EURASIP Journal on
Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

TCP's dynamic adjustment of transmission rate to packet losses in wireless networks

Mi-Young Park[1], Sang-Hwa Chung[2*] and Chang-Woo Ahn[2]

Abstract

Based on the assumption of transmission control protocol (TCP) that packets are lost due to congestion, TCP's congestion control algorithms such as fast retransmit/recovery (FRR) and retransmission timeouts (RTO) unconditionally reduce the transmission rate for every packet loss. When TCP operates in wireless networks, however, FRRs/RTOs are often triggered regardless of congestion due to sudden delay and wireless transmission errors. The congestion irrelative FRRs/RTOs incur TCP's misbehavior such as blindly halving the transmission rate, unnecessarily retransmitting the outstanding packets which may be in the bottleneck queue. Although many previous studies have been proposed to detect the congestion irrelative FRRs/RTOs, they paid little attention on effectively adjusting the transmission rate for the detected congestion irrelative FRRs/RTOs.

In this article, we propose an enhanced TCP to dynamically adjust its transmission rate according to network conditions. Our scheme adjusts the transmission rate in proportion to the available bandwidth in order to quickly utilize the available bandwidth, and also re-adjusts it in inverse proportion to the loss rate in order to avoid burst losses and long go-back-*N* retransmissions. By doing so, our scheme has significant effects to avoid the performance degradation caused by the congestion irrelative FRRs/RTOs. Throughout the extensive experiments, we evaluate our scheme and compare it with previous works in terms of goodput, fairness, and friendless under various network topologies. The results show that our scheme significantly outperforms previous studies while it maintains the fair and friendly behavior to other TCP connections.

Introduction

The transmission control protocol (TCP) [1-3] is the most popular protocol in the Internet, and its mission is to provide reliable data transfer between a TCP sender and a TCP receiver. For this, TCP has two mechanisms to detect packet losses for retransmissions: fast retransmit/recovery (FRR) and retransmission timeout (RTO) [2,4]. Assuming that packets are lost due to congestion, TCP reduces its transmission rate whenever an FRR or RTO is triggered in order to avoid further congestion losses.

That assumption works very well in wired networks where most FRRs or RTOs are triggered by the packets lost due to congestion, but it is not appropriate in wireless networks where most FRRs/RTOs are triggered by other reasons such as sudden delay [5-7], wireless errors, and mobility. The congestion irrelative FRRs/RTOs incur TCP's misbehaviors such as (1) blindly halving the transmission rate even when the available bandwidth is sufficient, (2) unnecessarily retransmitting the outstanding packets which may be in the bottleneck queue, and (3) needlessly increasing the back-off value exponentially. As a result, TCP underutilizes available bandwidth and its performance degrades severely in wireless networks.

To date, many previous studies have been proposed to solve such TCP's problem in wireless networks. Some loss differentiation schemes such as Westwood [8], JTCP [9], and RELDS [10] have been proposed to differentiate wireless losses from congestion losses. Other schemes such as Eifel [11-13], F-RTO [14,15], and STODER [16] have been suggested to remove the unnecessary retransmissions by detecting spurious FRRs/RTOs. The common problem is that they paid little attention on adjusting appropriately the transmission rate when the FRRs/RTOs are triggered regardless of congestion even though controlling the transmission rate is very critical in improving TCP's performance.

*Correspondence: shchung@pusan.ac.kr
[2] Department of Computer Engineering, Pusan National University, Busan, South Korea
Full list of author information is available at the end of the article

In case of TCP Westwood and Prairie, they estimate the available bandwidth, and adjust TCP's transmission rate according to the estimated bandwidth. Although they have more significant impact on improving TCP's performance, these schemes could have a high possibility to cause more frequent burst losses and long go-back-*N* retransmissions as the rate of wireless transmission errors increases. This is because these schemes do not consider the packet loss rate when they adjust the transmission rate. In this article, we propose an enhanced TCP to dynamically adjust its transmission rate according to network conditions such as the available bandwidth and the loss rate. When an FRR/RTO is triggered, our scheme adjusts the transmission rate in proportion to the available bandwidth in order to quickly utilize the available bandwidth, and also readjusts it in inverse proportion to the loss rate in order to avoid burst losses and long go-back-*N* retransmissions. In addition, when successive RTOs are triggered, our scheme initializes the back-off value if the network is not congested in order to avoid a long idle time period of an RTO. By doing so, our scheme has significant effects to avoid the performance degradation caused by the congestion irrelative FRRs/RTOs in wireless networks.

