Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 592502, 16 pages
http://dx.doi.org/10.1155/2013/592502

Research Article

T-DLRP: Detection of Fast Retransmission Losses Using TCP Timestamp for Improving the End-to-End Performance of TCP over Wireless Networks

Prasanthi Sreekumari,[1] Sang-Hwa Chung,[2] Meejeong Lee,[1] and Won-Suk Kim[2]

[1] *School of Computer Science and Engineering, Ewha Womans University, Seoul 120-750, Republic of Korea*
[2] *School of Computer Engineering, Pusan National University, Busan 609-735, Republic of Korea*

Correspondence should be addressed to Sang-Hwa Chung; shchung@pusan.ac.kr

Received 2 May 2013; Accepted 14 November 2013

Academic Editor: Peter Csaba Ölveczky

Improving the end-to-end performance of TCP over wireless networks has been an active research area. When TCP operates in wireless networks, the performance of TCP degrades significantly. It is well known that frequent retransmission timeouts are the leading source of TCP performance degradation over wireless networks. Retransmission timeouts are unavoidable, when the fast retransmission of a lost packet fails to reach the destination. There have been many schemes developed for reducing the retransmission timeouts of TCP. However, these schemes have no mechanism to detect the loss of fast retransmitted packets over wireless networks. In this paper, we introduce an efficient TCP mechanism, called T-DLRP, which is able to detect the loss of fast retransmitted packets and react accordingly. To justify our contributions, we compared T-DLRP against TCP New Reno, Vegas, New Jersey, Veno, Cerl, and DAC via extensive simulations using Qualnet 4.5 simulator. The results demonstrate that our mechanism achieves significant improvement over existing TCP variants especially in a coexisted network with packet loss and fast retransmission losses. Compared to DAC, T-DLRP gains more than 70% improvement in terms of accuracy over wireless networks. Also, our experiments of multiple connections of the same TCP flows reveal that T-DLRP has better fairness compared to TCP NewReno.

1. Introduction

In the current Internet, Transmission Control Protocol (TCP) is used as the most popular transport layer protocol [1]. The congestion control algorithms of TCP are very essential for the stability of the Internet. However, the performance of TCP degrades rapidly when TCP operates in wireless networks [2]. Improving the end-to-end performance of TCP over wireless network has been an active research area in network community. The performance of TCP highly depends on its loss recovery algorithms such as fast retransmission and retransmission timeouts. The main reasons for the degradation of TCP throughput over wireless networks are the unnecessary reduction of congestion window size due to the inability of TCP sender to differentiate the transmission errors from network congestion, the causes of frequent retransmission timeouts [3]. TCP often loses several packets from one window of data and recovers these packets only after the expiration of retransmission timeouts which leads to reducing its end-to-end performance. In TCP, the sender detects packet loss by receiving three duplicate acknowledgments or the expiration of retransmission timeouts and treats every loss as an indication of network congestion. When the sender detects packet loss by retransmission timeouts, it goes to slow start algorithm and set the congestion window size to one maximum segment size (mss). Also, it takes many round trip times (RTTs) to bring the transmission rate to the previous level [4]. This leads TCP to reduce the flow of packets drastically and thereby decrease the throughput in wireless networks. In [5], the authors show that more than 50% of the dropped packets are recovered after the expiration of retransmission timeouts. According to [6], a

small percentage of timeouts are caused by the loss of fast retransmitted packets. In wireless networks, a retransmission timeout is inevitable, when the fast retransmission of a lost packet fails to reach the destination particularly due to the changing level of congestion and bit error rate in wireless channel [7–9]. It is obvious that TCP has no inbuilt mechanism to recover the packets caused by the loss of fast retransmission without depending on retransmission timeouts. The existing loss differentiation algorithms [10–13] can increase the performance of TCP over wireless networks but they do not have the mechanism to detect and differentiate the loss of fast retransmitted packets. As a result, it is an important issue whether the sender can recover the packets caused by the loss of fast retransmission without retransmission timeout or not.

Recently, two schemes [6, 14] were proposed for detecting the loss of fast retransmission based on TCP NewReno. However, they do not have the approaches to detect the loss of fast retransmitted packets when multiple packets are lost from a single window of data. By considering the drawbacks of existing schemes, our research mainly focused on improving the performance of TCP in wireless networks by detecting the loss of fast retransmitted packets.

In this paper, we propose an efficient TCP mechanism called Timestamp based Detection of the Loss of Fast Retransmitted Packets (T-DLRP), which is capable of detecting the loss of fast retransmitted packets and thereby reduce the frequent retransmission timeouts for improving the performance of TCP over wireless networks. T-DLRP consists of three main components.

(i) Detection of Fast Retransmission Loss (FRL): This scheme detects the loss of fast retransmitted packets using TCP Timestamp option defined in RFC 1323. We modified the TS Echo reply (Tsecr) field of Timestamp option for detecting the loss of fast retransmissions.

(ii) Immediate Fast Recovery (IFR): this scheme is used for guiding the TCP sender to react upon the detection of fast retransmission loss without waiting for the expiration of retransmission timeouts.

(iii) Immediate Retransmission Timeout Recovery (IRTOR): It helps to guide the TCP sender to adjust the sending rate upon the expiration of retransmission timeouts caused by transmission errors.

This paper describes the design of T-DLRP and shows the result of our experiments using Qualnet simulator in several scenarios over wireless networks. The results demonstrate that our mechanism achieves more than 86%, 72%, and 40% throughput improvement over TCP NewReno, DAC and TCP NewJersey, respectively, in a co-existed network which contains the loss of normal packets and the loss of fast retransmitted packets. Compared to DAC, T-DLRP gains more than 70% improvement in terms of accuracy over wireless networks and the experiments of multiple connections of the same TCP flows reveal that T-DLRP has better fairness compared to TCP NewReno.

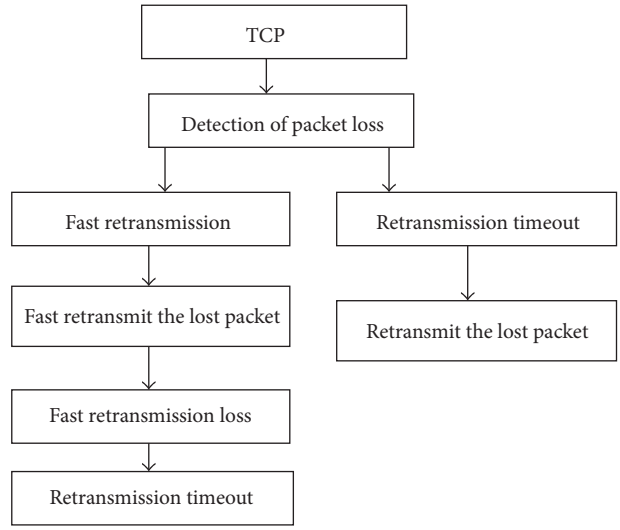

Figure 1: TCP and loss of its fast retransmission.

The remainder of this paper is organized as follows. Section 2 describes the problem of the loss of fast retransmitted packets in wireless networks. Section 3 briefly summarizes the related work. We introduce our new TCP mechanism T-DLRP in Section 4. Section 5 describes the performance evaluation of T-DLRP and finally Section 6 concludes our work.

2. TCP and Loss of Fast Retransmissions

The current implementation of TCP contains four main congestion control algorithms such as slow start, congestion avoidance, fast retransmit, and fast recovery, aimed to control and recover packets from the loss of network congestion. Together, TCP defines two main variables: congestion window size (cwnd) and slow start threshold (ssthresh). The former variable is used as an estimation of the maximum number of segments that can be sent to the receiver without overloading the network and the latter variable is used to determine the state of a TCP sender. TCP infers a packet loss based on the receipt of three duplicate acknowledgments or retransmission timeouts as shown in Figure 1. Whenever TCP transmits a segment, the sender starts a timer which keeps track of how long it takes for an acknowledgment (ack) for that segment to return. This timer is known as the retransmission timer. If an ack is returned before the timer expires (by default, it is often initialized to 1.5 seconds), the timer is reset with no consequence.

However, if an ack for the segment does not return within the timeout period, the sender would retransmit the segment and double the value of retransmission timer for each conservative timeout up to a maximum of about 64 seconds [4]. When the sender detects packet loss by the expiration of retransmission timeouts, the sender retransmits all the packets following the one that was lost and resets the cwnd size to one mss and increases the sending rate according to the slow start algorithm. When the sender detects packet loss

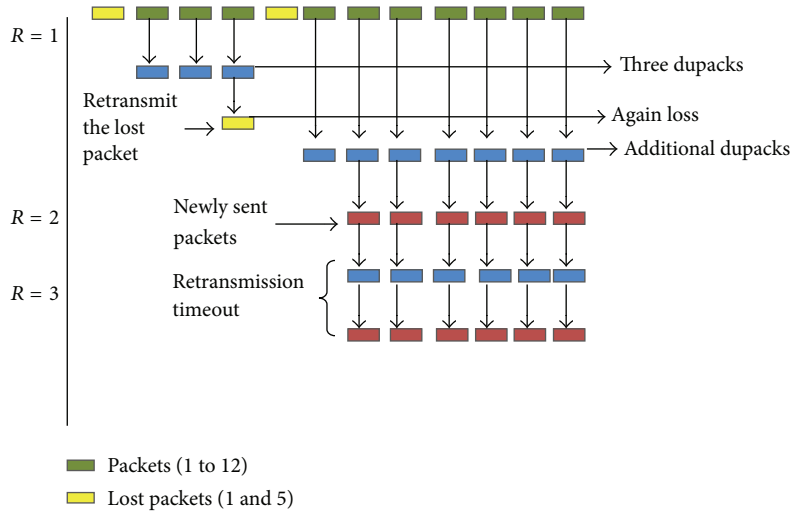

Figure 2: Retransmission timeout due to the loss of fast retransmitted packets.

via three duplicate acknowledgments (dupacks), it invokes fast retransmission algorithm immediately by setting the value of ssthresh to half of the cwnd size and retransmits the missing segment without waiting for the expiration of retransmission timeouts. After retransmitting the missing segment, fast recovery algorithm takes over and the sender receives more dupacks. In fast recovery, the value of cwnd sets to ssthresh plus three and increment by one for each additional dupacks received by the sender. The sender sends new packets allowed by the value of cwnd. When the sender receives a complete ack, cwnd resets its value to ssthresh and put the sender in congestion avoidance algorithm. Note that, TCP recovers the lost packets only if the retransmitted packet by fast retransmission reaches the destination successfully. Otherwise, the TCP sender recognizes the loss of fast retransmitted packets only at the end of retransmission timeouts. This will affect the throughput of TCP. As we mentioned in the above section, the loss of fast retransmitted packets may actually be related to various factors such as changing level of congestion, queue management scheme, and bit error rate in a wireless channel.

