Improving the Start-up Behavior of a Congestion Control Scheme for TCP

1.Updation of slow start threshold :

We need to update the value of initial slow start threshold as the bandwidth delay product's byte equivalent.
Delay is calculated by calculating the rtt value at the time of connection establishment.That is when syn ack is sent,wait for syn+ack and calculated the time by using m_rtt->Getestimate().
The result was same even when we used m_lastRtt.The values weren't different.
Bandwidth should be calculated by sending two data packets.when acknowledgement for each packet was received we noted down their time of arrival as t1 and t2.So bandwidth will be equal to (segment size)/(t2-t1).

In paper it was mentioned that we need to calculate the bandwidth by sending three packets and noting down their time of arrival.Using least square estimation method we need to find the time difference.

But when we searched for least time square estimation method there should be some other coordinate which we require to calculate it.For instance if there is no other variable then we can take it as average value.

First procedure:
We implemented first by sending only two packets.Calculated the bandwidth delay product.
The code we wrote for it is as follows:

```
static uint32_t pcount=0;
uint32_t bdproduct;
pcount++;
static double t1,t2,rtttime,t3,t4;
static Time rtt;
 rtt= m_rtt->GetEstimate () ;
 uint32_t size=m_tcb->m_segmentSize;


 if (pcount==1)
        {
        rtt= m_rtt->GetEstimate () ;
        t1=Simulator::Now().GetSeconds();
        }
if (pcount==2)
        {
         t2=Simulator::Now().GetSeconds();
        }
  else if( pcount==3)
  {
        t3=Simulator::Now().GetSeconds();
        rtttime=rtt.GetSeconds();
          t4=((t3-t2)+(t2-t1))/2;
        bdproduct=size/t4;
        bdproduct=bdproduct*rtttime;

        m_tcb->m_ssThresh=bdproduct;
 }
```

Problems faced:
Not working of updatessthresh() :
We used updatessthresh(oldvalue,newvalue) to update the ssthresh with the bandwidth delay product calculated.We first tried to print old value and then used the above function and then printed it.The observations were the ssthresh old value wasn't updated.

Resolved:
So instead of using updatessthresh() we assigned the value of bandwidth-delay product to m_tcb->m_ssThresh.We followed the same approaches above.When we tried to print the old ssthresh value and new ssthresh value the update has occurred.

Problem 2:
Case 1:We simulated this code in fifth.cc.When the pcount value mentioned above when it was 1 and 2,output showed as updation of ssthresh but AI algorithm didn't work.That is after congestion window crossed ssthresh,when acknowledgement is received congestion window is increasing as cwnd+1 but not like cwnd+1/cwnd.

Case 2:When pcount value was 2 and 3.The AI algorithm worked.That is after congestion window crossed ssthresh and when acknowledgement is received the cwnd increased by value of 1/cwnd.

Case 3:
Similarly we changed the values of pcounts to 3,4 and 4,5 and 10,11.None of them worked like case 2.

We doubted whether sshthresh value was updated or not.For this we used ns3TcpSocket->TraceConnectWithoutContext("SlowStartThreshold",MakeCallback (&UpdateSsThresh)); in fifth.cc.In updatessthresh() we printed the new value of ssthresh.
When simulated we found that whenever ssthresh is updated this function prints the ssthresh.
This printing of ssthresh is performed only for case 2.So we found out that ssthresh is correctly updated only for pcount 2 and 3.

Our Assumption:
The problem we assumed is there receivedack() in tcpsocketbase.cc also receives syn+ack.
So as we tcp header we tried to check whether syn+ack flag is also received.

So the code we wrote is

```
bool check=tcpHeader.GetFlags();
bool has_syn= tcpHeader.GetFlags() & TcpHeader::SYN;
 bool has_ack= tcpHeader.GetFlags() & TcpHeader::ACK;
 bool synack = has_ack & has_syn;
 std::cout<<has_syn<<" syn "<< has_ack<<" ack"<<std::endl;
```

So if received ack has both syn and ack it should print 1 else zero.When we simulated we found out that value printed was zero.That means syn+ack is not received in tcpsocketbase.cc.Hence our assumption was wrong.

So we now simulated the code in tcp-variants-comparison.cc present in examples/tcp in ns3. So when we tried executing it,we found that it too worked only for case.Hence this was also failure.

Final:
Now instead of calculating bandwidth using two packets,we calculated it for three packets.So we have now two differences that is t2-t1(first and second ack) and t3-t2(second and third ack).So according to least square estimation we took average value of times and found the bandwidth delay product using this bandwidth.Required results were produced.Threshold was updated and cwnd changed according to AI(additive increase algorithm).

2.Fast Retransmission:

In normal fast retransmission multiple losses cannot be recovered from same window.
So the lost packets are retransmitted only after timeout.If many multiple packets are lost,the total time to send to lost packets will be high.
In our paper algorithm is such after retransmission and when state enters into recovery phase and two dupacks are received we send a new packet,(the next packet to highest sent packet by tcp so far).In this way we tried to keep the flow of packets in the network.

Our first approach:
In dupack() (in tcpsocketbase.cc) when we wrote the following condition:

```
if(m_dupackcount==2 && tcpcongestionstate==CA_Recovery)
{
SendPendingData(m_connected)
}
```

Problem:
When we simulated this code there weren’t any changes.We confirmed it by using wire shark.We wrote code for generating pcap file.When the two extra dup acks were received sender didn’t send any extra packet.

Approach 2:
Now we understood that even when state is recovery phase dupack value increases.So it is not correct to check condition as m_dupackcount==2.

Hence we made changes to the codes

```
if(m_tcb->m_congState == TcpSocketState::CA_RECOVERY)
{
            if(m_dupAckCount>m_retxThresh)&& (m_dupAckCount-m_retxThresh)%2==0)
            {
                  Sendpendingdata(m_connected);
            }
}
```

We wrote some print statements in between to check whether it was working or not.It was working fine.But when we used wireshark and checked the sender wasn't sending any new packet after two dup acks were received.

Approach 3:

So we assumed that there might be something wrong with sendpendingdata() function.So we tried to print the return value of sendpendingdata() and observed that it returned value 0 i.e zero packets were sent.
So instead we used the function senddatapacket to check whether it was working or not.This senddatapacket() function was brought from sendpendingdata().And now we simulated it.
Observation was the sender was able to send the packet upon receiving two dup acks.
But the sent packet was the same packet as retransmitted one.

Approach 4:

As mentioned in approach 3 the error was due to in senddata,the first parameter sent was unacknowledged packet.Thats why when two dupacks were received the packet which was retransmitted was being sent again.We found the data sent was retransmitted packet from wireshark.So in the first parameter we need to send the packet which was the highest packet sent by tcp till now.This value is stored in m_highTxMArk.So the first parameter to be sent is m_highTxMark.When this code was executed and hecked in wireshark the sender sends a new packet upon arrival for every two dupacks.Hence our condition was achieved.

We tested our codes on fournodes,fifth.cc and gfc-dumbbell topologies in every case required outputs were produced.