EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

# TCP NRT: a new TCP algorithm for differentiating non-congestion retransmission timeouts over multihop wireless networks

Prasanthi Sreekumari and Meejeong Lee*

**Abstract**

In multihop wireless networks, reliable data transfer is one of the most difficult tasks. When transmission control protocol (TCP) operates in multihop wireless networks, the performance of TCP reduces drastically. TCP retransmission timeouts (RTOs) related to non-congestion events such as spurious and random packet losses have been reported as one of the main problems in the performance degradation of TCP in multihop wireless networks. The RTOs triggered by random packet losses due to transmission errors lead to unnecessary reduction of TCP congestion window size, and the spurious RTOs due to sudden delay of packets on the network paths often cause unnecessary retransmissions as well as reduction of congestion window size. Existing solutions for detecting non-congestion RTOs have no mechanism to differentiate the spurious RTOs from RTOs caused by random packet loss. In this paper, we introduce an efficient algorithm called non-congestion retransmission timeouts (TCP NRT) which is capable of recovering packets after RTOs by reducing unnecessary retransmissions and needless reduction of congestion window size in order to improve the performance of TCP in multihop wireless networks. TCP NRT consists of three key components: NRT-detection, NRT-differentiation, and NRT-reaction. We implemented the algorithm in Qualnet network simulator and compared its performance to existing TCP versions. Results from the experiments show that our algorithm achieves significant performance improvement in terms of throughput and accuracy. Also, the results showed that our algorithm, TCP NRT, maintains a fair and friendly behavior compared to the most widely deployed TCP, NewReno.

**Keywords:** Wireless networks; TCP; RTOs; Congestion loss; Non-congestion events

## 1. Introduction

In wireless local area and cellular networks, communication using wireless links occurs only in the last link between a base station and the wireless end system. Multihop wireless networks (MWNs) are a wireless network adopting the multihop wireless technology without deployment of wired backhaul links. MWNs have increased in importance and usage at the edge of the Internet over the past several years. It can be deployed in a cost-efficient way and can avoid wide deployment of cables. In MWNs, there are one or more intermediate nodes along the path that can receive and forward packets via wireless links [1]. One of the most important benefits of MWNs is the capability to extend the coverage of a network and improve the quality of connectivity, compared to network with single wireless links. Several paths might become available in the case of dense MWNs that can be used to increase the robustness of the network. Although MWNs are very useful, the most important task to be accomplished in MWNs is the reliability of data transmission due to current limitations in wireless communications.

Transmission control protocol (TCP) is very important for reliable data transmission as it is the most popular transport protocol, and also, it is the de facto standard in the Internet. In recent years, the performance of TCP has acquired great attention, and it is an active research area among research topics on MWNs [1]. TCP is implemented as a reliable data transfer protocol in wired networks. The congestion control algorithms of TCP are very essential for the reliability of data transmission as

* Correspondence: lmj@ewha.ac.kr
Department of Computer Science and Engineering, Ewha Womans University, Seoul 120-750, Republic of Korea

well as stability of the Internet. The success of TCP in wired networks motivates its extension to wireless networks [2]. However, the data transmission of TCP is no longer stable in wireless networks. Particularly, when TCP operates in MWNs, the performance degrades significantly by increasing the number of hops. It is well known that the main reason for TCP performance degradation is its inability to distinguish the causes of three duplicate acknowledgments and retransmission timeouts (RTOs). Among these, the RTOs caused by non-congestion events have been reported as one of the main problems in TCP performance degradation in MWNs [3]. Mainly, there are two main types of non-congestion RTOs. They are as follows: (1) spurious RTOs due to sudden delays or reordering of packets and (2) RTOs due to random packet losses caused by wireless transmission errors. These types of RTOs are unavoidable in MWNs.

Our work is motivated by three main observations. First, recent Internet measurement studies [4,5] show that about 70% of the dropped packets are recovered after the expiration of RTO. In wireless networks, congestion is very rare, and frequent RTOs are often due to transmission errors [3]. As a result, the sender reduces its sending rate unnecessarily, and it affects the performance of TCP in MWNs. Recently, researches on the performance of TCP reveal that spurious timeouts due to the increase in unexpected delay can considerably degrade the performance of TCP. In addition, the authors [2] show that in MWNs, more than 20% of the RTOs are caused spuriously by transmission delay of packets which results in unnecessary retransmissions and needless reduction of congestion window (cwnd) size. It takes a long time to reopen the window and is costly for high-bandwidth links [6]. In conventional TCP, when RTO is invoked, the sender assumes that the packet is lost due to network congestion and immediately retransmits all the outstanding packets and sets the cwnd size to one maximum segment size (mss), which then increases according to slow-start algorithm.

Second, previous research indicates that spurious RTOs are not rare events in MWNs [7,8]. After the spurious RTO, the late-arriving acknowledgments (Acks) can encourage TCP to retransmit all the packets in flight, which may all have been correctly received by the TCP receiver. This unnecessary retransmission behavior leads to the under utilization of the network resources such as available bandwidth. Not only that, each redundantly retransmitted packet will be responded to with a duplicate Acks by the TCP receiving side. Fast retransmission algorithm can be triggered if the number of duplicate Acks accumulates over the fast retransmit threshold (normally, it is equal to three). In addition to that, after the spurious RTOs, TCP starts the slow-start algorithm, and each original Ack received in the slow-start will trigger out two or three segments, more than the number of packets which have actually left the pipe. This TCP behavior is a violation of the principle of TCP packet conservation [2].

Third, several modifications have been proposed for TCP loss recovery and congestion control mechanisms to improve the performance of TCP. However, the existing solutions [2,6,9-11] proposed for detecting congestion RTOs and non-congestion RTOs have no mechanism to differentiate the two different types of non-congestion RTOs such as random loss and spurious RTOs. When spurious RTOs coexist with RTOs due to random packet losses, it results in unnecessary retransmissions and needless reduction of cwnd size, which leads to the performance degradation of TCP in MWNs. As a result, it is an important issue whether the TCP sender can control unnecessary retransmissions by differentiating the types of RTOs due to random packet losses from spurious RTOs.

In this paper, we develop a new TCP algorithm for differentiating non-congestion RTOs (TCP NRT), which are caused by transmission errors and sudden delays on the network path, and thereby improve the performance of TCP in MWNs. TCP NRT consists of three key components. They are as follows:

1. Detection of non-congestion RTOs (NRT-detection): To detect non-congestion RTOs from congestion RTOs using the improved explicit congestion notification (ECN) mechanism.
2. Differentiation of non-congestion RTOs (NRT-differentiation): To differentiate the RTOs due to random packet losses from spurious RTOs by using the comparison of the first Ack after the expiration of RTOs.
3. Reaction to non-congestion RTOs (NRT-reaction): To guide the TCP sender to control the unnecessary reduction of cwnd size and to control the needless retransmissions according to the types of RTOs.

With the help of these components, TCP NRT can improve the performance of TCP in MWNs. We implemented TCP NRT in a Qualnet network simulator and compared its performance using important metrics such as throughput, accuracy, fairness, and friendliness with the existing TCP versions such as Eifel, F-RTO, DSACK, EQRTO, and NewReno. The results demonstrate that TCP NRT achieves significant improvement in throughput and accuracy compared to other TCP versions, especially when RTOs occur in non-congested environments. Moreover, the simulation results show that when RTOs occurred in a congestion-free network with packet loss rate ranging from 1% to 9%, the TCP NRT performance achieves 29% higher throughput than EQRTO and more than 40% throughput improvement over Eifel, DSACK,

and F-RTO especially at 9% packet loss rate due to transmission errors. In addition, the experiment on multiple TCP flows clearly shows that TCP NRT maintains a fair and friendly behavior with respect to other TCP flows. The remainder of this paper is organized as follows: Section 2 describes the problem of TCP RTOs. In Section 3, we briefly summarize the existing research related to the work done in this paper. We introduce TCP NRT in Section 4, where the main features of each component are discussed in detail. Section 5 describes the performance evaluation of TCP NRT against existing protocols. Finally, Section 6 concludes our work by pointing out our major achievements.

## 2. The problem: retransmission timeouts of TCP

In the current implementation of TCP, whenever TCP transmits a segment, the sender starts a timer which keeps track of how long it takes for an Ack of that segment to return. This timer is known as the retransmission timer. If an Ack is returned before the timer expires (by default is often initialized to 1.5 s), the timer is reset with no consequence. However, if an Ack for the packet does not return within the timeout period, the sender would retransmit the packet and double the retransmission timer value for each conservative timeout up to a maximum of about 64 s [12]. When RTO expires, the sender assumes that the packet is lost and immediately retransmits the first unacknowledged packets and sets the cwnd size to one mss and the slow-start threshold (ssthresh) to half of the cwnd size. After retransmitting the packet, the sender continues to send packets by invoking the slow-start algorithm and increases the size of cwnd to one mss on each Ack. If the cwnd reaches the ssthresh size, the sender enters congestion avoidance algorithm in which the cwnd size increases linearly based on round trip time (RTT). In this way, the sender can recover the lost packets efficiently, and it works very well in wired networks because most of the RTOs are due to the congestion in the network. However, in the case of MWNs, this assumption is no longer true. In MWNs, the RTOs are due to transmission errors, sudden delay, link level recovery, physical disconnection of a wireless link, packet reordering, etc. rather than the congestion of the network [13]. As a result, the current RTO recovery mechanism of TCP causes performance degradation in MWNs. One of the main reasons is that TCP has no mechanism to detect and differentiate the non-congestion RTOs. As shown in Figure 1, we classify the RTOs into two main types. They are congestion RTOs and non-congestion RTOs. Congestion RTOs are those RTOs triggered by packet losses due to network congestion, whereas non-congestion RTOs are those triggered by packet losses without any network congestion.

As we mentioned in the previous section, mainly, there are two main types of non-congestion RTOs. One is the RTOs caused by random packet losses due to transmission errors, and the other is the RTOs triggered spuriously due to sudden delays or due to reordering of packets on the network path without any packet losses. If RTO expires due to random packet losses, the TCP sender retransmits the packet and unnecessarily reduces the size of cwnd. On the other hand, when RTO occurs spuriously, the sender not only reduces the size of cwnd but also retransmits the packet unnecessarily. As a result, the TCP sender cannot utilize the available bandwidth fully. This affects the performance of TCP in MWNs. To observe the throughput degradation of TCP in MWNs, we did an experiment using seven-hop chain topology and measured the throughput and the number of retransmissions triggered by fast retransmission and RTO procedures according to the number of hops. Although it is well known that the performance of TCP sharply decreases as the number of hop increases in MWNs, there is no experiment showing the variation of TCP performance according to the ratio of retransmissions caused by random losses and spurious packet losses. Thus, we did a simple experiment in order to know the ratio of retransmissions caused by random loss and spurious RTOs to that of retransmissions by triggering the fast retransmission algorithm. In this simulation,

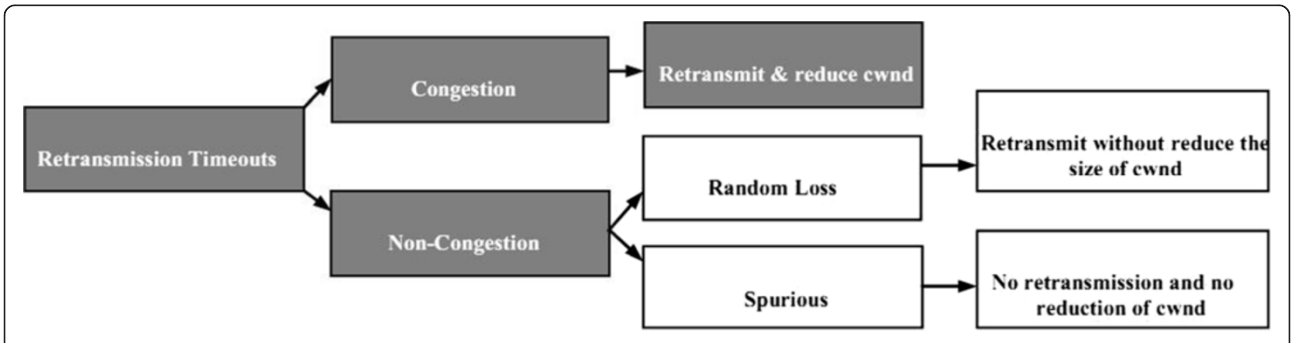


**Figure 1** Congestion control algorithms of TCP.

we used one TCP flow from source to destination in a chain topology with no network congestion and thereby evaluate the unnecessary degradation of throughput due to RTOs.

