Received: April 1, 2017 30

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Agility based safety growth of Slow-Start Congestion Avoidance and Control Scheme in TCP

Pavithra Krishna Moorthy[1*] **Karthikeyan Easwara Moorthy[1]**

[1]*Department of Computer Science, Government Arts College, Tamil Nadu, India*
* Corresponding author's Email: pkpavikrishna@gmail.com

Abstract: Wired and Wireless networks are two types of challenging environment for TCP congestion control. Most of the congestion control algorithms have been proposed to improve the performance of TCPs in these two environments. Although these improved algorithm can improve network utilization, and perform excellently over disparate networks that contain both wired and wireless residue a good performance. In this study, the Enhanced Slow-Start algorithm, which can concentrate for avoiding heavy packet loss and improved network utilization for keeping a Congestion Window (CWND) increment/decrement manner, and perform very well while controlling packet loss with the standard TCP Reno algorithm. A series of experimental results to demonstrate the performance of Enhanced Slow-Start compared with other state-of-the-art algorithms. The performance of the proposed algorithm is proved to better when compared to the standard Slow-Start, Agile-SD, Reno, Vegas, Hybrid Congestion control Algorithms. The parameters used for testing are CWND size, packet delivery ratio, RTT value, Packet drop.

Keywords: Congestion control, Agility factor, Slow-start, TCP.

1. Introduction

Internet performance is tightly depending on Transmission Control Protocol (TCP) congestion control function. When the number of packets sent to the network is much more than the network capacity, congestion occurs in the network and thereby packets are dropped. TCP congestion control aims to adjust the sending rate of flows in order to reduce traffic and also congestion. Several variants of TCP are provides to improve the performance of the TCP [1]. TCP variants can be classified as loss-based algorithms such as TCP Reno [2], TCP NewReno [3] and delay-based algorithm such as TCP Vegas [4]. The Fast TCP [5], Hybrid start (Hystart) [6, 7] can enhance TCP performance over high loss wireless links but cannot fully adapt to the rapid growth of network utilization over both networks. The Compound TCP [8], High speed TCP (HSTCP) [9], Scalable TCP [10] and TCP CUBIC [11] achieve remarkable throughput in wired networks. These advanced protocol to improve the TCP performance in high-speed networks and to manage efficiency friendliness methods are above, as loss based protocol using RTT metrics have been proposed. E.g., Gentle High speed TCP [12], Compound TCP [13], TCP-LP [14], TCP Africa [15]. They can adaptively switch their congestion control phase to the congestion level measurement estimate from RTT. The Congestion Control Algorithm (CCA) is one of the main part of TCP. It significantly affects the overall performance of such networks, because it still suffering from the problem of heavy packet loss, network under utilization, especially if the applied buffer regime is very small. This under utilization of bandwidth is caused by the performance of the networks which results in either a slow growth of congestion window (CWND) or an over injection of data into the network.

In order to solve the problem of heavy packet loss over wired/wireless networks, a modified version of CCA namely Enhanced Slow-Start has been proposed. By increasing the CWND in safety manner and maintaining the sender window size. For each connection, at the sender limits the

maximum amount of unacknowledged traffic in transit in networks.

The rest of this paper is organized as follows. Related works are described in Section 2. The section 3 discusses the proposed approach based on agility to increase the CWND in safety manner. Section 4 presents the simulation results, and finally Section 5 bring concluding remarks.

2. Concept of congestion control

Basically, to control congestion, is adjust the window of data transmission at sender side in such a way that is preventing buffer overflow in the recipient, but also in the intermediate routers. To achieve this, TCP uses another variable to control congestion window (CWND). The congestion control represents a number of segments of appreciation that can be injected in the network without causing congestion. The challenge is to take advantage of the available space in the network routers. Routers do not participate in the TCP layer and cannot be used to adjust the TCP ACK frame. To resolve this problem, TCP assumes network congestion as the retransmission timer expires and that it interacts with the network congestion by adjusting the congestion window using two algorithms, a Slow-Start and Congestion Avoidance, as shown in the Fig. 1.

In the slow start phase when the connection is established, initially set the value of *CWND* to 1 and then each received *ACK* value is updated as *CWND* = *CWND* ×2 which means doubling the *CWND* per *RTT*. The rapid growth of *CWND* continues until the packet loss was observed, causing the value of Slow Start Threshold (SSThreshold) is updated as *SSThreshold* = *CWND* / *2*.After losing the packet, the connection starts from slow start again with *CWND* = *CWND* ×2 and is increased exponentially until the window is equal to *SSThreshold*, the estimate of available bandwidth in the network. At this point, it goes to the congestion avoidance phase, where the value of CWND is less aggressive with the pattern *CWND = (CWND + (1/CWND))*, which implies a linear rather than exponential growth and also continue to increase until it incur the packet loss.

