ENSC 835: HIGH-PERFORMANCE NETWORKS

CMPT 885: SPECIAL TOPICS: HIGH-PERFORMANCE NETWORKS

TCP Fainess of CUBIC TCP In High-Speed NetWork

Spring 2006

FINAL PROJECT

Qing Chen

http://www.sfu.ca/~qingc/Index.htm

qingc@sfu.ca

List of Content

List of Content 2

Abstract 6

Abstract 6

I. Introduction 6

1.1 TCP congestion control 6

1.2 TCP problems in high speed long delay network 7

II. Advanced TCP Congestion Control Algorithms 8

2.1 BIC TCP 8

2.2 CUBIC TCP 9

2.3 TCP fairness 10

III. Simulation 11

3.1 Topology 11

3.2 Cases 11

IV. Simulation 1 and Analysis 13

4.1 Validation 13

4.2 TCP fairness of CUBIC in high speed network 14

4.3 Influence of queue type on fairness of CUBIC 15

4.4 Sub-conclusion 17

V. New approach based on RED to improve fairness 18

5.1 New idea based on RED 18

5.2 Differences from standard RED 21

5.3 Implementation 22

VI. Simulation 2 and Results 23

VII. Conclusion and Future Work 26

VIII. Reference 27

Appendix I: NS-2 Simulation Script (cubicnr_scr.tcl) 28

Appendix II: Codes for Implement New Approach 34

List of Figures and Tables

FIGURE 2. 1 WINDOW GROWTH CURVE OF BIC VS RTT [2] 8

FIGURE 2. 2 WINDOW GROWTH CURVE OF CUBIC VS RTT [2] 9

FIGURE 3. 1 NETWORK TOPOLOGY OF SIMULATION 11

FIGURE 4. 1 WINDOW GROWTH CURVE OF CUBIC AND NEW RENO 13

FIGURE 4. 2 THROUGHPUT OF CUBIC AND NEW RENO 13

TABLE 4. 1 LINK UTILIZATION WITH DIFFERENT BANDWIDTH AND QUEUE TYPE 14

FIGURE 4. 3 THROUGHPUT OF CUBIC AND NEW RENO 14

FIGURE 4. 4 THROUGHPUT OF CUBIC AND NEW RENO 14

TABLE 4. 2 FAIRNESS RATIO WITH BANDWIDTH OF 20MB/SEC AND 1 GB/SEC, QUEUE TYPE OF DROPTAIL AND RED 15

FIGURE 4. 5 COMPARE CONGESTION WINDOW OF NEW RENO 16

FIGURE 4. 6 COMPARE CONGESTION WINDOW OF NEW RENO 17

FIGURE 4. 7 COMPARE CONGESTION WINDOW OF NEW RENO 17

TABLE 4. 3 FAIRNESS RATIO WITH DIFFERENT START TIME AND QUEUE TYPE 17

FIGURE 5. 1 DROP PROBABILITY OF VARIANT ONE 19

FIGURE 5. 2 DROP PROBABILITY OF VARIANT TWO 20

FIGURE 5. 3 DROP PROBABILITY OF VARIANT THREE 20

FIGURE 5. 4 DROP PROBABILITY OF RED 21

FIGURE 6. 1 FAIRNESS RATIO WITH VARIANT 1,2,3 AND STANDARD RED 23

TABLE 6. 1 COMPARE LINK UTILIZATION AND FAIRNESS WITH VARIANT 1,2,3 AND

STANDARD RED 24

FIGURE 6. 2 CWND OF CUBIC WITH VARIANT 1,2,3 AND STANDARD RED.................. 25

FIGURE 6. 3 CWND OF NEW RENOWITH VARIANT 1,2,3 AND STANDARD RED 25

Abstract

With the usage of high speed network in the world, current TCP congestion control algorithm faces problems such as low link unitization, instability and etc. In order to solve these problems, some advanced TCP are brought forward. Since current TCP is widely used in the network, to be fair to current end users, the TCP fairness of a new design should be analyzed before it is put to use.

In this project, I will mainly discuss CUBIC TCP. After a brief introduction of CUBIC TCP in Session I, I will describe the simple topology that I used to simulate a two-flow network in ns-2. I will analyze the fairness performance of CUBIC TCP, as well compare link utilization and throughput in different cases. TCP fairness performance of CUBIC with RED and Drop Tail is compared. As a conclusion of my project, TCP unfairness of CUBIC TCP will exist in high speed network if without any other help such as queuing management. I will provide an approach based on RED queuing management to improve the fairness performance. I also implement three variants of my approach in a simplified way and compared the results.

I. Introduction

The network is now becoming more and more important in our society. In order to satisfy the need of huge data transportation volume, GB networks are put in use. For example, Energy Science Net (ESnet) has gigabyte link backbone which cover the entire and United States and links to other countries. Because of the physical media the network used and long distance they cover, the propagation delay is very long.

1.1 TCP congestion control

TCP is a data transmission protocol which is widely used in current network. It provides congestion control in the distributed network system for reliable data transmission. In TCP New Reno, the congestion control has four phases: slow start, congestion avoidance, fast retransmit, and fast recovery. The sender starts from slow start phase, in which the window size will increase exponentially. After the window size is greater than a threshold, it will enter congestion avoidance phase, where the window size increases linearly. If the sender receives triple (or more) duplicated ACKs, it will change the threshold to half of the window size, set the congestion window to threshold, and then go into fast retransmit and fast recovery phases. After exit the fast recovery phase, it will repeat congestion avoidance phase to probe the valid link capability. If a timeout happens, it will set the threshold to half of winder size and congestion window to 1 maximum segment size and enter slow start phase.

1.2 TCP problems in high speed long delay network

Due to slow increasing speed of congestion window in TCP, it will face some big problems in high speed long delay network.

Inefficient link utilization

Because the standard TCP cut the windows in half after a loss event and increase the window size by one packet every round trip time (RTT) in congestion avoidance stage, the slow increment of congestion window prevents TCP from complete usage of the spare link capability available in high speed network. The link utilization situation will be worse if a synchronized loss happens with multi TCP flows.

At the meantime, the long propagation delay makes window growth speed even much slower. It will keeps the link utilization at low lever most of the time.

Oscillation

From equilibrium condition of TCP Reno $p_i^* = \frac{3}{2w_i^{*2}}$ [5], the loss probability p_i^* is inversely proportional to the window size's square w_i^{*2}. But the window size can be very large in high speed network and hence the loss probability must be extremely small. Practically, it is impossible. Therefore oscillation of current TCP is unavoidable in high speed network.

II. Advanced TCP Congestion Control Algorithms

In order to solve these problems, many new approaches are brought forward. The most commonly discussed include HSTCP, STCP, H-TCP, Fast TCP, BIC TCP and CUBIC TCP, XCP, and SABUL. In this project, I will mainly discuss CUBIC TCP. Since CUBIC TCP is an improved version of BIC TCP, it is necessary to briefly introduce BIC TCP first.

2.1 BIC TCP

BIC TCP stands for binary increase control TCP. It uses a binary search algorithm to adjust the congestion window. It will start from TCP slow start. When a loss is detected, it will use multiplicative decrease as standard TCP and set the windows just before and after loss event as $cwnd_{\min}$ and $cwnd_{\max}$, and enter binary search phase immediately. In binary search phase, it will first set the middle point between $cwnd_{\max}$ and $cwnd_{\min}$ as its target window and set $cwnd$ equal to $cwnd_{\min}$. If the distance between target and $cwnd$ is larger than $S_{\max}$ (maximum increment), the window will increase linearly by $S_{\max}$ every RTT. Otherwise, the window will increase to target the next step. Once it reaches target, it will set $cwnd_{\min}$ equal to target window size, $cwnd$ equal to $cwnd_{\min}$, and calculate the new mid-point again, make that point as new target and repeat the search. This process will continue till distance between new $cwnd_{\min}$ and $cwnd_{\max}$ is less than $S_{\min}$. Until Then, BIC will start another slow start phase to probe new maximum point. The curve of window vs. RTT when close to $cwnd_{\max}$ is shown in Figure 2.1

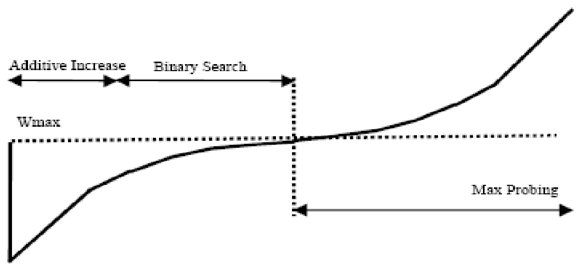

Figure 2. 1 Window Growth Curve of BIC vs RTT [2]

In congestion control procedure, the increment of $cwnd$ will be above 1 most of the time. It will become smaller and smaller when the window size is closer to the previous loss-happening point $cwnd_{\max}$.

