Received: December 5, 2016

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

TCP M-Start: A New Slow Start Method of TCP to Transfer Data Over Long Fat Pipe Network

Patel Ritesh Pravinbahi[1]* **Ganatra Amit Pravinchandra[1]**

[1] *U and P U. Patel department of Computer Engineering, CSPIT, Charusat, Changa, India*
* Corresponding author's Email: riteshpatel.ce@charusat.ac.in

Abstract: Transmission control protocol have gone through various revisions to develop new method of responding to network congestion control as per past, present and future requirements. In cloud computing moving large size of a VM from one data center to another data center over WAN is a challenging task. To regulate data flow, sending of data is carried out in three phases iteratively. Slow start, congestion avoidance and fast recovery. All these methods are functions of ACK reception. Specially in slow start, initially it is very slow but it progresses aggressively as times increases. Because of a slow start, it suffers from low network resource utilization in the high bandwidth delay scenario. At every receipt of ack, the congestion window is either step up by one or a zero. In this paper, efforts are made to improve slow start behavior by changing step up count (scnt) to improve network resource utilization. Experiments and results, observed during the slow start phase, show that throughput has increased to 51.23%, time to reach epoch has reduced to 40%, the position of epoch point has increased to 4 times higher than traditional but it leads to increased 30% of packet drops.

Keywords: TCP congestion control, Slow start, Step-up count, Epoch point, VM migration, Cloud computing, Data centre.

1. Introduction

About 2.6 billion mobile devices will be connected by the 2020 over the Internet [1]. It will raise the need of more services from data centre. Users also need data in a short response. Data Centre (DC) has to provide those services in very less time as per user needs. To achieve that DC needs to perform load balancing either by moving or copying VM nearest to clients [2]. This will reduce long distance traffic on the Internet as well as fast access to the required resources hosted nearby user.

To mitigate above service requirements data center has to provide VM migration efficiently over WAN. There are three types of VM migration [3] developed in local area network environment. It transfers VM over short distance with very high bandwidth i.e. 100 Gbps. WAN VM migration has issue of a latency and a resource occupancy for long time. To support same method in WAN, TCP protocol needs to be modified. Fiber optic network has solved the problem of backbone bandwidth over WAN. WAN has inherent problem of propagation delay. To support WAN VM migration along with LAN migration, data centre need of two types of hypervisor is suggested. One hypervisor takes care of internal VM migration. Other hypervisor takes care of external VM migration. In this kind of architecture, external hypervisor will take care of transferring VM on the high bandwidth and high delay where internal hypervisor is engaged to transfer VM internally in a very high bandwidth and very low delay network. Additionally, fiber optical network has reduced external interference while transferring data in WAN environment.

In TCP/IP protocol stack, network layer is engaged to transfer data from one end to the other end. TCP is responsible for regulating the flow between the end devices. TCP works at the transport layer, one layer above the network layer. The major concern of transport layer is to provide connection oriented and connection less services.

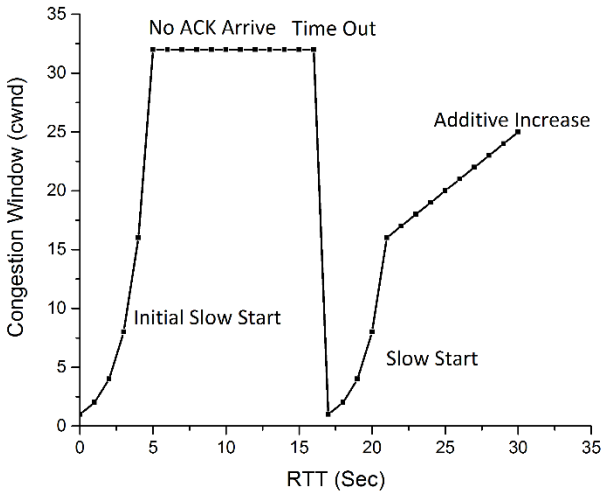

Figure.1 Behaviour of slow start

TCP is an example of a connection oriented protocol. A traditional TCP flow consist of three phases: Connection Establishment, Data Transfer and Connection Termination. During connection establishment client and server exchanges three packets. Both the ends also negotiate various parameters like value of initial sequence number, capacity to receive data during connection establishment. Congestion control mechanism is used to regulate data flow of connections over network

TCP is a connection oriented protocol which requires acknowledgment(ack) to be sent from the receiver to give feedback of data received successfully. If the packet is corrupted, sender would not receive any ack. Retransmission timer timeouts at sender and it resends the packet. In 1983, TCP did not have congestion control mechanism. In 3 years of span, the Internet had its first breakdown because of absence of network congestion control mechanism. V. Jacobson [4] has introduced a congestion control mechanism that have solved problem at some extents. Author gave theory to maintain three variables per connection: a congestion window(cwnd), a slow start threshold(ssthresh) and an advertised window (cwnd). In TCP flow, packets are sent from sender to receiver; receiver acknowledges receipt of packets. Congestion window (similar to sliding widow) is maintained on both the sides to control sending number of packets.