3. Related works

Congestion detection mechanism is an important module of TCP algorithms. The most widely used congestion detection mechanism is adopted by many popular TCP algorithms such as TCP Reno [2, 16]

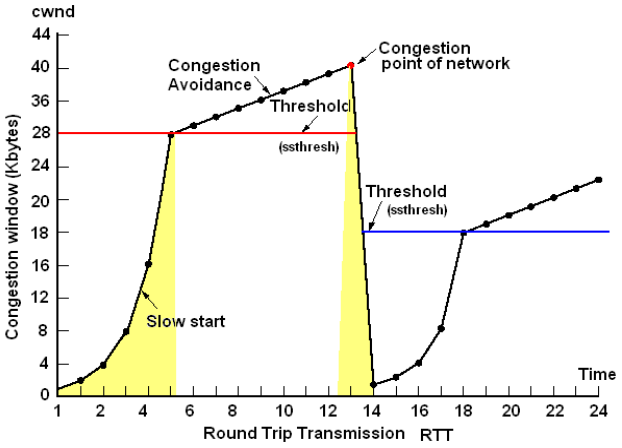

Figure. 1 TCP slow-start and congestion avoidance phase

and CUBIC [11] which inputs based on packet loss(loss based TCP) and used low speed wired and large Bandwidth Delay Product (BDP) environment. Delay based (queuing delay) Vegas [4] used low speed wired environment. Delay and Loss based Compound TCP [8, 17], TCP BIC [18], TCP Veno [19] used in Large BDP and TCP friendliness and TCP Westwood [20, 21]. TCP Westwood uses queuing delay to measure network bandwidth and thus avoid the impact packet losses to TCP congestion control in wireless networks.

TCP Veno adopts queuing delay as an index to adjust AIMD parameters for different random packet loss rates in fast-start-up [22, 23]. The Agility based Agile-SD [24] is used in high speed and short distance networks. The performance of these TCP variants are good for the application scenarios for which they were originally designed. However, an emerging generation of high-bandwidth wireless networks and heterogeneous networks that contain segments of both large BDP and wireless links still presents challenges to such algorithms.

3.1 TCP reno

The Reno retains the basic principle of Tahoe, such as slow starts and the coarse grained retransmit timer [16]. Reno performed very poorly if connection suffered from multiple packets dropping in one window of data. These because of Reno need to wait for the expiration timer of retransmission before restarting flow of data. Reno is applied diverse algorithm to control the network congestion which consists of four phases; Slow Start, congestion avoidance, fast retransmit and fast recovery. Reno is tried to exploiting the losses in packets to determining the existing bandwidth capacity in the network. It execute Slow Start

procedure in the TCP connection beginning as well as when timeouts within connection. In this progression the Reno primarily growths an exponential manner of congestion window and linearly when reaches Slow Start Threshold (SSThreshold) level to start the other phase known by congestion avoidance. When timeout occurs or if three duplicate ACKs are received, fast retransmit and fast recovery is initiated, where these algorithms enhancing the Reno performance by using the timeout interruption to indicate the congestion in network [2]. The congestion control of Reno does not decrease the transmission flow rate except if it notes a dropping in packet and that will happen only if network suffer from overload situation. Reno try to balance the size of window for different connections. The size of window in Reno is regularly changed in a distinctive situation. The size of window stays to be enlarged till packet loss happens. Whenever Reno receive 3 duplicate ACK‘s it will take it as a sign that the segment was lost, so the Reno re-transmit the segment without waiting for timeout. Thus it manage to re-transmit the segment with the pipe almost full. Another modification that Reno makes is in that after a packet loss, it does not reduce the Congestion Window (CWND) to 1.

