Received: April 15, 2018

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Discrete TCP: Differentiating Slow Start and Congestion Avoidance

Bhavika Gambhava [1]* **Chandu Bhensdadia [2]**

[1]*Charotar University of Science and Technology, India*
[2] *Dharmsinh Desai University, India*
* Corresponding author's Email: gambhava.b@gmail.com

Abstract: Deployment of wireless links (terrestrial and satellite) along with wired links has made extension of the Internet even in remote places feasible. TCP/IP protocol suite is an integral part of the Internet. Congestion control of TCP plays a vital role in the performance of the Internet. TCP's unconditional flow control in case of a packet loss has always been a concern for researchers. Further, halving congestion window in such conditions without taking in to account the current network state is also considered inappropriate. The problem is compounded in wireless networks where packet losses occur often due to channel errors rather than the shortfall in the available bandwidth. In this situation, TCP's conservative behaviour underutilises the bandwidth. We therefore, propose a scheme to address the issue of underutilization of network resources. The proposed approach, Discrete TCP (DTCP), differentiates slow start and congestion avoidance phases while tuning data flow over a transport connection. DTCP evaluates *ssthresh* and *cwnd* before setting up parameters, based on the existing network condition to enhance the performance. The proposed scheme is compared and analyzed with various existing schemes with the help of extensive simulations using ns2. Results of simulation based experiments indicate significant performance improvement of DTCP on erroneous links and in heterogeneous networks and confirm its suitability.

Keywords: SACK TCP, Discrete TCP, Sow start, Congestion avoidance, Congestion control, ssthresh, cwnd.

1. Introduction

The World Wide Web has seen an immense growth in past couple of decades. Because of everyday expansion of Internet, there is a requirement for efficient protocols. HTTP (web browsing) and FTP (file transfer) are two widely used protocols over the Internet. At the transport layer, both utilize TCP (Transmission Control Protocol) at the transport layer [1]. In Internet, most of the traffic is TCP-based. Thus, TCP has an imperative role in the performance of the Internet. TCP is used in the Internet that supports many applications such as web access, file transfer and email. Due to its extensive use in the Internet, it is desirable that TCP remains in use to offer reliable services for communications in wireless networks and in heterogeneous networks.

TCP is a reliable end-to-end transport layer protocol designed for wired networks characterized by negligible random packet losses [2]. TCP keeps increasing the sending rate of packets as long as no packets are lost. Due to inherent reliability of wired networks, there is an assumption made by TCP that any packet loss is due to congestion. TCP will invoke its congestion control mechanism whenever any packet loss is detected. Most of the congestion control mechanisms reduce sending data rate to relieve the network from congestion. The reduction is not decided based on the degree of congestion and it affects the performance. If the sender has crossed a certain threshold, then a drastic reduction in the data flow results in inferior performance. If the sender is still probing the network capacity, data rate should be significantly decreased to help the sender stabilize according to available network resources. Most of the TCP variants do not take these circumstances into account while setting up the data rate and offer the same treatment in both the scenarios.

In this paper, we present a new approach to setting up apposite data flow based on the state whenever any loss is encountered. The transmitter sets data flow related parameters differently during slow start and congestion avoidance phases unlike conventional TCP. A TCP sender enters in congestion avoidance phase after crossing slow start threshold and increases data flow linearly to avoid possible congestion. Congestion Window (cwnd) is doubled per Round Trip Time (RTT) in exponential increase, while it is increased just by 1 in linear rise. Hence, reduction in *cwnd* should also be as per the current phase of the TCP sender. The proposed scheme follows the behaviour of SACK TCP whenever it is in the exponential increase, while a new algorithm is followed during the linear rise. This leads to improved network performance as well as robustness. The proposed scheme is compared with well-known existing versions of TCP by numerous simulations.

The rest of the paper is organized as follows. In the next section, we discuss the existing approaches with their limitations. A new scheme named, Discrete TCP is proposed and explained with a state diagram in Section 3. Simulation environment and topologies are discussed in Section 4. Simulation results are presented and analyzed in Section 5. We conclude the paper in Section 6.

