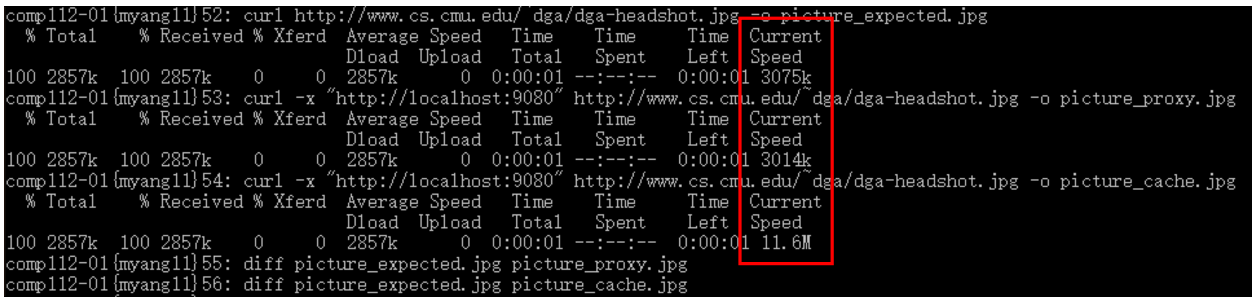
For this project, our initial design used the “select” to choose a read file descriptor set. Our proxy received connections and read requests from clients, but completed each request before processing other requests. This treated each client unequally - if a client is trying to download a huge file, it will block, which does not allow other clients to contact the server during that time.

Our current design integrates all read and write calls into being chosen by the “select” function. For this design, as the data is transferred back and forth between the client and server for “CONNECT” request, we have to use timeout for every client using “setsockopt()” to not let client’s “read” block our process when it is not its turn to read data.



Pic 2. current design structure

This design can handle multiple clients, respond with “GET” and “CONNECT” requests, and enables the caching feature. The download speed is also comparable to the baseline (3014k vs. 3075k as shown in the picture below).



```
comp112-01{myang11}52: curl http://www.cs.cmu.edu/~dga/dga-headshot.jpg -o picture_expected.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2857k  100 2857k    0     0  2857k      0  0:00:01 --:--:--  0:00:01 3075k
comp112-01{myang11}53: curl -x "http://localhost:9080" http://www.cs.cmu.edu/~dga/dga-headshot.jpg -o picture_proxy.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2857k  100 2857k    0     0  2857k      0  0:00:01 --:--:--  0:00:01 3014k
comp112-01{myang11}54: curl -x "http://localhost:9080" http://www.cs.cmu.edu/~dga/dga-headshot.jpg -o picture_cache.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2857k  100 2857k    0     0  2857k      0  0:00:01 --:--:--  0:00:01 11.6M
comp112-01{myang11}55: diff picture_expected.jpg picture_proxy.jpg
comp112-01{myang11}56: diff picture_expected.jpg picture_cache.jpg
```

Pic 3. baseline speed vs. proxy speed vs. cache speed

We furthermore added to the cache to allow for non-blocking cache searches. From the location of the proxy, you can enter commands into the console to search for message bodies that contain your search terms. The design we used for this is flexible and extensible, to allow for further commands - for example, we implemented a simple “EXIT” function from the console as well. We were able to make this non-blocking by taking advantage of the “select” call that was already in place. Since stdin is a file descriptor, it is able to cooperate with the select call just like sockets.

We had begun work on transcoding into gzip, but this was cut short by submission time, and is not integrated into our current design.

We use the buffer status to decide the read file descriptor set and the write file descriptor set. If a file descriptor id is assigned to a buffer which is not full, we set it to the read file descriptor set. If we find another buffer with remaining bytes to read in our write buffer look-up table using its id, we set it to our write file descriptor set. By doing this, we limit the max number of bytes to transfer during a successful "select." For "write," writing data from our cache to the client, we also use the value of our buffer size as the upper bound for writing.

We don't show the work flow here as it depends on the buffer status. We give the buffer a field named "size" and a field named "size_written". If size == size_written, we know we can't write (take) any data out of it. If size < the buffer size we set, we know we can read (store) any data into it.