3.2 TCP vegas

Vegas is a Transmission Control Protocol (TCP) implementation which is a modification of Reno [4]. It builds on the fact that proactive measure to encounter congestion is much more efficient than reactive ones. It tried to get around the problem of coarse-grained timeout by suggesting an algorithm which checks for timeout at a very efficient schedule. Also it overcomes the problem of requiring enough duplicate acknowledgements to detect a packet loss, and it also suggests a modified slow start algorithm which prevents it from congesting the network. It does not depend solely on packet loss as a sign of congestion. It detects congestion before the packet loss occur. However it still retains the other mechanism of Reno and Tahoe, and a packet loss can still be detected by the coarse-grained timeout of the other mechanisms fail. Vegas exploits the simple idea that the number of bytes in transit is directly proportional to the expected throughput. In Vegas the modified Slow Start mechanism which allows an exponential growth of Congestion Window (CWND) at every other Round Trip Time (RTT) and in between compares its current transmission rate with the expected rate see to the path has still some point to increase. Vegas maintains an estimate RTT minimum of the minimum measured RTT, corresponding to the RTT encountered when the bottleneck queue is empty. It allows an exponential growth of CWND at every other RTT and also, compares its current transmission rate with the expected rate to see whether the path has still some point to increase. The modified slow start of Vegas is known to incur a premature termination of slow start because of an abrupt increase of RTT caused by temporal queue buildups in the router during bursty TCP transmission. TCP Vegas records the smallest measured round trip time as Base RTT and computes the available bandwidth as:

Expected Bandwidth = Window Size/ Base RTT

Here Window Size is measured by current window size. During the packet transmission the RTT of packets are recorded. The actual throughput is calculated as:

Actual Bandwidth = Window Size/ RTT

The difference between the Expected Bandwidth and Actual Bandwidth is used to adjust the Window Size. The difference is calculated as:

Diff = Expected Bandwidth -Actual Bandwidth

If the actual throughput is smaller than the expected throughput, TCP Vegas takes this as indication of network congestion, and if the actual throughput is very close to the expected throughput, it is suggested that the available bandwidth is not fully utilized, so TCP Vegas increases the window size. This mechanism used in TCP Vegas to estimate the available bandwidth does not purposely cause any packet loss. Hence the oscillatory behavior is removed and a better throughput is achieved. Retransmission mechanism used by TCP Vegas is more efficient as compared to TCP Reno as it retransmits the corresponding packet as soon as it receives a single duplicate ACK and does not wait for three ACKs. TCP Vegas as compared to TCP Reno is more accurate and is less aggressive, thus it does not reduce its CWND unnecessarily. It has problems when packets do not follow the same route and when large delays are present. When routes change for a certain TCP Vegas flow, the Base RTT recorded from the previous route is not become unstable when there is large network delay for a flow; later established connections cannot get a fair share of the bandwidth, and when they coexist with TCP Reno connections, TCP Reno connections use most of the bandwidth.

3.3 Hystart (Hybrid start)

In general, the standard TCP doubles its congestion window (CWND) in every Round Trip Time (RTT) during Slow-start. However exponential growth of CWND results in burst packet loss. Hystart [25] reduces burst packet losses during Slow-Start and hence achieves better throughput and lower system load. This algorithm does not change Hybrid start [6] (Hystart) that finds a "safe" exit point of Slow-Start at which Slow-Start can finish and safely move to congestion avoidance without causing any heavy packet losses. This algorithm does not change the doubling of CWND during Slow-Start [7] but based on round trip delays it heuristically finds safe exit point at which it can finish Slow-Start and move to congestion avoidance, before CWND overshooting. When packet loss occurs during Slow-start, Hystart behaves the same way as the Slow-start.

3.4 Agile-SD

Agile-SD [13] mechanism which is geared to work on high-speed and short-distance networks to enhance the overall performance and bandwidth utilization while preserving the fairness. Agile-SD initializes its CWND by 2 packets in order to focus on the impact of Congestion avoidance on bandwidth utilization.

4. Proposed algorithm

4.1 Agility based safety growth enhanced slow-start algorithm

In this section we describe the enhanced Slow-Start algorithm that avoid and control the congestion, reduce the heavy packet losses during Slow-Start. This algorithm is a combination of Hybrid [25] and Agile-SD [24]. Main objective of this algorithm is to avoid the congestion before receiving three duplicate Acknowledgement (ACK). The algorithm need to change the doubling of CWND during the Slow-Start after finding the safe exit point, and also finding the agility factor mechanism to solve the problem of bandwidth under utilization over high speed networks. The latest studies have deployed that all of the current TCP variants have different to handle bandwidth utilization and queuing buffer. But every algorithm check their position and start the process in ACK basis. Hence the congestion incur very first then after they need to reduce the data flow. So it is very essential that to avoid the congestion and also control the congestion. Thus it becomes very essential to design a new Congestion Avoidance Algorithm to avoid the starting level congestion and also queuing delay, then simultaneously increase a bandwidth utilization.

In this paper we have to compare our proposed work to a latest techniques are Agile-SD [24], Hybrid [25], and also compared to basic algorithms are Reno [2], Vegas [4], finally the proposed technique has performed very well to compare those algorithms.