As shown in figure 1, cwnd is set to 1 initially, upon receiving each ack it gets incremented by one. This process is repeated till it suffers from congestion (No Ack arrive). It concludes network is congested by observing retransmission timed out. It again starts slow start to send packets. This process of slow start and congestion is repeated until all information is sent. Average throughput is decreased because of multiple slow start during transfer of long data. Our proposed method reduces time to slowly start thus increases average throughput.

Transmission of TCP streams is divided into two phases. (1) Slow start and (2) Congestion Avoidance. TCP stream comprises of progression of information sections (or parcels) sent from a source to a destination.

Acknowledgment packets travel in the reverse direction. Before receiving an acknowledgment, a source sends a specific number of packets (specified by cwnd), at the speed of local link. Therefore, cwnd controls the rate of packet transmission at any given point of time. Upon receiving ack from the receiver side, sender increases the congestion window according to the phase of connection transfer. During slow start phase, sender increases congestion window by one and sends two packets. Thus, congestion window of TCP gets doubled at every round trip time. In congestion avoidance phase, sender increases congestion window to fraction of 1. A fraction of 1 depends on by specific criteria defined by specific TCP variants.

Slow start method utilizes network bandwidth slowly. Following questions need to be answered to make an efficient slow start

- What should be the initial value of congestion window? There should be a proper selection of initial congestion window size for the slow start. If the initial congestion window is kept sufficiently large, then epoch point could be reached in less time thus improves network throughput. But wrong pick up of congestion window leads to poor efficiency of network.
- Does ssthresh is sufficient to move from slow start to congestion avoidance? Ssthresh divides slow start in two phase. Is there any need for multiple phases?
- How should react with reception of ACK in slow start? Traditional method implements congestion window increase by 1 upon reception of every ack. The system should react in such a way that it could identify congestion well in advanced in the bottleneck router.
- How could we move from slow start phase to congestion avoid phase without losing any packets.

Protocols are dependent on losses to detect the BDP (Bandwidth Delay Product). Losses of packets further reduces the efficiency of the network. At least one RTO is needed to detect the losses of the network. Exponential sending of data to learn a loss would create congestion in the network. Other flow

may get impacted because of exponential sending of data. In case of low bandwidth network, reaching to epoch point after failure would be very easy. But, in case of high speed network start from one to reach to the epoch point would take longer time because of multiple iteration of RTT.

Following are some tuning and responding parameters which effects network throughput and efficiency.

(1) cwnd: if congestion window is set to less than BDP then it would never make congestion in network and never makes any loss. But for that one requires to know the bandwidth, delay and queue size of the network well in advance.
(2) ssthresh: it is a point to transit from slow start to congestion avoidance phase. Congestion avoidance decreases the possibility of lots many losses in single congestion window as compared to aggressive behaviour of slow start. There is a trade-off between time to reach epoch and throughput with the value of ssthresh. If ssthresh is near to the congestion window(cwnd), it may lead to too many packet losses because TCP responds aggressively during slow start but results in better throughput. If ssthresh is far way to the congestion window cwnd, it leads to fewer packet losses but results in low throughput.
(3) RTT: Smooth RTT plays important role in the phase of congestion avoidance. Almost all the existing TCP variants are dependent on RTT to find out their next increase of congestion window.
(4) RTO: Larger value of RTO makes throughput very less. The Lesser value of RTO results in too many retransmissions which further decreases efficiency of path.

The remainder of this paper is organized as follows. The Next section covers literature review titled as related work. In Section 3, analysis of current slow start method is mentioned, proposed method is specified in section 4. Section 5 covers analysis of proposed method. Section 6 compares simulation results and concluded in section 7.

2. Related Work

As mentioned above, congestion control algorithms have two phases, namely slow start and congestion avoidance. Slow start, initially grows slowly but over a period of RTT it grows exponentially to estimate available bandwidth. It sets two parameters, ssthresh and MaxCwnd, at the end of the procedure. When size of congestion window reaches to ssthresh then it starts congestion avoidance. In congestion avoidance phase growth of congestion window is very slow as compared to slow start phase. The Objective of the congestion avoidance is to avoid too many dropping of packets as it is about to reach to the maximum capacity of network.

Lower value of ssthresh will take longer time to reach MaxCwnd in congestion avoidance system. Lower value of ssthresh will increase congestion avoidance time, which further reduces average throughput. Higher value of ssthresh will reduce congestion avoidance time and improves the average throughput.

Authors [5] have implemented new start up method called AFStart for the adaptation of changes of WAN. It dynamically sets slow start threshold and dynamically adjusts congestion window. AFStart can ramp up the congestion window from its initial value to the slow start threshold more quickly and smoothly than standard slow start, and AFStart achieves higher network link utilization and TCP throughput during the slow start than Fast TCP.

Initially, the congestion window(cwnd) is set to 4 packets. The initial 4 packets are sent back-to-back to measure the available bandwidth by the packet train algorithm. If calculated BDP based on reference bandwidth is above 16 then it jumps congestion window to 16 packets. Then after ImTCP method is used for bandwidth estimation.