2. Existing variants of TCP

In this section, we describe various TCP variants, which are used to compare with the proposed approach. TCP provides reliability by setting a retransmission timer when it sends data. In slow start phase, TCP increases *cwnd* each time an acknowledgement is received, by number of packets acknowledged [2]. This strategy effectively doubles TCP *cwnd* for every RTT. When *cwnd* exceeds a threshold named slow start threshold (*ssthresh*), it enters congestion avoidance phase. *cwnd* is increased by 1 for each RTT until a loss occurs. If the data is not acknowledged before expiration of the timer, it retransmits the data. TCP reduces *cwnd* to 1 when Retransmission timeout (RTO) takes place. It is because of the original design of TCP to operate over wired networks, where congestion was the main reason for packet losses.

TCP Tahoe is the first TCP variant, not depending on RTO to detect a packet loss [3]. In Tahoe TCP, a loss is detected by the arrival of three duplicate acknowledgements (*dupack*). When a loss is detected, fast retransmission is attempted. *ssthresh* is set to half of the current *cwnd* and slow start begins again from its initial *cwnd*. Tahoe TCP reduces *cwnd* to 1, which deteriorates performance of the connection.

TCP Reno involves fast recovery to reduce impact of *cwnd* reduction in contrast to TCP Tahoe [4]. When three *dupacks* are received, TCP Reno halves *cwnd*, performs a fast retransmit and enters fast recovery. Fast recovery sets new *cwnd* and *ssthresh*, both by half of the current *cwnd*. TCP Reno is effective to recover from a single packet loss, but it suffers when multiple packets are dropped from a window of data.

TCP New Reno tries to improve the TCP Reno's performance when a burst of packets is lost by modifying the fast recovery algorithm [5]. In TCP NewReno, a new data acknowledgement is not enough to take TCP from the fast recovery phase to congestion avoidance. Instead, it requires all the packets outstanding at the start of the fast recovery period are acknowledged. TCP NewReno assumes that the packet that immediately follows the partial acknowledgement received during fast recovery is lost, and retransmits the packet. However, this might not always be true because of reordering of packets and it affects the performance of TCP.

Selective Acknowledgement (SACK) TCP adds a number of SACK blocks in TCP header options, where each SACK block acknowledges a non-contiguous set of received data [6, 7]. SACK TCP's strength lies in its ability to avoid unnecessary retransmissions, based on SACK blocks available from the receiver. SACK TCP is able to recover from losses faster than New Reno TCP because of its ability to avoid retransmission of the packets which have certainly reached the receiver. By avoiding unnecessary retransmission, SACK TCP utilizes the available bandwidth more efficiently, which results in overall performance improvement. However, SACK TCP also does not reduce the data flow discreetly.

TCP Vegas [8] uses proactive measures to encounter congestion. It does not depend solely on packet loss as a sign of congestion. It detects congestion before the packet losses occur. It estimates the unacknowledged packets in the buffer of the bottleneck link. It maintains the minimum RTT as a reference to obtain the optimal/optimum throughput the network can achieve. However, it still retains the other mechanism of Reno, and a packet loss can still be detected by retransmission timeout if the other mechanisms fail. Issues identified with TCP Vegas are problems of rerouting, persistent congestion, and discrepancy in flow rate tied with starting times and link bandwidth [9].

Linux TCP [10] sender is governed by a state machine that determines the sender actions when

acknowledgements arrive. Linux implements a number of TCP enhancements proposed by Internet Engineering Task Force (IETF), such as Explicit Congestion Notification [11] and D-SACK [12].

The Forward Acknowledgements (FACK) algorithm [13] takes a more aggressive approach and considers the unacknowledged holes between the SACK blocks as lost packets. Although this approach often results in better TCP performance than the conservative approach, it is overly aggressive if packets have been reordered in the network, because the holes between SACK blocks do not indicate lost packets in this case.

TCP Fast Start [14] changes conventional TCP's slow start. The sender caches network parameters to avoid paying the slow start penalty for each page download. However, there is a risk of performance degradation if the cached information is stale. To shield the network as a whole from the ill-effects of stale information, packets sent during the fast start phase are assigned a higher drop priority than other packets.

AFStart TCP [15] dynamically sets *ssthresh* and *cwnd*. AFStart approaches *ssthresh* quickly than standard slow start. *Cwnd* is initialized with 4 packets. An abrupt increase of *cwnd* may acquire the available resources and which may force other traffic to get congested.

Novel Quick Start [16] optimizes slow start for the satellite communication networks. The value of *cwnd* is initialized to the detected network bandwidth. Error in estimated bandwidth is reduced in subsequent iterations. An abrupt change of *cwnd* may lead to congestion in the network.

An EQF (Explicit Queue-length Feedback) [17] uses the queue-length of the congested switch port as a congestion signal to trigger TCP congestion control for controlling the sending rate of the sender in a TCP connection.

