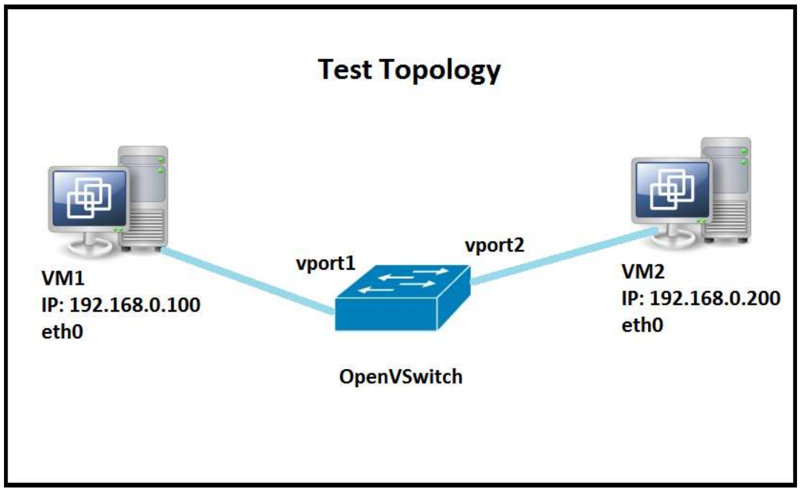
Objective

Our target for Phase 1 was to implement DCTCP in a virtual environment, preferably Mininet.

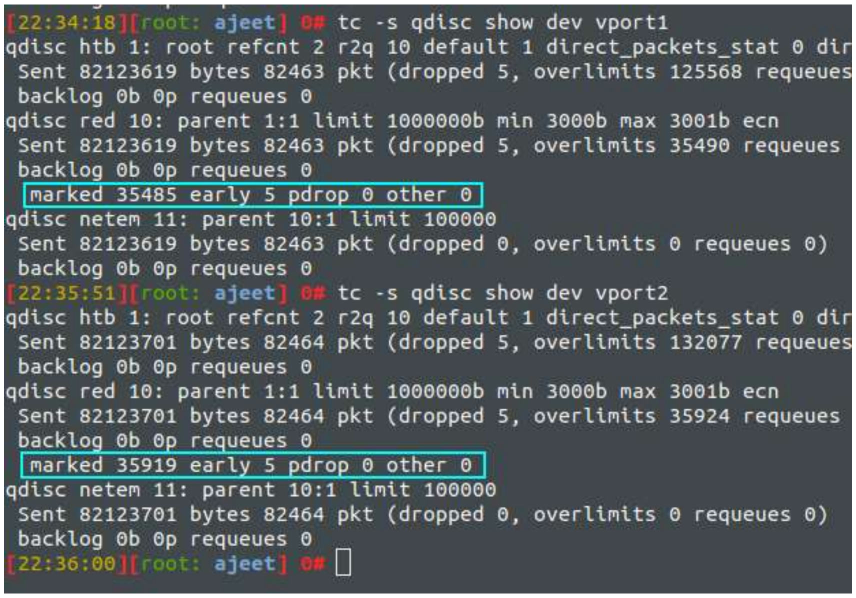
Topology and Configuration

Our topology consists of 2 VMs on Virtualbox running Ubuntu 14.04 (Kernel 3.19). The VMs are connected with each other using OpenVSwitch 2.0.2 as seen in figure.



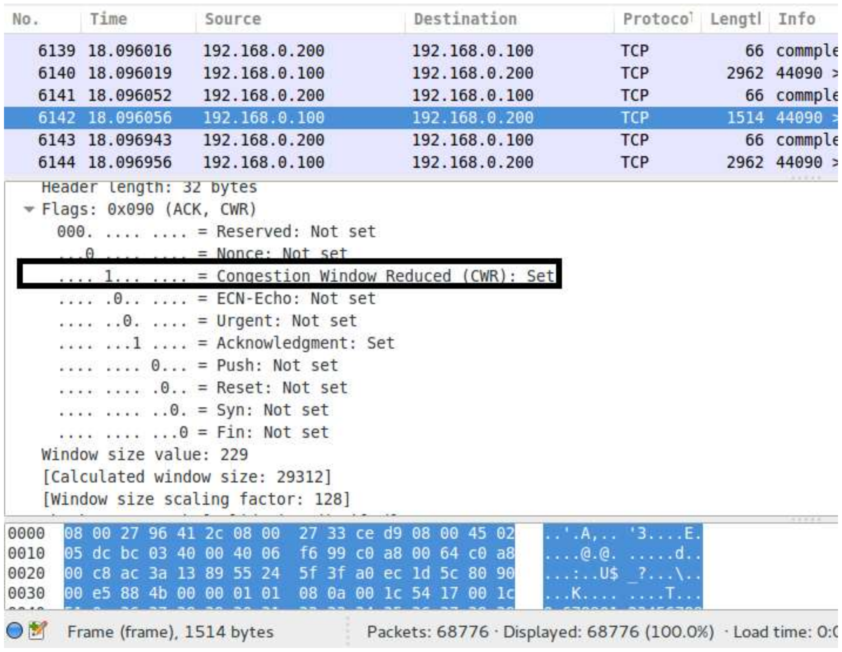
Kernel 3.19 supports DCTCP natively, so the configuration on VMs involved enabling ECN and DCTCP congestion control. The configuration on OpenVSwitch involved creating links to VMs and configuring their queue parameters to enable ECN marking using linux tool *tc*. Bandwidth of switch-host link was set to 75Mbps. DCTCP K value was set to 2 packets for test purpose.