2.2 CUBIC TCP

CUBIC TCP is designed on the basis of BIC TCP because the window control algorithm of BIC is too complicated, and also could be too aggressive in slow network with short round trip time. In CUBIC, they use a function of elapsed time to calculate the congestion window every RTT. The window calculation function is [2]:

Ack:

$$cwnd \leftarrow C(t-K)^3 + cwnd_{\max}$$

$$K = \sqrt[3]{cwnd_{\max}\beta / C}$$

Loss: $cwnd \leftarrow \beta \times cwnd_{\max}$

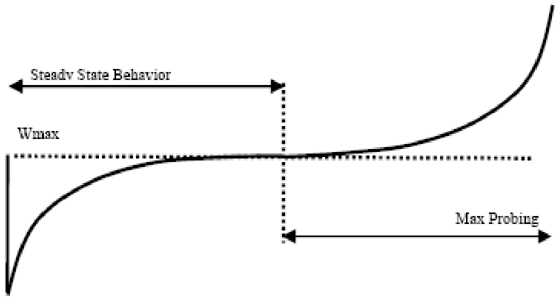

Figure 2. 2 Window Growth Curve of CUBIC vs RTT [2]

where C is a scaling factor, t is the time elapsed since last loss event, $cwnd_{\max}$ is the window just before the last loss event happens, β is a constant decrease factor. The shape of function curve vs. RTT (Figure 2.2) is similar to BIC TCP.

CUBIC TCP inherits the fast growing pattern of BIC TCP. But the window increment could become too large during target searching period with time elapsing. To limit the increment of window, a parameter $S_{\max}$ is also included. So the window will increase linearly by $S_{\max}$ if the window calculated from function is too big. But instead of maximum increment *each RTT*, $S_{\max}$ is the maximum increment *per second.*

CUBIC also has “low utilization detection” and “New TCP mode” [2].

2.3 TCP fairness

TCP fairness means if several flows compete one bottleneck link, each flow should share equal portion of bandwidth. However in the real world, TCP fairness is difficult to achieve. In TCP, the throughput in term of loss probability and round trip time is $\frac{1.22 \times MSS}{RTT\sqrt{L}}$ [9]. If sharing the same loss rate L, the flow with short round trip time will take more bandwidth, which results in unfairness. In high speed network, the round trip time will varies greatly, and therefore the unfairness will be much worse.

In CUBIC, since real time is used for calculating the congestion window, this will make CUBIC flows friendlier with each other in pure CUBIC world. But this will worsen the fairness in network if both CUBIC users and TCP users exist. If a CUBIC user and a TCP user has the same long round trip time, since long RTT doesn’t influence the calculation of CUBIC congestion window, CUBIC window size will increase much faster than TCP window. So, there will be serious TCP fairness problem with CUBIC if flows have same long round trip time. There are some other factors that influence TCP fairness performance, such as bandwidth and queue type. These factors will be discussed in next session.

To analyze TCP fairness, fairness ratio is used in this report for comparing fairness performance of CUBIC TCP in different situation. The fairness ratio (FR) in this report is defined as:

$$FR = \frac{Throughput_{TCP}}{Throughput_{CUBIC}} \qquad (1)$$

So, the bigger the FR, the better the TCP fairness of an approach.

III. Simulation

To analyze TCP fairness performance of CUBIC, I used ns-2.26 with CYNWIN in Windows XP. The CUBIC congestion control code was downloaded from http://www.csc.ncsu.edu/faculty/ rhee/export/bitcp/ and then emerged into ns-2.26 by replacing original tcp file.

3.1 Topology

The simple topology of my simulation has two flows. One is S1 → N1 → N2 → R1, and the other is S2 → N1 → N2 → R2. Sender S1 uses CUBIC and S2 uses New Reno. The traffic sources of two senders are both FTP. The receivers are TCP/sink on another side. Both flows share the link between N1 and N2. The propagation delay of bottleneck link between N1 and N2 is 50ms, and 1ms for all other links. Therefore, both flows have same round trip time – about 104ms if ignoring the other small delays. The bandwidth of each links is the same. Therefore, the total of flow 1 and flow 2 will be twice the capacity of bottle link between N1 and N2, which results in congestion and packet loss. Two flow monitors are used in order to get bdepature_ information. Figure 3.1 shows the topology that will be used in simulation.

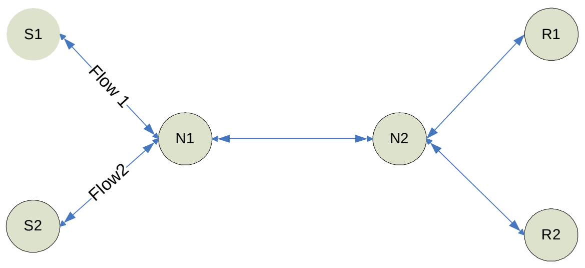

Figure 3. 1 Network Topology of Simulation

In order to compare the performance between CUBIC and New Reno, I will observe cwnd_ and bdepartures_ vaiables. cwnd_ of two senders will be recorded every 0.1 second. Throughput will be calculated from bdepartures_ and recorded every one second.

3.2 Cases

The simulation of the project can be separated into two parts. The first part focuses on TCP fairness of CUBIC and second part focuses on comparing the TCP fairness performance of CUBIC with three variants of improved RED and standard RED.

The first part consists of three cases, validation, influence of bandwidth, and influence of queue type. In first case, only one flow was enabled. In the second case, I will set the bandwidth of all links to 20Mb/sec or 1 Gb/sec to see the influence of different bottleneck bandwidth on fairness of CUBIC. The last one of this part is to observe the effect of queue management on fairness performance of CUBIC. Two queue types used in simulation are DropTail and RED. There are some other simulations with different parameter settings after that in order to verify the analysis. In all simulation, two flows will stop at the same time.

The second part of simulation is to observe TCP fairness performance of CUBIC with the help of my new approach. It consists of three cases for three variants of my approach. The bottleneck link bandwidth is set to 1 Gb/sec.

IV. Simulation 1 and Analysis

4.1 Validation

All validation simulation ran 400 seconds. I mainly used 20Mb/sec for all links instead of 1 Gb/sec in order to observe the CUBIC window growth pattern. If we choose 1 Gb/sec bandwidth, the windows size after loss will decrease a relatively big amount and therefore the link will have a big spare capability available. Because the fast increase of CUBIC and its maximum window increment restriction, the window will increase linearly by maximum window increment every RTT shortly after loss. This will results a saw shaped curve for CUBIC rather than the pattern shown in Figure 2.2.

After validation simulation, the congestion window growth curves and throughput of New Reno and CUBIC was plotted and shown in Figure 4.1 and Figure 4.2

From Figure 4.1, we can see the typical saw shaped New Reno window growth pattern as well as the quick changed CUBIC window growth pattern. Although the congestion window size of CUBIC also changes, its peak-to-peak change of window size is much smaller than that of New Reno. The average window size of New Reno congestion window is 169.2 packets, which is smaller than that of CUBIC (202.7 packets). With respect to throughput, oscillation happened with New Reno flow, but the change of CUBIC's throughput is so small that it shows a straight line shortly after start, (Figure 4.2). To compare the link utilization in high speed network, a quick simulation with 1 Gb/sec bandwidth is run. Table 4.1 shows the link utilization of New Reno and CUBIC when at 20Mb/sec and 1 Gb/sec. We can see that the link utilization of New Reno degrades from 89.44% to 50.10%. Comparatively, CUBIC just change

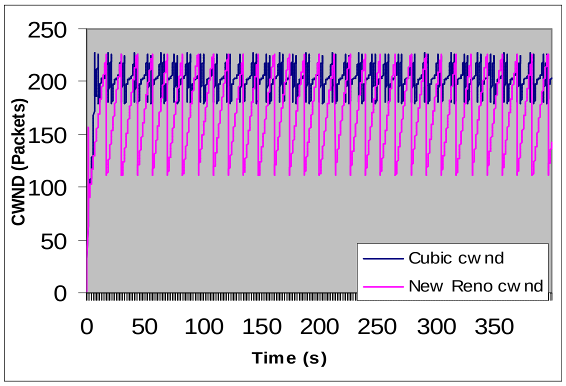

Figure 4. 1 Window Growth Curve of CUBIC and New Reno

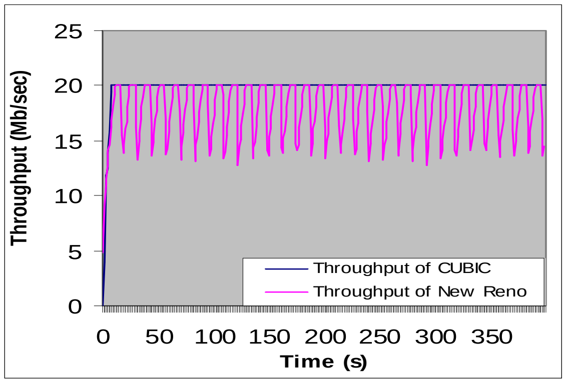

Figure 4. 2 Throughput of CUBIC and New Reno

from 99.36% to 94.62%. The link utilization of CUBIC is much higher than that of New Reno in high speed network.