Congestion control of Reno TCP or NewReno TCP is recently modified in [18 - 20]. Agility based safety growth enhanced slow start algorithm [18] tries to reduce Epoch time to increase *cwnd* quickly. This may lead to congestion if bandwidth estimation fails or concurrent traffic by other users increases. Slow start is modified to increase step up count to improve the efficiency [19]. However, packet drops are increased by 30% [19], which is wastage of network resources. Other connections can utilise the available bandwidth if these excess packet drops can be avoided. TCP LR-Newreno congestion control is an algorithm for IEEE 802.15.4 based standard [20]. It increases drop rate of the packets as compared to Vegas TCP when channel is error free. It also consumes more energy than Vegas TCP, which is a crucial parameter in wireless sensor networks (WSN).

Authors of [21] try to reduce the packet drops by proposing an enhanced queue management scheme for TFRC over wired networks. All the routers/intermediate nodes need to change the queuing mechanism which is a task. TCP variants demanding changes at sender or receiver are viable mechanisms.

3. Discrete TCP

TCP performance is strongly influenced by its congestion control algorithms that limit the amount of transmitted traffic based on the estimated network capacity. Most of the congestion control mechanisms assign new values to *cwnd* and *ssthresh* after an indication of the loss. It is conventional to halve *cwnd* whenever 3 *dupacks* are received by the sender. We propose an algorithm, which considers the state of the sender before setting up the parameters, *cwnd* and *ssthresh*. Hence, we call it a Discrete TCP. Discrete TCP (DTCP) tunes parameters differently after a packet loss based on slow start and congestion avoidance.

The sender doubles *cwnd* after each round trip time (RTT) in the slow start. *cwnd* is incremented only by 1 after an RTT if the sender is in the congestion avoidance. 3 *dupacks* trigger the sender to attempt fast retransmission and fast recovery is evoked after that. DTCP modifies the fast recovery based on the slow start/exponential increase or the congestion avoidance/linear rise. DTCP reduces *cwnd* and *ssthresh* by half of the previous *cwnd* in the slow start because the last doubling of *cwnd* probably caused congestion and resulted in the packet loss. DTCP reduces *cwnd* and *ssthresh* by 3/4 of the previous *cwnd* in linear rise because the last update in *cwnd* caused just one additional packet over the network. It is obvious that one additional packet cannot cause severe congestion and reduction of *cwnd* to half harms the network performance. It is also to be noted that this loss can also be due to errors of the wireless link in today's heterogeneous networks. Reducing *cwnd* by half in such circumstances is irrational. DTCP gracefully reduces *cwnd* to relieve the network from potential congestion, while improving the performance as discussed in a later section. The pseudo code of DTCP is given in Fig. 1.

The state diagram of DTCP is shown in Fig. 2. The behaviour of the slow start and the congestion avoidance phases are as per SACK TCP. Whenever *cwnd* exceeds *ssthresh*, the TCP sender enters the congestion avoidance from the slow start.

```
if cwnd < ssthresh
        cwnd_{n+1} = cwnd_n + 1                          // slow start
else
        cwnd_{n+1} = cwnd_n + 1/cwnd_n                   // congestion avoidance
if dupacks =3
        retransmit the lost packet                       // fast retransmission
if previous state = slow start
        ssthresh_{n+1} = cwnd_n / 2 & cwnd_{n+1} = ssthresh_{n+1}   // standard fast recovery
if previous state = congestion avoidance
        ssthresh_{n+1} = ¾ cwnd_n & cwnd_{n+1} = ssthresh_{n+1}     // modified fast recovery
if retransmission timer expiration
        ssthresh_{n+1} = cwnd_n / 2 & cwnd_{n+1} = 1     // timeout
```