Figure 2a shows the throughput degradation of TCP according to the number of hops in a congestion-free network with random packet loss RTOs and spurious RTOs when the bandwidth of the wireless channel is 9Mbps. When the number of hop increases to seven, the TCP cannot achieve more than 1.5 Mbps of throughput. The main reason for this degradation is shown in Figure 2b. From this figure, it is clear that the number of retransmissions due to RTOs is higher than that of fast retransmissions. Among that, in our experiment, more than 40% of the TCP retransmissions happen needlessly due to spurious RTOs. As a result, it is very important to differentiate random loss RTOs from spurious RTOs to reduce unnecessary retransmissions and reduction of the cwnd size.

## 3. Related work

There have been many solutions proposed for improving the performance of TCP in MWNs. In this section, we briefly summarize the existing research related to the work done in this paper. We classify the related work into two parts. The first part depicts the solutions to detect congestion and spurious RTOs, and the second part presents the solutions to detect congestion and non-congestion RTOs.

### 3.1. Solutions to detect congestion RTOs and spurious RTOs

In MWNs, spurious RTOs are inevitable because by no means can TCP predict and avoid the link delay spikes. The major difference in spurious RTO algorithms lies in how a spurious timeout is detected, which is equivalent to solving retransmission ambiguity in many circumstances. By clarifying the retransmission ambiguity, TCP can confirm whether or not data have been unnecessarily retransmitted and further conclude if a spurious RTO has happened. The following are the existing algorithms for detecting congestion and spurious RTOs.

- Eifel. The Eifel algorithm [9], specified in RFC 3522, uses the TCP timestamp option [14] for detecting spurious and congestion RTOs. The key idea is when the sender sends packet, it stored the current value of the timestamp clock into the header of each outgoing packets. When the receiver receives that packet, it echoes the timestamp value back to the sender in the corresponding Acks. After the expiration of a RTO, the sender stores the timestamp of the retransmitted packet. Upon the arrival of the first Ack after RTO, the TCP sender compares the timestamp of the Acks with the previously stored values. If the value of the Ack is smaller than the stored value, then the sender assumes that the RTO was spurious. The Eifel algorithm uses extra information in the Acks to eliminate retransmission ambiguity, thereby solving the problem of spurious retransmissions.
- DSACK. The DSACK algorithm [10], specified in RFC 3707, is an extension to the 'selective acknowledgment' (SACK) option; the sender uses the first SACK block to specify a segment that triggers a duplicate Ack at the receiver. The DSACK algorithm uses this feature of SACK for detecting spurious RTOs. The general idea is that if a retransmitted packet has been acknowledged for the second time, the sender assumes that the earlier retransmission

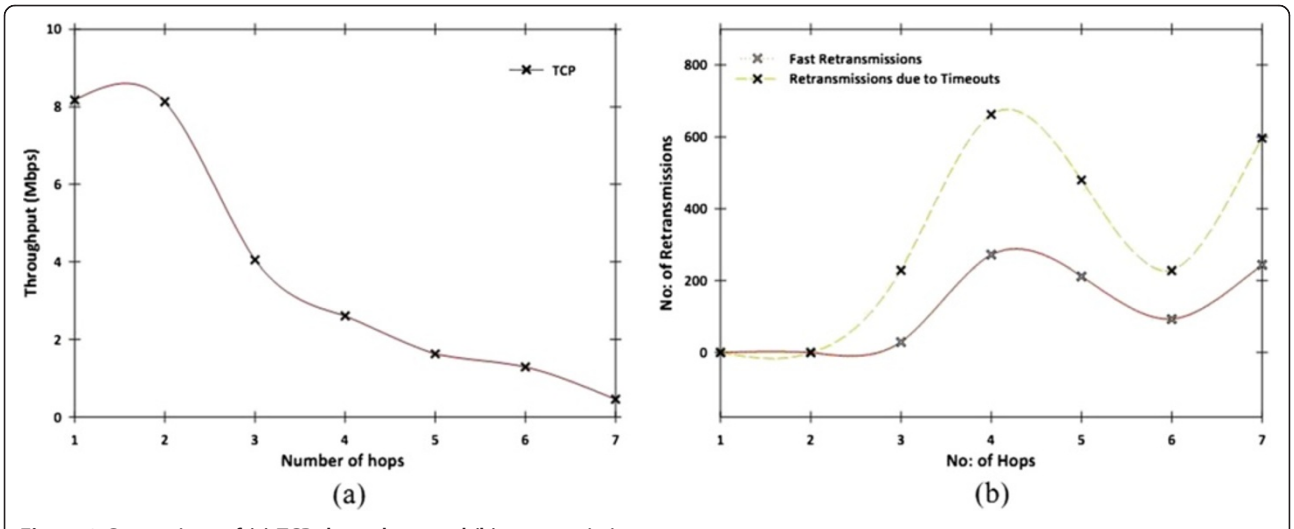


**Figure 2** Comparison of (a) TCP throughput and (b) retransmissions.

was spurious. DSACK introduces only less traffic overhead than Eifel does since the TCP receiver only needs replying with SACK or DSACK options upon the detection of out-of-order packets, while a timestamp option has to be attached to every TCP packet and Ack in the case of Eifel. However, the slow reaction of DSACK judgment makes the TCP retransmit several packets mistakenly, or in the worst case, a whole window of packets has to be retransmitted before the situation is clear.

- F-RTO. The forward RTO-recovery (F-RTO) algorithm [6], specified in RFC 4138, follows a distinct rationale for detecting spurious RTOs without using any TCP options. The key idea is that at the time of RTO, the TCP sender enters the slow-start algorithm, retransmits one outstanding packet, and then monitors the first incoming Ack; after retransmission advances the value of cwnd, the TCP sender will retransmit two new packets. Again if the second Ack advances the cwnd, then the sender interprets that the RTO is spurious. Otherwise, a congestion RTO occurs, and the sender will revert to the traditional slow-start algorithm to retransmit the outstanding packets. F-RTO requires no TCP extension for its application and has higher link utilization than TCP Eifel. However, F-RTO may postpone the response for a genuine RTO if the packet loss and the delay spike run in the same transfer window.
- STODER. Spurious timeout detection by repacketization (STODER) [11] detects retransmission ambiguity using TCP repacketization. When a RTO expires, a TCP sender repacketizes a packet which is k-bytes smaller than the original packet and retransmits it instead of the original packet which triggered the RTO. Then, it detects the spurious RTOs when an Ack arrives after the RTO by examining the number of bytes that are acknowledged. If the acknowledged bytes are the same with the size of the original packet, it assumes that the RTO is a spurious RTO. These schemes are very effective to distinguish spurious RTOs from congestion RTOs and let TCP avoid unnecessary retransmissions and the improvement of TCP to cope better with spurious RTOs. However, these schemes have no mechanism to detect the RTOs caused by random packet loss due to transmission errors, which is harmful to the performance degradation of TCP in MWNs.

### 3.2. Solutions to detect congestion RTOs and non-congestion RTOs

To the best of our current review of literatures, the only study in the direction of detecting non-congestion RTOs and congestion RTOs has been carried out in [7] for avoiding of unnecessary cwnd size reduction, thereby solving the performance degradation of TCP in MWNs.

Estimation of queue usage for detecting RTOs (EQRTO) [7] is able to distinguish spurious and random packet loss RTOs from congestion RTOs to improve the performance of TCP in MWNs. The main idea is that in order to identify congestion RTOs, EQRTO estimates the rate of queue usage during the go-back-N retransmissions in the network path between a TCP sender and a TCP receiver. When the go-back-N retransmission ends, EQRTO checks if the network queue usage indicates whether the network is congested or not. If yes, the TCP sender assumes the RTO as congestion RTO; otherwise, it assumes that RTOs occurred due to random packet losses or spurious RTOs. The existing algorithms affect the behavior of the TCP sender only after the expiration of RTOs. In the case of three duplicate Acks, these algorithms work the same as the conventional TCP. Unfortunately, the above solutions have limitations in differentiating the RTOs caused by random packet losses from spurious RTOs. In the case of EQRTO, the algorithm can detect congestion and random loss RTOs as well as congestion and spurious RTOs. However, the algorithm cannot differentiate the random loss RTOs from spurious RTOs. If the TCP sender cannot differentiate the spurious RTOs from random loss RTOs, it results in the degradation of the TCP performance in MWNs. By considering the limitations of the above solutions, we develop a new TCP algorithm for the differentiation of non-congestion RTOs, thereby increasing the performance of TCP in MWNs.

## 4. Proposed algorithm: TCP NRT

The main motivation of our algorithm, TCP NRT, is to recover packets efficiently from RTOs by differentiating random packet loss RTOs from spurious RTOs to improve the performance of TCP in MWNs. This algorithm affects the behavior of the TCP sender only when RTO occurs; otherwise the sender behaves similarly to conventional TCP. As shown in Figure 3, TCP NRT consists of three key components, namely NRT-detection, NRT-differentiation, and NRT-reaction. The contributions of each component are explained in detail in the following subsections.

### 4.1. Detection of non-congestion RTOs (NRT-detection)

To differentiate the RTOs due to random packet losses from spurious RTOs (together we call 'non-congestion RTOs'), first we should distinguish these RTOs from congestion RTOs. In order to detect non-congestion RTOs and congestion RTOs, we modified the ECN as it has been approved as an Internet official protocol

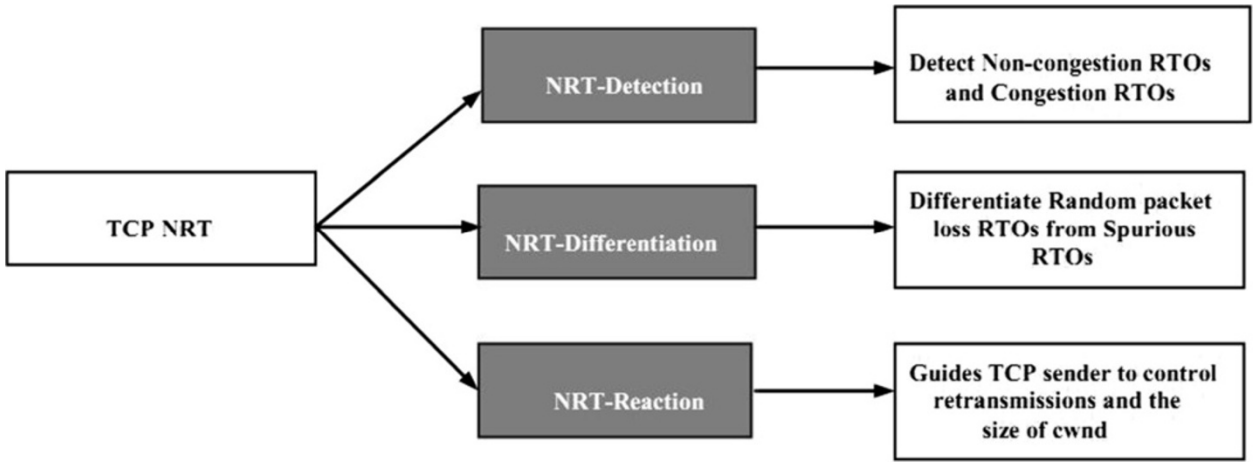


**Figure 3** Components of the TCP NRT algorithm.

standard in RFC 3168 [15] and is recommended to be widely deployed as a router mechanism. This is because ECN provides higher network efficiency, fairer distribution of bandwidth, less packet losses, and less bursty traffic; and benefits to short flows present a strong case for its implementation. ECN requires an active queue management mechanism in order to work [16]. Most of the active queue management schemes can achieve high performance with enabled ECN even if the load is more than 85% [17]. ECN is an extension to the random early detection (RED) algorithm for dropping packets. First, we explain the mechanism of RED. RED is the most widely deployed queue management scheme in commercial Internet routers [18]. The key idea behind the RED active queue management scheme is to inform the sender about the detection of incipient congestion by dropping the packets even when there is available buffer space. This type of probabilistic packet drop prevents the router from entering the fully congested state and helps reduce the queueing delay. However, in the case of large TCP flows, RED drops packets in higher loss rates and thereby degrades network performance since it requires extra time and causes TCP retransmissions which add to network congestion. As a result, instead of dropping packets randomly, an ECN router marks the packets to alert the sender of incipient congestion. Recent studies show that the performance of RED in the absence of ECN is worst than drop tail queue [17].