Bandwidth	New Reno	CUBIC
20Mb/sec	89.44%	99.36%
1Gb/sec	50.10%	94.62%

Table 4. 1 Link Utilization with Different Bandwidth and Queue Type

Therefore, CUBIC has very good link utilization and is very stable. But CUBIC will have unfairness problem in network environment where both CUBIC and New Reno exist at the same time.

4.2 TCP fairness of CUBIC in high speed network

The purpose of this simulation is to observe the fairness performance of CUBIC in high speed network. In the simulation, the bandwidth of bottleneck link is 1 Gb/sec. both flows start at the same time.

Figure 4.3 and Figure 4.4 show that the throughout of CUBIC is much higher than that of New Reno after 70 seconds of simulation no matter in DropTail or RED queuing management, presenting serious unfairness of CUBIC. The average fairness ratio with DropTail is 0.142 and that with RED is 0.085 as listed in Table 4.2. Both are very low.

To compare the fairness of CUBIC with smaller bottleneck bandwidth, I changed the bandwidth to 20Mb/sec and ran another simulation. Table 4.2 compares the average fairness ratio of each case. We can see that the fairness ratio of CUBIC with 20Mb/sec bottleneck link is much better than that with 1Gb/sec bandwidth, no matter which kind of queuing mechanism is used.

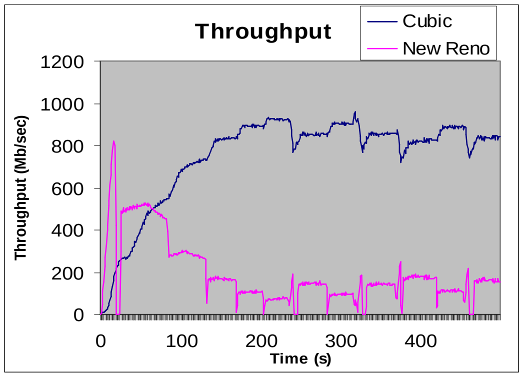

Figure 4. 3 Throughput of CUBIC and New Reno With Drop Tail

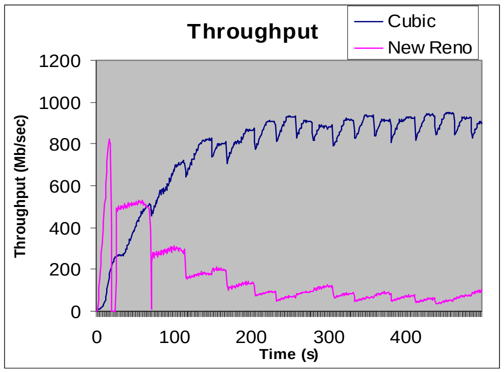

Figure 4. 4 Throughput of CUBIC and New Reno With RED

	DropTail Queue	RED Queue
20 Mb/sec bottleneck link	0.404	0.688
1 Gb/sec bottleneck link	0.142	0.085

Table 4. 2 Fairness Ratio with Bandwidth of 20Mb/sec and 1 Gb/sec, Queue Type of DropTail and RED

The decrease of fairness performance is resulted from CUBIC congestion control algorithm. In CUBIC, if a sender detects a loss, it will change the window size to 0.8 of its previous size instead of half in New Reno. On the other hand, the window increasing rate of CUBIC is based on a cubic function which related to the elapsed time, which is much faster than New Reno’s congestion avoidance algorithm. Also, the larger the bandwidth, the less the congestion happens. Therefore, comparing that in slow network, the CUBIC sender will have longer time to increase its window size at a fast increasing rate in high speed network before encountering a new loss event. So, the distance of window size between CUBIC and New Reno will be increased much larger in a high speed network than that in slow network. As a result, the fairness of CUBIC is worse in high speed network.

4.3 Influence of queue type on fairness of CUBIC

From above simulation, we can see that queue type may affect the fairness of CUBIC. In this part, we will observe and analyze the influence of DropTail and RED on fairness of CUBIC.

Theoretically, the TCP fairness with RED queue should be better than that with DropTail queue. In RED, there are three major parameters: $q_{\min}$, $q_{\max}$ and $p_{\max}$. If the packets in queue is less than $q_{\min}$, there is no packet drop. If packets in queue is more than $q_{\min}$ but less than $q_{\max}$, the arrival packets will be dropped based on a probability of $\frac{p_{\max}}{q_{\max} - q_{\min}} \times (q - q_{\min})$. If more than $q_{\max}$, all arrival packets will be dropped. As a result, the flow with higher sending rate will have higher drop probability. In other words, the flow will suffer more loss events and hence slow down its sending rate. To slow flow, the drop probability is smaller. Therefore the slow flow will has less loss events and less chance to decrease its windows size than fast flow. DropTail handles buffer by receiving all packets if there is space in buffer but dropping all new received packets when buffer is full. Therefore, the number of packets of a certain flow in buffer is decided by its sending rate. The higher the sending rate, the more the packets in buffer and later the more packets will be transmitted by bottle link. Therefore, RED is more fair than DropTail.

From Table 4.2, we find that, with 20Mb/sec bottleneck link, fairness of CUBIC with RED is better than that of DropTail as expected. But with 1 Gb/sec, fairness of CUBIC with RED is worse than that of DropTail. This is because the maximum window increment of CUBIC is 16. But the New Reno maximum window increment observed in simulation is 50. From Figure 4.5, we can find that, the New Reno flow with RED didn't have serious congestion. So, the windows size just halved its previous size then increased slowly by 1 packet each RTT after loss. With DropTail, the system suffered heavy congestion which causes New Reno into slow start mode. Since the maximum window increment of New Reno is 50 and that of CUBIC is set to 16, the New Reno has a chance to increase its congestion window faster than CUBIC and grow to a relatively large size and take most of bottleneck bandwidth valid after the previous serious loss before a new loss event happens. Even halved the congestion window after a new loss, the New Reno's congestion windows with DropTail queue is still comparatively larger than that with RED most of time, as shown in Figure 4.5. Therefore the throughput of New Reno in DropTail case is bigger then that in RED case. Based on equation (1), the fairness ratio is proportional to the throughput of New Reno. So, the fairness ratio in DropTail case is larger than that RED case in this simulation.

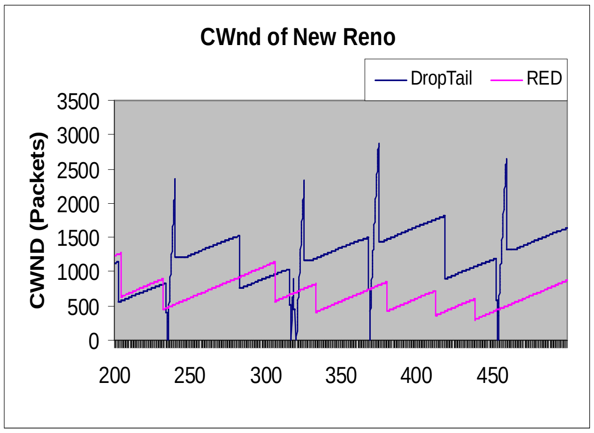

Figure 4. 5 Compare Congestion Window of New Reno With DropTail and RED in 1 Gb/sec Bottleneck Link

In order to verify the analysis, I set the maximum window increment of CUBIC to 50 packets so that New Reno flow has less chance to take away most of valid bottleneck bandwidth after a serious loss. After the verification simulation, the New Reno congestion window growth curve was plotted as Figure 4.6. The congestion window of New Reno with RED is sometimes larger than that with DropTail and sometimes less. The New Reno congestion window curves of DropTail case with different CUBIC' max. window setting are compared in Figure 4.7. This time, New Reno window, after setting CUBIC's maximum window to 50, almost had no chance to increase as large as before and was much less than that when CUBIC's max window was 16. The average fairness ratio with DropTail was now 0.045 and average fairness ratio with RED was 0.065, which proved my analysis and matched the theoretically analysis result as well.