Figure. 1 Pseudo code of discrete TCP

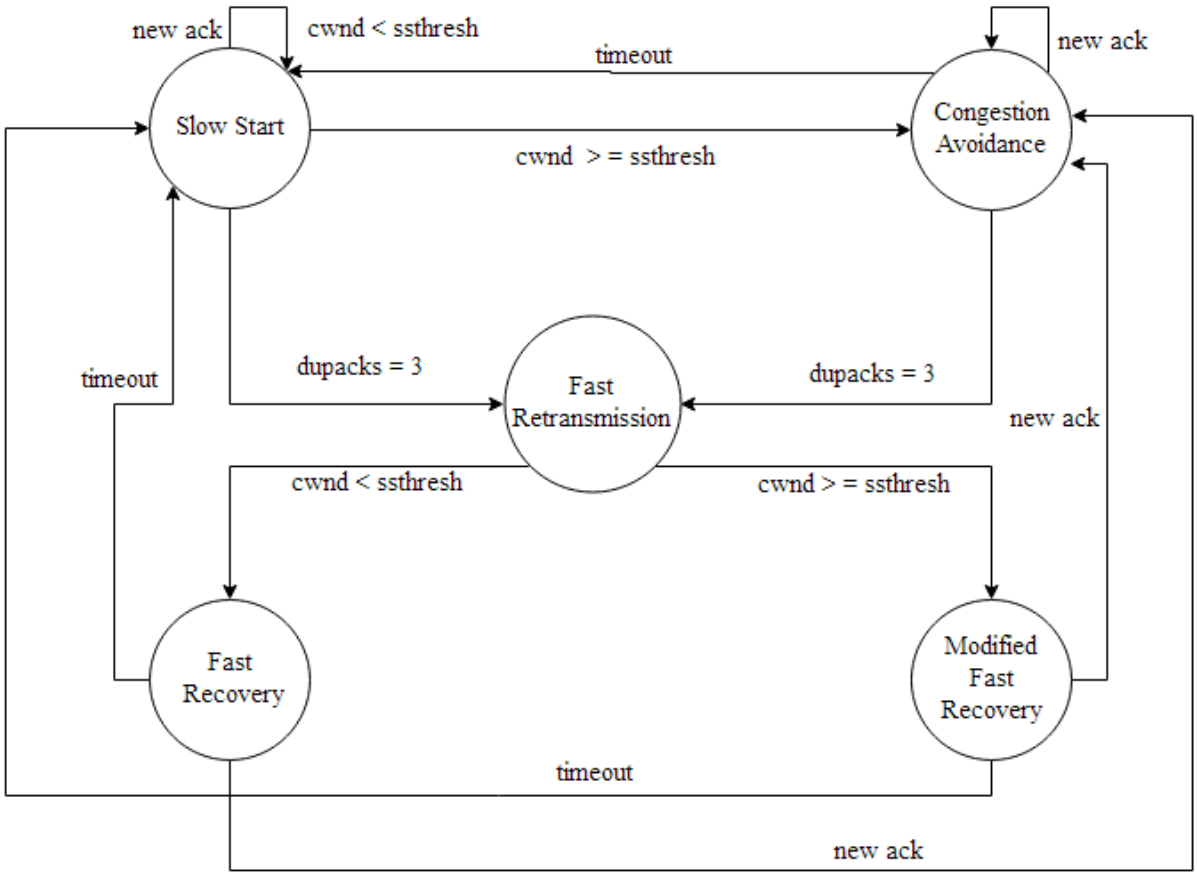

Figure. 2 State diagram of DTCP

The congestion avoidance is continued as long as new acknowledgements are received. Whenever a packet loss takes place on the network, *dupacks* are generated by the receiver. On arrival of 3 *dupacks*, the sender fast retransmits the lost packet and changes the state. If the previous state was slow start, then standard fast recovery is followed as shown in Fig. 2. Whereas modified fast recovery is followed if the previous state was congestion avoidance. Pseudo code of the proposed algorithm is shown in the figure.

No change is proposed in Discrete TCP for RTO. The sender completely follows SACK algorithm in the event of the timeout. DTCP does not require changes at intermediate nodes or at the receiver side. The TCP header and options are also not modified. DTCP needs changes only in SACK TCP sender implementation, which makes is easily deployable and also interoperable with senders and receivers, involving different variants.

4. Simulation environment

The performance of Discrete TCP was evaluated with several TCP variants, using simulations under identical conditions. The simulations were carried out using network simulator *ns-2*[22]. Selective Acknowledgement TCP(SACK)[6], Vegas TCP(Vegas)[8], Linux TCP(Linux)[10], Forward acknowledgement TCP(FACK)[13], Fast Start TCP(FS)[14], and Reno TCP(Reno)[4] are used for evaluation in *ns-2*. In order to examine the performance of DTCP, experiments were conducted for three types of environments: 1. only errors, no congestion 2. congestion but no error 3. errors and congestion.