Shortcomings: It starts with initial 4 packets for packet train which is insufficient for the estimating bandwidth. Again, an abrupt step increase of congestion window might acquire the available resources in short duration of time. which forces either source itself for a retransmission time out OR it forces other traffics to get congested soon

Authors [6] have developed a method which is known as TCP FastOpen. In this paper, they have utilized messages exchange of connection establishment to negotiate network capability parameters to define reduces network latency by one RTT to slow start.

Shortcomings: It does not suggest any modification of slow start rather it suggests new ways of reducing network latency by one RTT.

Authors [7] have presented a new slow-start variant which improves the throughput of TCP-Vegas. In this method, rate of congestion window size is increased between exponential growth and linear growth during slow-start phase. It improves throughput significantly during the slow-start phase.

Shortcomings: It suffers from low throughput and considerable packet loss.

Standard slow start has exponential increase phase, which causes high packet losses when it comes near to the epoch position. Recovering from heavy packet losses puts extremely high load on

sender which puts system in unresponsive for a long time, resulting in a long blackout period of no transmission. Authors [8] specified a new method called Hystart, its aim to prevent from the packet loss when it reaches to the epoch position of window size. HyStart improves the start-up throughput of TCP more than 2 to 3 times.

Shortcomings: In order to address the packet dropping problem of slow start, it slowdowns increase of congestion window near to epoch point. It achieves through observing two congestion indicators (1) ACK train length (2) the increase of RTT delays. Throughput gets decreased as it decreases as rate of congestion window drops near to epoch point.

Authors [9] aim to prevent multiple packet losses which occur at the end of standard slow start caused by exponential growth of congestion window. In this paper authors have divided process of standard slow start into two phases. In the first phase, up to ssthresh/2, it works as traditional slow start. In the second phase, it grows like negatively exponential growth, which greatly reduces multiple packet losses from a congestion window.

Shortcomings: It has low throughput as compared to standard slow start. But it reduces the probability of packet loss.

Authors [10] have developed this protocol for the satellite communication network where BDP is very high, but the probability of loss is also very high. The aim of the method is to optimize slow start phase. At the start of the connection, the initial value of cwnd is set to detected network bandwidth. It estimates the bandwidth and the window by using the time interval during each two subsequent ACKs. Also, error to estimate bandwidth is measured and used in subsequent iteration to reduce error. The two weights, exponential weighted filtering the smooth bandwidth last time and the estimated bandwidth in current iteration, can be adjusted, and its size can be changed dynamically according to the different network environment.

Shortcomings: Abrupt change of congestion window in case of slow start may fill up bottle neck queue quickly, which may lead to congestion of the network.

Authors [11] developed variation of slow start called smooth-start, which provides a smooth transition from slow start to congestion avoidance phase. Smooth-start solves this problem by approaching the slow-start threshold more gradually. The aim of the author is to significantly reduce both packet losses and traffic burstiness. It puts necessary lag points between ssthresh/4 and ssthresh to reduce the effect of traffic burstiness. In smooth- start, there are two sub-phases defined by a chosen separator ssthresh, ssthresh/4, In the first sub-phase, Smooth-start behaves the same way as slow-start. In the second sub-phase, called the Smooth-start period, cwnd is incremented only upon receipt of two or more ACKs. So, at every RTT cwnd increases by a factor of 1.5 or less. Increase of cwnd depends on gradient called coarse-grained and fine-grained.

Shortcomings: If there is no congestion occurs, the performance of smooth-start is worse than Slow-start. Other worst case happens when TCP connection is closed exactly when the congestion window size cwnd reaches ssthresh. If the transmission of data lasts after reaching ssthresh, the degradation of TCP performance caused by the Smooth-start will become more trivial.

TCP uses self-clock based mechanism, ACK reception, to update value of congestion window. Authors [12] have suggested a new method of updating congestion window by the way of system timer. They took 200 microseconds of interval to generate interrupt and update the congestion window. In slow start, normally data are sent when the ACK are received. That happens after one RTT period of time where sender receives ack number in burst. That triggers an increase of congestion window at the end of each RTT and leads to send too many packets at the same time. This method disperses sending of data during RTT, so that burstiness would not happen.

Shortcomings: Congestion window is updated at the rate of 200 microseconds. It leads to static variation of change of congestion window.

Authors [13] has given research direction to improve method of slow start to improve TCP performance. They also discussed and compared suitable startup speed with respect to TCP variants. They have also given research direction towards reducing queuing delay at each router, finding suitable start up speed for slow start mostly in high BDP network. TCP modification should support future network bandwidth as well as it should be backward compatible. They also investigated new research direction on congested ACK packets. The authors have shown hope for alternate method to slow start mechanism of slow start given name as e-speed which selects protocols depending on the network condition and gives maximum performance whether it is deployed on larger bandwidth or lower bandwidth.

