Introduction to Computer Networks: TCP Implementation Report

Xudong Wang / 2015012571

April 27, 2018

1 Set up the network topology

Setting up the network seems like an easy work, but I still meet some troubles here.

When I studied the example codes and tried to set up the required network, I thought that it can be done in a short time. The first problem is about the IPv4 address. When I first finished my network, I found that there is little flow among the hosts. I debugged it for hours until I read another code from the internet. I understood that *Ipv4AddressHelper.Setbase ()* must be called each time you assign IPv4 address to a NetDevice. The second problem is the config and attributes setting for each class. It also took me hours until I found the right way to set those required attributes, like *UseEcn* and *UseHardDrop* of *RedQueueDisc*. Then I learned to read the ns-3 model's library, where I can understand the right way to set up classes.

This work is easy and not challenging at all, but it is really time-comsuming. Luckily, with example codes provided by ns-3 and other users, I can finally complete it.

2 Implement DCTCP Protocol

The DCTCP[1] algorithm consists of three parts. I would like to outline these parts and explain how I implement the algorithm part by part. I commented "DCTCP" wherever I modified the code, so Ctrl + F can be used to jump to modified codes.

2.1 Simple Marking at the Switch

In brief, there is a marking threshold K. When the queue occupancy is greater than K, label the ECN field with CE, who is in the IP header. The author hint us that the RED marking scheme can help.

First, we need to set both the low and high thresholds to K. Thus I set both *minTh* and *maxTh* by 5. I tried several assignment of K, and I think 5 is a good assignment. Second, the algorithm require the RED marking to mark on instantaneous instead of average queue length. Then I slightly modified the *RecQueueDisc* class. When the average queue length variable(m_qAvg) is reassigned, I reassign the variable by the instantaneous queue length instead if *m_useDCTCP* is true. Lastly, from the model library, I found that RED marking works only when the ECN field has been set to ECT(0) or ECT(1). Thus I modified the *SendEmptyPacket()* and *SendDataPacket()* function of *TcpSocketBase*, and assured that all packets sent by these two functions have ECN field assigned by ECT(0).

2.2 ECN-Echo at the Receiver

The part says that a DCTCP receiver setting the ECN-Echo flag for ACK if and only if the received packethas a marked CE codepoint. Since delayed ACKs is adopted in TCP, the DCTCP receiver uses a trivial two state state-machine to set ECN-Echo bit.

First, I introduced a new boolean variable, *m_CEstate*, meaning the state of the state machine in Figure 10. I implement the state machine in the *ForwardUp()* function, since all packet received would be passed to this function. Then I modified *ReceivedData()* function, in order to labelled ACK with ECN-Echo flag when *m_CEstate* is true.

2.3 Controller at the Sender

The sender maintains α, an estimate of fraction of packets that are marked, and g, the weight given to new samples against the past in the estimation of α. Roughly, in each RTT, α is updated as follow:

$$\alpha \leftarrow (1-g) \times \alpha + g \times F,$$

where F is the fraction of packets that were marked in the last window of data. Moreover, while TCP always cuts its window size by a factor of 2 in response to a marked ACK, DCTCP uses α:

$$cwnd \leftarrow cwnd \times (1-\alpha/2).$$

First, I added corresponding variables to *TcpSocketState* class, who is the class to maintain the congestion control algorithm. α is initialized by 1.0, and the weight g is set to 0.125. In the function *EstimateRtt()*, I added some codes for updating α. The variables *m_markedAckCount* and *m_ackCount* is used to count the number of ECN-Echo-marked ACK packets and the total number of ACK packets in the *DoForwardUp* function, respectively. Besides, when α is updated, these two variables will be assigned by 0. When an ECN-Echo-marked ACK is received, its flags will be set to pure ACK, in order to correspond the orginal parts of the code.

There are still some modifications I have not mentioned above.

3 Design MYTCP (Improve TCP Hybla)

Compared to design a brand new congestion control algorithm, I choose to try improving existing congestion control algorithms. TCP Hybla[2] attracts me, because of its brevity and idea.

3.1 TCP Hybla Introduction

In brief, TCP Hybla introduced a new parameter, called "Referene RTT". The idea is simple. First evaluate the factor(called parameter ρ) by dividing actual RTT with Reference RTT. Then in each RTT, the algorithm acts like ρ RTTs has passed, and as if doing slow start and congestion avoidance for ρ times instead.

3.2 My Improvement

In the ns-3 implementation, when actual RTT gets smaller, the algorithm will update the factor ρ. However, if the actual RTT gets larger, the algorithm would not to update the factor ρ, which loses the correspondences. Thus I introduce a new parameter *m_mulThresh*, and modified the code in the *TcpHybla::PktsAcked()* function. Now actual RTT getting either smaller or larger sharply will cause the updating of ρ.