The basic error model of *ns* was chosen for experiments. It simulates link-level errors by marking the packet's error flag or dumping the packet to a drop target. A random variable is uniformly distributed from 0 to 1 to cause packet drops according to the percentage error rate. Different error-rates (0.00,0.001 and 0.01) in terms of percentage of packets were configured during simulations to analyze the impact on the performance.

FTP traffic was generated for 100 seconds. The purpose of conducting simulations for longer duration is to examine the impact of changes on a settled TCP connection. Packet size was kept 1500 bytes to be compatible with Ethernet. All the parameters are summarized in Table 1.

Simulation Topologies

A simple network topology shown in Fig. 3, was used to evaluate the performance in the presence of only errors. Node 1, node 2 and node 3 are transmitter, router and receiver respectively. The propagation delay of all duplex links is 50 msec with data rate of 100 Mbps. The erroneous packets were dropped from the intermediate node 2, resulting in a gap in sequence numbers at the receiver node 3. Because of equal incoming and outgoing data rates at a router, congestion never takes place. Hence, all packet losses are because of corruption only. This scenario was created to characterize a simple wireless network. The performance of DTCP along with previously mentioned variants was tested in this scenario.

Table 1. Simulation parameters

Simulator	ns2
Application	FTP
Packet size	1500 byes
Link type	Full duplex
Time	100 sec
Error rates	0, 0.001, 0.01
Error model	Random. Uniform (0-1)
Queue length	50
Type of the Queue	Droptail

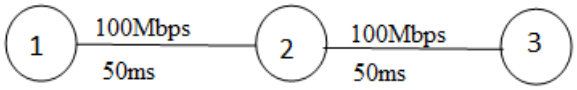

Figure. 3 Erroneous network topology

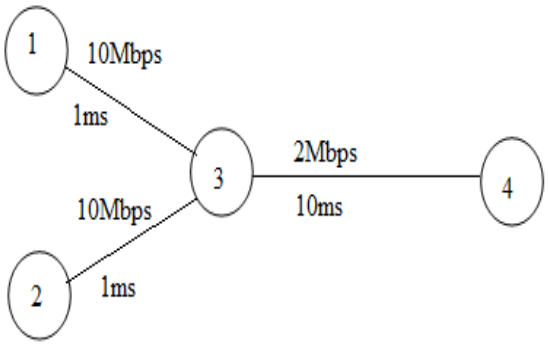

Figure. 4 Congested network topology

Simulations were repeated on a topology shown in Fig. 4. It shows a typical network with two sources (node 1 and node 2), transmitting on 10 Mbps full duplex link with a 1 msec delay. The link between router (node 3) and a common destination (node 4), is a bottleneck link with 2 Mbps data rate with 10ms delay. A router with a finite buffer of size 50, drops packets in case of overflow. This obviously creates congestion at a router because of total incoming traffic from sources is 20 Mbps whereas the capacity of the outgoing line is only 2 Mbps. In the second experiment, we assume error-free environment to study the impact of only congestion.

Simulations were also carried out on a topology shown in Fig. 5. The topology consists two sources (node 1 and node 2), transmitting on 10 Mbps full duplex link with a 1 msec delay. The link between router1 (node 3) and router 2(node 4) is a bottleneck link with 5Mbps data rate and 100ms delay. Source 1 and source 2 are sending data to destination 1(node 5) and destination 2(node 6) respectively. Router 2 is connected to destinations by 10Mbps full duplex link with 1ms Delay. Error model is configured at router1 to cause losses due to corruption also. Error rates were changed during simulations in the presence of congestion. The packet losses in this environment may be either due to congestion or because of errors. This environment resembles a heterogeneous network with wired and wireless links.

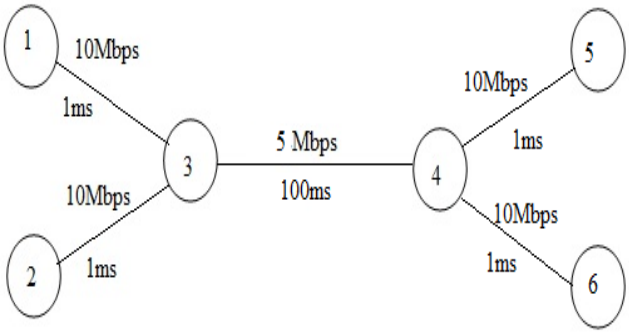

Figure. 5 Congested and erroneous network topology

5. Simulation results and analysis

The results are analyzed on the basis of number of packets successfully delivered over the simulation period of 100 seconds. The error rate is varied from 0.00 to 0.001 and 0.01 in order to check response of the network in absence of errors, in presence of moderate error rate and in presence of severe error rate. Results of three topologies are discussed in this section.