To evaluate our scheme, we design about 100 different simulation scenarios by setting different values of network parameters, and conduct simulation-based experiments using a network simulator, QualNet 4.5 [17]. Throughout the extensive simulations, we (1) observe how often congestion irrelative FRRs/RTOs are triggered, (2) measure the performance enhancement of our scheme and compare it with previous studies such as F-RTO, JTCP, and Prairie [18], (3) evaluate our scheme's fairness and friendliness. The results show that our scheme significantly outperforms previous studies while it maintains the fair and friendly behavior to other TCP connections.

In the following section, we describe TCP's misbehaviors caused by the congestion irrelative FRRs/RTOs. After that, we introduce previous studies to handle such TCP's problem, and explain our motivation. In "Dynamic responding algorithm" section, we propose our scheme to dynamically adjust the transmission rate according to the available bandwidth and the loss rate. Lastly, we evaluate and compare our scheme with previous studies in "Experimentation and analysis", and conclude this article in the last section.

TCP's misbehavior caused by wireless losses

TCP [1-3] uses packet loss as an indicator of congestion. Thus, whenever packets are lost in the networks, TCP's congestion control algorithms such as FRR and RTO are triggered to reduce transmission rate by updating the two variables: the congestion window (*cwnd*) and the slow start threshold (*ssthresh*) [2,4].

The two variables are critical to TCP's performance since the transmission rate depends on the two variables. *cwnd* is the size of the packets which can be sent without receiving any acknowledgements from a TCP receiver, and as *cwnd* is larger TCP's transmission rate is higher. *cwnd* is initialized by one segment when a connection starts or when an RTO is triggered. Then, a TCP sender increases *cwnd* exponentially whenever a new ACK arrives until *cwnd* reaches *ssthresh*. After reaching *ssthresh*, *cwnd* increases linearly. *ssthresh* is a critical border for a TCP sender to decide if the transmission rate increases exponentially or linearly. The value of *ssthresh* is initialized by an arbitrarily high value, and it is updated to the halved value of *cwnd* whenever an FRR/RTO is triggered.

In wireless networks, *cwnd* and *ssthresh* are often unnecessarily reduced whenever an FRR/RTO is triggered due to some characteristics of wireless networks such as sudden delay, wireless errors, etc. Moreover, TCP has no ability to identify if an FRR/RTO is triggered by congestion or not. It responds to all FRRs/RTOs by blindly reducing the transmission rate as well as by retransmitting the outstanding packets unnecessarily. Thus, it is unavoidable for TCP to underutilizes available bandwidth and its performance degrades severely in wireless networks [19,20].

In this section, using a simple scenario, we observed how TCP's congestion control algorithms respond to packet losses caused by wireless transmission errors. The scenario is designed to have only one TCP flow between a TCP sender and a TCP receiver in order to avoid causing congestion. Thus, none of RTOs or FRRs is triggered due to congestion in the scenario. First, we simulated the scenario without adding wireless transmission errors between the sender and the receiver, and traced all the variables related to TCP's congestion control into a trace file. After that, we simulated it again after adding 1% wireless losses (packets are lost due to wireless transmission errors), and then we compared the differences of TCP's behavior with/without wireless losses.

Figure 1 shows the results. In Figure 1a, we checked the differences of TCP's *cwnd* before/after adding wireless transmission errors. In the figure, we can see that *cwnd* dramatically increases after a TCP connection starts at 1 s, and it reaches its maximum value at 1.25 s. After that, *cwnd* does not decrease and keeps its maximum size until the simulation ends because there is no packet loss (the black line in Figure 1a). On the other hand, when wireless errors are added, *cwnd* sharply shrinks when an RTO expires at 1.25 s (the yellow line in Figure 1a). After that, it starts to increase, but it decreases again due to another wireless loss. Finally, *cwnd* could not reach its maximum value again even though the network is not congested.