4.2 Algorithm overall behavior

The intention of the Slow-Start is to find an appropriate sending rate and to follow the self-clocking mechanism. However the Slow-Start is not perfect in all situations. First, it takes a long time until a sender can fully utilize the available bandwidth on the path. Second the exponential increase may also be too strong and reason multiple packet drops if large CWND is reached. Finally the Slow-Start not ensures that new flows converge quickly to a reasonable share of resources.

The network limit C is computed using the following equation:

$$C=B+S \tag{1}$$

where B is a unused buffer space and S is the available buffer space. The congestion window size checked initially, if the value of CWND is greater than the C, next it wait for the three duplicate ACK. Suppose the *CWND* is less than the C, it check whether the *CWND* < *Slow-start* Threshold (*ssthreash*) value for control the exponential increment of the CWND.When the result is true it will increment the congestion window by exponentially as $CWND = CWND \times 2$, otherwise congestion window is modified as $CWND = CWND+CWND / 2$ and $CWND = CWND + CWND / 4$ manner. If the sender receives the three duplicate ACK this algorithm work in Agile-SD [13] basis. The main contribution of Agile-SD is the unique *CWND* growth function which relies on the agility factor mechanism which symbolized by λ, as shown in the following equation:

$$\lambda = \max\left(\frac{\lambda max\_gapcurrent}{gaptotal}, \lambda min\right) \tag{2}$$

$$gaptotal=max((cwnd\_loss-cwnd\_degraded),1) \tag{3}$$

$$gapcurrent=max((cwnd\_loss-cwnd),1) \tag{4}$$

Moreover, Fig. 2 shows the control flow diagram of Enhanced slow start algorithm.