Figure 2 illustrates how RTO takes place due to the loss of multiple packets from a single window of data and the loss of its fast retransmissions. As we discussed above, fast retransmission algorithm triggers only after receiving three dupacks. In this figure, the sender sent 12 packets (1 to 12) with cwnd of 12. Among that, the packets 1 and 5 were lost. Upon the arrival of three dupacks by packets 2, 3, and 4 the sender retransmitting the lost packet 1 at round 1 ($R = 1$) by invoking the fast retransmission algorithm and enters into fast recovery state at $R = 2$ after retransmits the lost packet 1. During fast recovery, the sender receives additional dupacks by packets 6 to 12. For each additional dupacks the sender increments the cwnd size by one mss and sent new packets (13 to 18) allowed by the value of cwnd.

Consider that the fast retransmitted packet is again lost while sending to the receiver. As a result, instead of getting a normal ack the sender receives more additional dupacks of the lost packet 1 by the arrival of newly sent packets at the receiver side. Without receiving the normal ack, the sender cannot exit fast recovery and wait for the expiration of retransmission timeouts. This type of retransmission timeouts is inevitable and degrades the performance of TCP over wireless networks.

3. Related Work

There have been a set of TCP protocols proposed for improving the performance of TCP in wireless networks. In this section, we briefly describe the TCP protocols that T-DLRP is compared to in this paper.

TCP NewReno. TCP NewReno [15] modified the fast recovery algorithm of TCP Reno in order to avoid retransmission timeout due to multiple packet losses in a single of window of data. TCP NewReno improves the performance of TCP by distinguishing the normal and partial acks. A normal ack acknowledges all the outstanding packets at the time of detecting the lost packets, whereas a partial ack acknowledges some but not all of this outstanding data. Unlike TCP Reno, where a partial ack terminates fast recovery algorithm, TCP NewReno retransmits the packet next in sequence based on the partial ack and reduces the cwnd size to one less than the number of segments acknowledged by the partial ack. In this way TCP NewReno recovers multiple packet losses in the same window by retransmitting one lost segment per RTT, remaining in fast recovery stage until a full ack is received.

TCP Vegas. The main goal of Vegas [16] is to keep the amount of extra bytes in the network between two threshold values. It uses the estimated number of bytes in the network to decide whether or not the cwnd size should increase linearly, not change, or decrease linearly during the next RTT. The Vegas version of TCP tries to use the minimum RTT (minRTT) value to estimate the expected connection capacity. The minRTT value is set as the smallest RTT sample observed

over the duration of the connection. When Vegas sender receives an ack, it will use the current RTT value to estimate the actual bandwidth and try to compare with the expected bandwidth. If the actual rate of a connection is much less than the expected rate, the sender should reduce the sending rate. On the other hand, if the actual rate of a connection approaches the value of the expected rate, the sender should increase the sending rate. Otherwise, the sender will keep its sending rate. Unfortunately, TCP-Vegas have big problems on fairness in wireless environment. When several connections start at different times or the circumstances between the source and the destination change, the minRTT value may become an unbefitting history value and not suitable for the current network. Specifically, we can find the Vegas gives up the addictive increase multiplicative decrease (AIMD) mechanisms and tries to hit the bottleneck bandwidth without using losses event.

TCP Veno. TCP Veno [12] proposed a solution to differentiate the random error losses from congestion losses. TCP Veno adopts the same mechanism as TCP Vegas to estimate the size of the backlogged packets (N) in the buffer of the bottleneck link. The calculation of N is given below:

$$N = \text{Diff} * \text{BaseRTT}, \tag{1}$$

where Diff is the difference between expected and actual rates and BaseRTT is the minimum measured RTTs. The Expected and Actual rates are measured as

$$\begin{aligned} \text{Expected} &= \frac{\text{cwnd}}{\text{BaseRTT}} \\ \text{Actual} &= \frac{\text{cwnd}}{\text{RTT}}, \end{aligned} \tag{2}$$

where cwnd is the current cwnd size and RTT is the smoothed round trip time measured. TCP Veno used the measurement of N to differentiate the type of packet loss. Specifically, when a packet is lost, Veno compares the measured value of N with β (backlog threshold).If $N < \beta$, TCP Veno assumes the loss to be random rather than congestive; otherwise veno assumes that the loss to be congestive.

TCP Cerl. TCP congestion control enhancement for random loss (CERL) by distinguishing random losses from congestion losses is based on a dynamic threshold value. TCP CERL [11] is a sender side modification of TCP Reno. TCP CERL and TCP Veno are similar in concept. However, the mechanisms utilized by TCP CERL differ greatly from those used in TCP Veno. TCP CERL utilizes the RTT measurements made throughout the duration of the connection to estimate the queue length (l) of the link and then estimates the congestion status. The calculation of l is as shown below:

$$l = (\text{RTT} - T)\, B, \tag{3}$$

where RTT is the measured round trip time, B is the bandwidth of the bottleneck link, and T is the smallest RTT observed by the TCP sender and l is updated with the most recent RTT measurement received. Using the value of l and A (a constant which is equal to 0.55), TCP CERL used to set the dynamic threshold value (N),

$$N = A * l_{\max}, \tag{4}$$

where $l_{\max}$ is the largest value of l observed by the sender. If $l < N$ when a packet loss is detected via three dupacks, TCP CERL assumes the loss to be random rather than congestive. Otherwise, TCP CERL assumes that the loss is caused by congestion.

TCP NewJersey. TCP NewJersey [10] is an extension of TCP Jersey [17] as a router assisted solution for differentiating noncongestion loss from congestion loss and reacts accordingly. TCP NewJersey has two key components in its scheme, timestamp based available bandwidth estimation (TABE) and congestion warning (CW) scheme. To estimate the available bandwidth, TCP Jersey follows the same idea of TCP Westwood's rate estimator to observe the rate of acknowledged packets by acks but with a different implementation. Upon receiving "n" acks, the available bandwidth B_n is estimated as shown below:

$$B_n = \frac{\delta B_{n-1} + L_n}{(t_n - t_{n-1}) + \delta}, \tag{5}$$

where δ is the TCP's estimation of the end-to-end round trip time delay at time t_n, L_n is the size of the data, t_{n-1} is the arrival time of the previous ack, and t_n is the arrival time of nth packet at the receiver. The sender interprets the estimated rate as the optimal congestion window (ownd) in unit of the size of segment (S) and is calculated as

$$\text{ownd} = \frac{(\delta * B_n)}{S}. \tag{6}$$

CW is a packet marking scheme of TCP Jersey for determining whether network congestion has occurred or not. This scheme is originated from Explicit Congestion Notification (ECN), but it is different from ECN in parameter settings and marking of packets. When the sender receives three dupacks, TCP NewJersey checks whether the received ack has CW mark or not. If it has CW mark, TCP NewJersey will assume that the loss is caused by network congestion and proceeds as TCP Reno, whereas if the ack has no CW mark, TCP NewJersey assumes that the loss is random and calls the rate control procedure to adjust the window size. Although there are several existing schemes for the improvement of TCP in wireless networks, TCP NewJersey is known as the best existing scheme in terms of throughput [13]. Even though the above schemes can increase the performance of TCP, these schemes has no mechanism to detect the loss of fast retransmitted packets and thereby suffer from frequent retransmission timeouts. Recently two schemes were proposed for detecting the loss of fast retransmitted packets. Below, we briefly explain those schemes.

DAC. Dynamic Acknowledgment Counting (DAC) [6] proposed a solution for reducing retransmission timeout due to the loss of retransmissions. DAC operates at the sender side of

TCP NewReno during fast recovery. The sender keeps some variables to store the expected number of dupacks for a packet loss. The cwnd size just before the first fast retransmission is stored in another variable. During fast recovery the TCP sender counts the number of dupacks for a packet loss. If the sender receives more dupacks for the packet loss than the stored value, DAC determines that the retransmission is lost again and retransmits the packet without waiting the expiration of timeout and reduces the cwnd size again to half of the size of the current value of ssthresh. In this way, DAC can detect the retransmission lost and reduce the retransmission timeout. Unfortunately, it has some limitations. First, DAC starts to count the dupacks only after fast retransmission and results in spurious retransmissions. Not only that, DAC has a serious problem of unauthorized segment transmission, one is new segment due to increment of cwnd during fast recovery and other one is the retransmitted lost segment. This is due to the lack of cooperation between the fast recovery and loss recovery algorithms. Second, DAC reduces the cwnd size blindly into two times. Third, DAC has no mechanism to detect the type of retransmission loss. DAC assumes that the normal packet losses and retransmission losses are caused by network congestion.

DDLRP. The key idea behind DDLRP [14] is to use the flight size information in order to set a dynamic threshold value when the sender receives three dupacks for detecting the retransmission loss. When the additional dupacks are greater than the threshold value, then the sender of DDLRP confirms that the retransmitted packet is again lost and retransmits the lost packet immediately without waiting for a retransmission timeout. In addition, DDLRP can be able to differentiate the type of retransmission loss and increase the performance of TCP over wireless networks. The major drawback of DDLRP is that it is unable to detect the loss of fast retransmission when multiple packets are lost from a single window of data and the loss of its fast retransmissions. In this situation, DDLRP degrades the performance of TCP over wireless networks. Opposed to above schemes, T-DLRP can detect the loss of fast retransmitted packet even multiple loss of packets from a single window of data and can increase the performance of TCP over wireless networks. In [18], we present our idea of T-DLRP for detecting the loss of fast retransmitted packets. In this paper, we modified our previous idea in the IFR scheme of T-DLRP for differentiating the loss of fast retransmitted packets and provide a brief experimental analysis in terms of throughput and accuracy to show that T-DLRP can increase the performance of TCP by reducing the expiration of timeouts in wireless networks.