Growth of congestion window is a function of rate of reception of ACK from the receiver. Authors [14] have introduced new model of generating ack from receiver to control the growth function of congestion window. The authors presented that

ACKs can be split so that an acknowledged portions of a segment instructs the sender to open more aggressively its congestion window. It focuses on the reducing the time of reaching to maximum congestion window thus improving throughput.

Shortcomings: Sender and receiver both need to be modified to take an effect.

Authors [15] have suggested to improve fast retransmit mechanism for quick recovery. The Initial value of ssthresh is set to the BDP. Bandwidth is calculated using least square estimation. Approximate delay is calculated during SYN segments exchanged during connection establishment. Once the congestion window, reaches to the ssthresh, it grows traditionally. The authors have put more focus on fast recovery. Instead of acting on 3rd ack, start reacting to 2nd acks which decreases further loss of data, and increases throughput.

Authors [16] have suggested modification into slow start named as conservative slow start. From starting of sending data it estimates capacity, buffer size, buffer level and calculates value of fractional increments of congestion window. Authors have used path estimators to detect TCP session queues build up.

Shortcomings: It avoids a significant amount of segment losses, but not so much as the slow start. This performance comes with a 5 times penalty on transaction speed than the other protocols.

Following are the research category of above research articles.

- Determining exit time from slow start to congestion avoidance (Value of ssthresh): This becomes exit point for the next iteration to jump into congestion avoidance. Examples are AFast Start [5], Hystart[8], Smooth Start [11].
- Estimating value of MaxCwnd: This becomes a target for the subsequent iteration. Examples are Novel Quick Start [10], [15].
- Multiple phases of Slow start method. Example is P-TCP [9].
- Change of ACK parameter. Example is Gallops Vegas [7].
- Change of slow start phases [16].
- Change into Three-way handshake, data transfer and connection termination. Example is TCP Fast Open [6].
- Change of method of responding to ACK. Example is Smooth Slow Start [12].
- Change of method of sending to ACK. Example is Conservative Slow start [14].

3. Analysis of Slow Start Method

Table 1. Evolution of Window of slow start size with respect to RTT in slow start

RTT Cycle #	Time	Acked Packet	Window Size	Packet Released	Queue length
0	0	-	1	1	1
1	T	1	2	2,3	2
2	2T	2	3	4,5	2
2	2T+1/μ	3	4	6,7	3
3	3T	4	5	8,9	2
3	3T+1/μ	5	6	10,11	3
3	3T+2/μ	6	7	12,13	4
3	3T+3/μ	7	8	14,15	5
4	4T	8	9	16,17	2
4	4T+1/μ	9	10	18,19	3
4	4T+2/μ	10	11	20,21	4
4	4T+3/μ	11	12	22,23	5
4	4T+4/μ	12	13	24,25	6
4	4T+5/μ	13	14	26,27	7
4	4T+6/μ	14	15	28,29	8
4	4T+7/μ	15	16	30,31	9

In traditional slow start, TCP sends two packets upon receiving each ACK. This way of evolution of window size is shown in Table 1.

RTT cycle specifies one RTT iterations. At time *t=0*, first packet is released by source to next node. At the end of one RTT, sender receives ack for the first packet and its window size gets incremented by one. Thus, sender sends two packets, numbered as 2,3. From the table, it is easy to see that the window size and queue size satisfies the following equations for *n >= 1*,

$$W(nT + m/\mu) = 2n-1 + m + 1,\ 0 <= m <= 2^{n-1} - 1.....n >= 1 \quad (1)$$

$$Q(nT + m/\mu) = m + 2,\ 0 <= m <= 2^{n-1} - 1.....n >= 1 \quad (2)$$

where W denotes window size, n denotes iteration number, T denotes n^{th} iteration of RTT, m denotes ack of i^{th} packet received during n^{th} iteration, $1/\mu$ denotes service rate.

Note that the maximum queue length in n^{th} iterations $Qmax\text{-}n = 2^{n-1}\text{-}1 + 2 = 2^{n-1}+1$ and the maximum window size in a n^{th} iteration $Wmax = 2n$. So forth $Qmax\text{-}n = Wmax/2$. Buffer overflow does not occur in the slow-start phase if $Qmax\text{-}n \leq Bs$, where Bs is buffer queue size. But $Qmax\text{-}n$ =

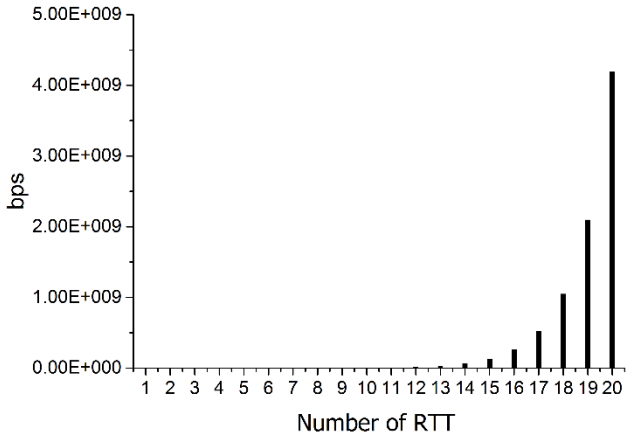

Figure 2. Theoretical Round trip time vs Bits per second

Wmax/2 ≤ (cT+Bs)/4 where cT is number of packets in transit. Thus, a sufficient condition for no buffer overflow in slow start is *(cT + Bs)/4* which is equivalent to *Bs ≥cT/3.*