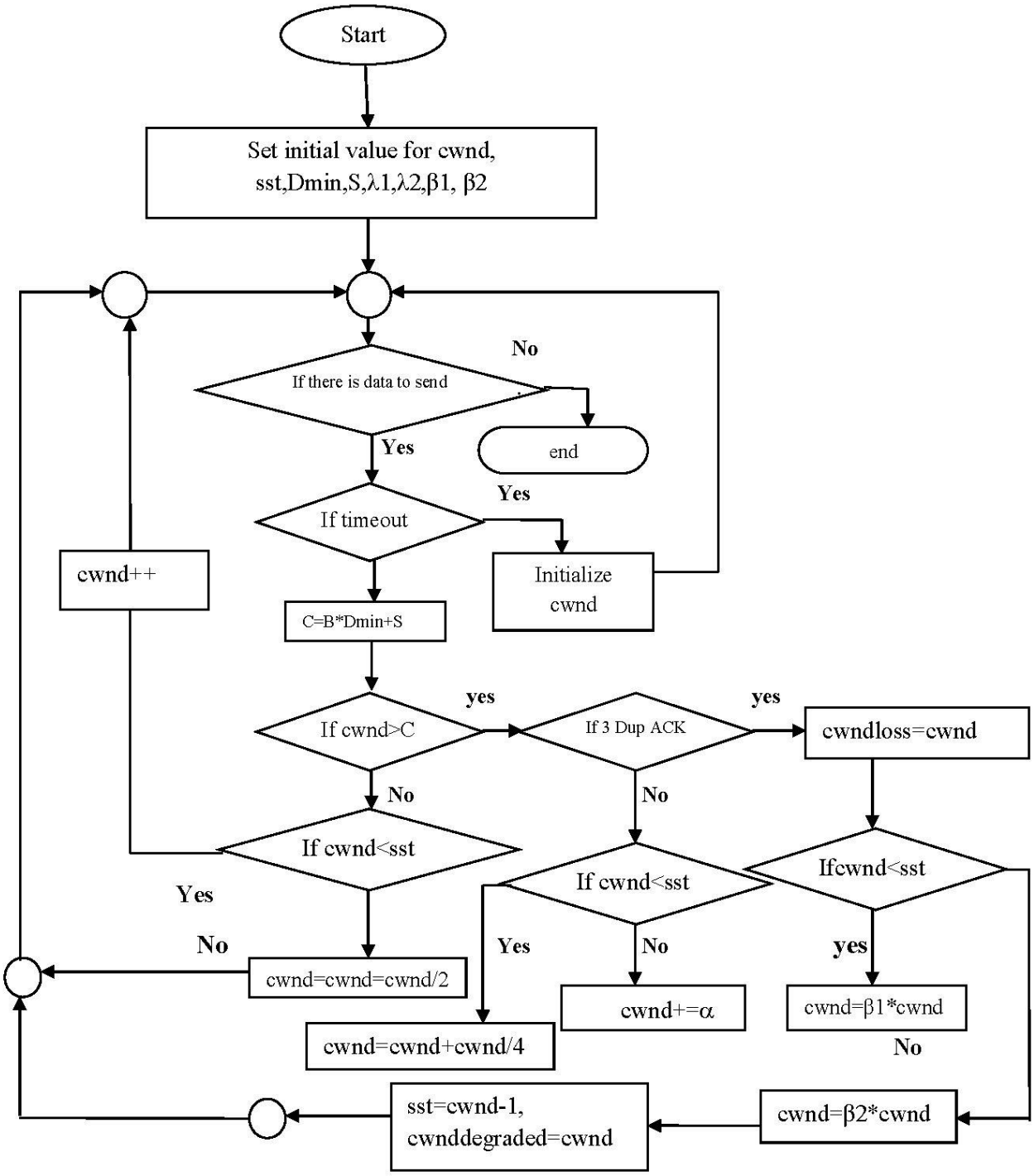

Figure. 2 The control flow diagram of modified slow start algorithm

The Algorithm - 1 explains the proposed strategy to enhance the overall performance and control the packet loss and improve the network utilization.

Algorithm–1: Enhanced Slow-Start Algorithm

Initialization:
λmin←1, λmax←3,
β1←0.90, β2←0.95,
cwnd ← 5
Event on No timeout C Value Reception do
Calculate C as in Eq. (1)
*cwnd = cwnd *2*
cwnd = cwnd + cwnd /2
End
Event on Ack Reception do
cwnd=cwnd+cwnd/4
Calculate gapcurrent as in Eq. (4)
Calculate gaptotal as in Eq. (3)
Calculate λ as in Eq. (2)
α= λ/ cwnd
cwnd ← cwnd +α

17 end
18 Event on Loss Detection of 3-duplicated Acks do
19 Cwnd Loss← cwnd
20 If tcp-status=slowstart then
*21 cwnd ← cwnd * β1*
22 else
*23 cwnd ← cwnd * β2*
24 end
25 ssthresh← cwnd -1
26 cwnddegraded← cwnd
27 end

4.3 Improved round trip time (RTT) independent

The RTT independent epoch time method enhances the performance of Agile-SD [24] algorithm. In RTT dependent every transaction depending to RTT and followed time out for every segment. So need to wait the time out for new transmission so the delay increased and also network utilization decreased. In that situation, to overcome the problem of unnecessary delay, the RTT independent epoch time methodology is utilized. This methodology used to reduce the delay, and also increase the network utilization. Agile-SD [24] increases its CWND independently from the Round Trip Time (RTT). The epoch time needed by the standard TCP, which is "RTT-dependent" [9], is the number of needed cycles time RTT, so it will equal to 80ms. For more understanding, assume that there is a TCP link with *cwnd_loss* = 12*; cwnd_degraded* = 9 and a constant *RTT* equal to 20ms, and the congestion avoidance stage is just started after the loss directly. Thus, the number of cycles needed by any *CCA* to reach cwnd_loss is 4 cycles is equal to (*cwnd_loss–cwnd_degraded*+1)

which consequently, the epoch time needed by the standard TCP, which is "RTT-dependent", is the number of needed cycles times *RTT*, so it will be equal to 80ms. Instead, Agile-SD increases its *CWND* independently from the RTT. Thus, every cycle consumes a time of $RTT = \lambda$ to send a number of $CWND = \lambda$ packets during that cycle, then it increases its *CWND* by 1.

Epoch Time is the number of needed cycles. Suppose λmin and λmax are set to1and 4, respectively. So, λi will take the value of (4, 3, 2, 1) sequentially, which will result in an epoch time equal to 41.66ms. Thus, the epoch time of Agile-SD [24] will be shrunk by around 48% from the epoch time of the standard TCP on the same network link. The epoch time of this algorithm will be shrunk by around 50% from the epoch time of the Agile-SD on the same network link calculated by interval difference of RTT.

Table 1. Experimental Parameter

Parameter	Value
CCA	Standard Slow-Start, Reno, Vegas, Hybrid, Agile-SD, Enhanced Slow-Start
Link capacity	0.2 mb to all
Link delay	1ms
Buffer size	10 Packets
Packet Size	1000bytes
Queuing algorithm	Drop tail
Traffic type	TCP

$$EpochTime = \sum_{i=1}^{k} \frac{MaxRTT - LastRTT}{\lambda i} \quad (5)$$

This behavior helps this algorithm to increase its CWND more quickly and consequently improves the bandwidth utilization.

Here also we have to compare our proposed work to a latest techniques are Agile-SD [24], Hybrid [25], and also compared to basic algorithms are Reno [2], Vegas [4], finally the proposed technique has performed very well to compare those algorithms.