An ECN-capable router is configured with three main parameters. They are the minimum threshold ($\min^{th}$), maximum threshold ($\max^{th}$), and the maximum marking probability ($P_{\mathrm{Max}}$). When a packet arrives at the router, ECN calculates the average queue length (AQL), and if the AQL is below $\min^{th}$, the router will not mark the packet [19]. If the AQL exceeds $\min^{th}$ and below $\max^{th}$, the router marks the packet with probability ($P$),

$$P = \left((\mathrm{avg} - \min^{\mathrm{th}})/(\max^{\mathrm{th}} - \min^{\mathrm{th}})\right) P_{max}$$

As a result, the sender reduces the size of cwnd to control the sending rate in advance before the network becomes heavily congested. If the AQL exceeds $\max^{th}$, the TCP router marks the packets with probability 1. As we mentioned in the previous section, in MWNs, network congestion is very rare. As a result, there is no need to mark and reduce the size of cwnd when the AQL is between $\min^{th}$ and $\max^{th}$. This slows down the sending rate, and the network cannot utilize the bandwidth fully. To avoid the slowdown of sending rate unnecessarily, TCP NRT marks the packets only when the AQL is greater than or equal to $\max^{th}$, as shown in Figure 4. As a result, the sender can send more packets than the original ECN without reducing the cwnd size needlessly before the sender detects a packet loss by duplicate Acks or RTOs. To check the effectiveness of the modified ECN, we did an experiment using ten-hop chain topology and trace the size of cwnd during the transmissions to node 11 from node 1 while causing packet losses by congestion and non-congestion. Figure 5a

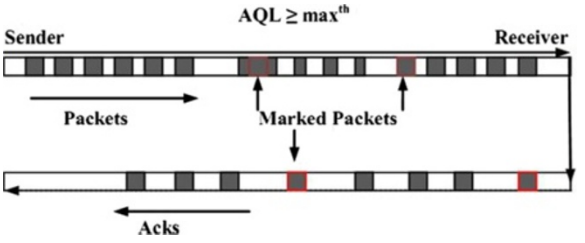


**Figure 4** Packet marking method of the modified ECN.

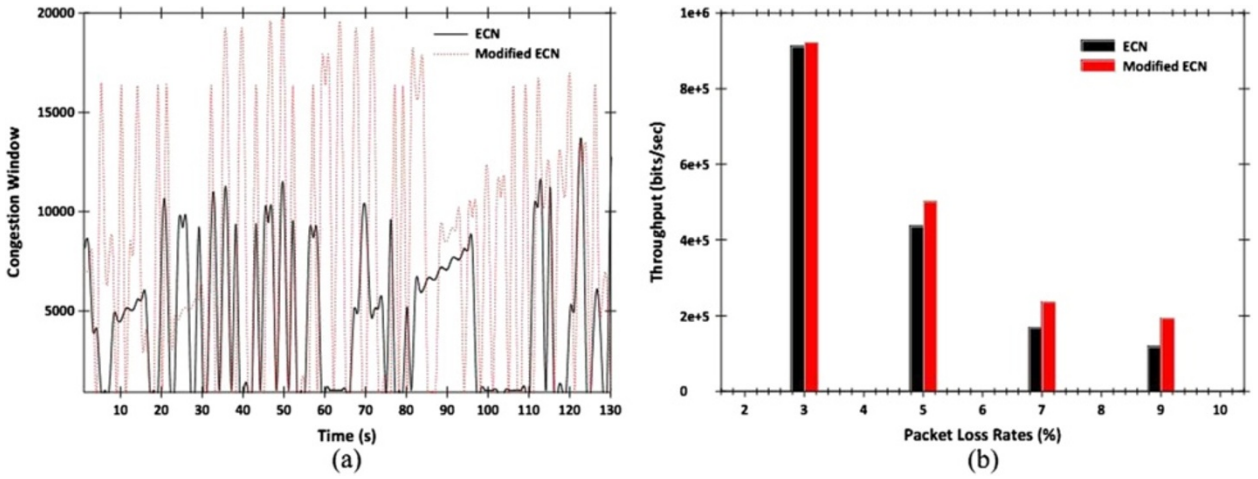


**Figure 5** Congestion window size (a) of the ECN and modified ECN (b) throughput performance.

shows the cwnd size of ECN and the modified ECN. The original ECN is not able to increase the size of cwnd because of its unnecessary cwnd reduction due to packet marking when AQl reaches between minimum and maximum thresholds.

On the other hand, the modified ECN can increase its sending rate and leads to improvement in TCP performance. This is because the modified ECN reduces its window size only when the threshold reached maximum. As a result, the sender can send more packets by increasing the window size. This type of marking helps the sender to use the proper congestion control strategy in various circumstances. In addition, the TCP connections with large window sizes are more tolerant of packet losses than those with small windows. It takes only one RTT to recover from multiple packet losses [16]. Moreover, we evaluated the throughput performance of the modified ECN against the original ECN by causing packet losses. Figure 5b shows that the modified ECN achieves better throughput compared to the original ECN. When RTO expires during packet transmission, the sender checks whether the last received Ack before RTO is marked or not by using the modified ECN mechanism, as shown in Figure 6a, to detect congestion and non-congestion RTOs. If the Ack is marked, the sender assumes that the network is congested, and the corresponding packet is lost. Otherwise, the sender assumes that the RTO was triggered due to non-congestion events such as random packet loss or spurious RTOs due to sudden delays on the network path.

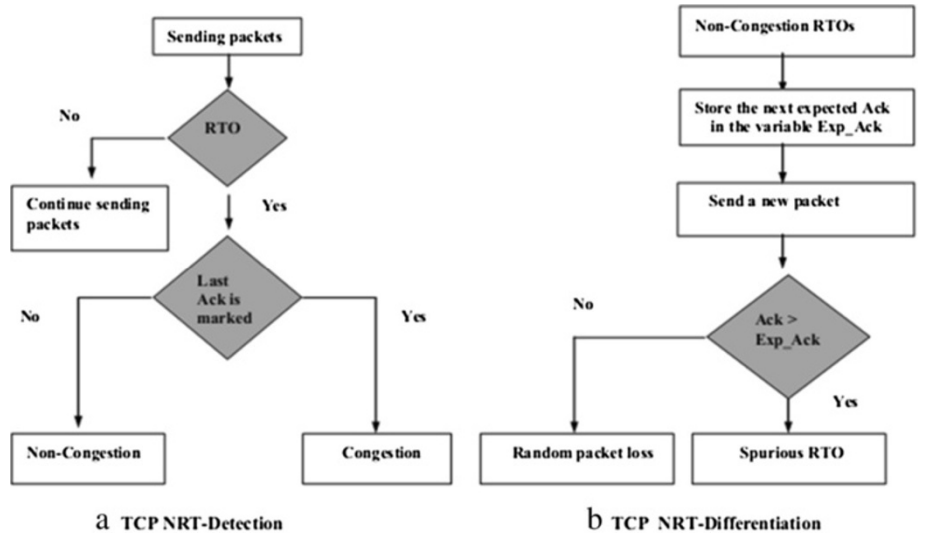


**Figure 6 Detection and differentiation of non-congestion RTOs. (a)** TCP NRT-detection and **(b)** TCP NRT-differentiation.

### 4.2. Differentiation of non-congestion RTOs

In this subsection, we explain in detail how TCP NRT differentiates the RTOs due to random packet losses from spurious timeouts. Spurious timeouts due to sudden delay are not rare in MWNs. As a result, it is very important to differentiate the RTOs due to random packet loss from spurious timeouts in order to reduce unnecessary retransmissions and needless reduction of cwnd size. As we explained in the previous subsection, when RTO expires at the sender, it checks the last received Ack to confirm whether it is marked or not. If it is not marked, it means that the RTO is due to random packet loss or spurious RTOs. As a result, the sender stores the value of the next expected Ack in a variable 'Exp_Ack'. Then the sender sends a new packet instead of retransmitting the timeout packet, as shown in Figure 6b. The reason is that non-congestion RTO happens due to transmission errors or sudden delays. It means that the buffer has space to accommodate a new packet. When the sender receives an Ack after sending a new packet, it checks whether the Ack is greater than the stored value. If it is greater, it means that the RTO was spurious; otherwise the sender retransmits the lost packet immediately without reducing the size of cwnd.

Recent Internet measurement studies [20] show that the time taken for the newly sent packet to be delivered is greater than the time taken to reach the delayed packet at the destination. As a result, the comparison of the first incoming Ack after the non-congestion RTOs with the stored value in the variable Exp_Ack at the time of RTO helps the sender to confirm accurately that the RTO triggered due to random packet losses or spurious timeouts. To understand better, Figure 7 demonstrates an example of how TCP NRT differentiates the RTOs due to random packet losses from RTOs that occur spuriously. Consider that the sender transmits 6 packets from P1 to P6, and packet P6 is lost due to transmission error. When the receiver receives the packet P5, the receiver sends an Ack of packet P5 by sending the next expected packet P6. As a result, the sender sends the packet P6, and it is lost. Due to the lack of enough duplicate Acks to trigger fast retransmission, the sender needs to wait for the expiration of RTO. When RTO is triggered, TCP NRT checks whether the last Ack is marked or not.

In this example, consider that the last Ack is not marked. As a result, the sender confirms that the RTO happened due to a non-congestion event, either random packet loss or sudden delay. Instead of retransmitting the packet P6 immediately, the sender sends a new packet P7 and stores the next expected Ack in the variable Exp_Ack. As soon as the packet P7 reaches the destination, the receiver again sends an Ack for the lost packet P6. The sender compares the Ack with the stored value in the variable Exp_Ack to check whether the first Ack after the non-congestion RTO is greater than Exp_Ack. Here, it is equal to the value in the Exp_Ack. As a result, the sender confirms that the packet P6 is lost and immediately retransmits the lost packet P6 without reducing the size of cwnd. On the other hand, if the non-congestion RTO was triggered due to sudden delay of packet P6, it reaches the destination before reaching the packet P7. As a result, the TCP sender receives an Ack of the next expected packet which is greater than the value of Exp_Ack. Then the sender confirms that the RTO was spurious. In this way, TCP NRT can reduce unnecessary retransmissions and needless reduction of cwnd size and thereby improve the performance of TCP in MWNs.