Author [17] have presented analysis of linear path model as contextual part is presented here. Let P be the time to send an entire packet in units of T. The contention phase has length 2^{e-1}, so the total time to send one packet (contention + packet time) is $2^{e-1}+P$. The useful fraction of this is, of course, P, so the effective maximum throughput is $P/(2^{e-1}+P)$.

Inter frame space interval is 96 ns for Gigabit Ethernet(GE), 9.6 ns for 10GE and 0.96/2.4 ns for 100/40 GE, respectively. The sum of the propagation delay and transmission delay is given by $\tau + 1/\mu$ where τ is propagation delay of signal, $1/\mu$ is service rate of sending device. Suppose that the link capacity is 1 Gbps and packet size is 1000 bytes, then $1/\mu$ would be 0.008 msec per packet. Sender sends data according to its speed of local link. This speed fills up a queue of intermediate bottleneck node very quickly. If the transmission distance is 10000 km and if we consider queuing delay is same as propagation delay, then round-trip propagation delay T would be 4 X 106 / 2 X 108 = 200 msecs.

There is trade-off between underestimation and overestimation of queue buffer size of intermediate node. If buffer size is underestimated, then it leads to poor utilization of network resources. If buffer size is overestimated, then it could lead to packet retransmission henceforth reduces efficiency.

4. Proposed Work

Initial start-up method of TCP known as slow start is further divided into two parts. Slow start itself and exponential increase. Slow start and exponential increase aims to find out the maximum capacity of network. Slow start method starts with congestion window 1, and it increments its congestion window as it receives acknowledgment. Initially this is termed as a slow start, because its growth of window is very slow over RTT. But when it progresses, same method of increments of congestion window leads to exponential increase. Author [18] have suggested to initialize window size to 10 to save 4 RTT times for web traffic. There is a need to develop protocol which follows slow start, but jumps to exponential increase very soon at the start of each iteration and also becomes steady when it's about to reach the next iteration. In our proposed method, we follow the same methodology as mentioned above. M-increase increments congestion window by additive factor P. Here initial value of P is set to 10 as mentioned in [18]. Gradually additive factor is getting decrease by one upon receiving each ack. When P becomes 0 it would reset to 10.

As per equation 1, congestion window is a function of number of acknowledgment in n^{th} iteration. At the beginning of iterations congestion window is increased by one at reception of every acknowledgment, although a large amount of bandwidth is available. As shown in table 1, initial iterations are having less number of sub iterations because it receives few packets and sends double packets of received packets. Then it waits for one RTT to receive ack and continues to send packets as mentioned above. It's waiting time is larger than the sending time. During the start of iterations only less bandwidth is utilized. In this proposal, increment of congestion window is a function of n^{th} iteration in k^{th} acknowledgment. Queue buffer size in the intermediate node gets empty at the end of each iteration. Increment of congestion window is higher at the start of iteration and low at the end of iteration, which allows high data rate at the start of iteration where whole bandwidth is empty and low congestion. To find out minimum delay, first two iterations are not changed.

The proposed work is based on the function of acknowledgment number(k) in i^{th} iteration. When it starts new iteration, queue would be empty. To fill up the queue aggressively it starts with higher number of increments in congestion window i.e. scnt, which increases utilization of network. When it reaches near to the end of iteration it progressively slowdowns increment to 0. This scenario could be achieved through equation 3.

$$W(nT + m/\mu) = [2^{k-1} + m + 1 + (p - (k\%p))],$$

$$0 <= m <= 2^{n-1} - 1, k <= n \quad (3)$$

where W denotes window size, n denotes iteration number, T denotes n^{th} iteration of RTT, m denotes ack of i^{th} packet received during n^{th} iteration, $1/\mu$

denotes service rate, P denotes initial value of step up count. K is used to rotate value of P from 10 to 0.