5. Performance evaluation of enhanced slow start

5.1 The experiments setup

The proposed work is tested using the standard well known simulator called NS2. The result is obtained with three main scenarios such as single-flow, packet send, packet drop, CWND stability. The parameters used for simulation is given in Table 1.

5.2 Results and discussion

This subsection presents an analytical discussion of the behavior exhibited by the proposed algorithm and compared to various congestion control algorithms. As well as, it presents the results of the performance evaluation and shows the measurements of the CWND, loss ratio, send ratio.

5.2.1 The CWND evolution

Fig. 3 shows the CWND evolution of the studied Congestion Control Algorithms (CCAs) based on

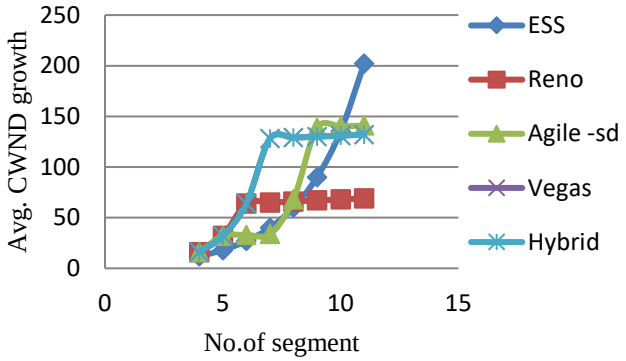

Figure. 3 The congestion window evolution for different TCP variants with enhanced slow-start.

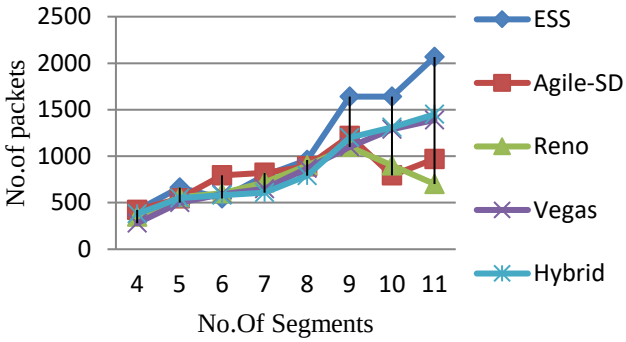

Figure. 4 Packet send ratio for different congestion control algorithm with enhanced slow-start

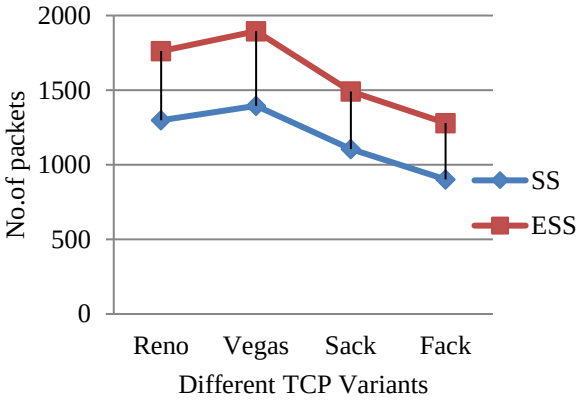

Figure. 5 Packet send ratio for standard slow-start (SS) and enhanced slow-start of different TCP variants

the buffer size change. Due to the mechanism of agility this method expectedly the average CWND growth to compare other congestion control techniques as followed.

5.2.2 Packet send ratio

In the next scenario, as shown in Fig.4, Enhanced Slow-Start (ESS) has overcome the various Congestion Control Algorithm (CCA) in

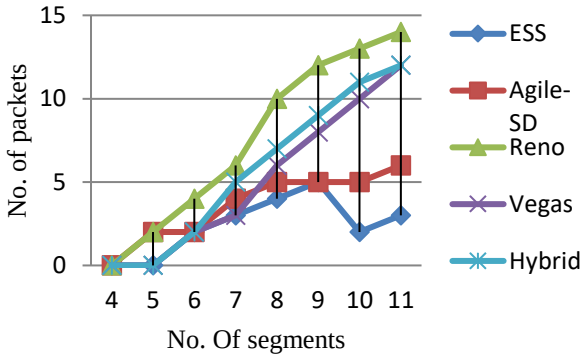

Figure. 6 Packet drop ratio for various slow-start and enhanced slow-start

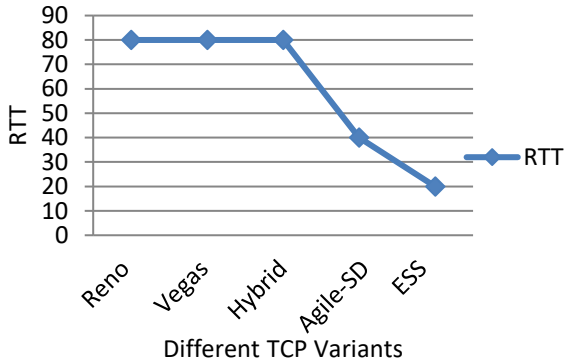

Figure. 7 RTT level for different congestion control algorithms

terms of sending rate due to its safety growth of CWND resulted by the combination of safe exit point and agility factor.