4. Proposed Mechanism: T-DLRP

For improving the end-to-end performance degradation of TCP over wireless networks due to retransmission timeouts cause by the loss of fast retransmitted packets, we propose an efficient TCP mechanism called T-DLRP, which is able to detect fast retransmission loss even multiple packets that are lost from one window of data. The key idea of our mechanism is to use TCP Timestamp option by modifying the Tsecr field which is capable of updating the value when new packets reach the destination during fast recovery. T-DLRP consists of three main components as shown in Figure 3.

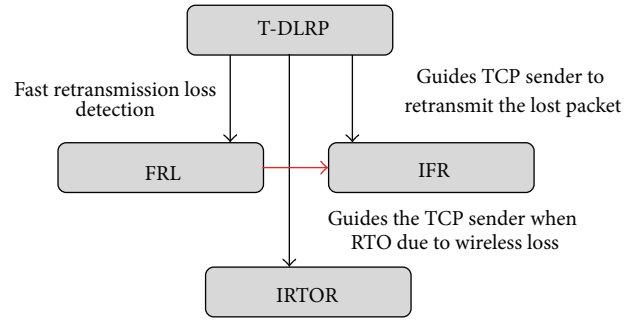

Figure 3: Key components of T-DLRP.

Table 1: TCP Timestamp option.

Kind: 8			
Length: 10 bytes			
Kind = 8	Length = 10	TS value (Tsval)	TS echo reply (Tsecr)
1	1	4	4

4.1. FRL. As we mentioned in Section 2, if TCP detects packet loss via three dupacks, the sender immediately triggers the fast retransmission algorithm and then follows the fast recovery algorithm for recovering the lost packets. However, TCP recovers from the loss only if the fast retransmitted packet reaches the destination successfully. Otherwise the TCP sender recognizes the retransmission loss only at the end of retransmission timeout. For avoiding this type of retransmission timeout, T-DLRP present its main component called Fast Retransmission Loss Detection (FRL) as shown in Figure 3.

For detecting the loss of fast retransmitted packets, we use TCP timestamp option defined in RFC 1323 [19]. The Timestamp option carries two 4 byte timestamp fields. One is TS Value and the other is TS Echo Reply as shown in Table 1. The working of TCP Timestamp can be seen in Figure 4. In Figure 4(a), when the sender sends packets A, B and C to the receiver at timestamp t, t_1, and t_2, it stores the current timestamp in the Tsval field. Upon the arrival of those packets at the destination, the receiver copies the timestamp of the Tsval field and sends it back to the source by attaching the copied timestamp value into the Tsecr field of timestamp option via ack. Consider that the packet B is lost as shown in Figure 4(b). In that case, the receiver does not update the value of Tsecr until the lost packet reaches the destination even the duplicate acks caused by the arrival of new packet which is equal to C and its Tsval is t_2. For every duplicate acks generated by the arrival of packets, the value in the Tsecr field contains the timestamp of last successfully received packet because the timestamp option is designed for measuring the accurate RTT. However, our main goal is to detect the loss of fast retransmitted packets. As a result, we cannot use the original Timestamp option. To achieve our goal, we modified

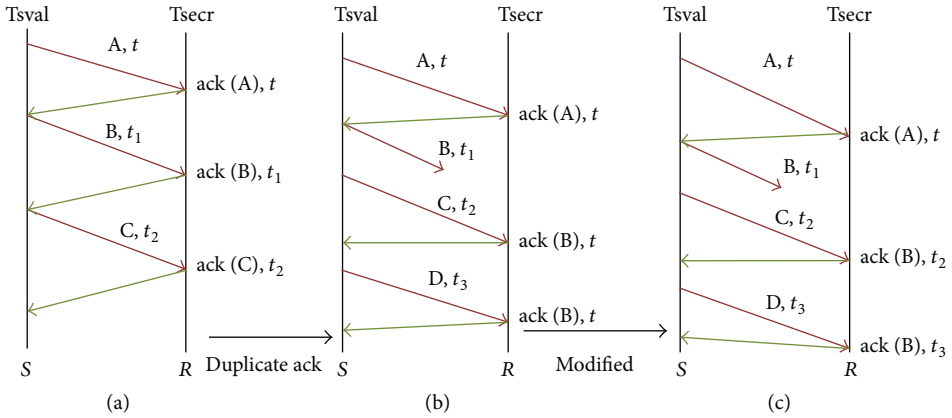

FIGURE 4: Working of TCP timestamp, (a) normal data transmission, (b) at the time of dupacks, and (c) modified Timestamp option.

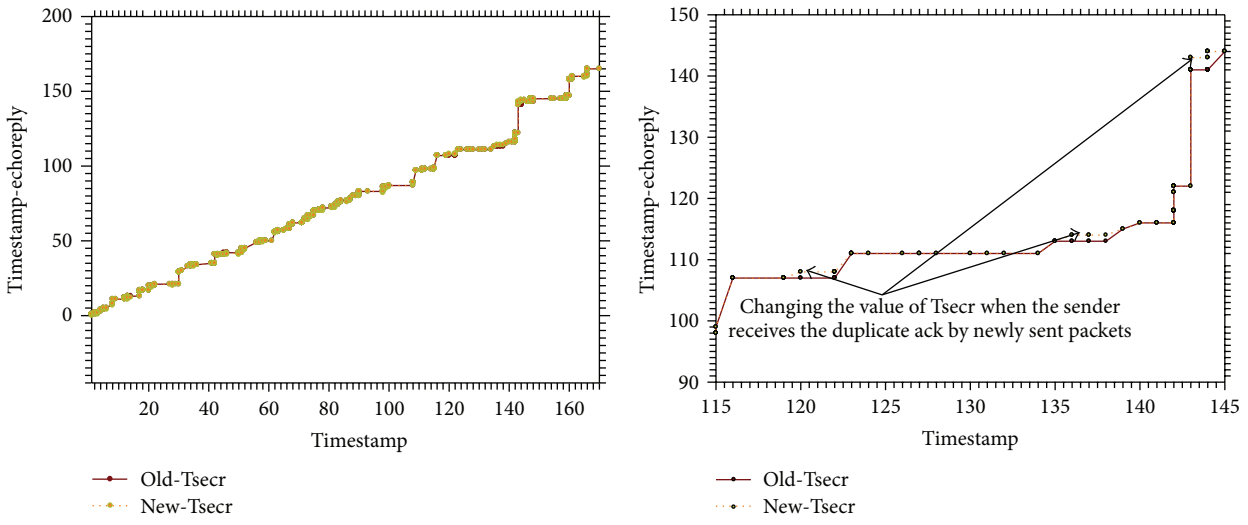

FIGURE 5: (a) Comparison of Tsecr values. (b) Zoomed graph.

the Tsecr field of TCP Timestamp as shown in Figure 4(c). When receiver receives a new packet C with Tsval t_2, the value in the Tsecr field updates to t_2 from t according to the timestamp attached to the field Tsval.

As a result, when the receiver sends dupack, the Tsecr field does not attach the timestamp of the Tsval field of the last successfully received packet as shown in Figure 4(b). Instead of that, receiver updates the value of Tsecr according to the arrival of new packets with Tsval. Figure 5 presents the comparison of original (Old) and modified (New) Tsecr values. For this comparison, we used a five hops chain topology with 2 Mbps bandwidth and 50 ms propagation delay. The distance between each hop is 200 m apart. We sent a TCP flow from source 1 to destination 6 using FTP-Generic. The duration of our experiment lasts upto 200 seconds. We traced the Old and New Tsecr values. The x-axis shows the timestamp at the time of receiving the acks at the sender and y-axis presents the timestamp echo reply when the sender receives the acks. Old Tsecr contains the original value in the Tsecr field and New Tsecr contains the modified value when the sender receives new packets. As shown in Figure 5(a), it is clear that the New Tsecr value updates its value upon receiving the new packets. Figure 5(b) presents the zoomed graph of Old and New Tsecr values, when the sender receives dupacks caused by the newly sent packets during fast recovery.

Using this modification, we detect the loss of fast retransmitted packets according to the following way. As we explained before, when the sender receives three dupacks, it invokes fast retransmission algorithm and retransmits the lost packet. Then it enters into fast recovery algorithm and sends new packets allowed by the size of cwnd for each additional dupacks. After fast retransmission, we store the timestamp of newly sent first packet during fast recovery to the variable "$\text{Timestamp}_{\text{new}}$" and attach that timestamp to the field Tsval as shown in Algorithm 1. Upon receiving the newly sent packet, the receiver changes the Tsecr value and attaches the recent timestamp of the Tsval field contained in the new

```
(1) If (fastrecovery)
(2) {
(3)    Send new packet
(4)    Timestamp_New = time of first sent packet after fast retransmission
(5)    Receive dupack
(6)      If (Timestamp_New ≥ Tsecr)
(7)       {
(8)          Report retransmission loss
(9)          trigger IFR
(10)       }
(11)    Continue fast recovery until receiving the complete ack
(12) }
```

Algorithm 1: Whenever the sender sends new packet during fast recovery.

packet to the field Tsecr and sends it back to the sender via dupacks of the lost packet. When the sender receives a dupack during fast recovery, it checks the value in the Tsecr field and compares that value to the variable "$\text{Timestamp}_{\text{new}}$."

If the value is greater than or equal to the stored value in the variable, then the sender clearly confirms that the fast retransmitted packet is again lost and immediately triggers the IFR algorithm without waiting for RTO. After executing the IFR algorithm, the sender continues the fast recovery procedure until receiving the complete ack including the ack of lost packet. By using this modified Tsecr field of TCP timestamp option we can detect the loss of fast retransmission accurately even multiple packet loss from a single window of data.