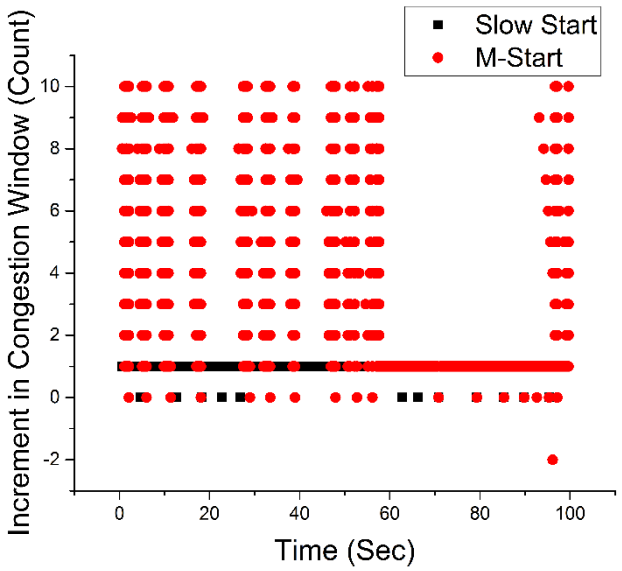

Figure 3. Increment of Congestion window with respect to time

Algorithm

Input: ACK Number
Method: Repetitive increment with ceiling
Output: Increment in congestion window

```
P=0;
scnt=P;
scnt--;
If(scnt<>0)
While(tp→snd_cwnd_cnt>=tp→snd_cwnd) {
    tp→snd_cwnd_cnt-= tp→snd_cwnd;
    if (tp→snd_cwnd < tp→snd_cwnd_clamp)
            tp→snd_cwnd+=scnt;
    if (scnt==0)
       scnt=P;
    }
```

5. Analysis of proposed work

To analyze proposed method, we have used NS-2.35 event driven network simulator. A CUBIC variant of TCP is used to send and receive data. Hystart[8] is implemented in current linux and NS 2.35 in addition to CUBIC. TCP-Cong.c file has been modified to observe an effect of proposed approach. As mentioned in algorithm, P is initialized to 10. scnt is step up count. Scnt step up count is added to the congestion window if all the condition is satisfied by congestion control algorithm. When scnt becomes 0, it is reinitialized with P.

Table 2. Evolution of Window size of M-Start with respect to RTT

RTT Cycle #	Time	Acked Packet	Window Size	Packet Released	Queue length	Throughput (Kbp
0	0	-	1	1	1	8
1	T	1	2	2,3	2	16
2	2T	2	12	4 to 16	11	88
2	2T+1/μ	3	21	17 to 26	18	144
3	3T	4	29	27 to 35	9	72
3	3T+1/μ	5	36	36 to 42	16	128
3	3T+2/μ	6	42	43 to 48	22	176
...						
3	3T+23/μ	26	139	137 to 145	119	952
4	4T	27	146	146 to 152	8	64
4	4T+1/μ	28	152	153 to 158	14	112
4	4T+2/μ	29	157	159 to 163	19	152
...						
4	4T+121/	166	901	899 to 907	754	6032

Table 3. Comparative summary analysis of both the methods at end of each iteration

Iteration #	Traditional Slow Start			Modified Slow Start (M-Start)		
	Window Size	Queue Length	Throughput	Window Size	Queue Length	Throughput
0	1	1	8 Kbps	1	1	8 Kbps
1	2	2	16 Kbps	2	2	16 Kbps
2	4	3	32 Kbps	27	18	232 Kbps
3	8	5	64 Kbps	139	119	10392 Kbps
4	16	9	128 Kbps	901	754	426712 Kbps

In research mentioned [6-16] in related work, they are applying algorithm after cwnd reach to ssthresh/2. In our approach, we applied count increase method before ssthresh/2. It becomes very aggressive than other methods, but also increases network resource utilization when the next iteration starts.

We present comparative analysis between traditional slow start and M-start shown in table 3. M-start is also function of ack, it changes congestion window on receipt of every ack with different count. There is no change in first iteration. M-Start starts with sending 1st packets in the network. Then it waits for one RTT. It sends two packets in iteration 1 and waits for the acknowledgment. In iteration 2, after receiving ack it increments cwnd by 10 so cwnd becomes 12. Table 2 shows progression of congestion window with respect to RTT. As mentioned in table, upon receiving second ack in the same iteration 2, scnt becomes 9 and cwnd becomes 21. During that it release 10 packets as congestion window to send 10 packets.

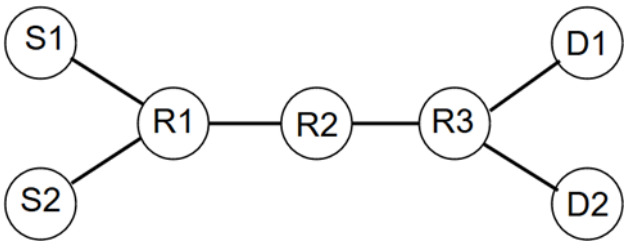

Figure 4. Simulation Topology

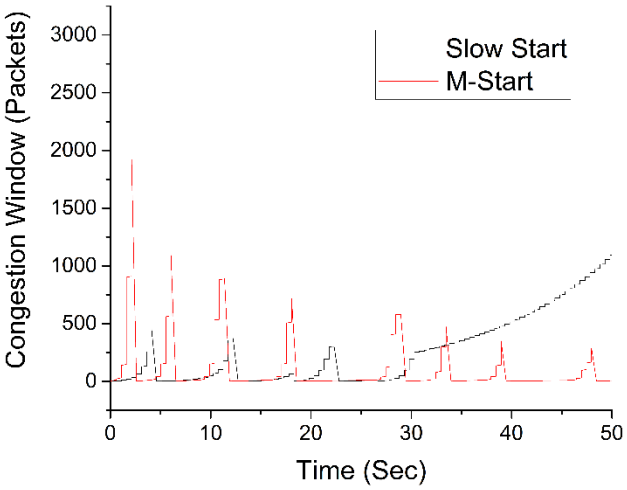

Figure 5. Comparative Result: Over 50 Sec

Comparative graph, as shown in Figure 3, shows increments of congestion window of traditional slow start and M-Start with respect to time. Slow start increments cwnd by on 1 only while M-start increments cwnd from 10 to 1 over period of time.