In the next scenario, as shown in Fig. 5, Enhanced Slow-Start (ESS) has performed in different TCP variants better than the other CCAs in terms of sending rate due to its safety growth of CWND.

5.2.3 Packet drop ratio

In the next scenario, as shown in Fig. 6, Enhanced Slow-Start (ESS) has control the packet drop to compare other Congestion control algorithms (CCA) in terms of safety growth of congestion window.

5.3 RTT independent

In the next scenario, as shown in Fig. 7, RTT level of different Congestion control algorithms. Here the result in an epoch time equal to 20ms. Thus, the epoch time of this method will be shrunk by around 80%, 50% from the epoch time of Agile-SD [24], Hybrid [25], Reno [2], Vegas [4], finally the proposed technique has performed very well to

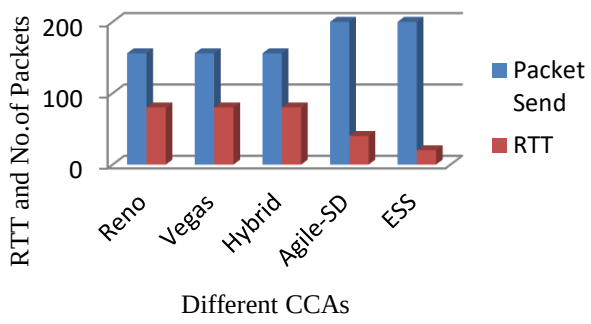

Figure. 8 Packet send ratio for different CCAs with RTT independent ESS

compare those algorithms. This behavior helps this method to increase its CWND more quickly than the another compared other CCA and consequently improves the bandwidth utilization.

As we can see from the graph the sending rate reach the maximum in RTT independence to compare RTT dependence and also decrease the waiting time for change a CWND this technique has smoothly increase bandwidth utilization and also ESS is better than other CCAs. The normal approach has increase the CWND in Congestion avoidance and standard Slow-Start manner, but the Enhanced Slow-Start approach increased the Congestion window in safety manner, hence Packet Send ratio is automatically increased in Enhanced Slow-Start method. From the results obtained, it is observed that the performance of network.

6. Conclusion

In this paper, the performance of the network is enhanced in terms of packet delivery ratio, delay, packet drop. The main contribution of this Congestion Control Algorithms (CCAs) is to implement the mechanism of agility factor. The need of the proposed CCA has the inability of the existing standard TCP CCAs in achieving a full bandwidth utilization over high-speed networks, especially when a small buffer regime is applied. Our enhanced Slow-Start protocol, detecting safe exit points of Slow-Start that does not lead to heavy packet losses or low network utilization, preventively avoiding heavy system overload or low performance during the start-up of TCP. It uses the concept of packet trains and RTT delay increase to find the safe exit points. Main objective of this algorithm is to avoid the congestion before receiving 3 Ack. The algorithm need to change the doubling of CWND during the Slow-Start after finding the safe exit point, and also finding the agility factor mechanism to solve the problem of bandwidth under utilization over high speed networks. Finally, this method helps this algorithm to increase its CWND in safety manner and consequently improves the bandwidth utilization and control the packet loss smoothly. The result shown prove that the proposed algorithm, ESS performs well when compared to Agile-SD [24], Vegas [4], Reno [2], and Hybrid [25]. In the future, the proposed work is integrated with Retransmission Time Out (RTO) based congestion avoidance, for improving the performance of TCP.

References

[1] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links", *IEEE/ACM Transactions on Networking*, Vol.5, No.6, pp.756-769, 1997.

[2] V.V. Jacobson, "Congestion avoidance and control", In: *Proc. of the ACM SIG COMM.*, pp. 314–329, 1988.

[3] S. Floyd and T. Henderson, "The New reno modification to TCPs fast recovery algorithm", *IETF Network Group*, RFC 2582, 1999.

[4] L.S. Brakmo and L.L. Peterson, "TCP vegas: end to end congestion avoidance on a global internet", *IEEE Journal Sel. Areas Communication*, Vol. 13, No.8, pp. 1465–80, 1995.