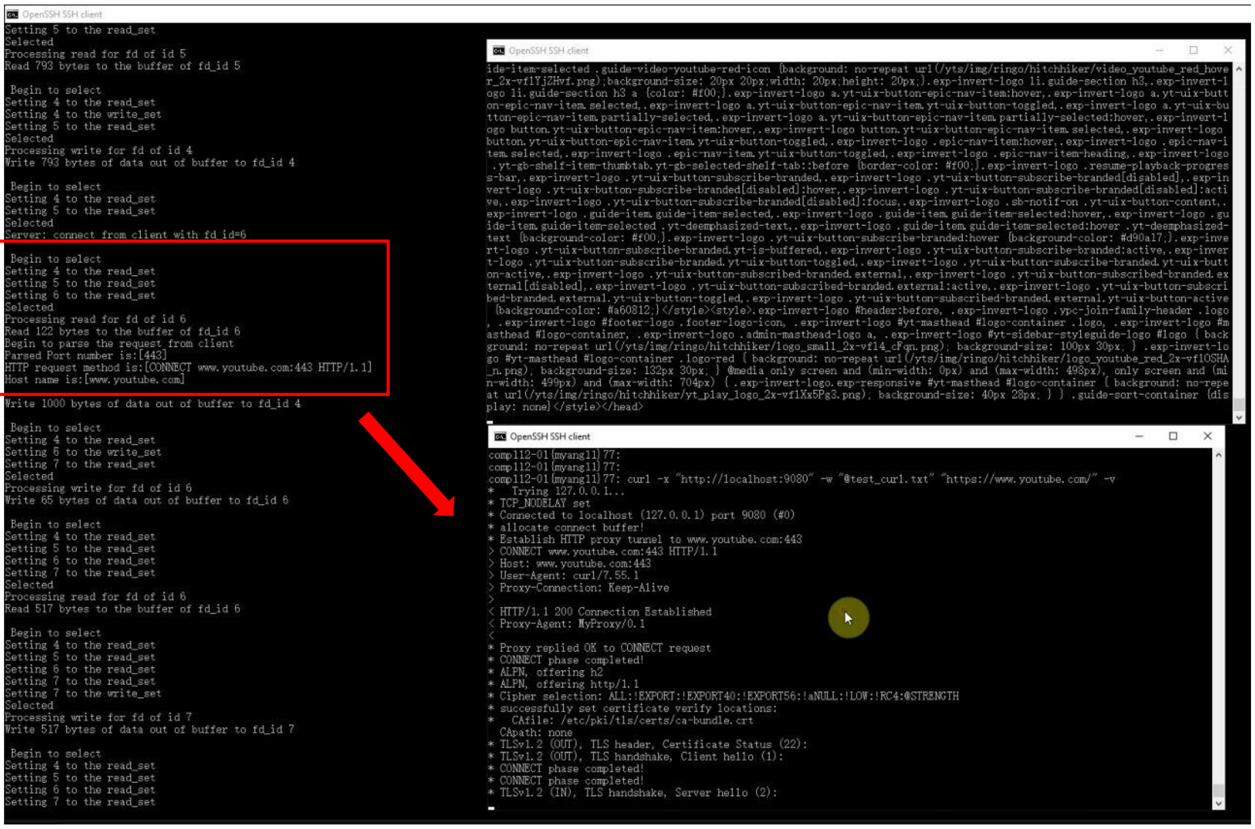
DEMO DESCRIPTIONS

"test_timeout_for_5_seconds.mp4"

In this video, the customized client uses "read()" to take over the proxy server without doing any "write" actions. We purposely change the constant of timeout value from 60 seconds to 5 seconds so that it is easy for demo.

"test_two_connections.mp4"