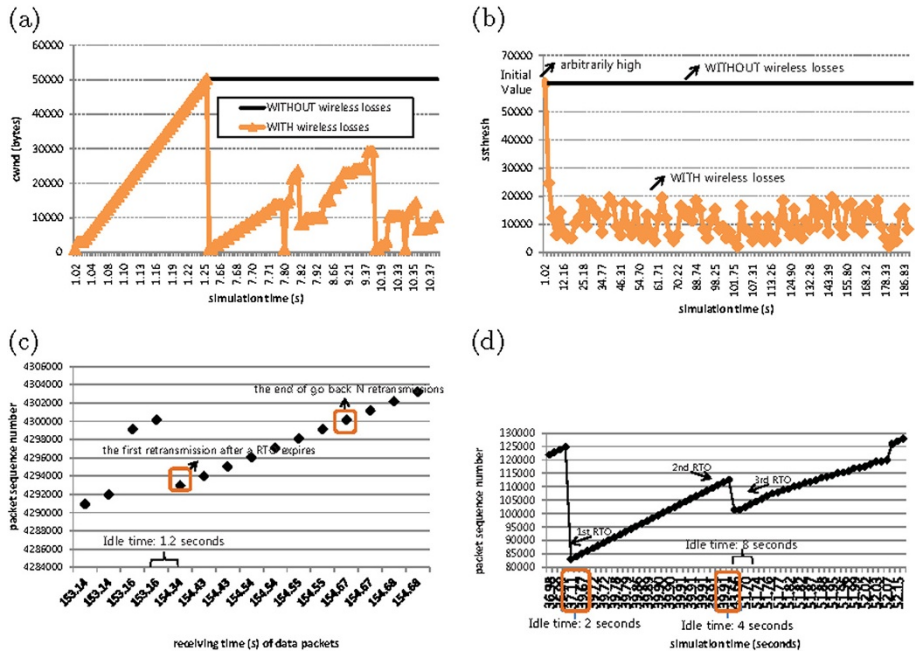

Figure 1 TCP's misbehaviors with wireless losses.

Figure 2 shows the variations of *cwnd* increasing exponentially or linearly before/after *ssthresh* around 8 s in Figure 1a. When *ssthresh* is set to 12,000 and *cwnd* is smaller than *ssthresh* (12,000), *cwnd* increases exponentially until it reaches *ssthresh*. After *cwnd* reaches *ssthresh* around 8.7 s, *cwnd* increases linearly.

When a packet is lost, generally TCP requires a long time to recover its maximum transmission rate. Using the periodic model [21], we can calculate the recovery time (*rt*) using the following formula.

$$rt = \frac{\text{bandwidth} \times \text{RTT}^2}{2 * \text{packetsize}} \quad (1)$$

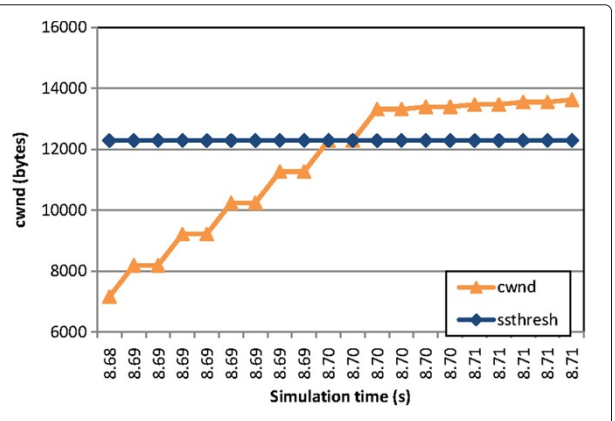

Figure 2 *cwnd*'s exponential or linear increases before/after *ssthresh*.

For example, let us suppose that the bandwidth is 11 Mbps, the packet size is 1000 Bytes, and the round-trip time is 100 ms. Then, it takes approximately 7 s for TCP to reach its maximum transmission rate after a single FRR. As the bandwidth or RTT increases, the recovery time also increases. If the bandwidth is 54 Mbps, the recovery time becomes 34 s. In other words, a TCP connection could end without fully utilizing the available bandwidth even from a single packet loss.

Figure 1b shows the differences of TCP's *ssthresh* before/after adding wireless transmission errors in the scenario. When there is no wireless losses, *ssthresh* does not decrease from its initial value which is arbitrarily high (the black line in Figure 1b). But, when we add wireless losses into the simulation scenarios, *ssthresh* decreases sharply and frequently due to wireless losses (the yellow line in Figure 1b). *ssthresh* is the border for TCP to decide if it increases its transmission rate exponentially or linearly after a packet loss. As *ssthresh* becomes smaller, TCP enters earlier the phase to linearly increase its transmission rate. As a result, the recovery time from a loss significantly increases.

Figure 1c shows the packet sequence number before/after an RTO expires. In the figure, each dot indicates the sequence number of a data packet which is received at the time shown in *x*-axis, For example, two data packets arrived at a TCP receiver in order at 153.14 s. Also, we can see that two out-of-order packets are received at 153.16 s. After 1.2 s of the idle time, an RTO is

expired at 154.34 s and 8 packets are retransmitted until 154.67 s.

Such retransmissions are necessary only when those packets are lost. However, if the RTO is triggered by sudden delay and the data packets are still in transit or in the queue, the retransmissions might be just a waste of time and bandwidth. Also, such unnecessary retransmitted packets could create a new series of duplicate ACKs that can be long enough to cause another FRR.