Table 2 show detail analysis of change of cwnd with respect to each packet received for M-Start and Table 3 gives comparative summary of traditional slow start and M-Start method for each iteration. M-Start gives better throughput that tradition slow start.

6. Performance Evaluation

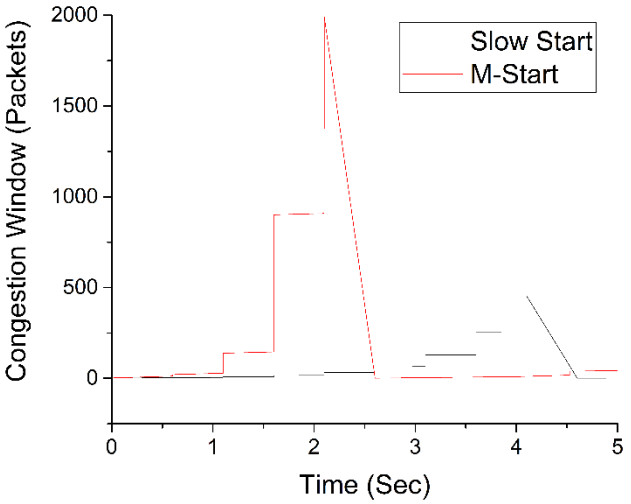

Figure 6. Comparative Result: Magnified result over 5 sec.

6.1. The Simulation Setup

In the simulation experiment, a standard single dumbbell topology is used as shown in Figure 4. Only one sender (S1) and one receiver (D1) are used. S1 sends data to D1 through three routers R1, R2 and R3.

Node S1 and D1 are connected to the routers over LAN with 1 Gbps speed and 1 ms propagation delay. Intermediate routers are also linked by 1 Gbps speed with 200 ms propagation delay. This network has a bottleneck at routers, and there is no other traffic present in this topology. Window size, throughput and packets loss are considered as a performance metric.

6.2 Simulation Result

As shown in the table 1, traditional slow start algorithm increment congestion window by 1 at all the time. Although sufficient bandwidth available to send large number of packets. Even if we choose to increment with specific value then congestion window increases, so aggressively it could fill the pipe and results in heavy congestion. We adopted variable congestion widow size with respect to previous change of congestion window.

Change of congestion window with respect to time for traditional slow start and proposed slow start is shown in figure 5. M-start changes congestion window with exponential growth, but when it reaches to change limits it slow down change of growth and restart the whole procedure.

So, it follows somewhat sine wave of change of congestion window. Shortcoming of this algorithm is that it has large number of packet drop as compared to traditional slow start. It is also noted that at the end of 15th iteration, average throughput of M-start is 1095 Mbps and traditional slow start is

549 Mbps. So, M-start fills 75% of available bandwidth as compared to traditional slow start as indicated in figure 6, tradition approach takes 4.10101 seconds to reach the saturation point at 454 packets while proposed approach takes 2.10097 seconds to reach the saturation point at 1992 packets.

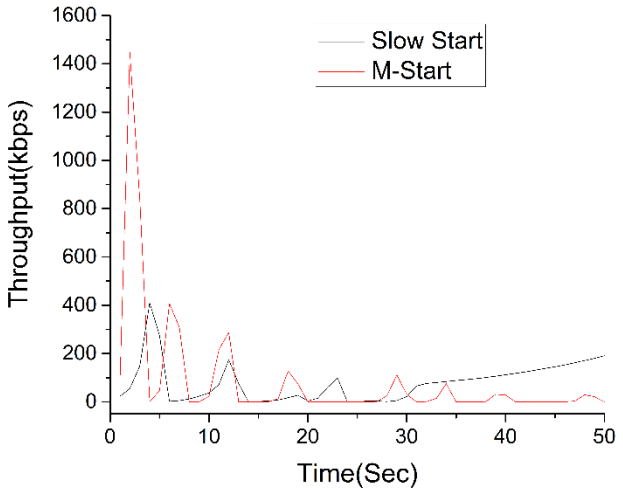

Figure 7. Throughput Comparative Result

Figure 7 and table 4 indicates simulated comparative result of traditional approach and proposed approach. M-Start approach is 51.23% faster as compared to traditional approach. Epoch point of M-Start is also 4 times higher than traditional approach. The graph indicates that tradition approach increases by one step while proposed approach increases with more aggressively with exponential approach.