[5] D.X. Wei, C. Jin, S.H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance", *IEEE/ACM Transactions on Networking*, Vol.14, No.6, pp.1246-1259, 2006.

[6] W. Xu, Z. Zhou, D. Pham, C. Ji, M. Yang, and Q. Liu, "Hybrid congestion control for high-speed networks", *Journal of Network and Computer Applications*, Vol. 34, No. 4, pp. 1416–28, 2011.

[7] K. Kaneko, T. Fujikawa, Z. Su, and J. Katto, "TCP-Fusion: a hybrid congestion control algorithm for high-speed networks" In: *Proc. of PFLDnet ISI*, pp. 31–36, 2007.

[8] K. Tan and J. Song, "Compound TCP: a scalable and TCP-friendly congestion control for high-speed networks", In: *Proc. of the 4th international workshop on protocols for fast long-distance networks*, pp. 80–83, 2006.

[9] M.C. Weigle, P. Sharma, and J.R. Freeman, "Performance of competing high-speed TCP flows", *Lecture Notes in Computer Science*, Vol.15, No.3976, pp.476, 2006.

[10] R. Morris, "Scalable TCP congestion control", In: *Proc. of the Nineteenth Annual Joint Conference of the IEEE Computer and*

Communications Societies, Vol.3, pp.1176-1183, 2000.

[11] S. Ha and I. Rhee, "CUBIC: a new tcp-friendly high-speed TCP variant", *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel*, Vol.42, No.5, pp. 64–74, 2008.

[12] K. Yamanegi, T. Hama, Gohasegawa, M. Murata, H. Shimonishi, and T. Murase, "Implementation Experiments of the TCP Proxy Mechanism", In: Proc. of the *6th Asia – Pacific symposium, Information and Telecommunication Technologies, APSITT*, pp. 17-22, 2005.

[13] K. Tan. J. Song, Q. Zhang, and M.A. Sridharan, "A compound TCP approach for high-speed and long distance networks", In: *Proc of INFOCOM*, 2006.

[14] A. Kuzmanovic and E. W. Knightly, "TCP-LP: Low priority service via end point congestion control", In: *Proc. of the IEEE/ACM Trans. Networking*, Vol.14, No.4, pp. 739-752, 2006.

[15] R. King, R. Baraniuk and R. Riedi, "TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP", In: *Proc. of the IEEE INFOCOM*, pp. 1838-1848, 2005.

[16] A. Mohamed, O. Mohamed, A. Borhanuddin, and H.Z. Mohd, "Comparative study of high-speed Linux TCP variants over high-BDP networks", *Journal of Network and Computer Application*, Vol. 43, pp. 66–75, 2014.

[17] C. Testa and D. Rossi, "Delay based congestion control: Flows vs Bittorrent swarm Perspectives", *Elsevier, Computer Networks*, Vol. 60, pp. 115–128, 2014.

[18] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion control (BIC) for fast long-distance networks", In: *Proc. of the Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 4, pp. 2514–2524, 2014.

[19] C. Fu and S. Liew, "TCP enhancement for transmission over wireless aceess network", *IEEE Journal of Set. Areas. Communication*, Vol. 21, No. 2, pp. 216-228, 2003.

[20] R. Wang, S. Mascolo, C. Casetti, M. Gerla, and M. Sanadidi, "TCP Westwood, Bandwidth estimation for enhanced transport over wireless links", In: *Proc. of ACM mobicom*, pp. 287-297, 2001.

[21] D. Kliazovich, F. Granelli, and D. Miorandi, "Logarithmic window increase for TCP Westwood for improvement in high speed, long distance networks", *Computer Networks*, Vol. 52, No. 12, pp. 395-410, 2008.

[22] M. Scharf, "Comparison of end to end network supported fast start up congestion control schemes", Elsevier , *Computer Networks*, Vol. 55, No. 8, pp. 1921-1940, 2011.

[23] K. Xu, Y. Tian, and N. Ansari, "Improving TCP performance in integrated wireless communications networks", *Computer Networks*, Vol.47, No.2, pp.219-237, 2005.

[24] M.A. Alrshah, M. Othman, B. Ali, and Z.M. Hanapi, "Agile-SD: a Linux-based TCP congestion control algorithm for supporting high-speed and short-distance networks", *Journal of Network and Computer Applications*, Vol.55, pp.181-190, 2015.

[25] S. Ha and I. Rhee, "Taming the elephants: New TCP Slow-Start", *Elsevier Journal of Computer Networks*, Vol. 55, pp. 2092–2110, 2011.