Results



```
[22:34:18][root: ajeet] 0# tc -s qdisc show dev vport1
qdisc htb 1: root refcnt 2 r2q 10 default 1 direct_packets_stat 0 dir
 Sent 82123619 bytes 82463 pkt (dropped 5, overlimits 125568 requeues
 backlog 0b 0p requeues 0
qdisc red 10: parent 1:1 limit 1000000b min 3000b max 3001b ecn
 Sent 82123619 bytes 82463 pkt (dropped 5, overlimits 35490 requeues
 backlog 0b 0p requeues 0
  marked 35485 early 5 pdrop 0 other 0
qdisc netem 11: parent 10:1 limit 100000
 Sent 82123619 bytes 82463 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
[22:35:51][root: ajeet] 0# tc -s qdisc show dev vport2
qdisc htb 1: root refcnt 2 r2q 10 default 1 direct_packets_stat 0 dir
 Sent 82123701 bytes 82464 pkt (dropped 5, overlimits 132077 requeues
 backlog 0b 0p requeues 0
qdisc red 10: parent 1:1 limit 1000000b min 3000b max 3001b ecn
 Sent 82123701 bytes 82464 pkt (dropped 5, overlimits 35924 requeues
 backlog 0b 0p requeues 0
  marked 35919 early 5 pdrop 0 other 0
qdisc netem 11: parent 10:1 limit 100000
 Sent 82123701 bytes 82464 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
[22:36:00][root: ajeet] 0# 
```

We were successful in implementing DCTCP on the test setup. We were able to observe the marked packets on switch queues as well as ECN, ECN-echo and CWR (congestion window reduced) packets on wireshark capture on the sender (as can be seen in the images).



```
No.    Time       Source          Destination     Protocol Length Info
  6139 18.096016  192.168.0.200   192.168.0.100   TCP        66 commple
  6140 18.096019  192.168.0.100   192.168.0.200   TCP      2962 44090 >
  6141 18.096052  192.168.0.200   192.168.0.100   TCP        66 commple
  6142 18.096056  192.168.0.100   192.168.0.200   TCP      1514 44090 >
  6143 18.096943  192.168.0.200   192.168.0.100   TCP        66 commple
  6144 18.096956  192.168.0.100   192.168.0.200   TCP      2962 44090 >

    Header length: 32 bytes
  ▾ Flags: 0x090 (ACK, CWR)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 1... .... = Congestion Window Reduced (CWR): Set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]

0000  08 00 27 96 41 2c 08 00  27 33 ce d9 08 00 45 02   ..'.A,.. '3....E.
0010  05 dc bc 03 40 00 40 06  f6 99 c0 a8 00 64 c0 a8   ....@.@. .....d..
0020  00 c8 ac 3a 13 89 55 24  5f 3f a0 ec 1d 5c 80 90   ...:..U$ _?...\..
0030  00 e5 88 4b 00 00 01 01  08 0a 00 1c 54 17 00 1c   ...K.... ....T...

Frame (frame), 1514 bytes     Packets: 68776 · Displayed: 68776 (100.0%) · Load time: 0:0
```

Challenges and Course Correction

During the implementation stage of DCTCP, we initially started with Mininet and OpenVSwitch (OVS) as our choice of tools. While setting up Mininet and OVS on our setup wasn't really a challenge, the major difficulty was to enable DCTCP on Mininet hosts. Although Mininet hosts share the kernel with the host machine, it creates a separate network namespace for each of the hosts created so that each host can maintain its own set of interfaces and network stack. The network stack on Mininet hosts is supposed to be a replica of the host machine but somehow the Mininet hosts failed to provide an option to set the tcp congestion control. We tried to get answers from the Mininet mailing list but even that proved to be unfruitful. Hence, we have decided to emulate the Mininet environment manually without the Mininet Python API using *ip netns* command on linux which allows creation of separate network namespaces. But, even then, a similar problem as Mininet persists. Upon probing into the Mininet/linux documentation, we have realized that it is an undocumented behavior and hence we decided to shift to implementing DCTCP with 2 separate Virtualbox VMs and connecting them with an OVS.

Future Work

So far we have looked into the vanilla implementation of the DCTCP on a set of virtual machines and OVS. By checkpoint 2, we would like to analyze the effect of modifying the parameters that were kept constant in the original paper as proposed initially. We plan on generating following graphs:

1) g vs throughtput
2) n vs throughput
3) xDCTCP vs DCTCP vs TCP performance

Effectively we would like to summarize the project with improvements to the original implementation of DCTCP.