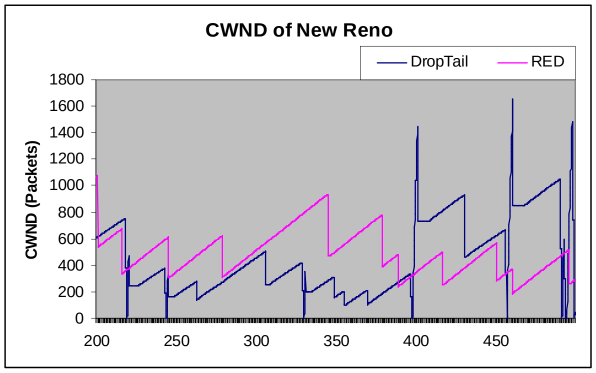

Figure 4. 6 Compare Congestion Window of New Reno With DropTail and RED in 1 Gb/sec Bottleneck link Max. Window Increment of CUBIC =50 Packets

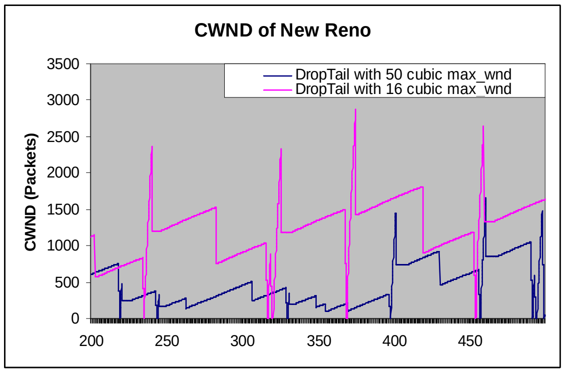

Figure 4. 7 Compare Congestion Window of New Reno with DropTail, When Max. Window of CUBIC =16 packets and =50 packets

Different Start Time	AVG FR with DropTail	AVG FR with RED
CUBIC flow starts first. After 50 seconds, New Reno flow join in	0.07	0.11
New Reno flow starts first. After 50 seconds, New Reno flow join in	0.052	0.08

Table 4. 3 Fairness Ratio with Different Start Time and Queue Type

Some other simulations were also run with 1 Gb/sec bottleneck link but with different start time. The average fairness ratio of each case is listed in Table 4.3. The results also show that the fairness of CUBIC with RED is better than that with DropTail.

4.4 Sub-conclusion

From above simulation analysis, the TCP fairness of CUBIC will be a problem in high speed network. It will exist unless slowing down its congestion window increasing speed and enlarging its decreasing factor. If so, CUBIC will loose its advantage in high speed network when comparing with TCP New Reno. In order to improve the fairness performance of CUBIC, we may use queuing management for help.

V. New approach based on RED to improve fairness

As we see from the simulation and analysis, the queuing management can affect the congestion control by dropping packets. The basic idea of my design is to “punish” the fast flow by dropping more packets. This improvement is based on current RED queue in ns-2.

5.1 New idea based on RED

My new idea is to adjust the drop probability by a factor based on flow id. If the queue finds that there are packets belong to different flows in queue when it receives a packet, it will increase the drop probability if there are a lot of packets that belong to the same flow of new arrival packet already in queue, and it will decrease the drop probability otherwise. If the queue finds all the packets in queue belong to the same sender of new arrival packet, it will act as standard RED. The queue will have a monitor which maintains a table that records the flow id and count relative packets in the queue. If the counter of a certain flow is zero, the queue will remove that flow id from the table. The detailed algorithm will be:

```
When packet arrives:
    if  (total number of packets >= $q_{\max}$)
        packet drop;
    else
    {
        if  (total number of packets < $q_{\min}$)
            no packet drop;
        else
        {
            if (flow id of new packet not find in table)
            {
                add new flow id into table;
                count(i)++;          //assume index of flow id in table is i
            }
            else
            {
                find index of the flow id in table;
                count(i)++;                //assume index of this flow id is I
            }
            calculate drop probability $p$ based on standard RED;
            if (number of flow id in table is more than 1)
            {
                calculate weight factor $wf$ ;   //three ways described later in report
                modify drop probability $p = wf \times p$ ; //should no more than 1
            }
            decide drop or keep based on adjusted $p$
            if a drop
            {
                count(i)--;
                if count(i) =0
                    remove flow id from table;
            }
        }
    }
When a packet is transmitted
    count(i)--;
    if count(i)=0
        remove flow id from table;              //cleanup table
```

There are three methods to calculate the factor. These variants are:

1. Calculate wf by

   $wf = count_i / count_{\max}$, where

   $count_{\max}$ is the maximum count in the table and i is the index of the flow id of new arrival packet in table. With the change of the counted packet number of certain flow, the drop probability will change in the range shown as

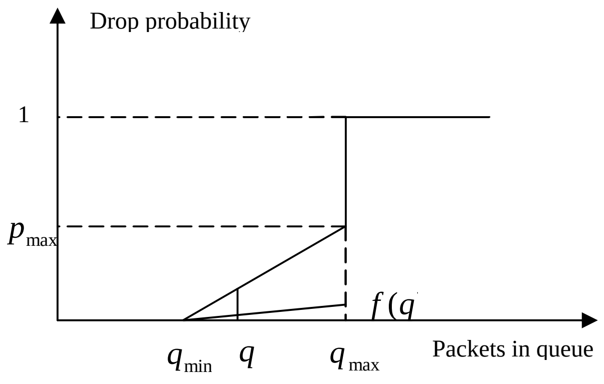

Figure 5. 1 Drop Probability of Variant One

Figure 5.1. The upper bound function is the same as standard RED drop probability function. The lower bound function is easy to get as follow:

$$f(q) = p_{\max} \times \frac{q - q_{\min}}{(q_{\max} - q_{\min}) \times (q-1)} \quad \text{where} \quad q \in [q_{\min} + 1, q_{\max}]$$

2. Calculate wf by

   $wf = count_i / count_{\min}$, where $count_{\min}$ is the minimum count in the table and i is the index of the flow id of new arrival packet in table. With the change of the counted packet number of a certain flow, the drop probability will change in the range shown as Figure 5.2. The lower bound function is the same as standard RED drop probability function. It is easy to get upper bound function:

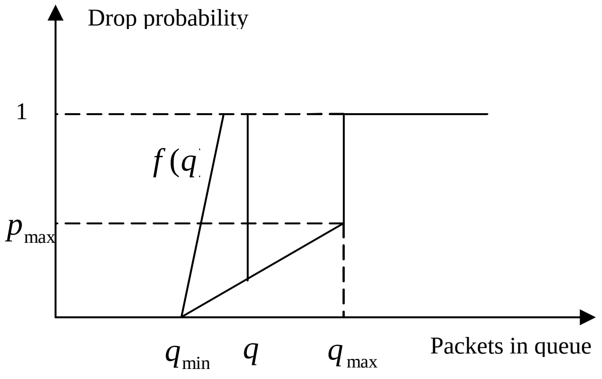

Figure 5. 2 Drop Probability of Variant Two

$$f(q) = \min\left\{ p_{\max} \times \frac{(q - q_{\min}) \times (q-1)}{(q_{\max} - q_{\min})}, 1 \right\} \quad \text{where} \quad q \in [q_{\min} + 1, q_{\max}]$$

3. Calculate wf by

   $wf = n \times count_i \Big/ \sum_{j=1}^{n} count_j$, where i is the index of the flow id of new arrival packet in table, and there are n flow id in the table. With the change of the counted packet number of a certain flow, the drop probability will change in the range shown as Figure 5.3. The lower bound and upper bound function will change based on number of flows in the table. If there are n flow packets in queue, the upper and lower bound functions are:

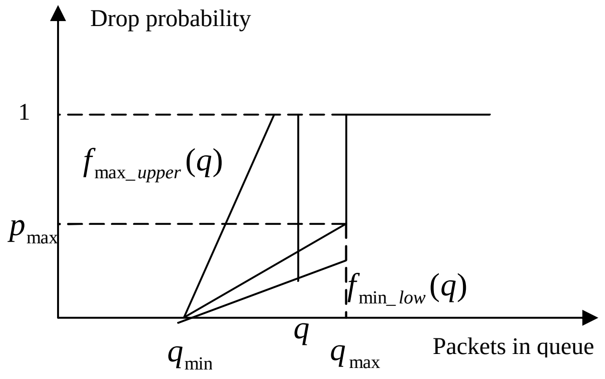

Figure 5. 3 Drop Probability of Variant Three

Lower Bound: $$f_{low}(q) = p_{\max} \times \frac{(q - q_{\min})n}{(q_{\max} - q_{\min}) \times q}$$

Upper Bound: $$f_{upper}(q) = \min\left\{ p_{\max} \times \frac{(q - n + 1) \ \times n \times (q - q_{\min})}{(q_{\max} - q_{\min}) \times q}, 1 \right\},$$

$$\text{where } q \in [q_{\min} + 1, q_{\max}]$$

It is easy to derive the minimum low bound and maximum upper bound from above equations.