### 4.3. Reaction to non-congestion RTOs (NRT-reaction)

This subsection explains how TCP NRT reacts after the TCP sender detects and differentiates the RTOs. As we explained in the above subsections, whenever the sender detects RTO, it checks whether the last Ack is marked

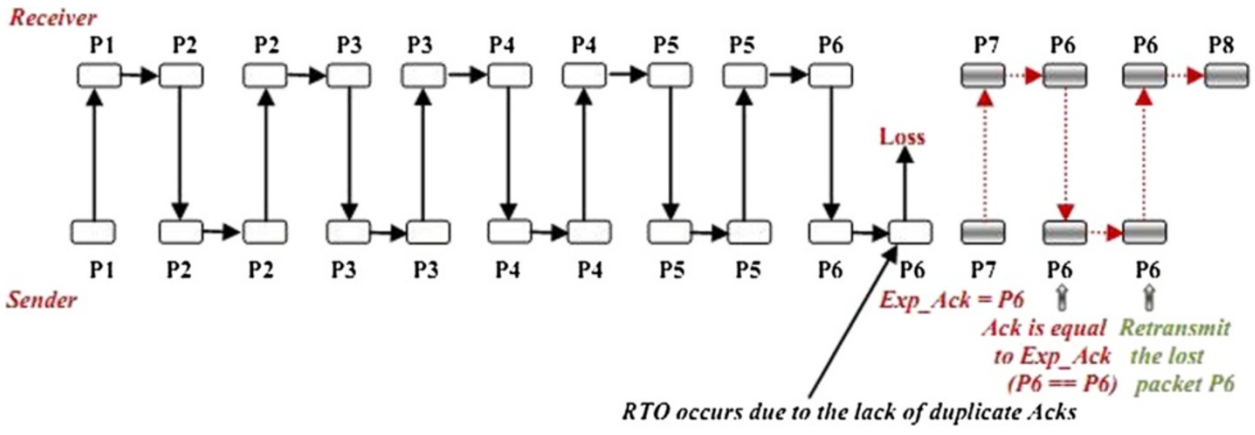


**Figure 7** Example of NRT-differentiation.

***Whenever the sender expires RTOs***

```
1:  If RTO () {
2:      If Congestion () {
3:        Retransmit the lost packet
4:        Reduce cwnd as TCP NewReno
5:        }else {
6:          If ( Non-Congestion packet loss) {
7:              Retransmit the lost packet
8:              Keep the current value of cwnd
9:          }else {
10:             The RTO is Spurious
11:             Continue sending packets until the sender
                reaches the value of ssthresh
12:         }
13:     }
14: }
```

**Algorithm 1** TCP NRT-reaction at the time of RTO expiration.

or not. Note that we mark the packets only if the AQL is greater than or equal to the maximum threshold. As shown in Algorithm 1, if the packet is marked, the sender immediately retransmits the lost packet and reduces the size of cwnd (lines 2 to 4) and follows the RTO loss recovery algorithm of TCP NewReno as it is the most widely deployed protocol. The reason is that a marked packet is an indication of network congestion. On the other hand, if it is not marked, the sender confirms that the RTO is caused by random packet loss due to transmission errors or spurious packet losses due to sudden delays. As a result, instead of retransmitting the packet, the sender sends a new packet and waits for the Ack. If the RTO is triggered due to random packet loss, the sender retransmits the lost packet while keeping the current size of cwnd (lines 6 to 8). If the RTO is triggered spuriously, the sender sends packets continuously without reducing the size of cwnd and ssthresh until the value of cwnd reaches the value of ssthresh. In this way, the sender of TCP NRT is able to reduce unnecessary retransmissions and needless reduction of the size of cwnd due to spurious RTOs and the RTOs due to random packet losses and can increase the performance of TCP in MWNs.

### 4.4. Operation of TCP NRT

In this subsection, we describe the rationale for proposing an efficient algorithm for the differentiation of RTOs due to random packet loss from spurious RTOs. We adopt the slow-start (SS), congestion avoidance (CA), fast retransmit (FRR), and recovery algorithms (FRE) algorithms of the original TCP NewReno [21] and changed the loss recovery algorithm in the case of RTOs. As shown in Figure 8, at the beginning of the TCP connection, the sender enters into the SS phase, in which the size of cwnd increases by one mss for every receiving Ack and the cwnd grows exponentially. When the value of cwnd reaches the value of ssthresh, the sender enters the CA state. During this phase, the sender increases its cwnd size linearly for every RTT. This linear growth of transmission rate helps the

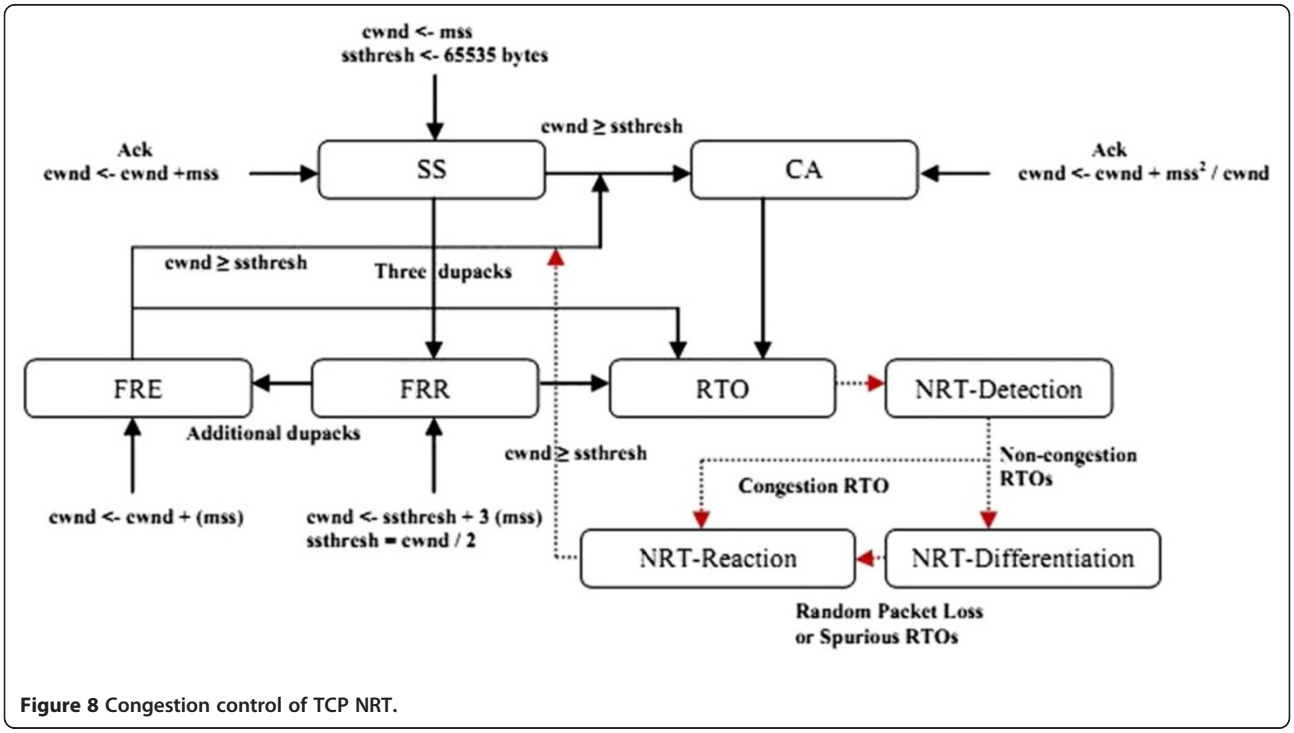


**Figure 8** Congestion control of TCP NRT.

sender to slowly probe the available network bandwidth. When the sender receives three duplicate Acks (dupacks), it proceeds as TCP NewReno because in our research we are focusing on improving the TCP performance using the loss recovery mechanism for the expiration of RTOs. As a result, we use the same FRR and FRE mechanisms of TCP NewReno.

As shown by the dotted lines in Figure 8, whenever the timer expires, the sender goes to NRT-detection to detect congestion and non-congestion RTOs. If the RTO is caused by congestion, the sender invokes the NRT-reaction algorithm to retransmit the lost packets and reduce the size of cwnd to one mss followed by the slow-start algorithm like conventional TCP. Otherwise, the sender goes to NRT-differentiation to check whether the RTO is caused by random packet loss or spurious RTO before retransmitting the packet and follows the component NRT-reaction. If the RTO is caused by non-congestion events, then the sender may not reduce the size of cwnd and can reduce unnecessary retransmissions using the component NRT-differentiation. When the value of cwnd reaches the value of ssthresh, the sender goes to CA state and continues sending packets until the sender receives three dupacks or the expiration of RTO. In this way, TCP NRT can improve the performance of TCP in MWNs by reducing unnecessary retransmissions due to spurious RTO as well as by avoiding unnecessarily reduction of cwnd due to the RTOs triggered by random packet losses or spurious RTOs.

## 5. Performance evaluation

In this section, we explain the simulation methodology and performance metrics we use to evaluate the effectiveness of the TCP NRT algorithm in MWNs. In order to make sure of the efficiency of our algorithm, the performance of TCP NRT is compared to that of related works such as Eifel, DSACK, F-RTO, EQRTO, and NewReno.

### 5.1. Simulation methodology

To evaluate the effectiveness of the TCP NRT algorithm, we implement our algorithm in the network simulator Qualnet version 5 [22] and test it in various conditions. As the main focus of our algorithm is to reduce unnecessary retransmissions and needless reduction of cwnd due to