Figure 6 shows an example of FRL-Detection. Consider that the sender sends 12 packets (1 to 12) with cwnd size of 12. Among that, the packets 1 and 5 were lost. Upon receiving three dupacks by packets 2 to 4, the sender invokes the fast retransmission algorithm and retransmits the lost packet and then enters into fast recovery. During fast recovery the sender receives additional dupacks by the remaining packets in the flight (6 to 12) and sends new packets allowed by the size of cwnd. In this example, the sender sent a new packet 13 at timestamp t_1 upon the arrival of dupack by packet 7 and stores the timestamp of packet 13 in a variable "$\text{Timestamp}_{\text{new}}$." Then the timestamp attaches to the Tsval field and sends it to the receiver. When the receiver receives a new packet 13 with a new timestamp, the receiver updates the value in Tsecr field and sends back to the sender. Upon the arrival of dupack of the lost packet by a newly sent packet 13 with Tsecr value t_1, the sender checks with the value stored in the variable "$\text{Timestamp}_{\text{new}}$." In this example the value of Tsecr field is equal to the value in the variable. Then the sender confirms that the fast retransmitted packet is again lost and retransmits the packet immediately by triggering IFR algorithm without waiting for the expiration of retransmission timeouts. In this way the sender can detect the lost packets from a single window of data without waiting for retransmission timeouts and thereby increase the performance of TCP over wireless networks.

4.2. IFR. For increasing the end-to-end performance of TCP, we not only reduce the retransmission timeouts by detecting the fast retransmission losses but also differentiate the type of fast retransmitted packet loss. For differentiating the fast retransmission loss, we adopt the CW mechanism of TCP NewJersey as it is the best existing solution for differentiating the type of packet losses. CW mechanism is a configuration of routers for giving alerts from the routers to end stations by marking all packets when the average queue length exceeds a threshold. This scheme is originated from ECN. In TCP NewJersey the threshold value was 30 percent of the buffer size. This means that the packet loss below the threshold value is judged as a transmission error, while the value exceeds the threshold as network congestion. In today's Internet, drop-tail queuing policy is the most widely deployed router queue management scheme. According to drop-tail queuing algorithm, the packet will be dropped only when the queue length becomes equal to the buffer size. As a result, the marking of packets when the buffer size becomes 30 percent causes misclassification of packet losses and we cannot use the available bandwidth fully and this results in the degradation of TCP throughput. In order to reduce the misjudgments of packet losses, T-DLRP mark the packets only when the average queue length exceeds or equal to 90 percent of the buffer size. Because when the average queue length reaches 90%, it means that the buffer will easily overflow, which results in that the packet may be lost due to network congestion.

For conveying the CW information, we assume that the routers along the TCP connection path use the original ECN because CW inherits the same information bits from the original implementation of ECN [16]. When the average queue length is greater than or equal to 90 percent (threshold) of the buffer size, the router may mark all the incoming packets. In T-DLRP, upon receiving "n" ($n = 3$) dupacks with CW mark (CW = 1), the sender can confirm that the lost packet was due to network congestion. Otherwise (CW = 0), the dupacks are due to transmission errors. Like that, when the sender detects a fast retransmission loss by comparing the value stored in $\text{Timestamp}_{\text{new}}$ variable with that of the value in the Tsecr field, the sender triggers IFR algorithm.

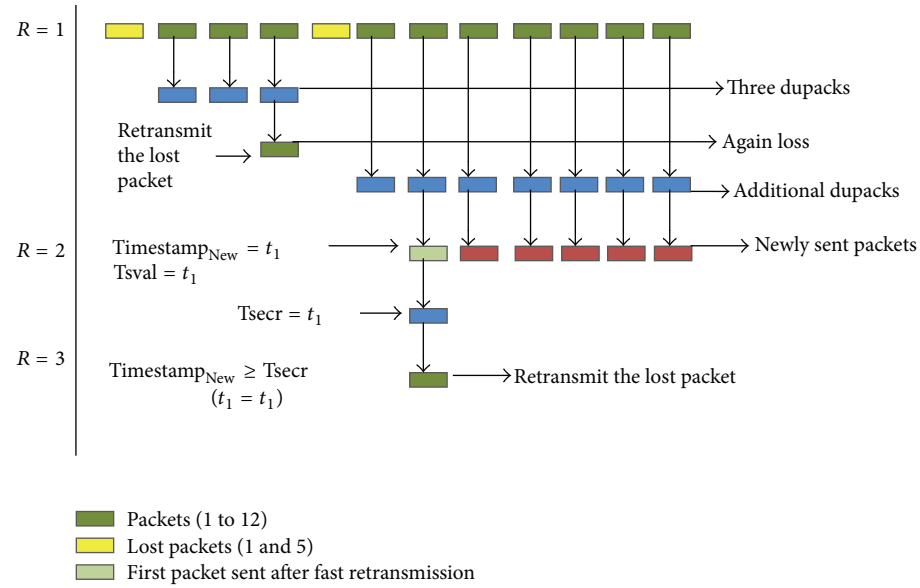

Figure 6: Example of FRL.

By executing this algorithm, the sender retransmits the retransmitted lost packet without waiting for retransmission timeout. In IFR algorithm, upon the detection of fast retransmission loss, the sender checks whether the packet is lost due to network congestion or wireless loss. If it is due to network congestion the sender reduces the size of cwnd as shown in the below equation:

$$\text{Cwnd} = \frac{\text{BW} * \text{RTT}}{\text{Pack\_size}}, \tag{7}$$

where BW is the bandwidth, RTT is the round trip time, and Pack_size is the size of the packet. Here, we only reduce the value of cwnd and keep the current size of ssthresh. On the other hand, if the packet is lost due to transmission error, then the sender retransmits the packet by keeping the current size of cwnd and ssthresh. After retransmitting the lost packet the senders continue the fast recovery algorithm until it receives a normal ack.

4.3. IRTOR. When retransmission timeout happens, the traditional timeout recovery mechanism of TCP retransmits the unacknowledged segments by reducing the cwnd size to one mss and sets the value of ssthresh to half the size of cwnd. After that, the TCP senders continue the slow start algorithm by increasing the size of cwnd by one segment on each ack and then retransmit the next unacknowledged segment allowed by the value of cwnd. However, if the RTO occurs due to transmission errors, the needless reduction of cwnd size leads to the performance degradation of TCP.

In wireless network, the expiration of retransmission timeouts is not a rare event. As a result, for reducing this performance degradation, T-DLRP uses the immediate retransmission timeout recovery algorithm when the sender detects the expiration of timeout due to transmission error. As shown in Algorithm 2, when the sender detects the retransmission timeout due to transmission error, the sender retransmit the lost packet and record the highest sequence number sent to the receiver in the variable "Maxsent" by keeping the current size of cwnd and ssthresh. Instead of retransmitting all the outstanding packets, the sender waits for the ack after retransmitting the first lost packet. If the next ack is greater than or equal to the highest sequence number sent by the sender at the time of retransmission timeout, the sender triggers the congestion avoidance algorithm. In this way, the sender avoids the unnecessary retransmissions of all outstanding packets and reduces the unnecessary reduction of the size of cwnd.

4.4. Congestion Control Policy of T-DLRP. In this subsection, we describe the congestion control policy of T-DLRP as shown in Figure 7. We adopted the Slow Start (SS) and Congestion Avoidance (CA) algorithm of TCP NewReno and modified the Fast Retransmission (FR) and Recovery (FRe) algorithm of TCP NewReno. At the beginning of the TCP connection the sender enters into SS state and increases the size of cwnd by one mss for every ack it receives. When cwnd reaches the value of ssthresh, the sender triggers the CA algorithm and increases the size of cwnd linearly for every ack it receives. The sender detects packet losses with the help of three dupacks or the expiration of retransmission timeouts. When the sender receives three dupacks, it invokes FR and FRe algorithm. During FR, the sender retransmits the lost packet and checks whether the loss is due to network congestion or wireless loss using the congestion warning mechanism. If the incoming dupacks have CW mark, then the size of cwnd reduces and enters into FRe algorithm by storing the highest sequence number sent to a variable. During FRe, the cwnd size increased to one mss and sent new packets allowed by the value of cwnd.

```
(1)  IRTOR()
(2)  {
(3)   Retransmit the lost packet
(4)   Maxsent = Record highest sequence number sent
(5)   cwnd = current
(6)   ssthresh = current
(7)   If (ack ≥ Maxsent)
(8)    {
(9)      Congestion avoidance
(10)  } else {
(11)     Partial ack
(12)     Retransmit the packet
(13)  }
(14) }
```

Algorithm 2: Whenever the sender detects retransmission timeout due to wireless loss.

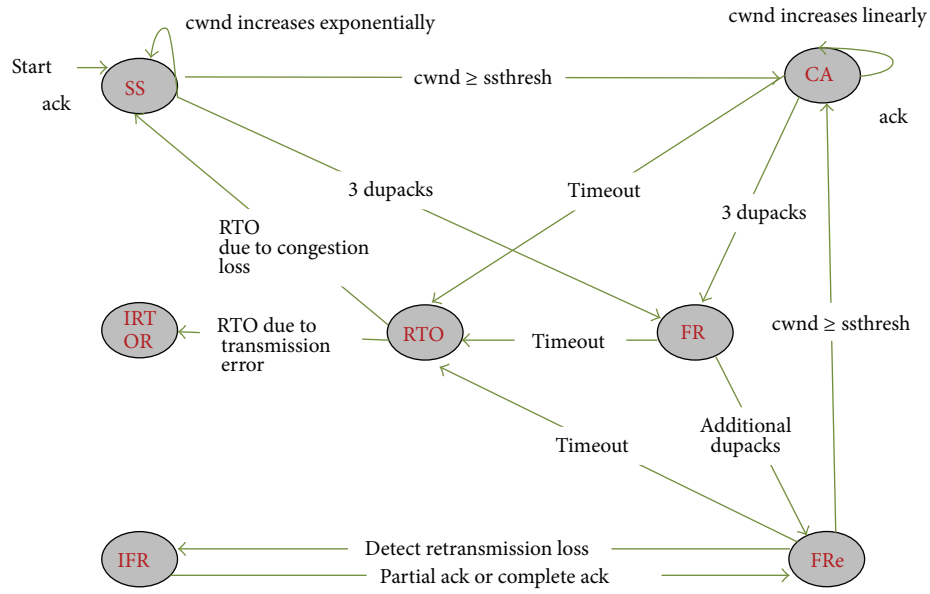

Figure 7: State diagram of T-DLRP.