Min Lower Bound: $$f_{\min\_low}(q) = p_{\max} \times \frac{2(q - q_{\min})}{(q_{\max} - q_{\min}) \times q}$$

Max Upper Bound: $$f_{\max\_upper}(q) = \min\left\{ p_{\max} \times \frac{(q+1)^2 (q - q_{\min})}{4(q_{\max} - q_{\min}) \times q}, 1 \right\},$$

$$\text{where } q \in [q_{\min} + 1, q_{\max}]$$

5.2 Differences from standard RED

In RED, the drop probability change with the change of packets accumulated as shown in Figure 5.4. At the certain packet number $q, q \in (q_{\min}, q_{\max}]$, there is a certain drop probability p. When next packet arrives, no matter it is sent by slow sender or fast sender, it might be dropped based on this probability p. But in my approach, if the packet belongs to a flow that has fewer packets in queue, the factor will be very small and its modified drop probability will be much smaller than that of standard RED. Therefore, that packet will be more likely to be kept. On another hand, the flow with more packets in queue will have comparatively more loss events which will trigger the congestion control algorithm to pull down the window size and slow down the sending rate.

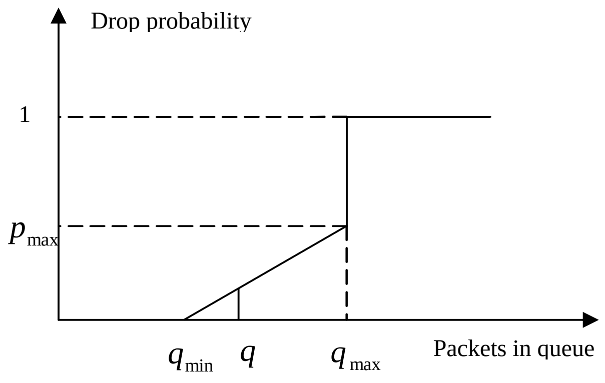

Figure 5. 4 Drop probability of RED

With the change of packet ratio with the packets of a certain flow accumulated up in buffer, its loss probability may change within the bounded area.

5.3 Implementation

In order to quickly analyze the results of my new approach, I simplify the algorithm so that I can quickly implement it into standard RED in ns-2. Because the simple topology in this project has only two flows and the flow id is defined, I will use CNT1 and CNT2 to count packets of these two flows that in queue. Another two variables are used to store maximum count (CNTmax) and minimum count (CNTmin). After simplification, the rules of my approach become:

1. Increment when enter queue, decrement when drop or transmitted. Update CNTmax, CNTmin

2. Calculate weighted loss probability
   For Variant 1:
   if flow1 packet received and CNT2 !=0, P=P*CNT1/CNTmax //P is RED drop //probability
   if flow2 packet received and CNT1 !=0, P=P*CNT2/CNTmax
   Otherwise, P is not changed

   For Variant 2:
   If flow1 packet received and CNT2 !=0, P=P*CNT1/CNTmin
   if flow2 packet received and CNT1 !=0, P=P*CNT2/CNTmin
   Otherwise, P is not changed

   For Variant 3:
   If flow1 packet received and CNT2 !=0, P=P*CNT1*2/(CNT1+CNT2)
   if flow2 packet received and CNT2 !=0, P=P*CNT2*2/(CNT1+CNT2)
   Otherwise, P is not changed

The codes will be merged into enque(), deque() and drop_early() function in red.cc and some small adjustments in red.h. The implementation codes are listed in Appendix II

VI. Simulation 2 and Results

I used the same topology of simulation1 with 1Gb/sec bottleneck link to simulate and compare the fairness performance of CUBIC with three different variants as well as standard RED in high speed network. The fairness ratio of CUBIC is plotted in Figure 6.1.

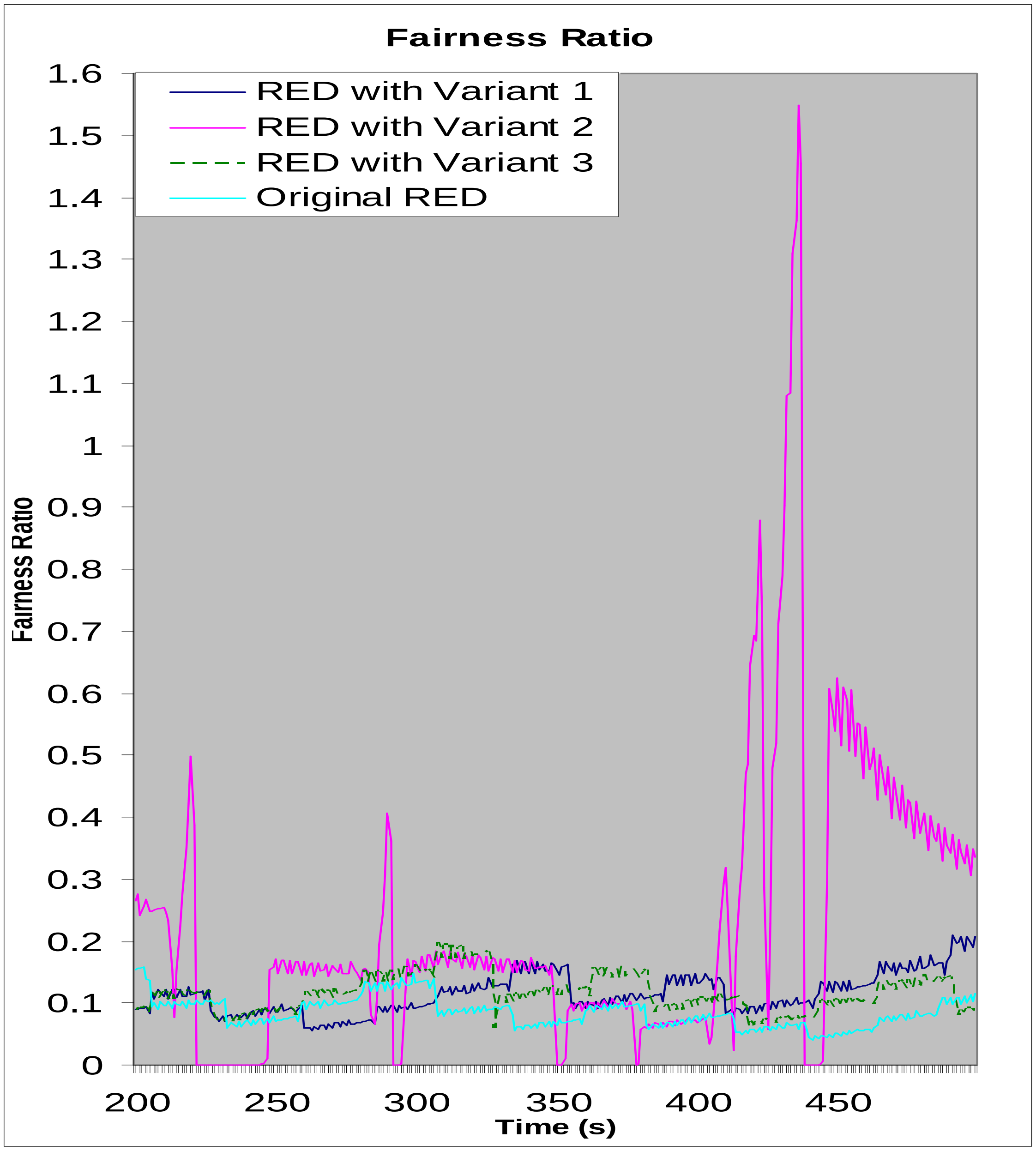

Figure 6. 1 Fairness Ratio With Variant 1,2,3 and Standard RED

From Figure 6.1 , we can see that the fairness ratio curves of CUBIC with Variant 1 and Variant 3 and standard RED are close to each other. With Variant 1 and Variant 3, the fairness ration of CUBIC is a bit higher than that with standard RED most of time. The average fairness ratio with Variant 1 is 0.115, with Variant 3 is 0.116, a bit higher than that of standard RED (0.085). So, RED with Variant 1 and 3 can help CUBIC to improve fairness by 0.03 in fairness ratio.

The effect of Variant 2 on TCP fairness of CUBIC is stronger than other variants. The curve shows that the RED with Variant 2 “punished” CUBIC too much so that the fairness ration goes beyond 1, maximum ratio up to 1.54 during 200sec to 500sec period. This is because, from drop probability range shown in Figure 5.2 , the drop probability of fast flow (here is the CUBIC flow) might keep at 1 if there are huge amount of fast flow packets in queue but only a few of slow flow packets in queue. When it happens, all packets from CUBIC will be dropped. But, at that time, the queue buffer is not full. Therefore, packets from New Reno can still be stacked into buffer at a probability of (1-standard RED drop probability). Therefore, the fairness ratio gets improved. The average fairness ratio of CUBIC with Variant 2 is 0.228. Table 6.1lists the fairness of CUBIC with different RED queuing management.

In Table 6.1, the link utilization of different cases is compared as well in order to see if the improved RED will result in degrade of link utilization in the network where both CUBIC flow and New Reno exists. The over all percentage values of average link utilization with standard RED, RED with Variant 1 are very close. With Variant 3, the link utilization decrease a little. With Variant 3, though the fairness of CUBIC increases, the over all link utilization degrades greatly as a trade.