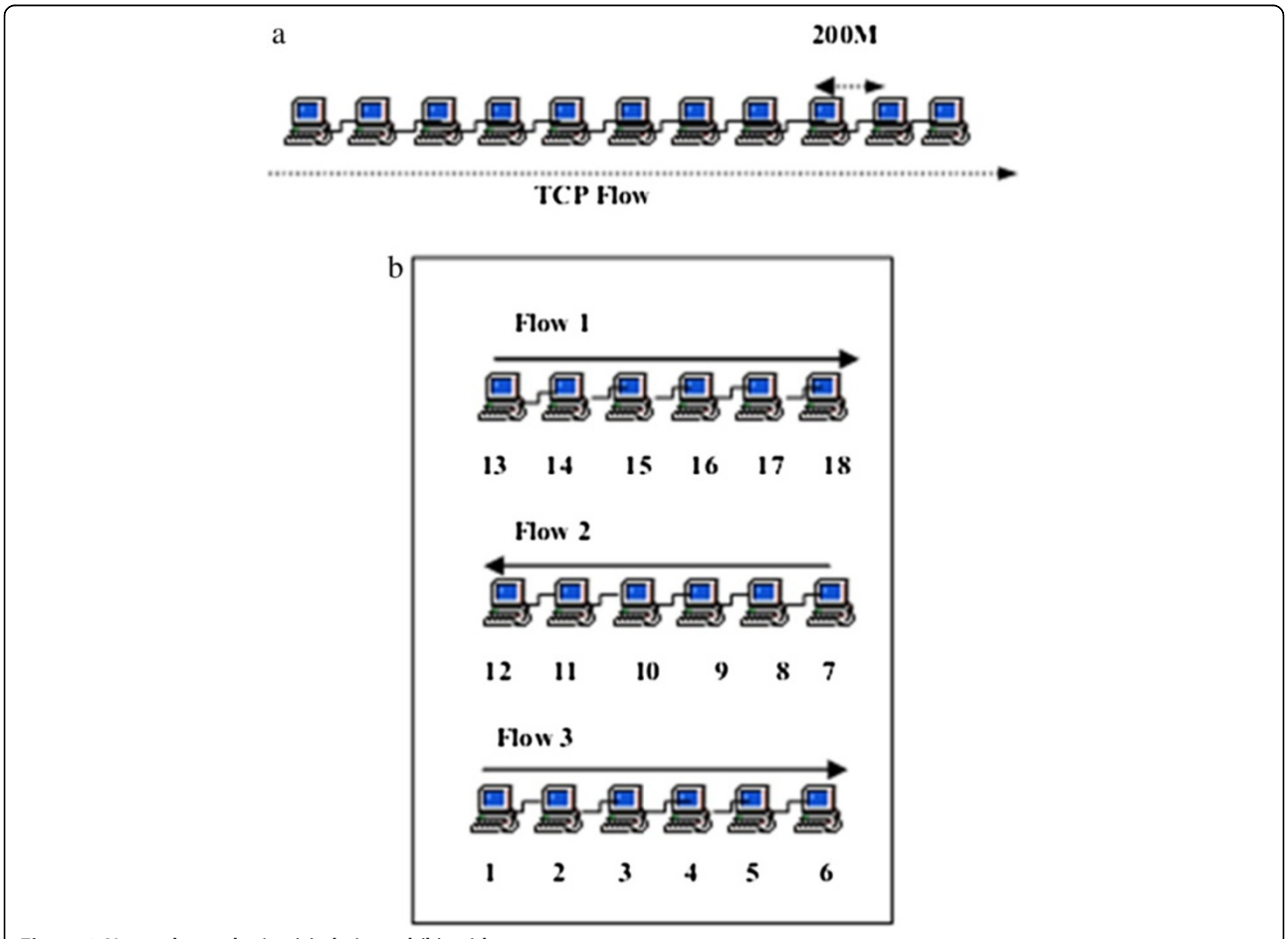


**Figure 9** Network topologies (a) chain and (b) grid.

spurious RTOs and RTOs caused by random packet losses, we adopt the methodology used in [2] for tracing RTOs. We use two types of multihop topologies, as shown in Figure 9. One is a chain topology consisting of ten hops or 11 nodes, and another one is the grid topology consisting of 18 nodes. All wireless nodes are considered as static, which are using IEEE 802.11a with a basic data rate of 9 Mbps in our experiments, unless stated otherwise. The distance between two neighboring nodes is given as 200 m so that the nodes can communicate with each other. The maximum size of cwnd is set to 32 packets. For routing, we use DSR protocol and FTP-generic used for application. We designed about 300 scenarios by setting different parameters in terms of the number of hops, packet loss rate, bandwidths, number of RTOs, number of unnecessary retransmissions, and number of unnecessary reduction of cwnd. Through experiments, we aimed to evaluate how much our algorithm improves the performance of TCP in MWNs.

For this, we grouped our experiments into four sets. The first set of experiments involves wireless links that drop packets randomly without any congestion in the network. The packet losses are distributed uniformly. The main purpose of this scenario is to test the performance when RTO occurs due to transmission errors. Since the primary motivation of TCP NRT is to improve TCP performance in MWNs when random packet loss causes unnecessary cwnd reduction and spurious RTOs cause unnecessary retransmissions and needless cwnd reduction, we start with a scenario that involves random packet losses. The second set of experiments involves wireless links that do not drop any packets but randomly inflicts sudden delays for some packets. The main goal of this experiment is to evaluate the performance when sudden delays cause spurious RTOs. To perform this experiment, we adopt the method described in [6].

The third set of experiments involves wireless links that cause RTOs due to random packet losses and sudden delays. The main aim of this experiment is to ensure the effectiveness of TCP NRT when the RTOs coexist with random packet loss and sudden delays. It is essential to check the performance improvement of TCP NRT by differentiating the two types of RTOs in the network. The fourth set of experiments involves wireless links that cause RTOs due to congestion as well as random packet loss and RTOs due to sudden delays. The main purpose of this experiment is to evaluate the performance of TCP NRT in a general environment. As mentioned in [2], in all scenarios, we traced packet losses at the Mac layer and the network layers and compared those lost packets with the packets triggered by RTO at the transport layer. If the packet triggered by RTO is not found at the Mac and network layers, we assume that the corresponding RTO is triggered due to sudden delay, and we treat those RTOs as spurious RTOs.

### 5.2. Performance metrics

To ensure the significant performance of TCP NRT, we evaluate our algorithm using four perspectives: throughput, accuracy, fairness, and friendliness.

#### *5.2.1. Throughput*

Throughput is one of the most important performance metrics in the TCP. We define throughput as the number

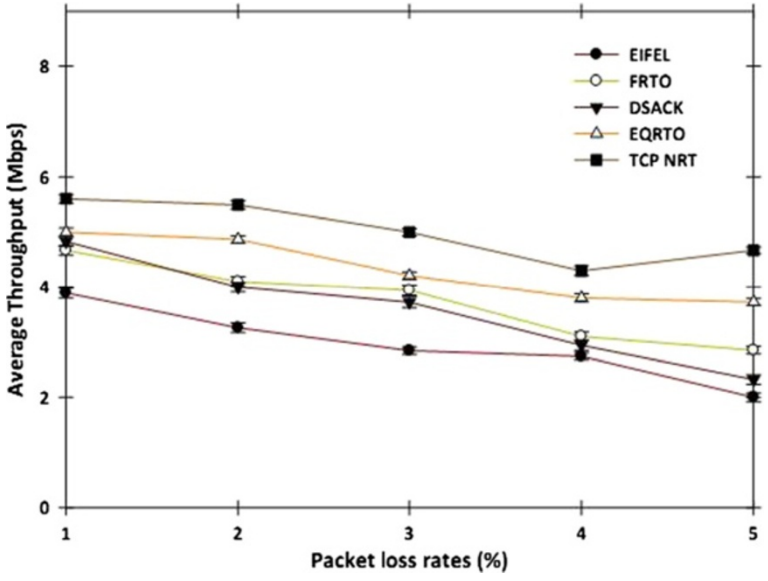


**Figure 10** Throughput performance in terms of packet loss rates.

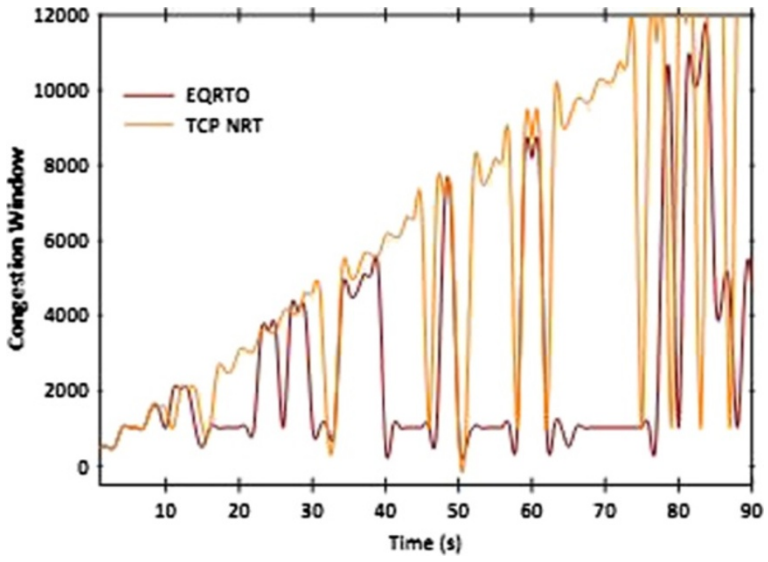


**Figure 11** Comparison of cwnd size in terms of 2% packet loss rate at ten hops.

of packets sent divided by the transmission time. We measured throughput in terms of the number of hops, number of flows, varying bandwidths, different packet loss rates, and varying delays using chain and grid topologies.

#### *5.2.2. Accuracy*

Another important metric is accuracy. It defines exactly how a scheme detects the different types of RTOs [7]. We measured accuracy in terms of RTOs due to random packet loss (RPL), congestion packet loss (CPL), spurious RTOs (S), and the combination of random packet losses and spurious RTOs (NC).

$\mathrm{RPL}_{\mathrm{Acc\text{-}RTO}}$ is defined as the accuracy in detecting RTOs caused by random packet loss. The formula used for calculating $\mathrm{RPL}_{\mathrm{Acc\text{-}RTO}}$ is shown below:

$$\mathrm{RPL}_{\mathrm{Acc-RTO}} = \frac{\mathrm{RPL}_{\mathrm{IdentRTO}}}{\mathrm{RPL}_{\mathrm{totalRTO}}} \times 100 \tag{1}$$

where $\mathrm{RPL}_{\mathrm{Ident_}}$RTO is the number of RTOs exactly identified as RTO due to random packet loss by TCP NRT compared to other existing algorithms, and $\mathrm{RPL}_{\mathrm{Total_}}$RTO is the total number of RTOs triggered due to random packet losses.

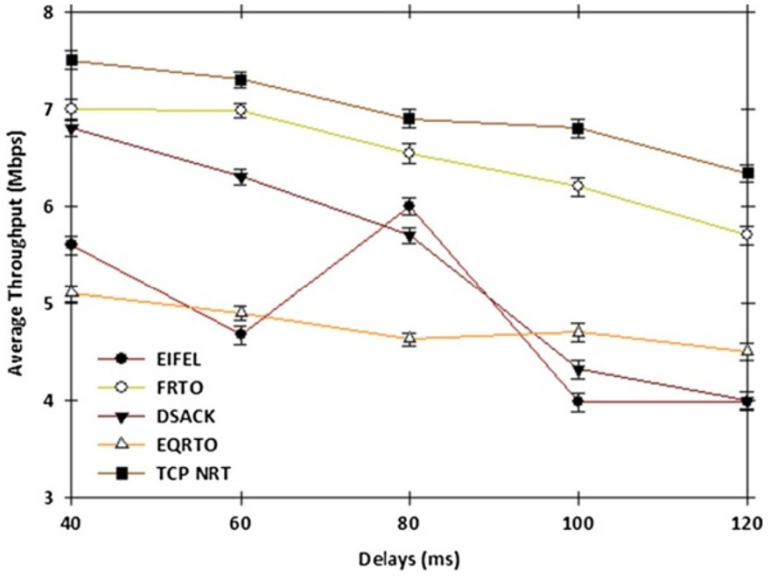


**Figure 12** Throughput performance in terms of varying delays.

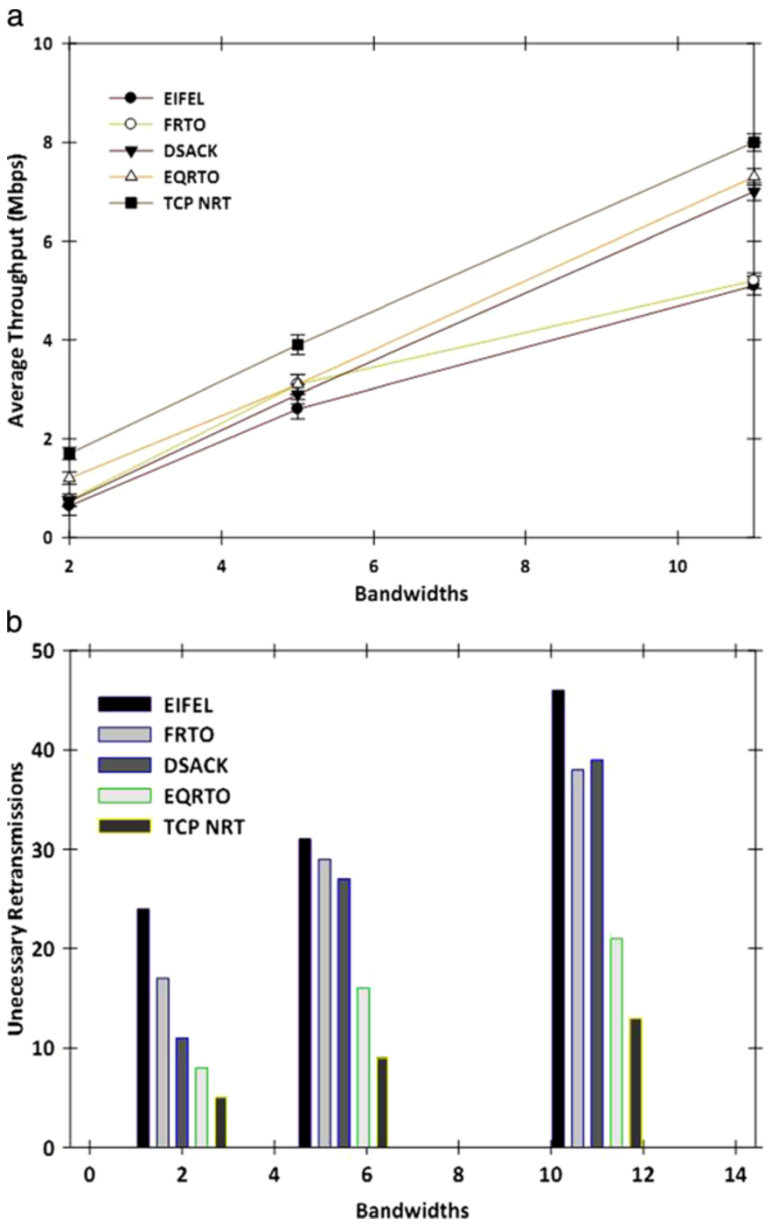


**Figure 13** Comparison of (a) average throughput and (b) unnecessary retransmissions in terms of bandwidths.

$CPL_{Acc\text{-}RTO}$ is defined as the accuracy in detecting RTOs caused by congestion loss. The formula used for calculating $CPL_{Acc\_RTO}$ is shown below:

$$CPL_{Acc-RTO} = \frac{CPL_{IdentRTO}}{CPL_{TotalRTO}} \times 100 \quad (2)$$

where $CPL_{Ident}\_RTO$ is the number of RTOs exactly identified as RTO due to congestion packet loss by TCP NRT compared to other existing algorithms, and $CPL_{Total}\_RTO$ is the total number of RTOs triggered due to congestion packet losses.

