Sankaradithyan Gunasekharan - gunasekh@usc.edu
CSCI 551 Lab 3 Part 2

Lab 3 Part 2: Build Your Own Transport

1. Introduction

In this lab I wrote code for BBR congestion control protocol developed for the Internet by Google. Since packet loss is a bad indicator of congestion condition, we need a true indicator of congestion which is Bandwidth Delay Product (BDP). We can accurately estimate maximum bandwidth and minimum RTT. Maximum bandwidth guarantees that TCP can run at 100 percent bandwidth utilization and minimum RTT guarantees there is enough data to prevent receiver starvation but not overfill the buffer. Since these two values change separately, we can maximize and minimize the values separately. There are four states of BBR:

a) STARTUP: exponential growth to quickly fill pipe (like slow-start), will stop growth when bandwidth estimate plateaus, not on loss or delay.
b) DRAIN: drain the queue created in STARTUP phase.
c) PROBE_BW: cycle pacing_gain to explore and fairly share bandwidth.
d) PROBE_RTT: occasionally send slower to probe min RTT.

2. Program Structure, Design and Tools Used

A. Program Structure

a) ctcp.c – Added code to handle changes to the pacing gain by calculating the timing for when we need to send the next packet. Whenever an ACK is received at the receiver end it will invoke a function check_bbr_state() which handles the state change. It will also update the rtt, rt_prop, bandwidth, btwbw, pacing_gain and congestion window gain.
b) ctcp.h – Added the definitions of the newly added handler function.
c) ctcp_bbr.h – Added code for the bbr structure which keeps track of the various parameters like, rtt, cwnd, rt_prop, bw, next packet send time, etc This is the structure which deals with all the main code of bbr. Also added array of float values for the pacing gain values. Contains an enum bbr_mode as well to track the bbr states.
d) ctcp_bbr.c – Added function to change_to_probe_bw() when we reach the plateau. Added function change_to_probe_rtt() when the min RTT remains the same for 10 seconds. Also added a function check_bbr_state() which is invoked every time we receive and ACK and will update the values of rtt, rt_prop, bandwidth, btwbw, pacing_gain and congestion window gain.

B. Design Details

a) The BBR will start from the STARTUP phase and then move to the other stages as per the condition.

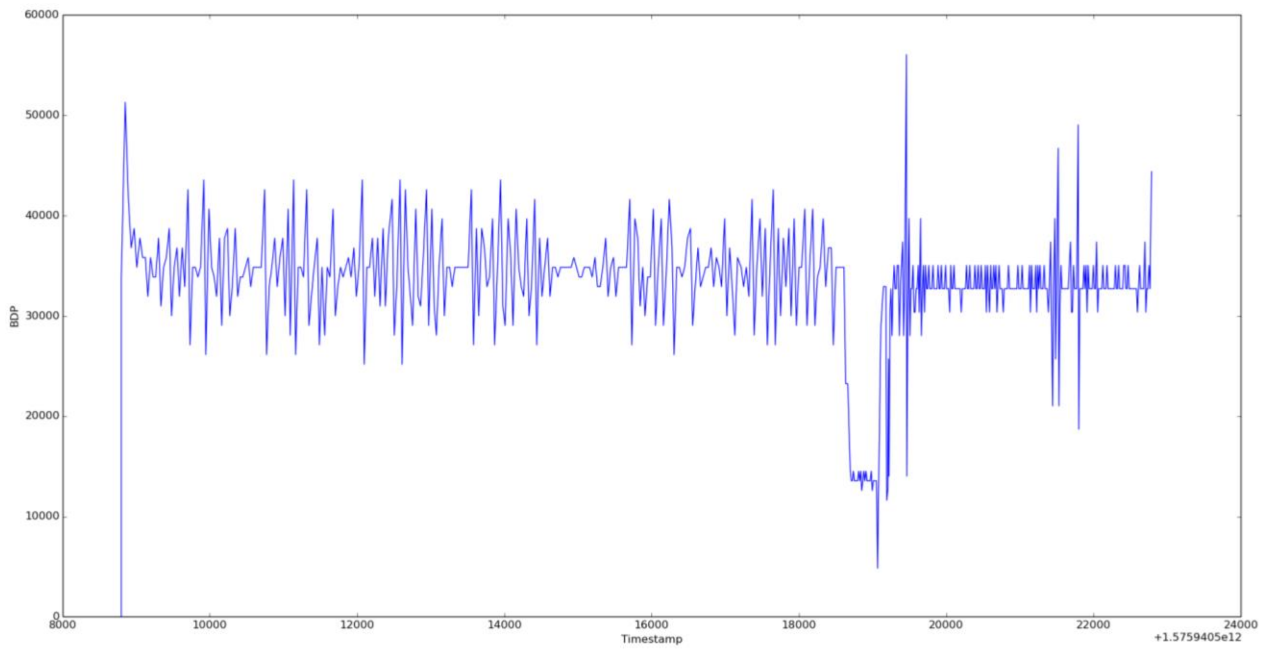
b) In the STARTUP phase the pacing gain is first set to 2/ln2 (2.89) and the congestion window gain starts from the same value. I bandwidth keeps increasing until it reached a plateau where the bandwidth remains constant for 3 consecutive times.
c) In the STARTUP phase the bandwidth keeps increasing which keeps filling the queues in the network and fills the queues up
d) Once there is no further increase in the bandwidth for three times, I move to the DRAIN phase.
e) In the DRAIN phase the pacing gain will become ln2/2 (1/2.89) which will ensure that there is no further drastic increase in the bandwidth and the queues will become empty.
f) At this state I keep checking the value of the inflight packets and if the value of the inflight packets reaches to a value which is less than or equal to the BDP, I then enter the PROBE_BW phase.
g) This is where I iterate through the different values of pacing gain such as [5/4, ¾, 1, 1, 1, 1, 1, 1] to keep the bandwidth to a steady state and provide equal share to all the flows and maximize utilization.
h) The congestion window gain in this phase will remain constant and only the pacing gain will keep looping.
i) If the value of min RTT does not change for 10 seconds, I move on to the PROBE_RTT phase where I am ensuring that there are only 4 packets inflight and until then I will not send new data.
j) On every ACK that I receive at the receiver end I make a check at for the BBR states and update the values of the various parameters according to the state they are in.
k) I calculate the BDP whenever the ACK is received and copy the BDP and current time value to the bdp.txt file which is used for plotting the graph.