As shown in table 4 modified slow start having problem of more freezing time period compared to traditional slow start. It can be seen from table that at 4th, 8th, 9th, 13th, 14th & 15th second throughput is near to zero.

It means sender is not sending any packet into network. Main cause for these phenomenon is waiting time for the acknowledgment is higher because of too many packet loss occurring at the end of RTT iteration.

7. Conclusion

In this paper, causes of low bandwidth utilization in slow start of TCP is investigated and proposed new method of slow start named as Modified start. This paper proposes changes to improve step up count of slow start method of congestion control used in TCP. Proposed method is tested in a simulated environment. It significantly improves throughput performance during slow start. Experiments and results, observed during slow start phase, show that throughput has increased to 51.23%, time to reach epoch has reduced to 40%, position of epoch point has increased to 4 times higher than traditional approach. Future work involves tuning initial parameter, P, with respect to different network conditions and different types of traffic. Also, finding the safe exit point to reduce packet loss.

Table 4. Simulated Throughput per second and Average Throughput over time

	Traditional Slow Start		Modified Slow Start (M-Start)	
Time (Sec)	Average Throughput	Throughput per second	Average Throughput	Through put per second
1	23	23	102	102
2	67	55	1498	1447
3	192	148	1840	841
4	555	410	1381	1
5	718	274	1151	46
6	601	2	1367	407
7	521	5	1481	309
8	468	12	1296	0
9	439	23	1152	0
10	432	37	1060	23
11	465	71	1180	216
12	601	174	1369	287
13	631	76	1264	0
14	587	0	1174	0
15	549	1	1095	0

References

[1] A. Technologies, *Q4 2015 state of the internet connectivity report.*

[2] W. Zhang, K. T. Lam, C.-L. Wang, “Adaptive live vm migration over a wan: Modeling and implementation.” in: *IEEE CLOUD*, IEEE, pp. 368–375, 2014.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, “Live migration of virtual machines.”, in: A. Vahdat, D. Wetherall (Eds.), NSDI, *USENIX*, 2005.

[4] V. Jacobson, M. J. Karels, “Congestion avoidance and control”, *ACM Computer Communications Review* 18 (4), pp.314–329, 1988.

[5] Y. Zhang, N. Ansari, M. Wu, H. Yu, “Afstart: An adaptive fast tcp slow start for wide area networks.”, in: *ICC, IEEE*, pp. 1260–1264, 2012.

[6] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, B. Raghavan, “TCP fast open.”, in: K. Cho, M. Crovella (Eds.), CoNEXT, *ACM*, p. 21, 2011.

[7] C.Y. Ho, Y.C. Chan, Y.C. Chen, “Gallop-vegas: An enhanced slow-start mechanism for tcp vegas.”,

Journal of Communications and Networks 8 (3) pp.351–359, 2006.

[8] S. Ha, I. Rhee, "Taming the elephants: New tcp slow start.", *Computer Networks* 55 (9) pp.2092–2110, 2011.

[9] X. Deng, Z. Chen, L. Zhang, "A parameterized model of tcp slow start.", in: H. Jin, G. R. Gao, Z. Xu, H. Chen (Eds.), NPC, Vol. 3222 of *Lecture Notes in Computer Science, Springer*, pp. 316–324, 2004.

[10] D. gan Zhang, K. Zheng, D. Zhao, X. dong Song, X. Wang, "Novel quick start (qs) method for optimization of tcp.", *Wireless Networks 22* (1) pp.211–222 2016.

[11] H. Wang, K. G. Shin, H. Xin, D. S. Reeves, "A simple refinement of slow-start of tcp congestion control.", in: *ISCC, IEEE Computer Society*, pp. 98–105, 2000.

[12] Y. Nishida, "Smooth slow-start: Refining tcp slow-start for large-bandwidth with long-delay networks.", in: *LCN, IEEE Computer Society*, pp. 52–60, 1998.

[13] K. I. Oyeyinka, A. O. Oluwatope, A. T. Akinwale, O. Folorunso, G. A. Aderounmu, O. O. Abiona, "TCP window based congestion control-slow-start approach.", *Communications and Network 3* (2) 85–98, 2011.

[14] A. Arcia-Moret, O. Diaz, N. Montavont, "A tunable slow start for tcp.", in: *GIIS, IEEE*, pp. 1–6, 2012.

[15] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme for tcp.", in: D. Estrin, S. Floyd, C. Partridge, M. Steenstrup (Eds.), *SIGCOMM, ACM*, pp. 270–280, 1996.

[16] K. Kumazoe, C. Marcondes, M. Gerla, D. Cavendish, M. Tsuru, Y. Oie, "Conservative slow start: Controlling losses in very high speed networks.", in: *ICC, IEEE*, pp. 5798–5803, 2008.

[17] R. Srikant, "*The Mathematics of Internet Congestion Control"*, 2004.

[18] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, N. Sutin, "An argument for increasing tcp's initial congestion window.", *Computer Communication Review* 40 (3) pp.26–33, 2010.