When the sender detects fast retransmission loss by comparing the values in the variable $\text{Timestamp}_{\text{new}}$ and Tsecr field of the incoming dupack, then the sender triggers the Immediate Fast Recovery (IFR) algorithm. After retransmitting the lost packet, the sender goes back to FRe algorithm and continues until it gets an ack which covers all outstanding packets including the value stored in the variable recover. On the other hand, if the sender detects the packet loss by the expiration of retransmission timeouts, it retransmits the lost packet. Before reducing the size of cwnd the sender checks whether the last ack is marked or not. If it is marked, it reduces the value of cwnd otherwise it triggers IRTOR algorithm of T-DLRP. By using the three schemes of T-DLRP we can increase the performance of TCP over wireless networks.

5. Performance Evaluation

In this section, we present the performance evaluation of T-DLRP by a set of experiments using qualnet 4.5 simulator [20]. We implemented T-DLRP and existing algorithms such as DAC, TCP Vegas, TCP NewJersey, TCP Veno, and TCP Cerl in qualnet. First, we compared the performance of T-DLRP in the presence of fast retransmission loss with TCP NewReno as it is the most widely deployed protocol in the Internet. Then, we performed a set of experiments in terms of throughput, accuracy, and fairness in order to ensure that T-DLRP improves the end-to-end performance of TCP. The subsections show the experimental setup and results of T-DLRP compared with other TCP variants. For our

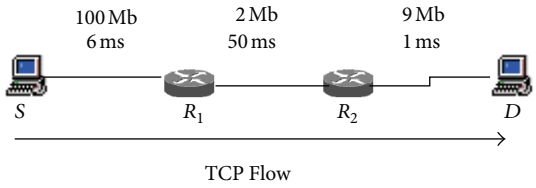

Figure 8: Mixed wired-wireless network topology.

experiments, we use three types of topologies such as mixed wired-wireless, multihop, and dumbbell shaped topology.

5.1. Comparison of T-DLRP with TCP NewReno in the Presence of Fast Retransmission Loss. For comparing the performance of T-DLRP with TCP NewReno in the presence of fast retransmission loss, we use a mixed wired-wireless network topology as shown in Figure 8. In this Figure, S and D denote the source and destination whereas R_1 and R_2 denote the routers. The router R_2 is connected to the destination via a wireless link. The bottleneck bandwidth is 2 Mbps between the routers R_1 and R_2 with propagation delay 50 ms. The bandwidth and delay between S and R_1 are 100 Mb and 6 ms whereas the bandwidth of the wireless link is 9Mb with 1 ms delay. In all simulations, the maximum segment size of TCP was set to 1024 bytes. We implemented the traffic source using FTP-Generic. The maximum window limit is set to 32 packets and the size of an ack packet is same as the size of data packet.

DSR is the main routing protocol in our simulation with a maximum message buffer size set to 50 packets. The duration of our simulations was set to 200 s. During simulations, data packets are continuously transmitted up to the end of the simulation time. We imposed packet loss and retransmission loss using the exponential packet error model which is available in qualnet. The simulations have been conducted using Qualnet version 4.5 software that provides scalable simulations of wireless networks. All nodes are static in our simulations. Figure 9 presents the behavior of TCP NewReno during packet transmission.

In Figure 9(a), in between 40 and 100 s, four retransmission timeouts happened during fast recovery due to the loss of fast retransmitted packet. This is due to the inability of NewReno to detect the loss of fast retransmitted packets. As a result, the size of cwnd goes to one mss as shown in Figure 9(b) and thus degrades the performance of NewReno unnecessarily. On the other hand, in case of T-DLRP, as shown in Figure 10(a), there is no retransmission timeouts due to its ability to detect the loss of fast retransmitted packets and Figure 10(b) shows the corresponding cwnd size of T-DLRP.

As we mentioned previously, T-DLRP can retransmit the loss of fast retransmitted packet by triggering the IFR algorithm upon the detection of the loss of fast retransmitted packets and can reduce the unnecessary retransmission timeouts. As a result, when compared to TCP NewReno, T-DLRP can increase the size of cwnd by sending more data packets. This will lead to an increase in the performance of TCP under the loss of fast retransmitted packets.

5.2. Comparison of Throughput Using Mixed Wired-Wireless Network. Throughput is one of the most important metric use for evaluating the performance of TCP. In this subsection, we present the throughput improvement of T-DLRP compared to other protocols in terms of packet loss rate, TCP connections, and different bandwidths over mixed network topology. In addition to that, we compare the size of T-DLRP cwnd with the size of another retransmission loss detection scheme called DAC. The experimental parameters we used for this experiment are same as the above mentioned experiment otherwise stated. For calculating the throughput we use the following equation:

$$\text{Throughput} = \frac{\text{Total-packet} * \text{Packet-Size}}{\text{Simulation time}}. \quad (8)$$

Figure 11(a) shows the throughput variations according to the packet loss rate ranges from 1 to 5%. In this experiment, we caused loss due to transmission errors and retransmission loss. For causing transmission error (wireless loss), we used the exponential error model which is available in qualnet 4.5 and for imposing the loss of fast retransmitted packets, we adjusted the uptime and downtime values of the error model. From the graph, it is evident that T-DLRP achieves higher throughput compared to other TCP variants. At 5% packet loss rate, T-DLRP gains more than 50% throughput compared to NewReno and 20% throughput compared to the loss differentiation algorithm of TCP NewJersey. Figure 11(b) shows the throughput comparison in terms of different numbers of TCP connections from 1 to 10. In this experiment, for causing retransmission loss we used CBR traffic in addition to FTP-Generic application. When the number of TCP connections increases, the throughput leads to a decrease. However, T-DLRP acheives higher throughput compared to other TCP variants. As shown in Figure 12(a), we compare the throughput of T-DLRP in terms of different bandwidth ranging between 2, 5.5, and 11 Mbps. In this experiment, we used congestion loss, wireless loss, and retransmission loss. For causing congestion loss we used 5 TCP connections and 3% packet loss due to wireless loss. In addition, we added 1% retransmission loss in order to evaluate the throughput performance of T-DLRP by the coexistence of different losses. TCP NewReno has the least throughput compared to other TCP variants and T-DLRP outperforms TCP NewReno and other schemes. When bandwidth reaches 11 Mbps, the throughput of T-DLRP increases to 70% more than TCP NewReno and 75% more throughput than DAC. However, when compared to loss differentiation algorithm, T-DLRP acheives more than 40% throughput of TCP NewJersey. This is because the threshold value of T-DLRP used for the differentiation of packet loss is more accurate than TCP NewJersey. Figures 12(b) and 12(c) present the size of T-DLRP cwnd with the cwnd size of DAC. From this figure, it is clear that T-DLRP can send more data packets than DAC. The main reason is that T-DLRP does not reduce the cwnd size blindly into half and it has an accurate retransmission loss detection scheme than DAC.

5.3. Comparison of Throughput Using Multihop Topology. For evaluating the throughput performance of T-DLRP in

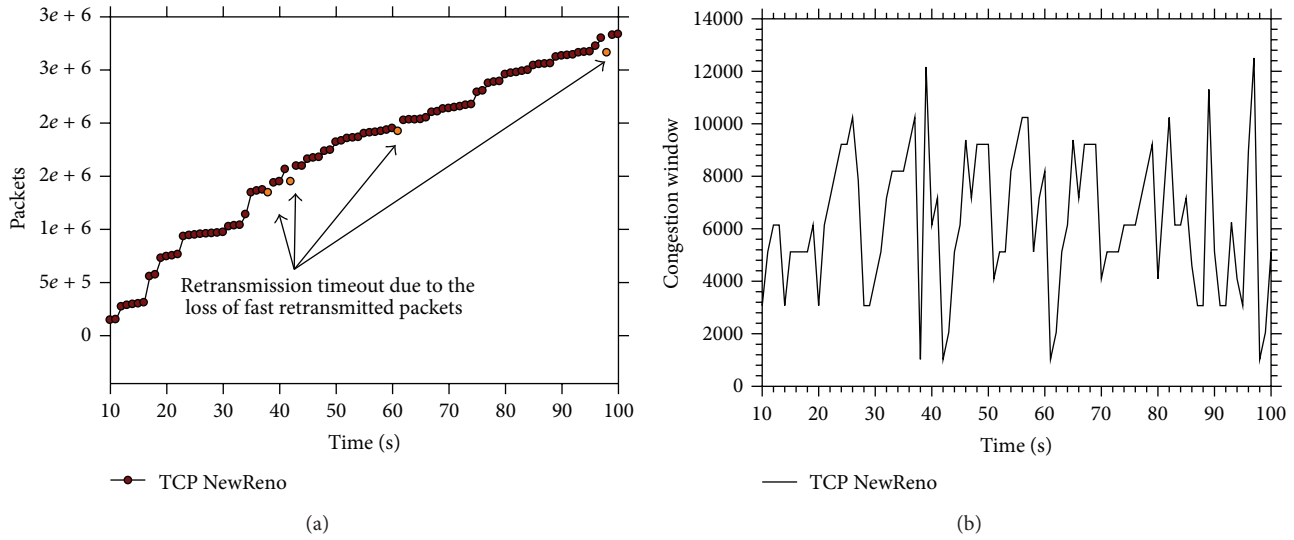

Figure 9: Behavior of TCP NewReno. (a) Timeouts due to the loss of fast retransmitted packets and (b) size of cwnd during FRR.