Figure 1d shows the case when three successive RTOs are triggered due to wireless losses in the scenario. At about 39 s, the first RTO is triggered, and the second and the third RTOs are triggered at 43 and at 51 s, respectively. TCP sender's waiting time in each RTO is approximately 2, 4, and 8 s due to TCP's back-off mechanism. During the idle time, a TCP sender cannot send any data packets wasting available bandwidth and time. If RTOs are triggered regardless of congestion, a TCP sender does not need to perform its back-off mechanism, and should retransmit the lost packets as soon as possible since the network is not congested.

As shown in Figure 1, TCP's FRRs/RTOs do not work anymore as they were originally intended to when they are triggered regardless of congestion. Such congestion irrelative FRRs/RTOs are the main cause of TCP's performance degradation in wireless networks. To avoid such performance degradation, TCP needs to know the cause of FRRs/RTOs, and should respond to each of FRRs/RTOs differently according to the causes.

Here, we classify FRRs/RTOs into three types according to the causes: *congestion FRRs/RTOs*, *spurious FRRs/RTOs*, and *wireless FRRs/RTOs*. Congestion FRRs/RTOs are those FRRs or RTOs triggered by congestion losses (packets are lost due to congestion), spurious FRRs/RTOs [5,6,22] are those FRRs or RTOs triggered by sudden delay without any packet losses, and wireless FRRs/RTOs are those FRRs or RTOs triggered by wireless losses (packets are lost due to wireless transmission errors). Among the three types, congestion FRRs/RTOs are conventional FRRs/RTOs, and both wireless and spurious FRRs/RTOs are the congestion irrelative FRRs/RTOs.

Related study and motivation

Since TCP's FRRs/RTOs [2,4] are originally designed to control congestion, their behaviors are not appropriate if they are triggered regardless of congestion. In case of wireless FRRs/RTOs, it might be necessary for TCP to perform go-back-N retransmissions, but TCP should not reduce sharply its transmission rate and does not need to increase exponentially its back-off value since the network is not congested. In case of spurious FRRs/RTOs, even the retransmissions are unnecessary since the FRRs/RTOs could have been avoided if a TCP sender had waited longer.

Unfortunately, TCP has no ability to distinguish the cause of an FRR or an RTO. It responds to all FRRs/RTOs blindly by reducing sharply the transmission rate and by increasing the back-off value unnecessarily. Thus, it is unavoidable for TCP to underutilize available bandwidth and its performance degrades severely in wireless networks.

To date, many previous studies have been proposed to solve such TCP's problem in wireless networks. Some loss differentiation schemes such as Westwood [8], JTCP [9], and RELDS [10] have been proposed to detect wireless FRRs/RTOs. These schemes distinguish wireless losses from congestion losses, and let TCP avoid reducing unnecessarily its transmission rate when any of FRRs/RTOs are triggered by wireless losses.

To detect spurious FRRs/RTOs, several schemes have been proposed such as Eifel [11-13], F-RTO [14,15], and STODER [16]. These schemes are very effective in detecting spurious RTOs, and they help TCP avoid unnecessary retransmissions caused by spurious RTOs. The common problem of the above schemes is that they paid little attention on adjusting the transmission rate appropriately when spurious or wireless FRRs/RTOs are detected even though controlling appropriately the transmission rate is critical in improving TCP's performance.

In case of TCP Westwood [8] and Prairie [18], they dynamically estimate the available bandwidth, and adjust the transmission rate by updating *ssthresh* and *cwnd* to the large values in proportion to the estimated available bandwidth. They have more significant impact on improving TCP's performance compared to the other previous studies. Unfortunately, updating *cwnd* and *ssthresh* to large values does not always improve TCP's performance. If the loss rate due to wireless errors is high, a large *cwnd* could have a high possibility to have frequent burst losses as well as to have long go-back-N retransmissions compared to a small *cwnd*. Thus, if the packet loss rate is high, a small *cwnd* could be better to improve TCP's performance.

To confirm our claim, we conducted a simple experiment using a simulation scenario to check the impact of *cwnd*'s size when the rate of wireless errors is high. The scenario is designed to have only wireless losses, and the rate of packet losses due to wireless errors is approximately 5%. In each simulation, we just limited the maximum size of *cwnd* and measured the performance using the same scenario. Figure 3 shows the result. The graph simply shows that the performance degrades as the value of *cwnd* increases. While it is well known that TCP's performance improves significantly when *cwnd* is proportional to the available bandwidth, this experiment shows that it is also necessary to make *cwnd* be smaller when the rate of packet losses is high.