	1Gb/sec bandwidth	
	Link Utilization	Fairness
Standard RED	96.8%	0.085
RED with Variant 1	97.2%	0.115
RED with Variant 2	79.2%	0.228
RED with Variant 3	92.6%	0.116

Table 6. 1 Compare Link Utilization and Fairness with Variant 1,2,3 and Standard RED

Figure 6.2 is the congestion window curves of CUBIC with three RED variants and standard RED. We can see that the congestion window curves of Variant 1, 3 and standard RED are close to each other. Only that of Variant 2 differs from others greatly. This is because, in first variant, the drop probability of fast flow, which is CUBIC flow, will follow the standard RED drop probability function most of the time. In second variant, since there are only 2 flows and the default maximum drop probability of standard RED is 0.1., the maximum possible drop probability is about 0.2 and minimum possible drop probability is around 0, which is also close to standard RED drop probability. So, the loss events, which driven by the drop probability of RED, will be similar. Therefore, these three curves are close to each other. The second variant is quite different. In two-flow case here, the fast CUBIC flow' drop probability will change in a bigger range from standard RED drop probability to 1. Therefore, the congestion window of CUBIC will change differently as shown in graph.

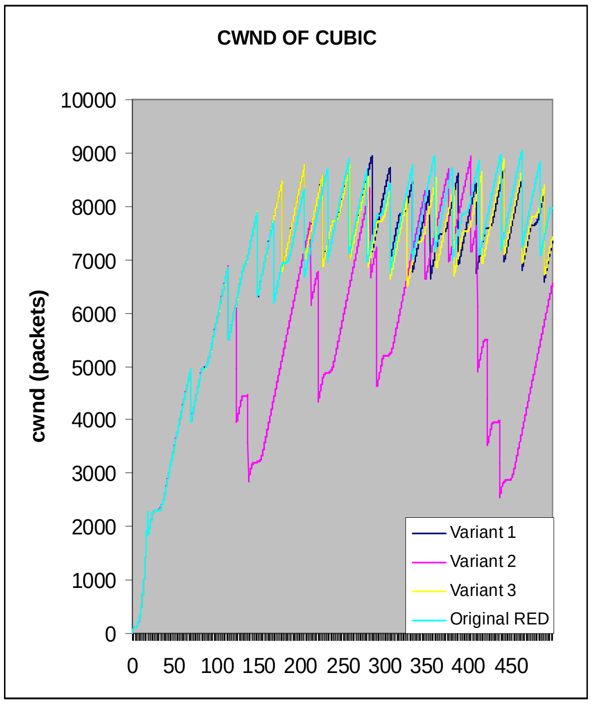

Figure 6. 2 CWND of CUBIC With Variant 1,2,3 and Standard RED

Figure 6.3 is the congestion window curves of New Reno with three variants and standard RED. Because of the similar reason, the congestion window curve of Variant 1, Variant 3 and standard RED are close together. With Variant 2, since the window size of CUBIC changes greatly, New Reno has chance to grow its window size larger than that of CUBIC and therefore sometimes do not follow the standard RED drop probability function. But most of the time, the slow New Reno flow will follow the standard RED drop probability curve because usually it has less packets in queue.

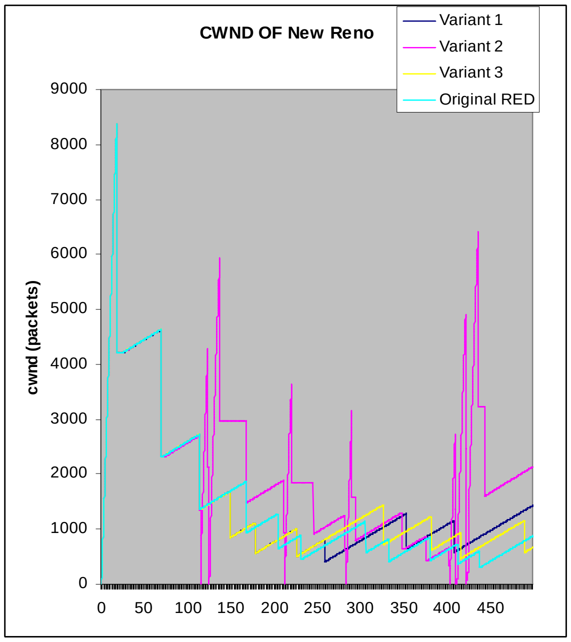

Figure 6. 3 CWND of New RenoWith Variant 1,2,3 and Standard RED

VII. Conclusion and Future Work

From the simulation and analysis, we can see that the performance of CUBIC is better than that of TCP New Reno in oscillation and link utility especially in high speed network. However, it also have serious unfairness problem when both CUBIC and New Reno exist in network. The higher the bottleneck bandwidth, the fairness performance of CUBIC is the worse.

Fairness of CUBIC can be improved by using proper queuing management. In simulation, the fairness of CUBIC with RED is a little bit better than that with DropTail if the right parameter combination is used.

Based on RED, I designed an approach to improve the fairness performance of CUBIC with the help of queuing management. The simulations show that the fairness performance is improved after including the new approach into RED, especially Variant 2. But the link utilization of Variant 2 decreases much larger. With new approach, the congestion window curve with Variant one, Variant three and standard RED will not change greatly. With Variant two, the congestion window size curve differs greatly from that with standard RED. The reasons of these patterns are analysis in the report.

In order to properly evaluate the performance of CUBIC TCP, it is necessary to do more simulations using various combinations of parameters such as bandwidth of bottleneck link, with or without Improve RED, buffer size, TCP settings and RED configuration. The performance of new approach also needs more tests to verify its influence on TCP fairness in high speed network in future.

VIII. Reference

[1] Lisong Xu, Khaled Harfoush, and Injong Rhee, “Binary Increase Congestion Control for Fast, Long Distance Networks”, In Proceedings of the IEEE INFOCOM, March 2004

[2] Injong Rhee, and Lisong Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant”, PFLDnet 2005, February 2005

[3] D.J. Leith, R.N. Shorten, Y.Lee, “H-TCP: A framework for congestion control in high-speed and long-distance networks”, HI Technical Report, August 2005, http://www.hamilton.ie/net/htcp/

[4] D. Leith, R. Shorten, “H-TCP: TCP for high-speed and long-distance networks”, Second International Workshop on Protocols for Fast Long-Distance Networks, February 16-17, 2004, Argonne, Illinois USA

[5] Cheng Jin, David X. Wei and Steven H, “FAST TCP: motivation, architecture, algorithm, performance”. In Proceedings of IEEE INFOCOM, March 2004 http://netlab.caltech.edu

[6] Junsoo Lee, Stephan Bohacek, Joao P. Hespanha, Katia Obraczka, “A Study of TCP Fairness in High-Speed Networks”, submitted to ICNP, 2005

[7] Dina Katabi, Mark Handley, Charlie Rohrs, “Congestion Control for High Bandwidth-Delay Product Network”, ACM SIGCOMM Computer Communication Review, Volume 32, Issue 4, October 2002

[8] Jim Kurose, Keith Ross Addison-Wesley, “Computer Networking: A Top Down Approach Featuring the Internet”, 3rd edition, July 2004, Chapter 3, Slides 3-99