5.1 Errors without congestion

As the incoming and outgoing link capacity is identical in the topology of Fig. 3, no packet is dropped because of congestion.

(i) Error rate 0.00

Initially, the error rate was kept 0.00 to check the response of all variants in utopian condition. In this congestion-free and error-free network, TCP scheme is expected to deliver the best performance in absence of channel errors. However, the TCP performance is obviously constrained by its inherent dynamic parameter like *cwnd* apart from other network parameters like the channel bandwidth, delay and router queue length etc. Performance of all schemes is observed to be identical in absence of errors except Linux TCP as shown in Table 2. This indicates that these schemes adopt the conventional TCP behaviour in absence of packet losses. Our detailed examination revealed that Linux TCP delivered almost significantly (almost 6 times) more packets than rest of the TCP schemes in the same period. We confirmed aggressive behaviour of Linux TCP by observing its *cwnd* at intermediate points. All other TCP variants never send more than 50 packets in one RTT because of specified maximum window *(maxwnd).* Linux TCP does not restrict its data flow to 50 and sends up to 543 by the end of 100 seconds simulation. We carried out other simulations with higher *maxwnd* values and other TCP variants were found to perform same as Linux TCP. Note that, this aggressive data flow is not friendly with other competing connections in the network and it may hamper their performance. In the absence of errors, DTCP performs same as other TCP as expected. We verified here that no action is taken by our proposed algorithm if no loss is encountered. DTCP is activated only when any packet loss is detected in order to revise the sending rate of TCP judiciously.

(ii) Error rate 0.001

Error rate was then increased to 0.001 to check behaviour of various TCP variants in the presence of moderate link errors. Vegas TCP shows better performance than other variants because of its ability of estimating network bandwidth and accordingly adjusting *cwnd.* Note that Vegas TCP adopts the conventional behaviour along with conservative *cwnd* value after a packet loss. Linux TCP reduces *cwnd* frequently in case of random error losses without considering network condition. This leads to underutilization of network. The proposed algorithm, DTCP sets up higher data rate as compared to other TCP variants, which helps in using the available bandwidth promptly and more efficiently. The same can be observed in the third column of Table 2.

DTCP delivers only 7.8% lesser packets in presence of 0.001 error rate than the most favourable network condition of 0.00 error rate as compared to other variants, which reduce by 19.3% to 87.9% as mentioned in Table 2. Vegas TCP is second best to DTCP, while Linux TCP's aggressive data rate is controlled by link errors, which deteriorates performance greatly.

(iii) Error rate 0.01

Error rate 0.01 was also configured over the network to check the response in the presence of a highly error-prone link. DTCP surpasses all other

Table 2. Number of delivered packets for erroneous network

	error=0	error=0.001	error=0.01
SACK[6]	24762	15940	6331
Vegas[8]	24587	19836	8456
Linux[10]	146302	17567	6561
FACK[13]	24762	17009	6226
FS[14]	24762	14598	5259
Reno[4]	24762	16225	5049
DTCP	24762	22818	8671

protocols once again as observed from the last column of Table 2. The performance gain in the number of successfully delivered packets varies from 2.54% to 71.7% in DTCP against other protocols. DTCP performs 43.14% and 36.96% better than one of the widely used implementation SACK TCP for 0.001 and 0.01 error rates respectively. The rationale behind this improvement is pertinent *cwnd* setting while encountering a loss at the sender.

5.2 Congestion without errors

In order to examine compatibility with the existing terrestrial wired network, the next set of experiments was carried out in a congested environment without link impairments. Incoming data flow at a router, as shown in Fig. 4, is much higher than the capacity of the outgoing link. Multiple losses in a single transmitted window tend to be present due to congestion. DTCP restricts packet sending rate in a congested network as there is no scope of utilizing bandwidth further. No improvement can be expected in this case. Therefore, there was no valid reason for DTCP to set up cwnd according to the proposed modifications as most of the losses were from the slow start phase. The bottleneck link in Fig. 4 can carry up to 2×10^6 bits/sec or 25×10^4 bytes/sec. Thus, maximum 16666 packets of 1500 byes each can be sent in 100 seconds simulation period. All TCP versions perform nearly same as seen in Table 3. However, it is noteworthy that DTCP does not deteriorate network performance when deployed in severely congested environment.