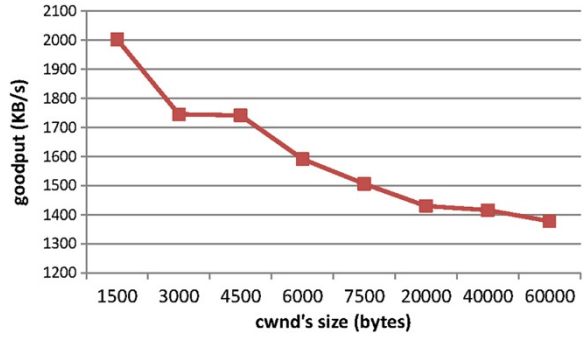

Figure 3 Performance according to *cwnd*'s maximum size when the loss rate is 5%.

Dynamic responding algorithm

As mentioned above, it has been overlooked that *cwnd* needs to be smaller when wireless transmission errors are high. In this section, we propose an enhanced TCP to dynamically adjust its transmission rate based on the packet loss rate as well as the available bandwidth. Since many previous studies have been proposed to estimate the available bandwidth, we use one of them, Prairie's estimation (ABW), which is more stable and accurate than TCP Westwood's. In this article, we describe how to estimate the packet loss rate at TCP sender side, and how to adjust *ssthresh* and *cwnd* according to the estimated bandwidth and the estimated loss rate.

Measuring accurately the rate of packet losses caused by wireless errors is almost not possible at transport layer. Thus, we will estimate the packet loss rate by checking how often FRRs/RTOs are triggered since TCP invokes FRRs/RTOs whenever packets are lost. Although our estimation includes the rate of packet losses caused by congestion, it is useful since *cwnd* needs to be smaller even when packets are lost due to congestion.

When a kth FRR or RTO is triggered, we assume that one packet is lost and the kth loss rate is computed based on the following formula.

$$Loss\_rate_k = \frac{1}{partial\_total\_data\_sent} \times 100 \tag{2}$$

where *partial_total_data_sent* is the number of data packets sent by a TCP sender during the time between $(k-1)$th FRR/RTO and kth FRR/RTO.

Since the loss rate could fluctuate under dynamic network conditions, we smooth the loss rate using the exponential weighted moving average like the following formula.

$$SLoss\_rate_k = SLoss\_rate_{(k-1)}(1-\theta_k) + Loss\_rate_k\theta_k \tag{3}$$

As θ_k ranging from zero to 1 increases, $SLoss\_rate_k$ is greatly affected by the current network status. Otherwise, $SLoss\_rate_k$ is more reflecting the past estimation.

In our scheme, θ_k is also dynamically determined to reflect network conditions as shown in the following formula.

$$\theta_k = \frac{Interval_{current}}{Interval_{max}} \tag{4}$$

where $Interval_{current}$ is the time between $(k-1)$th FRR/RTO and kth FRR/RTO, and $Interval_{max}$ is the maximum value of $Interval_{current}$ samples. In other words, as $Interval_{current}$ is larger, $SLoss\_rate_k$ reflects the current estimation greater than the past estimation.

If the loss rate is high, a large *cwnd* might result in burst losses or long go-back-N retransmissions. Thus, as the loss rate increases, *cwnd* needs to be smaller even when the bandwidth is sufficient. Our question is how much *cwnd* needs to be decreased as the loss rate increases.

To find our best answer, we represent the relation between *cwnd* and the loss rate in a simple way as shown in Figure 4. In the figure, y-axis represents *reduced_cwnd*, which is the *cwnd* reduced according to the loss rate, and x-axis represents the loss rate ranging from 0 to 1. In the graph, *cwnd_max* denotes the possible maximum value of *reduced_cwnd* when the loss rate (*Sloss_rate*) is higher than 0, and β is the minimum size of *reduced_cwnd* when the loss rate is higher than α. Thus, *reduced_cwnd* ranges from β to *cwnd_max*. As the loss rate (x) increases, *reduced_cwnd* decreases from its maximum size (*cwnd_max*) according to the solid line in Figure 4. When the loss rate is higher than α, *reduced_cwnd* is set to its minimum size (β). This is to avoid making *cwnd* be smaller than one segment.