$S_{Acc\_RTO}$ is defined as the accuracy in detecting spurious RTOs which are triggered due to sudden delay of packets on the network path. It is calculated by using the following formula:

$$S_{Acc-RTO} = \frac{S_{IdentRTO}}{S_{TotalRTO}} \times 100 \quad (3)$$

where $S_{Ident}\_RTO$ is the number of RTOs exactly identified as spurious RTOs, and $S_{Total}\_RTO$ is the total number of RTOs triggered due to spurious packet losses.

$NC_{Acc\text{-}RTO}$ is defined as the accuracy in detecting non-congestion RTOs when they coexisted with each other.

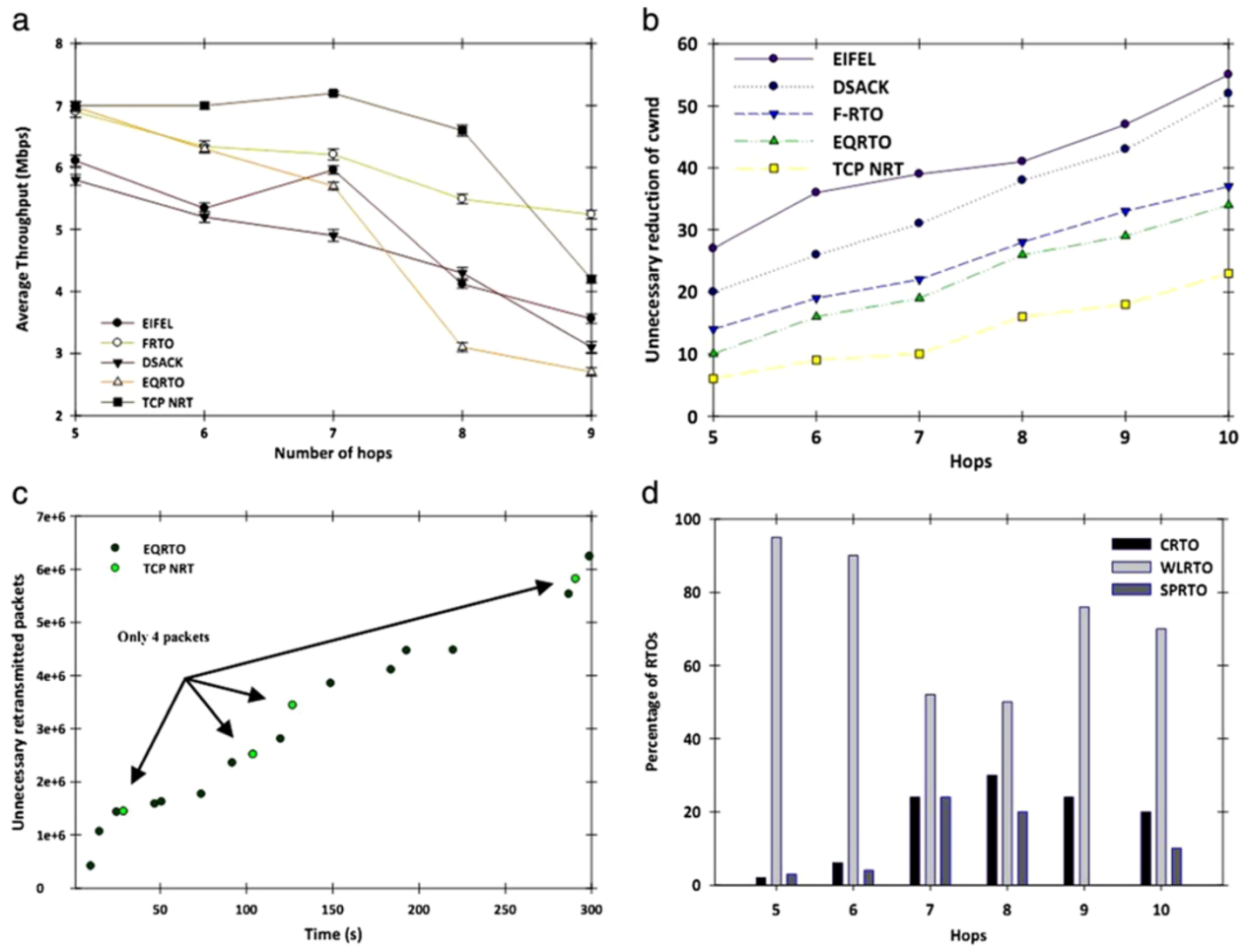


**Figure 14** Comparison of (a) average throughput, (b) unnecessary cwnd reduction, (c) unnecessary retransmissions, and (d) ratio of RTOs.

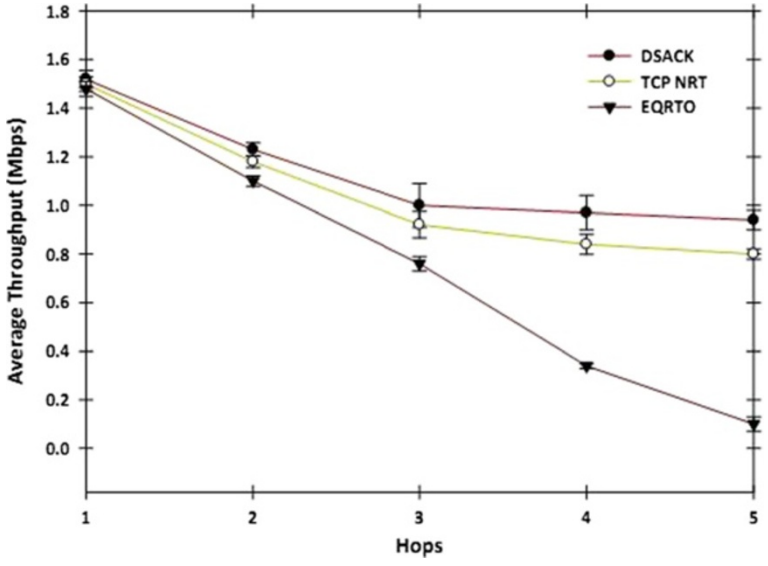


**Figure 15** Comparison of throughput in the presence of spurious fast recovery.

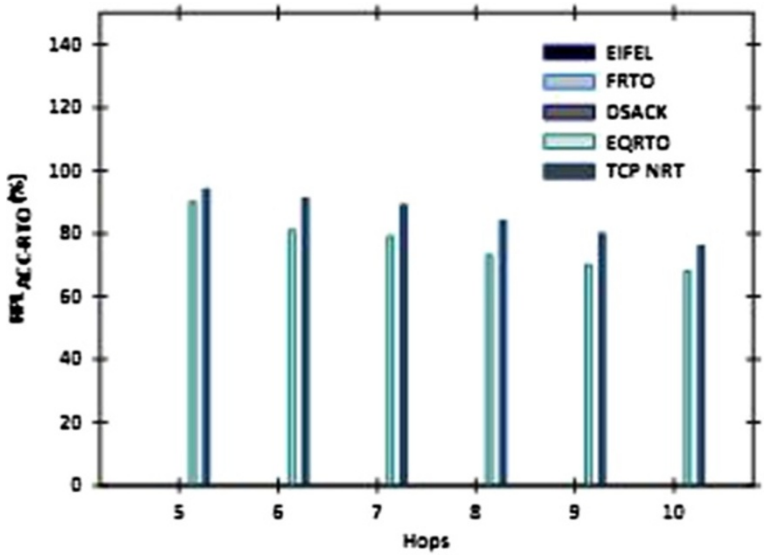


**Figure 16** Accuracy of random packet loss RTOs.

The formula used for calculating $NC_{Acc\text{-}RTO}$ is as follows:

$$NC_{Acc-RTO} = \frac{NC_{Ident}}{NC_{Total}} \times 100 \quad (4)$$

where $NC_{Ident}$ is the number of RTOs exactly identified as non-congestion RTOs, and $NC_{Total}$ is the total number of non-congestion RTOs. Finally, we compared the fairness and friendliness of TCP NRT. We explain these in detail in the next subsections.

#### *5.2.3 Fairness*

Another important performance metric of TCP is its fairness. Multiple connections of the same TCP scheme must interoperate nicely and converge to their fair shares. We use the fairness index function (Equation 5), proposed in [23], to justify the fairness of TCP schemes. The fairness index function is expressed as,

$$F(x) = \left(\sum x_i\right)^2 / n\left(\sum x_i\right)^2 \quad (5)$$

where $x_i$ is the throughput of the $i$th connection, and $n$ is the number of connections. $F(x)$ ranges from $1/n$ to 1.0. A

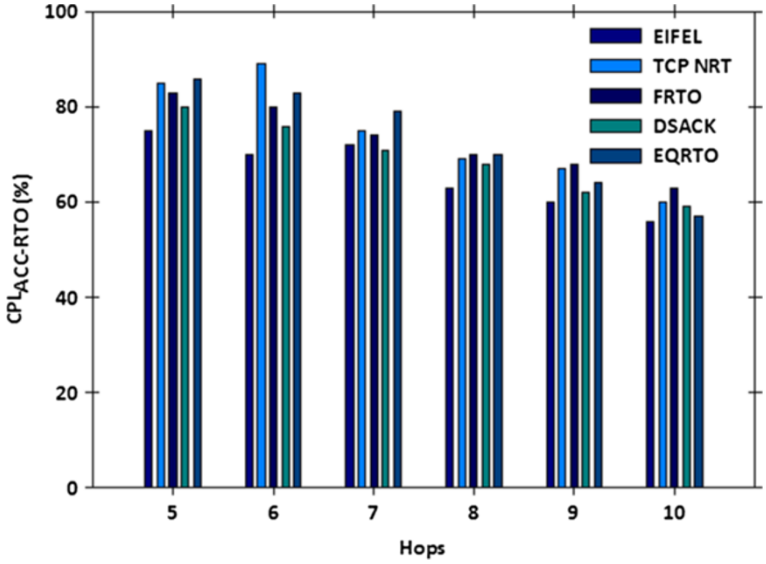


**Figure 17** Accuracy of congestion packet loss RTOs.

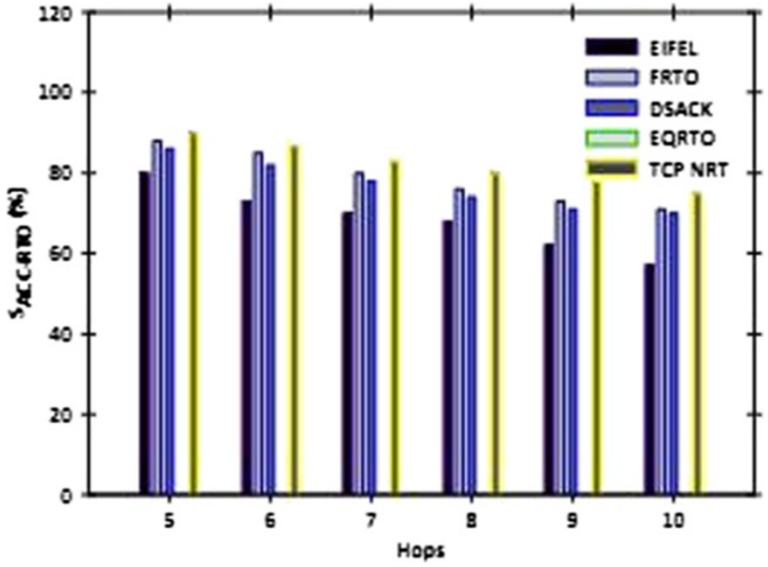


**Figure 18** Accuracy of spurious RTOs.

perfectly fair bandwidth allocation would result in a fairness index of 1.0. On the contrary, if all bandwidths are consumed by one connection, Equation 5 would yield $1/n$.