[9] “NS Manual”, http://www.isi.edu/nsnam/ns/ns-documentation.html

[10] “Marc Greis's tutorial ”, http://www.isi.edu/nsnam/ns/tutorial/index.html

[11] “NS by Samples”, http://nile.wpi.edu/NS/

[12] “BIC TCP”, http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/

Appendix I: NS-2 Simulation Script (cubicnr_scr.tcl)

```
# ===Procudures===

# End procedure
proc finish {} {
    global ns
    global fcwnd0
    global fcwnd1
    global fthr0
    global fthr1

    #Cloase file
    close $fcwnd0
    close $fcwnd1
    close $fthr0
    close $fthr1

    puts "Simulation Ends..."
    exit 0
}


#Indicate the progress during simulation
proc progress {elapsetime} {
    puts "$elapsetime completed..."
}

#Save congestion windows every 0.1 second
proc save_cwnd {flow interval fcwnd} {
    global ns
    set now [$ns now]
    puts $fcwnd "[format %.2f $now] [$flow set cwnd_]"
    #puts "$now [$flow set cwnd_]"

    $ns at [expr $now+$interval] "save_cwnd $flow $interval $fcwnd"
}

# Save throughput to file every one second
```

```
proc save_thr {fid fmon interval fthr} {
    global ns preKByte

    set now [$ns now]

    #define a classifier
    set fclassifier [$fmon classifier]
    set flow [$fclassifier lookup auto 0 0 $fid]

    if {$flow !=""} {
        set curByte [$flow set bdepartures_]
        set curByte_DBL [ns-int64todbl $curByte]
        set KByte [expr $curByte_DBL / 1000]

        set latestthr [expr ($KByte - $preKByte($fid) ) * 8 / $interval]
        set preKByte($fid) $KByte

        puts $fthr "[format %.2f $now] [format %.2f $latestthr]"
    }
    $ns at [expr $now+$interval] "save_thr $fid $fmon $interval $fthr"
}
#===End of Procudures===

#===Basic constant===

#End time in seconds
set endtime 500
set starttime0 0.0
set starttime1 50




# TCP constant Default
set low_window 0
set high_p 0.0000001
set high_window 83000
set high_decrease 0.1
set hstcp_fix 1
#===End of basic constant===
#===Network Topology===
remove-all-packet-headers; # removes all except common
add-packet-header Flags IP TCP; # hdrs reqd for TCP
set ns [new Simulator]
ns-random 0

# Copy from 'behavior.tcl'
Agent/TCP set timestamps_ 1
Agent/TCP set ecn_ 1
Agent/TCP set window_ 100000
Agent/TCP set packetSize_ 1460
Agent/TCP set overhead_ 0.000008
Agent/TCP set max_ssthresh_ 100
Agent/TCP set maxburst_ 2

# BIC - To avoid warning
Agent/TCP set bic_beta_ 0.8
Agent/TCP set bic_B_ 4
Agent/TCP set bic_max_increment_ 32
Agent/TCP set bic_min_increment_ 0.01
Agent/TCP set bic_fast_convergence_ 1
Agent/TCP set bic_low_utilization_threshold_ 0
Agent/TCP set bic_low_utilization_checking_period_ 2
Agent/TCP set bic_delay_min_ 0
Agent/TCP set bic_delay_avg_ 0
Agent/TCP set bic_delay_max_ 0
Agent/TCP set bic_low_utilization_indication_ 0

# Cubic
Agent/TCP set cubic_beta_ 0.8
Agent/TCP set cubic_max_increment_ 16
Agent/TCP set cubic_fast_convergence_ 1
Agent/TCP set cubic_scale_ 0.4
Agent/TCP set cubic_tcp_friendliness_ 1
Agent/TCP set cubic_low_utilization_threshold_ 0
Agent/TCP set cubic_low_utilization_checking_period_ 2
Agent/TCP set cubic_delay_min_ 0
Agent/TCP set cubic_delay_avg_ 0
Agent/TCP set cubic_delay_max_ 0
Agent/TCP set cubic_low_utilization_indication_ 0
```

```
#===end===

#===Create Nodes===
set s1 [$ns node]
set s2 [$ns node]
set s3 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set r1 [$ns node]
set r2 [$ns node]
set r3 [$ns node]

set bandwidth 20; #1000 for 20Mb and 1Gb cases
set quebuf 50; #2000 difference queue buffer for relative bandwidth
#===Create Links===
$ns duplex-link $s1 $n1 [expr $bandwidth]Mb [expr 1]ms RED; #DropTail case
$ns queue-limit $s1 $n1  [expr $quebuf]
$ns queue-limit $n1 $s1 [expr $quebuf]

$ns duplex-link $s2 $n1 [expr $bandwidth]Mb [expr 1]ms RED
$ns queue-limit $s2 $n1  [expr $quebuf]
$ns queue-limit $n1 $s2 [expr $quebuf]
$ns duplex-link $n1 $n2 [expr $bandwidth]Mb [expr 50]ms RED
$ns queue-limit $n1 $n2  [expr $quebuf]
$ns queue-limit $n2 $n1 [expr $quebuf]

$ns duplex-link $n2 $r1 [expr $bandwidth]Mb [expr 1]ms RED
$ns queue-limit $n2 $r1  [expr $quebuf]
$ns queue-limit $r1 $n2 [expr $quebuf]

#===Set Flow Monitor0===
set fmon0 [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $n2 $r1] $fmon0

$ns duplex-link $n2 $r2 [expr $bandwidth]Mb [expr 1]ms RED
$ns queue-limit $n2 $r2  [expr $quebuf]
$ns queue-limit $r2 $n2 [expr $quebuf]
#===Set Flow Monitor1===
set fmon1 [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $n2 $r2] $fmon1
```

```
#===Define Node===
# Refer to behavior.tcl downloaded from
#
http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/script.h
tm

# Refer to behavior.tcl downloaded from
#
http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/script.h
tm
set fid0 0
set tcp0 [$ns create-connection TCP/SackTS $s1 TCPSink/Sack1 $r1 $fid0]
$tcp0 set windowOption_ 13
$tcp0 set low_window_ $low_window
$tcp0 set high_window_ $high_window
$tcp0 set high_p_ $high_p
$tcp0 set high_decrease_ $high_decrease
$tcp0 set hstcp_fix_ $hstcp_fix
#$tcp0 set class_ 1
#===end===
set ftp0 [[set tcp0] attach-app FTP]
$ns at $starttime1 "[set ftp0] start"
$ns at $endtime "[set ftp0] stop"
#===save CWND===
set fcwnd0 [open tcpcwnd0.txt w]
save_cwnd [set tcp0] 0.1 $fcwnd0
#===calculate and save throughput===
set preKByte($fid0) 0
set fthr0 [open tcpthr0.txt w]
save_thr $fid0 $fmon0 1 $fthr0

set fid1 1
set tcp1 [$ns create-connection TCP/Newreno $s2 TCPSink/Sack1 $r2 $fid1]
#$tcp1 set class_ 2
set ftp1 [[set tcp1] attach-app FTP]
$ns at $starttime0 "[set ftp1] start"
$ns at $endtime "[set ftp1] stop"
#===save CWND===
set fcwnd1 [open tcpcwnd1.txt w]
```

```
save_cwnd [set tcp1] 0.1 $fcwnd1
#===calculate and save throughput===
set preKByte($fid1) 0
set fthr1 [open tcpthr1.txt w]
save_thr $fid1 $fmon1 1 $fthr1

$ns at 75 "progress 75"
$ns at 150 "progress 150"
$ns at 225 "progress 225"
$ns at 300 "progress 300"
$ns at 375 "progress 375"
$ns at 450 "progress 450"
$ns at 490 "progress 490"
$ns at $endtime "finish"

puts "Starting Simulation..."
$ns run
```