In the graph, "> 0" means the first number higher than 0, thus, we assume it as zero and formulate the inverse proportion relation between *cwnd* and the loss rate based

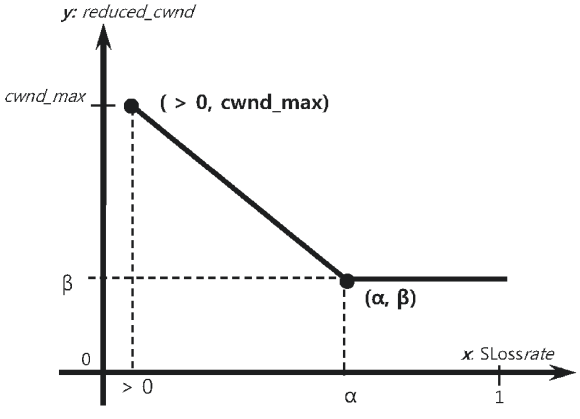

Figure 4 Relation between the loss rate and *cwnd*.

on the two points (0, *cwnd_max*) and (α, β) using the below linear equation.

$$y = \begin{cases} \frac{\beta - cwnd\_max}{\alpha} \times x + cwnd\_max & \text{if } 0 \le x < \alpha \\ \beta & \text{if } x \ge \alpha \end{cases} \quad (5)$$

The following is our algorithm to update *cwnd* and *ssthresh* according to our estimated bandwidth and loss rate.

An algorithm updating *ssthresh* and *cwnd* when a fast recovery or a RTO is triggered

1: *ssthresh* = *ABW*
2: *cwnd* = *ssthresh* / 2
3: *reduced_cwnd* = *cwnd*
4: **if**($SLoss_{rate} \succ 0$)
5: *reduced_cwnd* = $\frac{\beta - cwnd\_max}{\alpha} \times SLoss_{rate} + cwnd\_max$
6: *reduced_cwnd* = **max**(*reduced_cwnd*, β)
7: **end if**
8: *cwnd* = **min** (*cwnd*, *reduced_cwnd*)

When an FRR or an RTO is triggered, *ssthresh* is updated to the estimated available bandwidth (*ABW*) at line 1 instead of being halved. Since *ssthresh* is the border for TCP to decide if its transmission rate increases exponentially or linearly, the recovery time after a packet loss is affected by *ssthresh*. As *ssthresh* is larger, the recovery time after a packet loss is shorter. By setting *ssthresh* to *ABW*, it lets TCP increase its transmission rate exponentially until *cwnd* fully utilizes available bandwidth.

At line 2, *cwnd* is updated to the halved value of *ssthresh*. Although a TCP sender could send data packets as much as *ssthresh* without receiving an ACK from a TCP receiver, we update *cwnd* to the halved value of *ssthresh*. The reason is to avoid causing sudden congestion in a queue by a large *cwnd*. Since *cwnd* is smaller than *ssthresh*, *cwnd* will increase exponentially until it reaches *ssthresh*.

If the loss rate is zero, *cwnd* is finally determined to the halved value of *ssthresh* at line 8. Otherwise, *reduced_cwnd* is calculated according to Equation (5) as we mentioned before. Equation (5) is represented in lines from 5 to 6 of the algorithm. After computing *reduced_cwnd*, *cwnd* is readjusted to the minimum value between the halved value of *ssthresh* and *reduced_cwnd* at line 8. By taking the minimum value, it enables to avoid burst losses caused by insufficient bandwidth as well as the high loss rate.

The following shows another algorithm to remove unnecessarily increased idle time of an RTO.

An algorithm updating the back-off value when an RTO is triggered

1: **if**(*BDP* > the halved value of **cwnd_max**)
2: initializes the back-off value
3: **end if**

When successive RTOs are triggered, the back-off value exponentially increases, which incurs a huge idle time of the sender. If the network is not congested, such unnecessarily increased back-off value will be just a severe waste of bandwidth and time. Our scheme initializes the back-off value if the network is not congested when successive RTOs are triggered. In order to check if network is congested or not, our scheme uses the bandwidth-delay product (*BDP*), which is calculated by the product of the available bandwidth and the round trip time (it is well known that TCP's performance is maximized when *cwnd* is equal to *BDP*).

At line 1 in the algorithm, our scheme checks if current *BDP* is larger than the halved value of *cwnd_max* when an RTO is triggered. If the condition is satisfied, we assume that the network is not congested. In that case, our scheme initializes the back-off value in order to avoid an unnecessary idle time.

As shown in the algorithms, our scheme lets TCP respond to every FRR/RTO dynamically by considering the available bandwidth and the loss rate. By doing so, our scheme helps TCP avoid causing frequent burst losses while it utilizes the available bandwidth. As a result, it has significant effects to avoid the performance degradation caused by the congestion irrelative FRRs/RTOs.