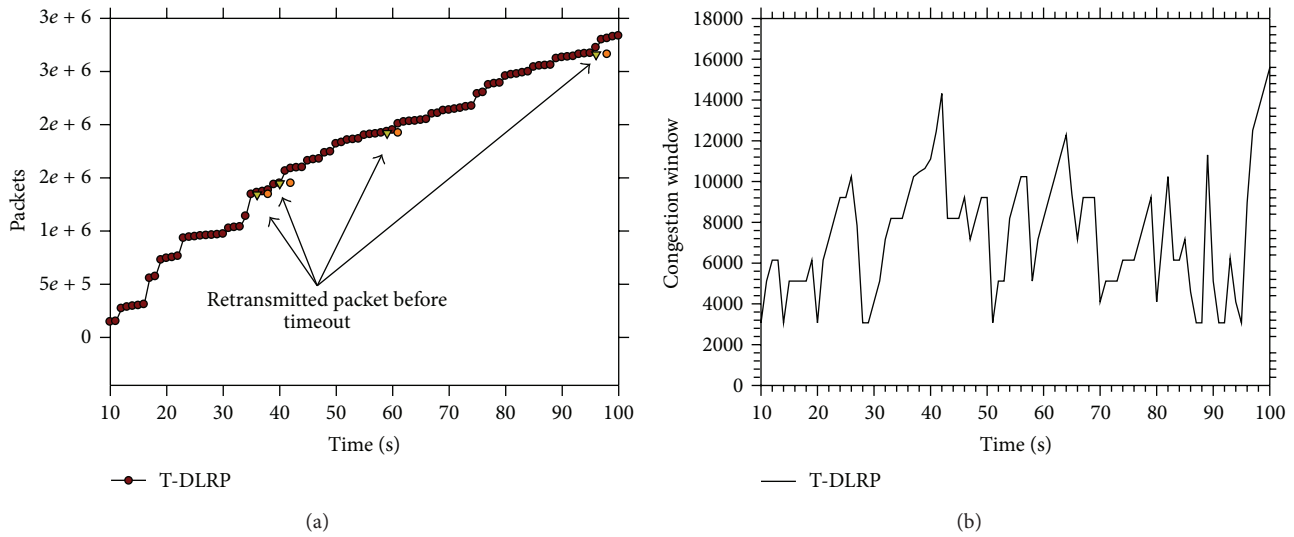

Figure 10: Behavior of T-DLRP. (a) Retransmission of lost fast retransmitted packets before timeout and (b) size of cwnd.

multihop wireless networks, we used 11 static nodes having 10 hops with an equal distance of 200 m apart as shown in Figure 13. As a result, all the nodes can communicate with each other. The bandwidth we used for the below experiments is 11 Mbps. Rest of the parameters are same as we mentioned in the Section 5.1. By using the multihop wireless topology, we evaluated the throughput performance of T-DLRP in terms of different number of hops, packet loss rate, and retransmission rate. In addition to that, we trace the number of timeouts happened during our simulations.

Figure 14(a) shows the throughput comparison of T-DLRP with other TCP variants in terms of number of hops. In this experiment, we used 3 TCP flows for causing congestion, 3% packet loss rate due to transmission errors, and 1% retransmission loss rate. As expected when hop increases, the throughput decreases. However, as expected the throughput of T-DLRP outperforms all other TCP variants. This is because even in the case of existence of packet losses and retransmission losses, T-DLRP can accurately differentiate and detect the losses and avoids the blind reduction of cwnd size and unnecessary retransmission timeouts. As a result, T-DLRP can send more data packets and can increase the performance compared to other schemes. When the number of hop reaches 10, the throughput of T-DLRP achieves 47% greater than TCP NewJersey and more than 50% greater than TCP NewReno. Figure 14(b) depicts the comparison of throughput in terms of packet loss due to fast retransmission which ranges from 0.01 to 0.1%. For this experiment, we used 2% packet loss by using the exponential error model and adjusted the downtime and uptime for imposing different rate of retransmission loss. T-DLRP achieves better throughput compared to DAC and other protocols. Even DAC can detect

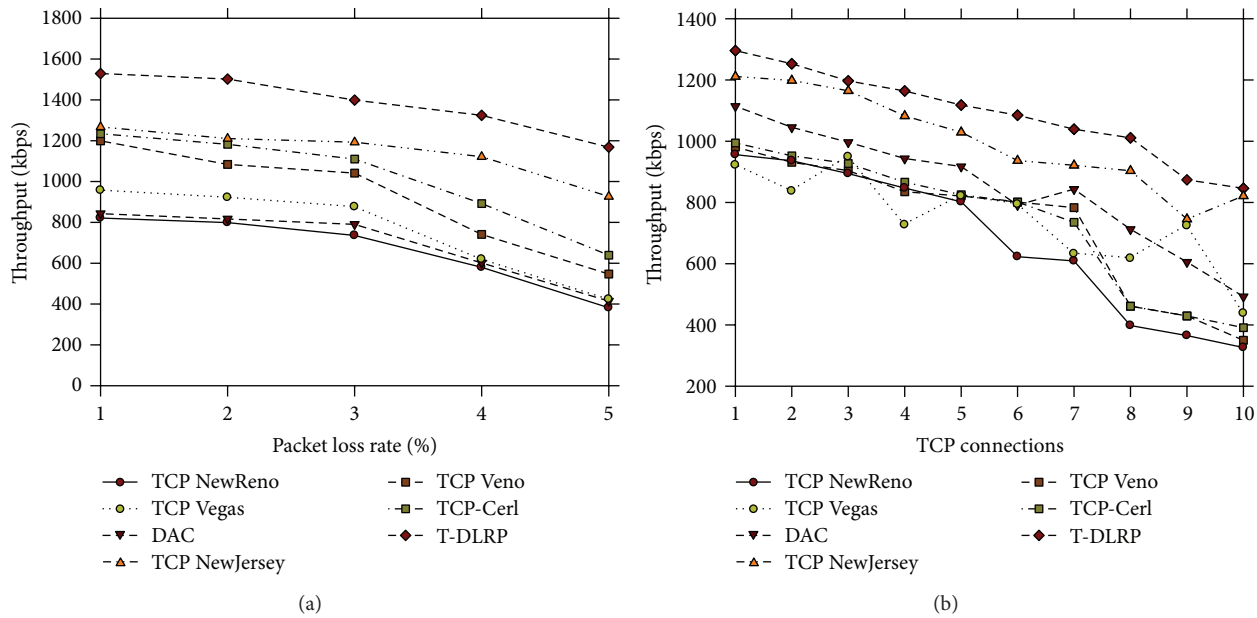

Figure 11: Comparison of throughput. (a) Throughput versus packet loss rate and (b) Throughput versus TCP connections.

the retransmission loss, and it assumes that all loss is due to network congestion and reduces the size of cwnd into half of the ssthresh value. Figure 15(a) presents the throughput performance at node 11 under different packet loss rate ranges from 1 to 5%. Node 1 acts as the source node and sends packets continuously to node 11. We use 0.05% retransmission loss in this experiment. At 5% packet loss rate except T-DLRP, all other protocols can achieve only less than 300 Kbps throughput. However, T-DLRP outperforms more than 85% throughput improvement with less number of timeouts compared to TCP NewJersey as shown in Figure 15(b).

5.4. Accuracy of T-DLRP with DAC. Accuracy is another performance metric that we used to evaluate the performance of T-DLRP with DAC. It is one of the most important metrics used in loss differentiation algorithms to evaluate the performance of TCP in addition to end-to-end throughput [21]. We measured the accuracy of retransmission loss due to congestion (A_{RLC}), wireless loss (A_{RLW}), and the average accuracy ($\mathrm{Average}_{\mathrm{RLCW}}$), where

$$A_{\mathrm{RLC}} = \left(\frac{N_{\mathrm{RLC}}}{N_{\mathrm{RLC\ Total}}}\right) * 100 \quad (9)$$
$$\left(A_{\mathrm{RLC}} \geq N_{\mathrm{RLC}} \geq 0,\ 100\% \geq A_{\mathrm{RLC}} \geq 0\%\right),$$

where N_{RLC} is the number of fast retransmission loss exactly identified as fast retransmission due to congestion loss by T-DLRP and DAC, and $N_{\mathrm{RCL\ Total}}$ is the total number of fast retransmission lost packet due to congestion:

$$A_{\mathrm{RLWL}} = \left(\frac{N_{\mathrm{RLWL}}}{N_{\mathrm{RLWL\ Total}}}\right) * 100, \quad (10)$$
$$\left(A_{\mathrm{RLWL}} \geq N_{\mathrm{RLWL}} \geq 0,\ 100\% \geq A_{\mathrm{RLWL}} \geq 0\%\right),$$

where N_{RLWL} is the number of fast retransmission loss exactly identified as fast retransmission due to congestion loss by T-DLRP and DAC, and $N_{\mathrm{RCL\ Total}}$ is the total number of fast retransmission lost packet due to congestion. The average accuracy is calculated using the equation below:

$$\mathrm{Average}_{\mathrm{RLCW}} = \frac{\left(A_{\mathrm{RLC}} + A_{\mathrm{RLWL}}\right)}{2}, \quad (11)$$
$$\left(0\% \leq \mathrm{Average}_{\mathrm{RLCW}} \leq 100\%\right).$$

Figure 16 presents the results for checking the accuracy of T-DLRP compare with DAC in terms of congestion based retransmission loss with different packet loss rates using mixed wired-wireless network topology. For this experiment, we used 10 TCP flows from source to destination via routers R_1 and R_2. For causing fast retransmission loss we changed the starting and ending time of each TCP flows and imposed 0.5% of retransmission loss due to network congestion. When the number of TCP connections increases, the accuracy decreases. However, compared to DAC, T-DLRP achieves highest accuracy because of its efficient detection of retransmission loss. At 10 number of TCP connection, T-DLRP outperforms more than 75% improvement in accuracy compared to DAC. The accuracy of DAC reduced to less than 60%. Figure 17 presents the accuracy of DAC and T-DLRP according to different packet loss rate ranges from 1 to 5% due to wireless loss. In this experiment, we used 0.5% retransmission loss for evaluating the accuracy. From the graph, it is clear that when the loss rate of packets increases, the accuracy tends to decrease. Although the accuracy is decreasing, T-DLRP gains higher accuracy compared to the existing scheme called DAC.

