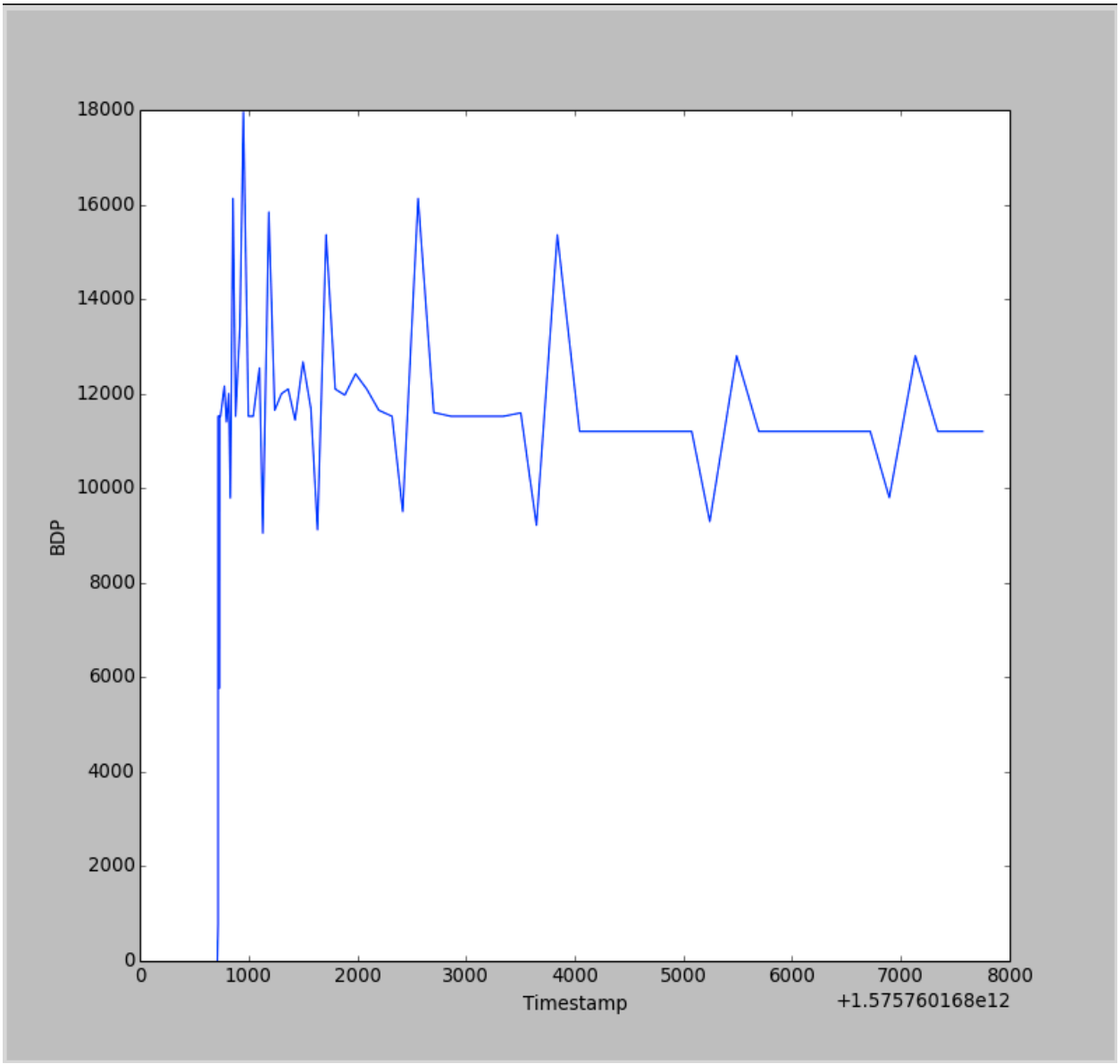
Lab 3 – Part 2: BBR

Plot of sending a large file



- Program Structure and Design -
  - My BBR implementation is loosely based on this description of BBR: https://tools.ietf.org/id/draft-cardwell-iccrg-bbr-congestion-control-00.html
  - The main difference is that I have everything based around one function bbr_on_ack(), which handles transitioning between modes, calculating the bandwidth and the rtt, and everything else that BBR needs to do.
  - I skipped out on a lot of the harder to implement details of BBR, because my code was working pretty well, and my bdp plot has the form of a nice BBR implementation.

    - There is nothing really groundbreaking about my implementation—I really followed the above link, and simplified it where needed.
    - In order to avoid implementing some of the harder parts of BBR, I used approximation, that I verified through testing. For example, I stay a set amount of time (2 Rtt's) in drain. This turns out to work pretty well.
    - On_ack can be broken down into a few major parts:
        - Update_btl_bw
        - Check for 10 second probe rtt timer
        - Check if the pipe is full
        - Set the cwnd
        - Calculate the next send time for pacing
        - Figure out if there are any other state changes (switch statement)
        - Update the RTT
        - Log data to the file for debugging.
        - Enforce the floor of 4 on cwnd
    - Chosen by testing and looking at log files and plot:
        - static const float bbr_high_gain = 2.5;
        - static const float bbr_drain_gain = 0.8;
        - static const float bbr_cwnd_gain = 1.0;
    - This was because my drain was draining too much and startup was spiking and increasing the queue sizes too much.
- Implementation Challenges
    - I was having trouble calculating rtt at first. I cast everything as ints and this type conversion to long was leading to BDP values that were way too high because the MSB's were being truncated
    - I also ran into the problem of seg faults due to the memory for the bbr struct being malloc'd in ctcp.c. I changed this to have the bbr_init function return a pointer to a bbr struct, and no longer had a seg fault.
    - Overall, I think that the hardest part of this was lab was picking and choosing what to features of BBR to actually implement in your code. I choose not to implement some of the features I thought would be very tough, and simply approximate them as described above.
- Testing -
    - Overall, my testing procedure went like this:
        - Import all my BBR code and fail miserably… I don't know what naïve hope always causes me to do this.
        - Restart with a ctcp file from part 1 (debugged version)
        - Slowly add in BBR functionality—testing every time and making sure that it compiled and could deliver packets (using ctcp_test.py)
        - Once I had all my bbr functioning, I began to test large file transfer, so I could tweak my parameters. This is where I tweaked all the parameters related to high_gain, drain_gain, and unity gain.

        - I also tweaked my code at this point to make the drain mode and probe RTT modes shorter, because that is where I would lose lots of Bandwidth.

        - I implemented an output file in BBR to plot specific values in BBR. This allowed me to debug more easily

- Remaining Bugs -
    - Sometimes, when I run the code on the same port twice in a row, it will see lots of errors related to a fin already received or waiting on a previous ACK (out of order delivery)
    - Sometimes the drain phase is too long, and I lose a lot of bandwidth.
    - If you encounter a really strange error where it does not work, just retry on a new port.
    - I have been sending the file exmovie.MP4 for testing. You can use this as well.
        - ./ctcp -p 9999 -c localhost:9995 < exmovie.MP4