If we use a quadratic equation instead of the linear equation in (5), it would be better to adjust the transmission rate more appropriately, but it could cause more overhead of the complexity. In order to avoid such overhead, we used the linear equation in (5). Compared to the conventional TCP, our scheme does more computations to control its transmission rate according to the network conditions. The complexity overhead of our scheme is based on the simple linear equation in (5). Since the value of x in the equation ranges from 0 to 1, the computation in the equation is simple and the overhead is reasonable.

To maximize the performance enhancement of our scheme, we conducted extensive experiments to find the best value for α, β, and *cwnd_max*, and the results will be shown in the following section.

Experimentation and analysis

In this section, we observe how our scheme adjusts the transmission rate per FRR or RTO, and evaluate our scheme in terms of goodput, fairness, and friendliness using a network simulator, QualNet [17]. For this, we design more than 100 different simulation scenarios by

Table 1 Simulation parameters

Simulator	QualNet 4.5
Topology	A chain topology, a dumbbell topology
Bandwidth	Wireless (1/2/5/11 Mbps), wired (10/100 Mbps)
Propagation delay	1/5/10/30/60/90 ms
Application	FTP/generic
Transport protocol	TCP Reno
TCP send/recv buffer	50000 Bytes
TCP packet size	1024 Bytes
UDP packet size	512 Bytes
Queuing policy	DropTail
Simulation time	200 s (including 35-s warm-up)

setting different values of network parameters such as the loss rate, and the hop count under various network topologies. Table 1 shows the detailed parameters used in our simulation scenarios. We used TCP Reno as conventional TCP and its congestion window size is limited to hold 50 packets. In all experiments, each scenario lasts about 200 s, and data packets of TCP are continually transmitted during the simulation time.

Each scenario has wireless losses as well as congestion losses. To make congestion losses, we increased the number of TCP flow gradually like 1, 3, 6, 12, 15, 18, and 21 flows between the sender and the receiver. As the number of TCP flow increases, the loss rate due to congestion also increases. To incur wireless losses, we added wireless transmission errors at the wireless link using QualNet's fault interface. The packet loss rate in the scenarios ranges from 0 to 15%, and the ratio of wireless losses to congestion losses is approximately one of 1:0, 2:8, 3:7, 4:6, 5:5, 6:4, 7:3, 8:2, 0:1 in a scenario.

Throughout the extensive scenarios, we aim (1) to observe how often congestion irrelative FRRs/RTOs are triggered, (2) to suggest the best values for α and β in our algorithm, (3) to check if our scheme works as we intended, (4) to measure the performance enhancement of our scheme and compare it with previous studies, (5) to evaluate our scheme's fairness and friendliness.

Three types of FRRs/RTOs

In our experiment, we observed how often congestion irrelative FRRs/RTOs are triggered in a simple topology shown in Figure 5a. The simple topology consists of two nodes and a base station. *S* denotes the sender, and *D* denotes the receiver. *S* connects to the base station via a 10-Mbps wired link with 10-ms propagation delay, and the base station is linked to *D* via a 2-Mbps wireless link with 1-ms propagation delay.

And we modified QualNet's source code to trace all the packet information related to the triggered FRRs/RTOs into trace files in a scenario. During each scenario simulation, we traced all information of the packets dropped at MAC Layer into a file named by "DroppedAtMac", traced all the packets dropped at network layer into "DroppedAtNetwork". Also, we traced all the packets which triggered FRRs or RTOs at Transport layer into "FRRatTransport" or "RTOatTransport", respectively.