The reason is, using the modified Timestamp option of RFC 1323, T-DLRP can accurately detect the retransmission loss and modified the threshold value of TCP NewJersey;

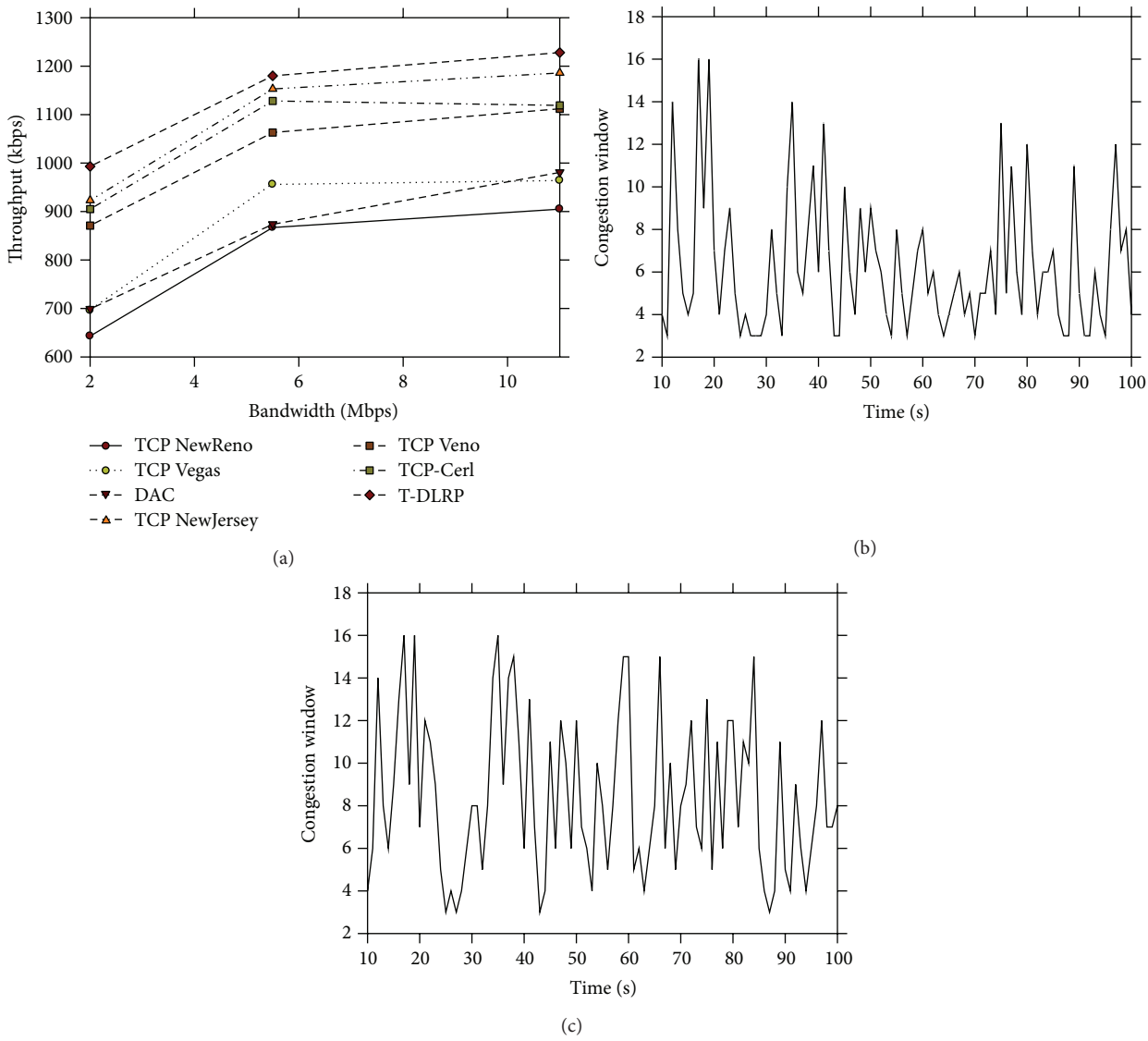

Figure 12: Comparison of throughput. (a) Different bandwidths and (b) cwnd size of DAC, (c) cwnd size of T-DLRP.

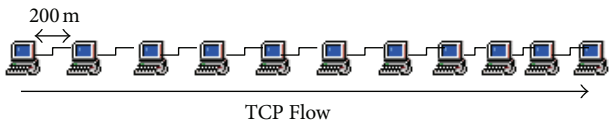

Figure 13: Ten-hop chain topology with single and multiple flows.

T-DLRP can efficiently differentiate the type of retransmission loss by reducing the misclassification of the types of retransmission loss. On the other hand, in the presence of wireless loss, DAC cannot achieve higher accuracy even the scheme can detect the retransmission loss because DAC has no method for differentiating the type of retransmission loss. As a result, DAC assumes that all retransmission loss was due to network congestion and the accuracy of DAC for detecting the retransmission loss due to wireless loss is 0% and it cannot increase or decrease according to different packet loss rates as shown in Figure 17. DAC continuously reduces the size of cwnd upon the detection of retransmission loss of each packet and thereby reduces the performance of TCP in wireless network. On the other hand, T-DLRP can efficiently differentiate the retransmission loss due to wireless loss compared to DAC and can send more packets without reducing the cwnd size. Figure 18 depicts the average accuracy of retransmission loss due to network and wireless loss under varying range of retransmission loss rate from 0.1% to 0.6%. Compared to DAC, T-DLRP outperforms DAC by showing the efficiency for detecting and differentiating the retransmission loss due to congestion and wireless loss. As a result, it is clear that when retransmission loss happens, T-DLRP can increase the performance of TCP by reducing the retransmission timeouts.

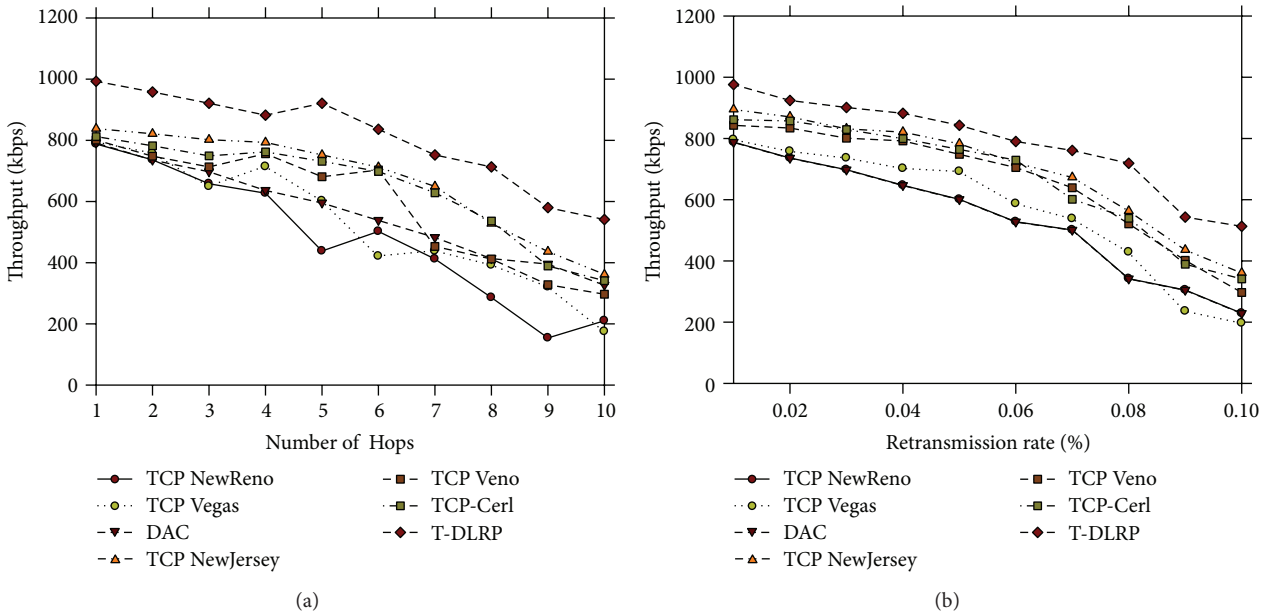

FIGURE 14: Comparison of throughput. (a) Throughput versus number of hops and (b) Throughput versus Retransmission rates.