C. Tools Used

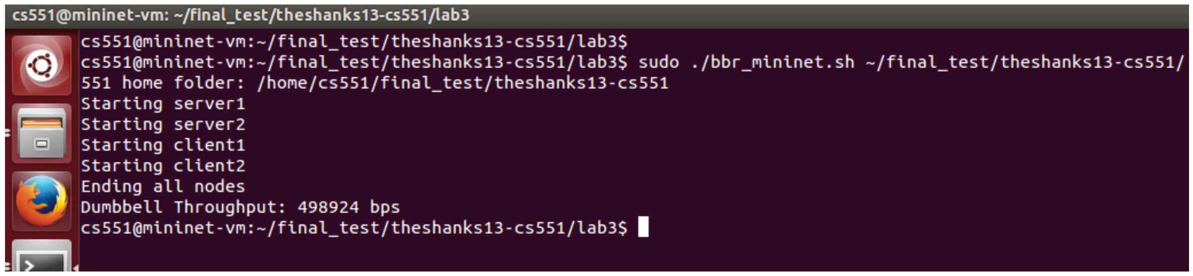
a) GDB to identify memory corruption bugs and isolate bugs in code
b) Git and Github used as version control systems to continuously save the code progress on the local repository and the cloud repository respectively
c) Sublime Text editor to write and edit code