In this video, we show that the proxy server can handle multiple clients. We use "curl" to print the result to I/O in order to make the process slower so that I can quickly fire another request from a second Command Prompt. In the picture below which is one frame taken from the video, the 2nd "CONNECT" request gets properly handled while processing the first one.



```
Setting 5 to the read_set
Selected
Processing read for fd of id 5
Read 793 bytes to the buffer of fd_id 5

 Begin to select
Setting 4 to the read_set
Setting 4 to the write_set
Setting 5 to the read_set
Selected
Processing write for fd of id 4
Write 793 bytes of data out of buffer to fd_id 4

 Begin to select
Setting 4 to the read_set
Setting 5 to the read_set
Selected
Server: connect from client with fd_id=6

 Begin to select
Setting 4 to the read_set
Setting 5 to the read_set
Setting 6 to the read_set
Selected
Processing read for fd of id 6
Read 122 bytes to the buffer of fd_id 6
Begin to parse the request from client
Parsed Port number is:[443]
HTTP request method is:[CONNECT www.youtube.com:443 HTTP/1.1]
Host name is:[www.youtube.com]
Write 1000 bytes of data out of buffer to fd_id 4

 Begin to select
Setting 4 to the read_set
Setting 6 to the write_set
Setting 7 to the read_set
Selected
Processing write for fd of id 6
Write 65 bytes of data out of buffer to fd_id 6

 Begin to select
Setting 4 to the read_set
Setting 5 to the read_set
Setting 6 to the read_set
Setting 7 to the read_set
Selected
Processing read for fd of id 6
Read 517 bytes to the buffer of fd_id 6

 Begin to select
Setting 4 to the read_set
Setting 5 to the read_set
Setting 6 to the read_set
Setting 7 to the read_set
Setting 7 to the write_set
Selected
Processing write for fd of id 7
Write 517 bytes of data out of buffer to fd_id 7

 Begin to select
Setting 4 to the read_set
Setting 5 to the read_set
Setting 6 to the read_set
Setting 7 to the read_set
```

```
ide-item-selected .guide-video-youtube-red-icon {background: no-repeat url(/yts/img/ringo/hitchhiker/video_youtube_red_hove
r_2x-vf1YjZHvf.png);background-size: 20px 20px;width: 20px;height: 20px;}.exp-invert-logo li.guide-section h3,.exp-invert-l
ogo li.guide-section h3 a {color: #f00;}.exp-invert-logo a.yt-uix-button-epic-nav-item:hover,.exp-invert-logo a.yt-uix-butt
on-epic-nav-item.selected,.exp-invert-logo a.yt-uix-button-epic-nav-item.yt-uix-button-toggled,.exp-invert-logo a.yt-uix-bu
tton-epic-nav-item.partially-selected,.exp-invert-logo a.yt-uix-button-epic-nav-item.partially-selected:hover,.exp-invert-l
ogo button.yt-uix-button-epic-nav-item:hover,.exp-invert-logo button.yt-uix-button-epic-nav-item.selected,.exp-invert-logo
button.yt-uix-button-epic-nav-item.yt-uix-button-toggled,.exp-invert-logo .epic-nav-item:hover,.exp-invert-logo .epic-nav-i
tem.selected,.exp-invert-logo .epic-nav-item.yt-uix-button-toggled,.exp-invert-logo .epic-nav-item-heading,.exp-invert-logo
 .yt-gb-shelf-item-thumbtab.yt-gb-selected-shelf-tab::before {border-color: #f00;}.exp-invert-logo .resume-playback-progres
s-bar,.exp-invert-logo .yt-uix-button-subscribe-branded,.exp-invert-logo .yt-uix-button-subscribe-branded[disabled],.exp-in
vert-logo .yt-uix-button-subscribe-branded[disabled]:hover,.exp-invert-logo .yt-uix-button-subscribe-branded[disabled]:acti
ve,.exp-invert-logo .yt-uix-button-subscribe-branded[disabled]:focus,.exp-invert-logo .sb-notif-on .yt-uix-button-content,.
exp-invert-logo .guide-item.guide-item-selected,.exp-invert-logo .guide-item.guide-item-selected:hover,.exp-invert-logo .gu
ide-item.guide-item-selected .yt-deemphasized-text,.exp-invert-logo .guide-item.guide-item-selected:hover .yt-deemphasized-
text {background-color: #f00;}.exp-invert-logo .yt-uix-button-subscribe-branded:hover {background-color: #d90a17;}.exp-inve
rt-logo .yt-uix-button-subscribe-branded.yt-is-buffered,.exp-invert-logo .yt-uix-button-subscribe-branded:active,.exp-inver
t-logo .yt-uix-button-subscribe-branded.yt-uix-button-toggled,.exp-invert-logo .yt-uix-button-subscribe-branded.yt-uix-butt
on-active,.exp-invert-logo .yt-uix-button-subscribed-branded.external,.exp-invert-logo .yt-uix-button-subscribed-branded.ex
ternal[disabled],.exp-invert-logo .yt-uix-button-subscribed-branded.external:active,.exp-invert-logo .yt-uix-button-subscri
bed-branded.external.yt-uix-button-toggled,.exp-invert-logo .yt-uix-button-subscribed-branded.external.yt-uix-button-active
 {background-color: #a60812;}</style><style>.exp-invert-logo #header:before, .exp-invert-logo .ypc-join-family-header .logo
, .exp-invert-logo #footer-logo .footer-logo-icon, .exp-invert-logo #yt-masthead #logo-container .logo, .exp-invert-logo #m
asthead #logo-container, .exp-invert-logo .admin-masthead-logo a, .exp-invert-logo #yt-sidebar-styleguide-logo #logo { back
ground: no-repeat url(/yts/img/ringo/hitchhiker/logo_small_2x-vf14_cFqn.png); background-size: 100px 30px; } .exp-invert-lo
go #yt-masthead #logo-container .logo-red { background: no-repeat url(/yts/img/ringo/hitchhiker/logo_youtube_red_2x-vf1OSHA
_n.png); background-size: 132px 30px; } @media only screen and (min-width: 0px) and (max-width: 498px), only screen and (mi
n-width: 499px) and (max-width: 704px) { .exp-invert-logo.exp-responsive #yt-masthead #logo-container { background: no-repe
at url(/yts/img/ringo/hitchhiker/yt_play_logo_2x-vf1Xx5Pg3.png); background-size: 40px 28px; } } .guide-sort-container {dis
play: none}</style></head>
```

```
comp112-01{myang11}77:
comp112-01{myang11}77:
comp112-01{myang11}77: curl -x "http://localhost:9080" -w "@test_curl.txt" "https://www.youtube.com/" -v
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 9080 (#0)
* allocate connect buffer!
* Establish HTTP proxy tunnel to www.youtube.com:443
> CONNECT www.youtube.com:443 HTTP/1.1
> Host: www.youtube.com:443
> User-Agent: curl/7.55.1
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 Connection Established
< Proxy-Agent: MyProxy/0.1
<
* Proxy replied OK to CONNECT request
* CONNECT phase completed!
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: /etc/pki/tls/certs/ca-bundle.crt
  CApath: none
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* CONNECT phase completed!
* CONNECT phase completed!
* TLSv1.2 (IN), TLS handshake, Server hello (2):
```

COMPILE AND RUN

The files included are the proj.c and a Makefile. These should be placed within the same directory. To start the server, navigate in the console to the directory of the files and run the following command, where "<port>" is the port number you would like to use:

```
make && ./proj <port>
```