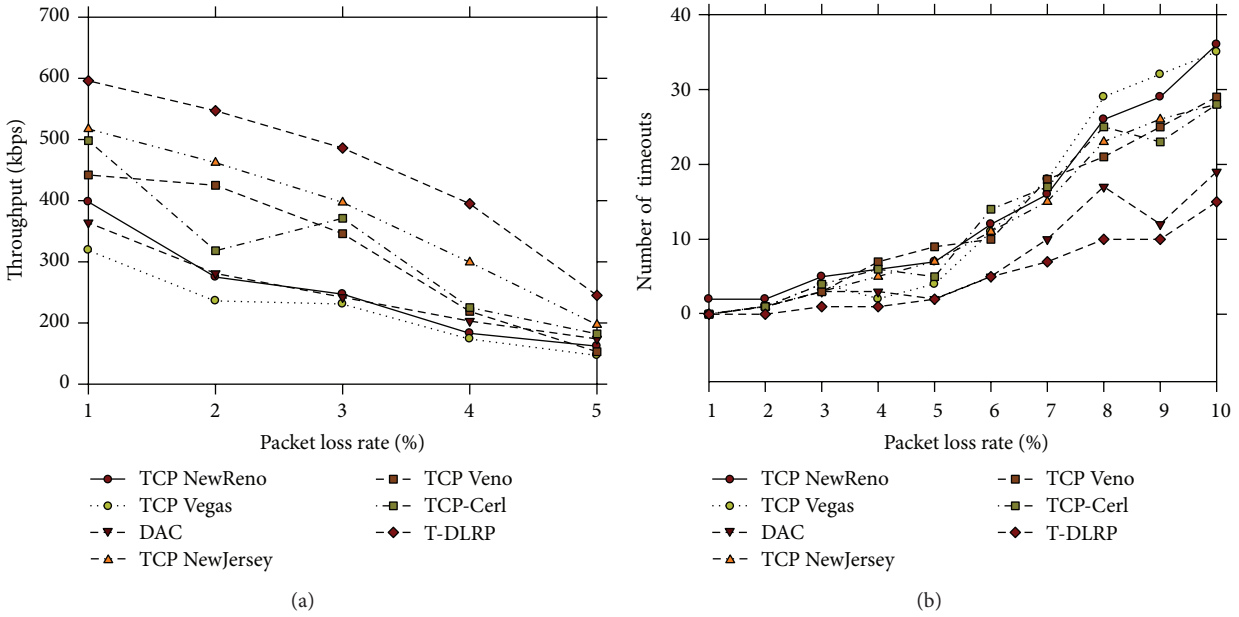

FIGURE 15: Comparison of throughput. (a) Throughput versus packet loss rates and (b) Timeouts versus Packet loss rates.

5.5. Comparison of Fairness Using Dumbbell Topology. Fairness is another important performance metric used for evaluating the performance of TCP over wireless networks. It is the bandwidth allocation measure for the multiple connections of the same TCP. The topology we used for measuring the fairness can be seen in Figure 19 which consists of sources (S to S_n) and destinations (D to D_n).

We used the Jain fairness index proposed in [22], to measure the fairness of T-DLRP compared to TCP NewReno. In our experiment, 10 TCP connections are used with 1% retransmission loss and 2% wireless loss with 6 Mb bottleneck link and 50 ms propagation delay. We run the simulation for T-DLRP and TCP NewReno and compare their fairness index. The results are summarized in Table 2. From Table 2, it is clear that T-DLRP can utilize more bandwidth compared to TCP NewReno. This is because T-DLRP can reduce the retransmission timeouts due to the loss of fast retransmitted packets and can guide the TCP sender to adjust the size of cwnd.

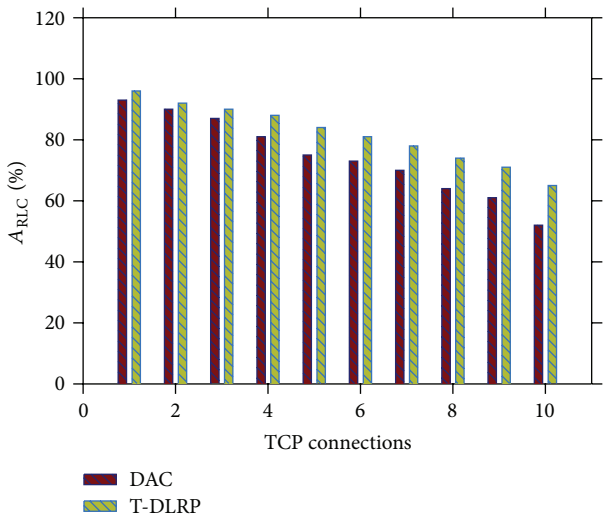

Figure 16: Accuracy of retransmission loss due to congestion.

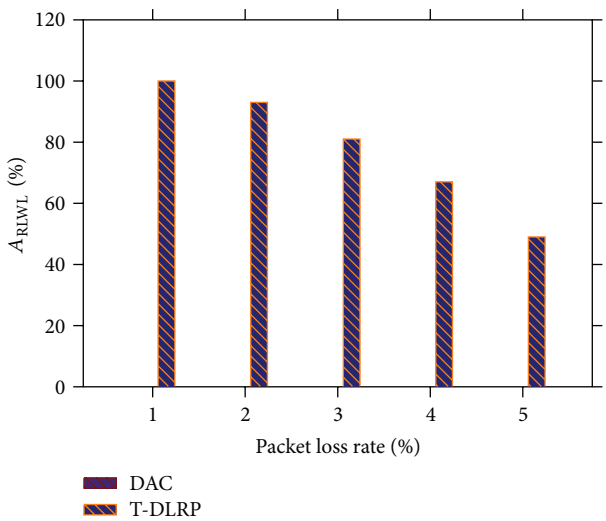

Figure 17: Accuracy of retransmission loss due to wireless loss.

Table 2: Comparison of fairness.

Error rate (%)	T-DLRP	TCP NewReno
0.1	0.9999	0.9999
0.5	0.9999	0.9997
1.0	0.9998	0.9992
5.0	0.9994	0.9986
10	0.9989	0.9981

6. Conclusion

An important issue, which we addressed in this paper, is reducing the frequent retransmission timeout due to the loss of fast retransmitted packets for improving the end-to-end performance of TCP over wireless networks. In order to reduce the retransmission timeouts, we have proposed a new TCP scheme called T-DLRP, which is capable of detecting the fast retransmission loss and reacting accordingly. T-DLRP modified the TCP timestamp option defined in RFC 1323 and used to detect the loss of fast retransmission even multiple loss of packets in a single window of data and the loss of its fast retransmission. T-DLRP consists of three key components, called FRL, IFR, and IRTOR. We used three types of network environments for our evaluation in terms of throughput, accuracy, and fairness. Results from experiments using qualnet demonstrate that our mechanism achieves more than 86%, 72%, and 40% throughput improvement over TCP NewReno, DAC, and TCP NewJersey, respectively; in a coexisted network. Compared to DAC, T-DLRP gains more than 70% improvement in accuracy over wireless networks. Also, our experiments of multiple connections of the same TCP flows reveal that T-DLRP has better fairness compared to TCP NewReno.

Figure 18: Average accuracy of retransmission loss due to network congestion and wireless loss.

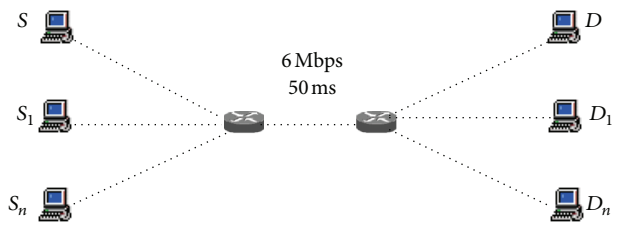

Figure 19: Network topology for the comparison of fairness.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A2043531).

References

[1] http://www.ietf.org/rfc/rfc793.txt.

[2] G. Xylomenos and G. C. Polyzos, "Internet protocol performance over networks with wireless links," *IEEE Network*, vol. 13, no. 4, pp. 55–63, 1999.

[3] K. V. Anusuya and C. B. Krishna, "Performance analysis of an efficient TCP variant under lossy environments," in *Proceedings of the International Conference on Advanced Computer Control (ICACC '09)*, pp. 657–660, Singapore, January 2009.

[4] N. J. Kothari, B. M. Gambhava, and K. S. Dasgupta, "RTT utilization by detecting avoidable timeouts," in *Proceedings of the 14th IEEE International Conference on Networks (ICON '06)*, pp. 1–6, Singapore, September 2006.

[5] D. Ciullo, M. Mellia, and M. Meo, "Two schemes to reduce latency in short lived TCP flows," *IEEE Communications Letters*, vol. 13, no. 10, pp. 806–808, 2009.

[6] D. Kim, B. Kim, J. Han, and J. Lee, "Enhancement to the fast recovery algorithm of TCP newreno," in *Information Networking. Networking Technologies for Broadband and Mobile Networks*, vol. 3090 of *Lecture Notes in Computer Science*, pp. 332–341, 2004.

[7] S. Prasanthi and S. H. Chung, "An efficient algorithm for the performance of TCP over multi-hop wireless mesh networks," in *Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG '10)*, pp. 816–821, Las Vegas, Nev, USA, April 2010.

[8] C. Y. Ho, Y. C. Chen, Y. C. Chan, and C. Y. Ho, "Fast retransmit and fast recovery schemes of transport protocols: a survey and taxonomy," *Computer Networks*, vol. 52, no. 6, pp. 1308–1327, 2008.

[9] B. Kim and J. Lee, "A simple model for TCP loss recovery performance over wireless networks," *Journal of Communications and Networks*, vol. 6, no. 3, pp. 235–244, 2004.

[10] 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.

[11] H. El-Ocla, "TCP CERL: congestion control enhancement over wireless networks," *Wireless Networks*, vol. 16, no. 1, pp. 183–198, 2010.

[12] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–228, 2003.

[13] J. Lee, J. Kim, M. Park, J. Koo, and H. Choo, "NJ+: an efficient congestion control mechanism for wireless networks," *KSII Transactions on Internet and Information Systems*, vol. 2, no. 6, pp. 333–351, 2008.

[14] S. Prasanthi, S. H. Chung, and C. W. Ahn, "An enhanced TCP mechanism for detecting and differentiating the loss of retransmissions over wireless networks," in *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications (AINA '11)*, pp. 54–61, Singapore, March 2011.

[15] http://www.ietf.org/rfc/rfc2582.txt.

[16] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[17] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 4, pp. 747–756, 2004.

[18] P. Sreekumari, S. H. Chung, and W. S. Kim, "A Timestamp based detection of fast retransmission loss for improving the performance of TCP NewReno over wireless networks," in *Proceedings of the IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob '2011)*, pp. 60–67, Wuhan, China, October 2011.

[19] http://www.ietf.org/rfc/rfc1323.txt.

[20] http://web.scalable-networks.com/content/qualnet.

[21] M. Y. Park, S. H. Chung, and P. Sreekumari, "End-to-end loss differentiation algorithm based on estimation of queue usage in multi-hopwireless networks," *IEICE Transactions on Information and Systems*, vol. 92, no. 10, pp. 2082–2093, 2009.

[22] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," Research Report TR-301, 1984.