3. BDP Plot, Dumbbell Throughput and I2 Throughput

A. BDP Plot

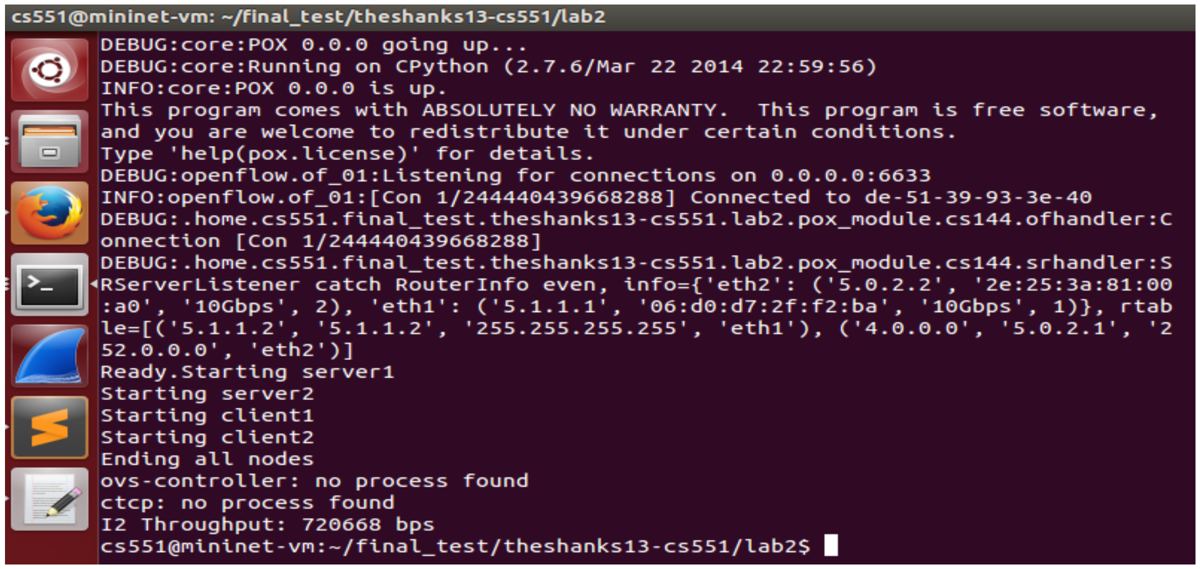


B. Dumbbell Throughput


cs551@mininet-vm: ~/final_test/theshanks13-cs551/lab3

```
cs551@mininet-vm:~/final_test/theshanks13-cs551/lab3$
cs551@mininet-vm:~/final_test/theshanks13-cs551/lab3$ sudo ./bbr_mininet.sh ~/final_test/theshanks13-cs551/
551 home folder: /home/cs551/final_test/theshanks13-cs551
Starting server1
Starting server2
Starting client1
Starting client2
Ending all nodes
Dumbbell Throughput: 498924 bps
cs551@mininet-vm:~/final_test/theshanks13-cs551/lab3$
```

C. I2 Throughput


cs551@mininet-vm: ~/final_test/theshanks13-cs551/lab2

```
DEBUG:core:POX 0.0.0 going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
INFO:core:POX 0.0.0 is up.
This program comes with ABSOLUTELY NO WARRANTY.  This program is free software,
and you are welcome to redistribute it under certain conditions.
Type 'help(pox.license)' for details.
DEBUG:openflow.of_01:Listening for connections on 0.0.0.0:6633
INFO:openflow.of_01:[Con 1/244440439668288] Connected to de-51-39-93-3e-40
DEBUG:.home.cs551.final_test.theshanks13-cs551.lab2.pox_module.cs144.ofhandler:C
onnection [Con 1/244440439668288]
DEBUG:.home.cs551.final_test.theshanks13-cs551.lab2.pox_module.cs144.srhandler:S
RServerListener catch RouterInfo even, info={'eth2': ('5.0.2.2', '2e:25:3a:81:00
:a0', '10Gbps', 2), 'eth1': ('5.1.1.1', '06:d0:d7:2f:f2:ba', '10Gbps', 1)}, rtab
le=[('5.1.1.2', '5.1.1.2', '255.255.255.255', 'eth1'), ('4.0.0.0', '5.0.2.1', '2
52.0.0.0', 'eth2')]
Ready.Starting server1
Starting server2
Starting client1
Starting client2
Ending all nodes
ovs-controller: no process found
ctcp: no process found
I2 Throughput: 720668 bps
cs551@mininet-vm:~/final_test/theshanks13-cs551/lab2$
```

4. Implementation Challenges

The major implementation challenge I faced was making changes to the existing ctcp implementation as I always had to be careful to not break any working of the ctcp code. Another major challenge was to identify when to exactly call the BBR state changes and recalculate the values of rtt, bw and bdp since retransmission should not be counted as part of rtt calculation.

5. Testing and Extra Tests

a) Dumbell Topology:
The dumbbell topology surpasses values of both A (430kbps) and B (460kbps) when run multiple times and reaches a value of 498kbps and on occasion 463kbps.

b) I2 Topology:
I2 test passes only 1/15-20 times when run. In most cases the file transfers are incomplete and even 0 bytes transferred sometime. I have attached the image of I2 Throughput when the attempt when the file was transferred successfully and achieved a throughput of 720kbps.

c) I also manually tested the by individually creating the client and server interface and transferring the file and they sometimes work and sometimes fail. I also get different values of throughput on different occasions.

6. Remaining Bugs and Unexpected Cases

a) I have not handled the case when I receive an ACK or FIN along with data segment. This is the case when there is piggybacking of segments.

b) The Dumbbell and I2 Throughput tests need to be run multiple times since the throughput values will come less on some occasions.

c) For the I2 throughput test needs to be run several times (1 out of 20 times) since on most occasions the file transfer is 0 bytes or either incomplete. The successful scenario can be viewed in the image attached above.