5.3 Errors and congestion

Real networks are likely to suffer from errors and congestion together. Most of heterogeneous networks experience packet losses due to congestion as well as errors. The same was tested on topology shown in Fig. 5 to examine all the possibilities. The error rate was increased to 0.001 to evaluate behaviour in a realistic environment. DTCP transfers highest number of packets followed by Vegas TCP [8]. DTCP delivered 4.48% and 23.2% more packets than Vegas TCP and SACK TCP, respectively. When error rate was further increased to 0.01 to observe the impact of highly noisy wireless links, DTCP outperformed all other schemes considered for experiments. DTCP's performance was found to be 10.55% and 58.47% higher than Vegas TCP and SACK TCP, respectively. DTCP delivers 92% more packets than FS TCP in case of congested network having 0.01 error rate. Fig. 6 illustrates the percentage improvement of DTCP in terms of delivered packets over other TCP schemes. It can be observed from that DTCP performs better when higher error rates were encountered. It validates DTCP's ability to rightly assign *cwnd* according to the state of the network.

Table 3. Number of delivered packets for error-free congested network

	SACK	Vegas	Linux	FACK	FS	Reno	DTCP
Sender 1	7274	8334	8192	8148	8560	7366	8097
Sender 2	8983	8330	8084	8119	7610	8728	8182
Total Packets	16257	16664	16276	16267	16170	16094	16279

Table 4. Number of delivered packets for erroneous and congested network

	error=0	error=0.001	error=0.01
SACK	40217	30631	10592
Vegas	40289	36118	15183
Linux	39373	32306	10644
FACK	40217	30833	10630
FS	40217	27979	8705
Reno	40217	30567	9690
DTCP	40217	37739	16786

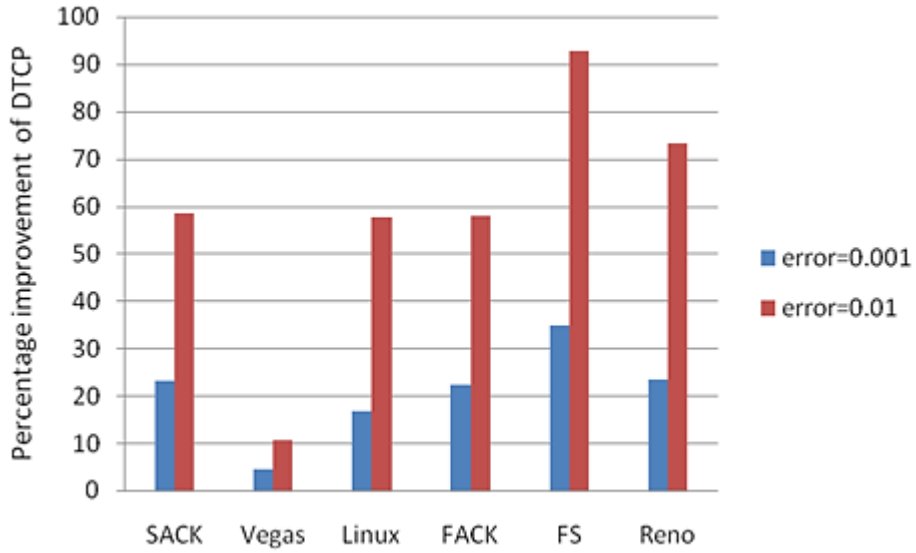

Figure. 6 Improvement of DTCP over other variants for erroneous and congested network

6. Conclusion

In this paper, we propose a modification to change congestion control algorithm of the conventional TCP with a more appropriate value of *cwnd* in case of a packet loss. The proposed scheme, Discrete TCP (DTCP) tunes data flow differently in case of slow start and linear rise. The objective is to use available bandwidth more efficiently.

The performance of the proposed scheme is evaluated over different topologies in presence of different error rates, delay, and levels of congestion. The performance of the proposed scheme is compared with other well-known TCP variants. Our observations are as under:

1. In absence of errors, performance of all TCP variants is observed to be same. DTCP adopts behaviour of the conventional TCP when links are error free. DTCP improves performance up to 56.3% and 71.7% over erroneous network with 0.001 and 0.01 error rates respectively.
2. There is no improvement in case of severe congestion, which in turn confirms network friendliness of DTCP.
3. Simulation results indicate substantial performance enhancement in DTCP as compared to many other TCP variants in the presence of errors in a congested network also.