#### *5.2.4. Friendliness*

Gradual deployment requires acceptable friendliness behavior when TCP NRT is sharing the medium with other flows [8]. This means that TCP NRT ideally should not suppress the regular flows but allow them to achieve at least the same throughput they would obtain without any improved flow in parallel. We show through xour experiments how friendly our mechanism can be when TCP NRT competes with regular flows of TCP NewReno in a MWNs.

### 5.3. Experimental results

In this subsection, we present the results of our experiments used to verify the effectiveness of our algorithm in MWNs using three different scenarios, namely chain, grid, and dumbbell topologies.

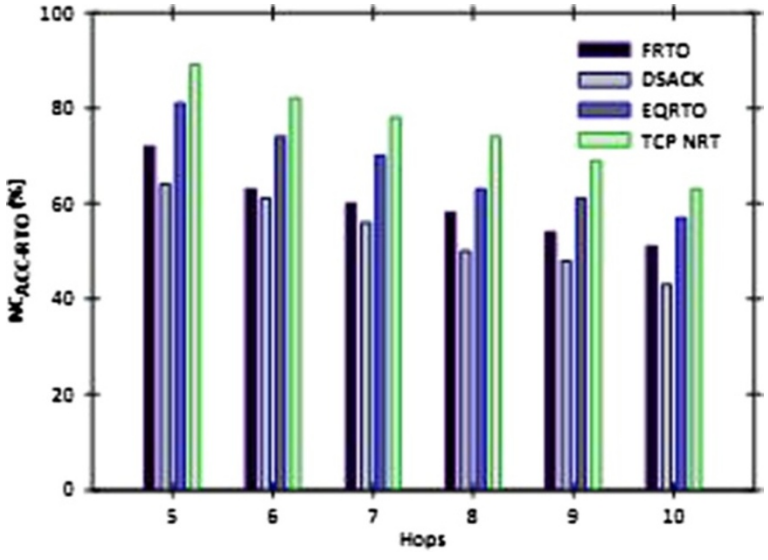


**Figure 19** Accuracy of random packet loss RTO and spurious RTOs.


**Figure 20** Simulation network for verifying fairness.

#### *5.3.1. Comparison of throughput in chain topology*

Using ten-hop chain topoloy, we compared the average throughput of each algorithms in terms of packet loss rates, bandwidths, delays, and varying number of hops. Simulations were run ten times with 95% confidence interval. Figure 10 shows the measured average throughput at ten hops by sending the traffic flow from node 1 to node 11 in terms of varying packet loss rates ranging from 1% to 5% using the first set of experiments, that is, the RTOs are triggered only by random packet losses. As we expected, when the packet loss rate increases, the throughput of each TCP version reduces drastically.

However, compared to other TCP versions, TCP NRT and EQRTO achieved better performance. When the percentage of packet loss rate becomes 5%, TCP NRT gains 18% throughput over EQRTO. This is because TCP NRT can detect and differentiate the non-congestion RTOs very efficiently. Figure 11 shows the comparison of the cwnd size of TCP NRT with EQRTO in terms of 2% packet loss rate at ten hops. The cwnd size of TCP NRT is increasing faster than the cwnd size of EQRTO. It reveals that TCP NRT can achieve higher throughput compared to EQRTO. Figure 12 presents the result of throughput comparison in terms of varying delays ranging from 40 to 120 ms using the second set of experiments. In this experiment, the RTO is triggered only due to spurious RTOs. The performance is decreasing according to the increase in delays. Even when the throughput decreases, TCP NRT outperforms the other TCPs. In this scenario, F-RTO achieves a relatively higher throughput compared to Eifel, DSACK, and EQRTO. The performance of DSACK and Eifel is the same for the 120-ms delays, and TCP NRT improves with more than 15% throughput at a delay of 100 ms compared to F-RTO. Figure 13 depicts the result of the comparison of throughput in terms of different bandwidths ranging from 2, 5, to 11 Mbps using the third set of experiments. In this experiment, the RTO is triggered only due to spuriously or random packet losses. As the bandwidth increases, the throughput also increases. However, the throughput of TCP NRT is better than those of other TCPs. The reason is that by reducing unnecessary retransmissions as shown in Figure 13b and unnecessary reduction of cwnd, the sender of TCP NRT can send more packets and thereby make utilization of the available bandwidth in the network.

**Table 1 Comparison of fairness**

| Loss rate (%) | TCP NewReno | TCP NRT |
|---|---|---|
| 0.1 | 0.9999 | 0.9999 |
| 0.5 | 0.9999 | 0.9999 |
| 1.0 | 0.9998 | 0.9998 |
| 5.0 | 0.9991 | 0.9992 |
| 10 | 0.9984 | 0.9986 |

Figure 14a demonstrates the comparison of average throughput in terms of the number of hops ranging from five to nine using the fourth set of experiments, that is, the RTOs are due to congestion, random packet loss, and spurious packet losses. In this experiment, we set node 1 as the source node and destination changes from node 6 to node 10 according to the number of hops. Although the throughput decreases when the number of hop increases, TCP NRT achieves better performance due to its capability to detect and differentiate the type of RTOs. Compared to other TCPs, TCP NRT has more than 80% throughput gain due to its capability to avoid unnecessary retransmissions and needless reduction of cwnd size, as shown in Figure 14b,c. In Figure 14b, we can clearly see that TCP NRT reduces the size of cwnd less than the other versions. In case of retransmissions, Figure 14c depicts packet retransmissions during the simulation time span of 300s. EQRTO frequently retransmits a lot more packets unnecessarily than TCP NRT, and this explains why TCP NRT improves the throughput of TCP in MWNs. Figure 14d shows the ratios of RTOs caused in our experiments. In this figure, we can see that the RTO due to random packet loss (WLRTO) is greater than the RTOs due to congestion (CRTO) and spurious RTOs (SRTO).

In addition to these experiments, we evaluate the throughput performance of TCP NRT in the presence of spurious fast recovery as it is not a rare event in wireless networks [24]. If the retransmission is triggered by fast recovery and the particular packet loss is not found at the Mac and network layers, we assume that the

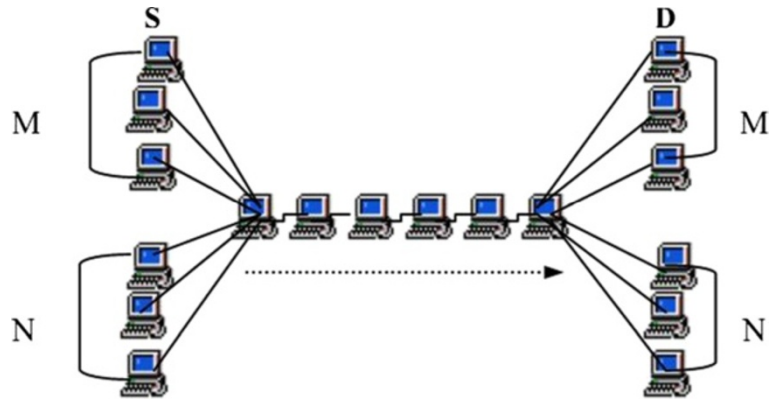


**Figure 21 Simulation network for verifying friendliness.**

corresponding fast recovery is triggered due to sudden delay, and we treat that fast recovery as spurious fast recovery. In Figure 15, we can see the average throughput of TCP NRT, DSACK, and EQRTO. In the presence of spurious fast recovery, DSACK outperforms TCP NRT and EQRTO. This is because TCP NRT and EQRTO have no mechanism to false fast retransmissions. However, compared to EQRTO, when the number of hop increases, the average throughput of TCP NRT is better due to its efficiency in handling the fast retransmission by using the features of TCP NewReno fast recovery mechanism.

#### *5.3.2. Comparison of accuracy in grid topology*

In this subsection, we present the results of accuracy in terms of the number of hops using grid topology. Figure 16 shows the accuracy of random packet loss RTOs. Only TCP NRT and EQRTO achieve the highest accuracy in detecting the RTOs due to random packet losses. When the number of hop increases, the accuracy of TCP NRT increases compared to EQRTO. At ten hops, TCP NRT achieves more than 80% accuracy in detecting random packet loss RTOs compared to EQRTO. Figure 17 shows the accuracy of congestion packet loss RTOs in terms of the number of hops. In this experiment, all the variants have similar performance. However, TCP NRT achieves more than 85% accuracy at six hops compared to other variants. Figure 18 shows the accuracy of spurious RTOs in terms of the number of hops. The results of this experiment show that, except for EQRTO, all of the compared TCP versions achieve good accuracy. This is because EQRTO misclassifies the RTOs as congestion packet losses. In the case of TCP NRT, it shows a slightly high accuracy

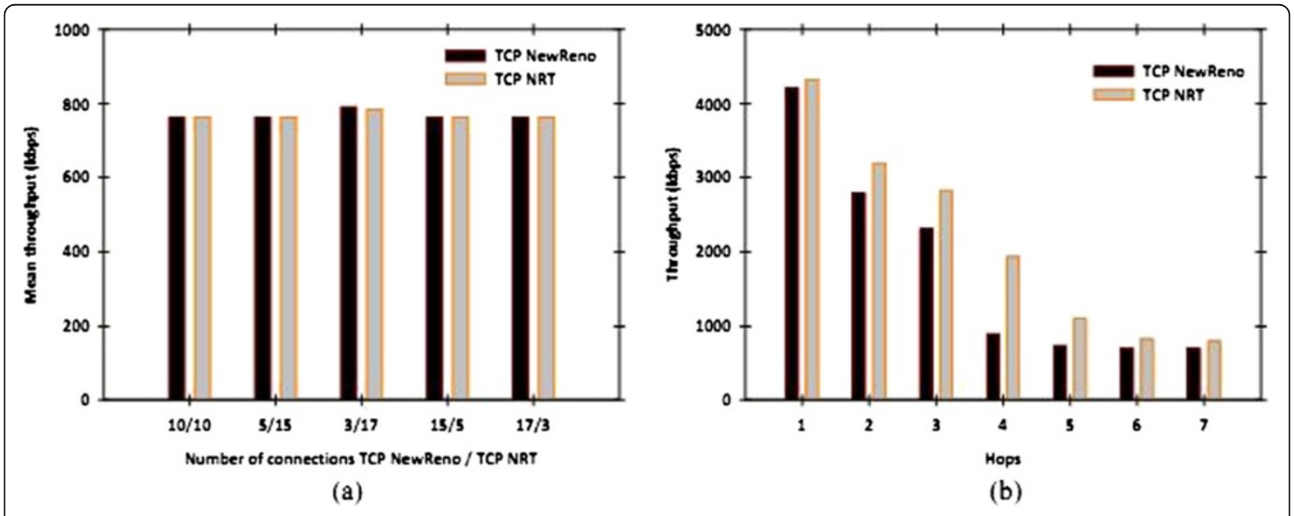


**Figure 22 Comparison of friendliness using multiple TCP connections.** (**a**) The result of fairness in terms of the number of connections of TCP NewReno and TCP NRT. (**b**) The fairness of TCP NRT in terms of the number of hops.