Appendix II: Codes for Implement New Approach

```
// ++++ red.h file ++++
.......
#ifndef ns_red_h
#define ns_red_h

#include "queue.h"

#include "trace.h"
class LinkDelay;

// Added by Steven Chen for improve RED
// Define two counters and two variables
static int CNT1=0;
static int CNT2=0;
static int CNTmax=0;
static int CNTmin=0;
// end of add
.......
.......

// ++++ red.cc ++++
#include "flags.h"
#include "delay.h"
#include "red.h"

//Added by Steven Chen to improve RED
#include "ip.h"
//end of add
.......
.......

// ++++ deque() ++++

Packet* REDQueue::deque()
{
    Packet *p;
    if (summarystats_ && &Scheduler::instance() != NULL) {
        Queue::updateStats(qib_?q_->byteLength():q_->length());
    }
    p = q_->deque();
    if (p != 0) {
        idle_ = 0;
    } else {
        idle_ = 1;
        // deque() may invoked by Queue::reset at init
        // time (before the scheduler is instantiated).
        // deal with this case
        if (&Scheduler::instance() != NULL)
            idletime_ = Scheduler::instance().clock();
        else
            idletime_ = 0.0;
    }

    //Added by Steven Chen for improve RED
    int fid;
    if (p!=0)
    {
        hdr_ip* iph = hdr_ip::access(p);
        fid=iph->flowid();
        //printf("fid_deque=%d\n",fid);
        if (fid==0)
            CNT1--;
        else
            CNT2--;
        if (CNT1>CNT2)
            CNTmin=CNT2;  // CNTmax=CNT1;
        else
            CNTmin=CNT1;  // CNTmax=CNT2;
    }
    //end of add

    return (p);
}

// ++++ drop_early () ++++

REDQueue::drop_early(Packet* pkt)
```

```
{
    hdr_cmn* ch = hdr_cmn::access(pkt);

    edv_.v_prob1 = calculate_p_new(edv_.v_ave, edp_.th_max, edp_.gentle,
      edv_.v_a, edv_.v_b, edv_.v_c, edv_.v_d, edv_.cur_max_p);

    //Added by steven to improve RED
    int fid;
    hdr_ip* iph=hdr_ip::access(pkt);
    //printf("CNTMax %d, CNT1 %d, CNT2 %d \n",CNTmax,CNT1,CNT2);

    fid=iph->flowid();
    if (fid==0)
    {
        if (CNT2>0)
        {
            edv_.v_prob1=edv_.v_prob1*CNT1/CNTmin; // /CNTmax;

            //Only used in impr2
            if (edv_.v_prob1>1.0)
                edv_.v_prob1=1.0;
        }
    }
    else
    {
        if (CNT1>0)
        {
            edv_.v_prob1=edv_.v_prob1*CNT2/CNTmin; // /CNTmax;

            //Only used in impr2
            if (edv_.v_prob1>1.0)
                edv_.v_prob1=1.0;
        }
    }
    //end of add

    edv_.v_prob = modify_p(edv_.v_prob1, edv_.count, edv_.count_bytes,
      edp_.bytes, edp_.mean_pktsize, edp_.wait, ch->size());

    // drop probability is computed, pick random number and act
```

```
if (edp_.cautious == 1) {
     // Don't drop/mark if the instantaneous queue is much
     //  below the average.
     // For experimental purposes only.
    int qsize = qib_?q_->byteLength():q_->length();
    // pkts: the number of packets arriving in 50 ms
    double pkts = edp_.ptc * 0.05;
    double fraction = pow( (1-edp_.q_w), pkts);
    // double fraction = 0.9;
    if ((double) qsize < fraction * edv_.v_ave) {
        // queue could have been empty for 0.05 seconds
        // printf("fraction: %5.2f\n", fraction);
        return (0);
    }
}
double u = Random::uniform();
if (edp_.cautious == 2) {
           // Decrease the drop probability if the instantaneous
    //   queue is much below the average.
    // For experimental purposes only.
    int qsize = qib_?q_->byteLength():q_->length();
    // pkts: the number of packets arriving in 50 ms
    double pkts = edp_.ptc * 0.05;
    double fraction = pow( (1-edp_.q_w), pkts);
    // double fraction = 0.9;
    double ratio = qsize / (fraction * edv_.v_ave);
    if (ratio < 1.0) {
        // printf("ratio: %5.2f\n", ratio);
        u *= 1.0 / ratio;
    }
}
if (u <= edv_.v_prob) {
    // DROP or MARK
    edv_.count = 0;
    edv_.count_bytes = 0;
    hdr_flags* hf = hdr_flags::access(pickPacketForECN(pkt));
    if (edp_.setbit && hf->ect() && edv_.v_prob1 < edp_.mark_p) {
        hf->ce() = 1; // mark Congestion Experienced bit
        // Tell the queue monitor here - call emark(pkt)
        return (0);   // no drop
        } else {
            return (1);   // drop
        }
    }
    return (0);           // no DROP/mark
}

// ++++ enque() ++++
void REDQueue::enque(Packet* pkt)
{

    /*
     * if we were idle, we pretend that m packets arrived during
     * the idle period.  m is set to be the ptc times the amount
     * of time we've been idle for
     */

    int m = 0;
    if (idle_) {
        // A packet that arrives to an idle queue will never
        //  be dropped.
        double now = Scheduler::instance().clock();
        /* To account for the period when the queue was empty. */
        idle_ = 0;
        // Use idle_pktsize instead of mean_pktsize, for
        //  a faster response to idle times.
        if (edp_.cautious == 3) {
            double ptc = edp_.ptc *
               edp_.mean_pktsize / edp_.idle_pktsize;
            m = int(ptc * (now - idletime_));
        } else
                m = int(edp_.ptc * (now - idletime_));
    }

    /*
     * Run the estimator with either 1 new packet arrival, or with
     * the scaled version above [scaled by m due to idle time]
     */
    edv_.v_ave = estimator(qib_ ? q_->byteLength() : q_->length(), m + 1,
edv_.v_ave, edp_.q_w);
```

```
//printf("v_ave: %6.4f (%13.12f) q: %d)\n",
//  double(edv_.v_ave), double(edv_.v_ave), q_->length());
if (summarystats_) {
    /* compute true average queue size for summary stats */
    Queue::updateStats(qib_?q_->byteLength():q_->length());
}

/*
 * count and count_bytes keeps a tally of arriving traffic
 * that has not been dropped (i.e. how long, in terms of traffic,
 * it has been since the last early drop)
 */

hdr_cmn* ch = hdr_cmn::access(pkt);
++edv_.count;
edv_.count_bytes += ch->size();



/*
 * DROP LOGIC:
 *  q = current q size, ~q = averaged q size
 *  1> if ~q > maxthresh, this is a FORCED drop
 *  2> if minthresh < ~q < maxthresh, this may be an UNFORCED drop
 *  3> if (q+1) > hard q limit, this is a FORCED drop
 */

register double qavg = edv_.v_ave;
int droptype = DTYPE_NONE;
int qlen = qib_ ? q_->byteLength() : q_->length();
int qlim = qib_ ? (qlim_ * edp_.mean_pktsize) : qlim_;

curq_ = qlen; // helps to trace queue during arrival, if enabled

if (qavg >= edp_.th_min && qlen > 1) {
    if ((!edp_.gentle && qavg >= edp_.th_max) ||
        (edp_.gentle && qavg >= 2 * edp_.th_max)) {
        droptype = DTYPE_FORCED;
    } else if (edv_.old == 0) {
        /*
         * The average queue size has just crossed the
```

```
            * threshold from below to above "minthresh", or
            * from above "minthresh" with an empty queue to
            * above "minthresh" with a nonempty queue.
            */
           edv_.count = 1;
           edv_.count_bytes = ch->size();
           edv_.old = 1;
       } else if (drop_early(pkt)) {
           droptype = DTYPE_UNFORCED;
       }
   } else {
       /* No packets are being dropped.  */
       edv_.v_prob = 0.0;
       edv_.old = 0;
   }
   if (qlen >= qlim) {
       // see if we've exceeded the queue size
       droptype = DTYPE_FORCED;
   }

   //Added by Steven Chen to improve RED
   int fid;
   //end of add

   if (droptype == DTYPE_UNFORCED) {
       /* pick packet for ECN, which is dropping in this case */
       Packet *pkt_to_drop = pickPacketForECN(pkt);
       /*
        * If the packet picked is different that the one that just arrived,
        * add it to the queue and remove the chosen packet.
        */
       if (pkt_to_drop != pkt) {
           q_->enque(pkt);

           //Added by Steven Chen for improve RED
           hdr_ip* iph = hdr_ip::access(pkt);
           fid=iph->flowid();
           //printf("fid_in = %d\n",fid);
           if (fid==0)
               CNT1++;
            else
            {
               CNT2++;
            //  printf("CNT2: %d\n",CNT2);
            }
            if (CNT1>CNT2)
               CNTmin=CNT2;  // CNTmax=CNT1;
            else
               CNTmin=CNT1;  // CNTmax=CNT2;
            //end of add

            q_->remove(pkt_to_drop);

            //Added by Steven Chen for improve RED
            iph=hdr_ip::access(pkt_to_drop);
            fid=iph->flowid();
            //printf("fid_drop=%d\n",fid);
            if (fid==0)
               CNT1--;
            else
               CNT2--;
            if (CNT1>CNT2)
               CNTmin=CNT2;  // CNTmax=CNT1;
            else
               CNTmin=CNT1;  // CNTmax=CNT2;
            //end of add

            pkt = pkt_to_drop; /* XXX okay because pkt is not needed anymore
*/
        }

        // deliver to special "edrop" target, if defined
        if (de_drop_ != NULL) {

        //trace first if asked
        // if no snoop object (de_drop_) is defined,
        // this packet will not be traced as a special case.
            if (EDTrace != NULL)
               ((Trace *)EDTrace)->recvOnly(pkt);
            reportDrop(pkt);
            de_drop_->recv(pkt);
        }
        else {
            reportDrop(pkt);
            drop(pkt);
        }
    } else {
        /* forced drop, or not a drop: first enqueue pkt */
        q_->enque(pkt);

        //Added by Steven Chen to improve RED
        hdr_ip* iph = hdr_ip::access(pkt);
        fid=iph->flowid();
        //printf("fid_in = %d\n",fid);
        if (fid==0)
            CNT1++;
        else
        {
            CNT2++;
        //  printf("CNT2: %d\n",CNT2);
        }
        if (CNT1>CNT2)
            CNTmin=CNT2;  // CNTmax=CNT1;
        else
            CNTmin=CNT1;  // CNTmax=CNT2;
        //end of add

        /* drop a packet if we were told to */
        if (droptype == DTYPE_FORCED) {
            /* drop random victim or last one */
            pkt = pickPacketToDrop();

            //Added by Steven Chen for improve RED
            iph=hdr_ip::access(pkt);
            fid=iph->flowid();
            //printf("fid_drop=%d\n",fid);
            if (fid==0)
                CNT1--;
            else
```

```
                CNT2--;
            if (CNT1>CNT2)
                CNTmin=CNT2;  // CNTmax=CNT1;
            else
                CNTmin=CNT1;  // CNTmax=CNT2;
            //end of add

            q_->remove(pkt);
            reportDrop(pkt);
            drop(pkt);


            if (!ns1_compat_) {
                // bug-fix from Philip Liu, <phill@ece.ubc.ca>
                edv_.count = 0;
                edv_.count_bytes = 0;
            }
        }
    }
    return;
}
```