All changes in implementation are confined to the sending side. No changes are needed in the TCP header, routers or at the receiver implementation. Hence, it is interoperable with any other TCP implementation at the receiver.

Acknowledgments

Authors would like to thank Dr. Nikhil Kothari and Dr. Brijesh Bhatt for their helpful suggestions and proofreading drafts of this paper.

References

[1] M. Duke, E. Blanton, A. Zimmermann, R. Braden, and W. Eddy, “A roadmap for transmission control protocol (TCP)”, *RFC 7414*, 2015.

[2] Y. Tian, K. Xu and N. Ansari, “TCP in wireless environments: problems and solutions”, *IEEE Communications Magazine*, Vol. 43, No. 3, pp.S27-S32, 2005.

[3] V. Jacobson, "Congestion avoidance and control", *ACM SIGCOMM Computer Communication Review*, Vol. 18, No. 4, pp. 314-329, 1988.

[4] M. Allman, V. Paxson, and E. Blanton, “TCP congestion control”, *RFC 5681*, 2009.

[5] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, “The NewReno modification to TCP's fast recovery algorithm”, *RFC 6582*, 2012.

[6] M. Mathis, M. Jamshid, S. Floyd, and A. Romanow, “TCP selective acknowledgment options”, *RFC 2018*, 1996.

[7] S. Floyd, M. Jamshid, M. Matt, and P. Matthew, “*An extension to the selective acknowledgement (SACK) option for TCP”, RFC 2883*, 2000.

[8] U. Hengartner, B. Jürg, and G. Thomas, "TCP Vegas revisited", In: *Proc. of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3, pp. 1546-1555, 2000.

[9] K. Srijith, J. Lillykutty, and A. Ananda, "TCP Vegas-A: solving the fairness and rerouting issues of TCP Vegas", In: *Proc. of IEEE conference on Performance, Computing and Communications*, pp. 309-316, 2003.

[10] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP", In: *Proc. of USENIX Annual Technical Conference, FREENIX Track*, pp. 49-62, 2002.

[11] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP", *RFC 3168*, 2001.

[12] E. Blanton and M. Allman, "Using TCP duplicate selective acknowledgement (DSACKs) and stream control transmission protocol (SCTP) duplicate transmission sequence numbers (TSNs) to detect spurious retransmissions", *RFC 3708*, 2004.

[13] M. Mathis and J. Mahdavi, "Forward acknowledgement: Refining TCP congestion control", *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 4, pp. 281-291, 1996.

[14] V. Padmanabhan and R. Katz, "TCP fast start: A technique for speeding up web transfers", In*: Proc. IEEE Globecom*, 1998.

[15] Y. Zhang, N. Ansari, W. Mingquan, and H. Yu, "AFStart: An adaptive fast TCP slow start for wide area networks", In: *Proc. of IEEE International Conference on Communications*, pp. 1260-1264, 2012.

[16] D. Zhang, K. Zheng, D. Zhao, X. dong Song, and X. Wang, "Novel quick start (QS) method for optimization of TCP", *Wireless Networks*, pp. 211-222, 2016.

[17] Y. Lu, X. Fan, and Lei Qian, "EQF: An explicit queue-length feedback for TCP congestion control in datacenter networks", In: *Proc. of the Fifth International Conference on Advanced Cloud and Big Data*, pp. 69-74, 2017.

[18] P. Moorthy and K. EaswaraMoorthy, "Agility based safety growth of Slow-Start Congestion Avoidance and Control Scheme in TCP", *International Journal of Intelligent Engineering and Systems*, Vol.10, No.6, pp. 30-38, 2017.

[19] R. Patel and A. Ganatra, "TCP M-Start: A New Slow Start Method of TCP to Transfer Data Over Long Fat Pipe Network", *International Journal of Intelligent Engineering and Systems*, Vol.10, No.1, pp. 124-133, 2017.

[20] A. Sembiring, M. Abdurohman, and F. Yulianto, "TCP LR-Newreno Congestion Control for IEEE 802.15.4-based Network", *International Journal of Intelligent Engineering and Systems*, Vol.10, No.5, pp. 181-190, 2017.

[21] R. Nalavala, C. Pakanati, and P. Mokkala, "An Enhanced Queue Management Scheme for TFRC Over Wired Networks", *International Journal of Intelligent Engineering and Systems*, Vol.10, No.1, pp. 22-27, 2017.

[22] *http://www.isi.edu/nsnam/ns/*,Network Simulator ns-2.