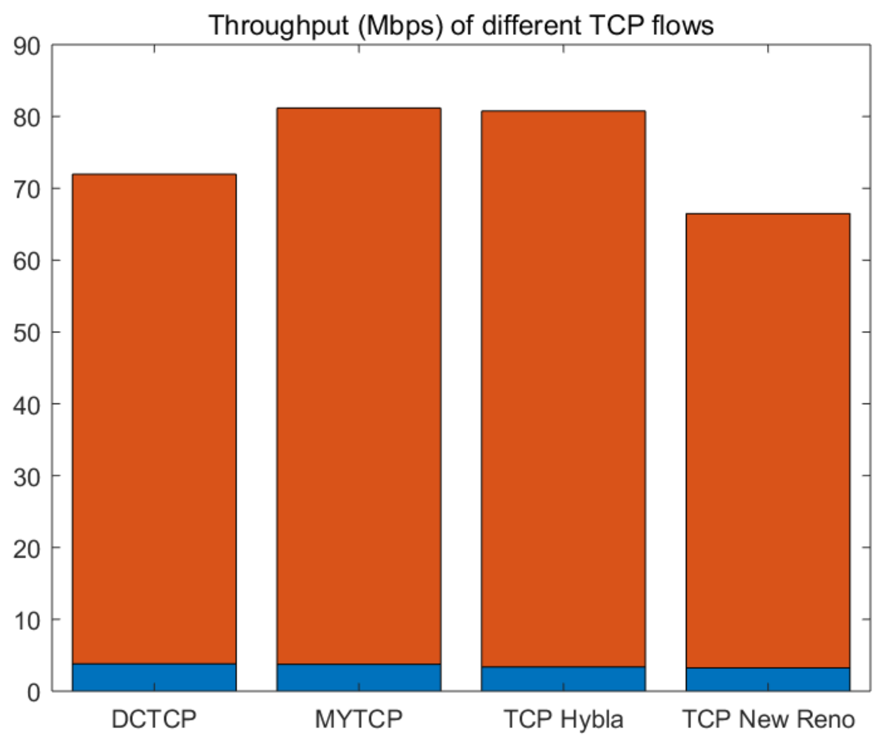
4 Outcomes and Conclusion

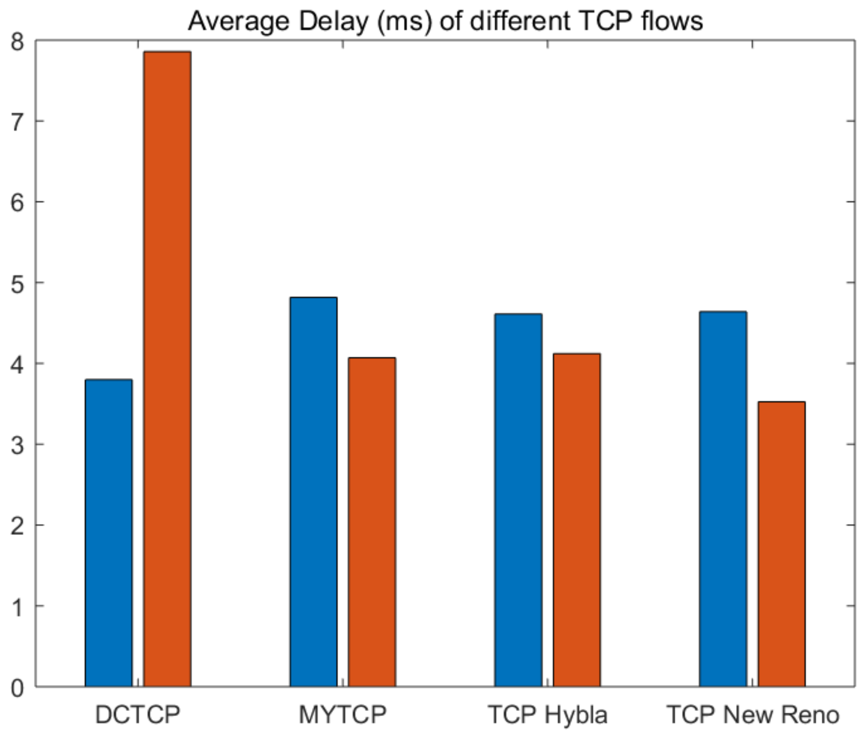
4.1 Flow Statistics Figure

	Throughput (Mbps)	Delay (ms)	Packets Received	Packets Loss
TCP New Reno A	3.245	4.641	694	185
TCP New Reno B	63.218	3.525	13454	44
DCTCP A	3.810	3.800	814	307
DCTCP B	68.153	7.858	15071	298
TCP Hybla A	3.363	4.611	719	185
TCP Hybla B	77.399	4.120	16468	44
MYTCP A	3.740	4.818	799	169
MYTCP B	77.438	4.070	16476	62

	Packets Marked	Flow Jitter Sum (ms)
TCP New Reno A	\	70.630
TCP New Reno B	\	694.810
DCTCP A	14	81.188
DCTCP B	92	710.506
TCP Hybla A	\	73.019
TCP Hybla B	\	829.987
MYTCP A	\	79.017
MYTCP B	\	830.309

4.2 Throughput and delay data of different TCP flows





4.3 Conclusion

- Implemented DCTCP has larger throughput than TCP New Reno.
- Implemented MYTCP has slightly larger throughput than TCP Hybla.

References

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[2] Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. *International Journal of Satellite Communications Networking*, 22(5):547–566, 2004.