**Table 2 Comparison of existing solutions with TCP NRT**

| Solutions | Sender-side modifications | Receiver-side modifications | Type A | Type B | Type C |
|---|---|---|---|---|---|
| Eifel | Yes | Yes | No | Yes | No |
| DSACK | Yes | Yes | No | Yes | No |
| F-RTO | Yes | No | No | Yes | No |
| STODER | Yes | Yes | No | Yes | No |
| EQRTO | Yes | No | Yes | Yes | No |
| TCP NRT | Yes | No | Yes | Yes | Yes |

compared to other TCPs. F-RTO algorithm has accuracy close to that of TCP NRT in the case of spurious RTOs. Figure 19 shows the accuracy in detecting non-congestion RTOs in the presence of congestion RTOs. For this experiment, we caused 1% spurious RTOs and 3% random packet loss RTOs. From the figure, it is evident that EQRTO and TCP NRT have the highest accuracy when the network coexists with spurious and random packet loss RTOs. Compared to the other TCPs, EQRTO is effective in detecting the RTOs caused by random packet loss like TCP NRT, while F-RTO is effective in detecting spurious RTOs in addition to TCP NRT. At ten hops, TCP NRT has more than 70% accuracy compared to F-RTO, EQRTO, DSACK, and Eifel. If accuracy is better, the performance of TCP is also better. Due to the highest accuracy of TCP NRT, it can achieve significant improvement in the performance of TCP in MWNs.

#### *5.3.3. Comparison of fairness using dumbbell topology*

In order to assess the ability of our mechanism to allow a fair distribution of bandwidth, we simulate here another scenario for the dumbbell topology in Figure 20 which consists of two source nodes (S and S1) and two destinations (D and D1). We measure the fairness of TCP NRT compared to TCP NewReno as it is the most widely deployed protocol. We use the Jain fairness index proposed in [23] to measure the fairness of TCP NRT and TCP NewReno. In our experiment, we use ten TCP connections with packet loss rates varying from 0.1% to 10% and 1% delay with 6Mb bottleneck link and 20-ms propagation delay. We run the simulation for TCP NRT and TCP NewReno and compare their fairness index. The results are summarized in Table 1. From Table 1, it is clear that TCP NRT achieves fairly satisfactory fairness index.

#### *5.3.4. Comparison of friendliness using dumbbell and chain topologies*

A friendly TCP scheme should be able to coexist with other TCP variants and not cause them starvation [8]. To verify the friendliness of TCP NRT, we use the dumbbell topoloy, where TCP NRT coexists with TCP NewReno. The simulation network is shown in Figure 21. There are 20 pairs of connections, of which 'M' are TCP NRT connections and 'N' are TCP NewReno connections. Using different parameters, in our experiment, we caused congestion and non-congestion RTOs and measured the mean throughput of TCP NRT and TCP NewReno. We calculated the mean throughput by summing up the throughput of the same TCP scheme and dividing by the number of connections, respectively. Figure 22a depicts the mean throughput in terms of the number of connections of TCP NewReno and TCP NRT. It is observed that the bandwidth allocation of each TCP connection is close to its fair share at the bottleneck links. In addition to that, we measured the friendliness of TCP NRT using the chain topology. Figure 22b presents the fairness of TCP NRT in terms of the number of hops. For this experiment, we use three TCP connections by keeping node 1 as the source node and the destination changes according to the number of hops. Compared to Figure 22a, TCP NewReno is not friendly with TCP NRT because TCP NRT is more dominating due to its advantage in controlling the size of cwnd if both TCPs are communicating separately. TCP NRT can send more data packets compared to TCP NewReno especially when the number of hop increases. On the other hand, when TCP NRT coexists with TCP NewReno, TCP NRT shows a friendly behavior to TCP NewReno, as shown in Figure 22a.

#### *5.3.5. Comparison of existing solutions with TCP NRT*

In this subsection, we compare existing solutions with TCP NRT using type A, type B, and type C classifications. Type A classification is the solutions which are able to detect congestion RTOs and the RTOs due to random packet loss, whereas type B classifies the solutions proposed for detecting congestion RTOs and spurious RTOs. Finally, type C is able to classify solutions which are capable of differentiating random loss RTOs from spurious RTOs. Table 2 shows the classifications of existing solutions with TCP NRT in terms of sender and receiver-side modifications, detection of congestion loss and random loss RTOs, detection of congestion loss and spurious RTOs, and finally, differentiation of random loss RTOs from spurious RTOs. From the table, we can see that only TCP NRT can guide the sender congestion window mechanism by detecting and differentiating the

congestion and non-congestion RTOs and thereby increase the performance of TCP in MWNs.

## 6. Conclusions

In this paper, we have proposed a new TCP algorithm, called TCP NRT, to improve the performance of TCP in MWNS by differentiating the types of non-congestion RTOs. TCP NRT detects the non-congestion RTOs by means of the modified packet marking scheme of ECN mechanism. Our approach is easy to implement and requires only simple changes in the sender side of the TCP without altering the protocol itself. Our simulation results show that TCP NRT is a viable solution to the TCP performance degradation in MWNs. Results show that under 5% packet loss rate, a typical characteristic of MWNs, TCP NRT outperforms other TCP variants by 20% to 60% improvement in throughput when the RTO is caused in the absence of network congestion and presence of random loss due to transmission errors. On the other hand, the results show that under 140 ms delay, TCP NRT achieves high throughput compared to other existing solutions in the presence of spurious RTOs. Furthermore, when the network coexists with both congestion and non-congestion RTOs, the throughput of TCP NRT increases more than 25% compared to the other solutions. There are three main reasons for these significant improvements in the throughput, namely NRT-detection, NRT-differentiation, and NRT-reaction. Our experiments also show that TCP NRT maintains a fair and friendly behavior with other TCP flows. Our future research in this direction would be to improve TCP NRT so that the sender could further control its congestion control mechanism when it receives duplicate acknowledgments or RTOs in case of loss of acknowledgments in addition to the sudden delay or random loss of packets.

#### Competing interests

The authors declare that they have no competing interests.

#### Acknowledgements

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2013020769).

Received: 30 April 2012 Accepted: 13 May 2013
Published: 21 June 2013

#### References

1. Z Fu, P Zerfos, H Luo, S Lu, L Zhang, M Gerla, The impact of multi-hop wireless channel on TCP throughput and loss, in *22th Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM 2003)*, vol. 3 (IEEE, New York, 2003), pp. 1744–1753
2. P Mi-Young, C Sang-Hwa, Distinguishing the cause of TCP retransmission timeouts in multi-hop wireless networks, in *12th IEEE International Conference on High Performance Computing and Communications (HPCC-2010)* (Melbourne, 1–3 Sept 2010), pp. 329–336
3. S Xu, T Saadawi, Performance evaluation of TCP algorithms in multi-hop wireless packet networks. Wireless Communications and Mobile Computing **2**, 85 (2001)
4. NJ Kothari, BM Gambhava, KS Dasgupta, RTT utilization by detecting avoidable timeouts, in *14th IEEE International Conference on Networks, Singapore*, 13–15 Sept 2006
5. D Ciullo, M Mellia, M Meo, Two schemes to reduce latency in short lives TCP flows. IEEE Communications Letters **13**, 10 (2009)
6. P Sarolahti, M Kojo, Forward RTO-recovery (FRTO): an enhanced recovery algorithm for TCP retransmissions timeouts. ACM SIGCOMM Computer Communications Review **33**, 2 (2003)
7. P Mi-Young, C Sang-Hwa, Detecting TCP retransmission timeouts non-related to congestion in multi-hop wireless networks. IEICE Transactions on Information and Systems **E93-D**(12), 3331–3343 (2010)
8. R de Oliveira, T Braun, A smart TCP acknowledgment approach for multihop wireless networks. Mobile Computing, IEEE Transactions **6**(2), 192–205 (2007)
9. R Ludwig, RH Katz, The Eifel algorithm: making TCP robust against spurious retransmissions. ACM SIGCOMM Computer Communication Review **30**, 1 (2000)
10. E Blanton, M Allman, *Using TCP duplicate selective acknowledgment (DSACK) and stream transmission control protocol, RFC Information-3707*, 2004. http://tools.ietf.org/html/rfc3707. Accessed 10 March 2012
11. K Tan, Q Zhiang, Z Wenwu, STODER: a robust and efficient algorithm for handling spurious retransmission timeouts in TCP, in *Proc. IEEE Global Telecommunications Conference, St. Louis, 20 Nov–2 Dec 2005*, pp. 3692–3696
12. M Allman, H Balakrishnan, S Floyd, Enhancing TCP's loss recovery using limited transmit, RFC 3042. (2001). [http://tools.ietf.org/html/rfc3042]. Accessed 10 March 2012
13. CP Fu, SC Liew, TCP Veno, TCP enhancement for transmission over wireless access networks. IEEE Journal on Selected Areas in Communications **21**(2), 216–228 (2003)
14. V Jacobson, R Braden, D Borman, TCP extensions for high performance, RFC 1323. (1992). http://www.ietf.org/rfc/rfc1323.txt. Accessed 23 March 2012
15. S Floyd, TCP and explicit congestion notification. ACM Computer Communication Review **24**, 8 (1994)
16. D Lin, R Morris, Dynamics of random early detection. ACM SIGCOMM Computer Communication Review **27**(4), 127–137 (1997)
17. L Long, J Aikat, K Jeffay, FD Smith, The effects of active queue management and explicit congestion notification on web performance. Networking, IEEE/ACM Transactions **15**(6), 1217–1230 (2007). doi:10.1109/TNET.2007.910583
18. LB Lim, L Guan, A Grigg, IW Phillips, XG Wang, IU Awan, LB Lim, L Guan, A Grigg, IW Phillips, XG Wang, IU Awan, RED and WRED performance analysis based on superposition of N MMBP arrival process, in *2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), Perth, 20–23*, 2010, pp. 66–73
19. K Xu, Y Tian, N Ansari, TCP-Jersey for wireless IP communications. IEEE Journal of Selected Areas of Communication **22**, 747–756 (2004)
20. W Yi, L Lu, A study of internet packet reordering, in *Proceedings of Information Networking Technologies for Broadband and Mobile Networks International Conference* (, Busan, 18–20 Feb 2004)
21. S Floyd, T Henderson, *The NewReno modification to TCP's fast recovery algorithm, RFC 2582* (IETF, 1999). http://www.hjp.at/doc/rfc/rfc2582.txt. Accessed 18 March 2012
22. Scalable Network Technologies, (2013). http://www.scalable-networks.com/index.php. Accessed 12 February 2012
23. R Jain, D Chiu, W Hawe, *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems, Research Report TR-301(Digital Equipment* (Hudson, 1984)
24. P Mi-Young, C Sang-Hwa, A simulation-based study on spurious timeouts and fast retransmits of TCP in wireless networks, in *2010 Third International Joint Conference on Computational Science and Optimization (CSO)* (Huangshan, 28–31 May 2010), pp. 273–277

doi:10.1186/1687-1499-2013-172
**Cite this article as:** Sreekumari and Lee: **TCP NRT: a new TCP algorithm for differentiating non-congestion retransmission timeouts over multihop wireless networks.** *EURASIP Journal on Wireless Communications and Networking* 2013 **2013**:172.