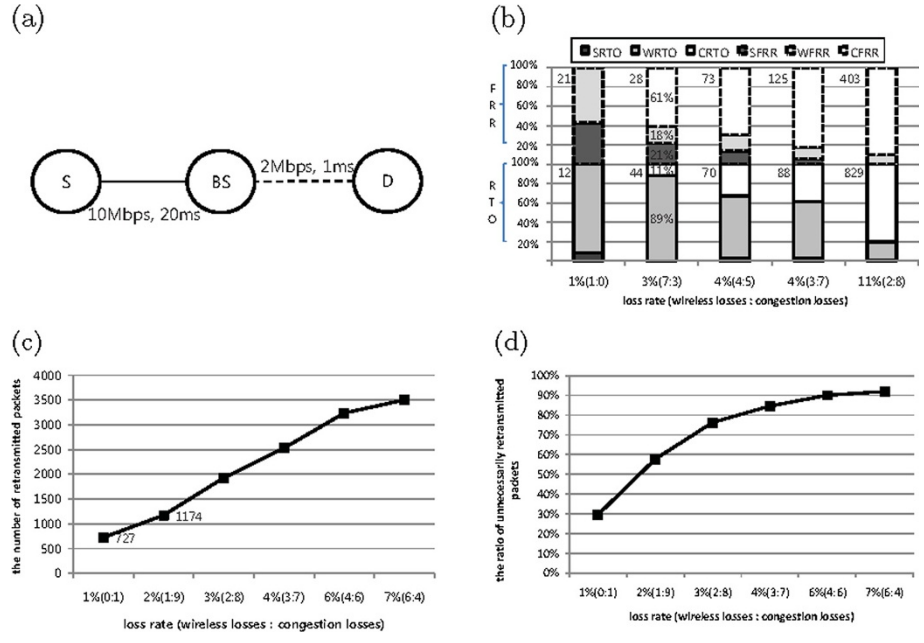

Figure 5 Three types of FRRs/RTOs.

When the simulation ends, we checked if each packet at "FRRatTransport" or "RTOatTransport" exists at "DroppedAtMac" or "DroppedAtNetwork". If a packet of "FRRatTransport" exists at "DroppedAtMac", we assumed that the FRR is triggered due to wireless errors, and we treated it as wireless FRRs. If the packet is found at "DroppedAtNetwork", we assumed that the FRR is triggered due to congestion, and we treated it as congestion FRRs. In a similar way, if a packet of "RTOatTranport" exists at "DroppedAtMac", we treated the RTO as wireless RTOs. If the packet is found at "DroppedAtNetwork", we treated the RTO as congestion RTOs. Lastly, if the packet is not found either at "DroppedAtMac" or "DroppedAtNetwork", we assumed that the FRR or the RTO is triggered by sudden delay, and we treated it as spurious FRRs or RTOs.

Figure 5 shows the results of our observation. In Figure 5b, each of CRTO, WRTO, and SRTO represents congestion, wireless, and spurious RTOs, respectively, and the ratio of each of them is shown in a solid line bar graph. Each of CFRR, WFRR, and SFRR represents congestion, wireless, and spurious FRRs, respectively, and the ratio is shown in a dashed line bar graph.

For example, when the loss rate is 3% and the ratio of wireless losses to congestion losses is 7:3, 44 RTOs are triggered in a scenario, and each ratio of congestion and wireless RTOs is, respectively, 11 and 89% among 44 RTOs. In case of FRRs, 28 FRRs are triggered, and each ratio of congestion, wireless, and spurious FRRs is 61, 18, and 21%, respectively.

In Figure 5b, we can see that as congestion increases (to the right direction in the figure), the ratio of CRTO or CFRR increases. On the other hand, as the ratio of wireless losses increases (to the left direction), the ratio of congestion irrelative FRRs/RTOs increases. In either case, however, the ratio of congestion irrelative FRRs/RTOs is always higher than 30%. If we consider the fact that most packets are lost due to wireless transmission errors in wireless networks, the ratio of congestion irrelative FRRs/RTOs might be much higher than 30% in real networks.

Figure 5c shows the number of retransmitted packets in a scenario according to the loss rate, and also shows the ratio of "unnecessary" retransmitted packet among all the retransmitted packets. For example, when the packet loss rate is 1% and the ratio of wireless losses to congestion losses is 0:1 in a scenario, 727 packets are retransmitted (in Figure 5c), and 30% of them are unnecessarily retransmitted (in Figure 5d). From the two graphs, we can see that, as the ratio of wireless losses increases, the ratio of "unnecessary" retransmitted packet significantly increases. In the worst case, almost 90% of retransmitted packets are unnecessarily retransmitted.

Testing for α and β

Whenever an FRR/RTO is triggered, our scheme updates *cwnd* and *ssthresh* according to the linear equation (5) as described in "Dynamic responding algorithm" section. The slope of the equation represents the degree of reduction in *cwnd* according to the loss rate, and that is determined by α, β, and *cwnd_max*.

To find the best values for α, β, and *cwnd_max*, we tested various values between the lower bound and the upper bound of each variable. For β, which is the minimum size of *cwnd*, we tested the values ranging from one segment to the halved size of the maximum value of *cwnd*, which is 65535 in conventional TCP. For α, which is a threshold to prevent *cwnd* from being smaller than β, we tested the values ranging from 0.1 to 10%. For *cwnd_max*, which is the possible maximum value of *reduced_cwnd*, we tested the values ranging from 10000 to the maximum value of *cwnd*, 65535.

After testing various values, we chose the strong candidates for the values, and again conducted experiments to see the best combination. Figure 6a,b shows the results. First, Figure 6a shows the performance comparison according to α and β when *cwnd_max* is fixed to 65535. In the figure, we chose the strong candidates of

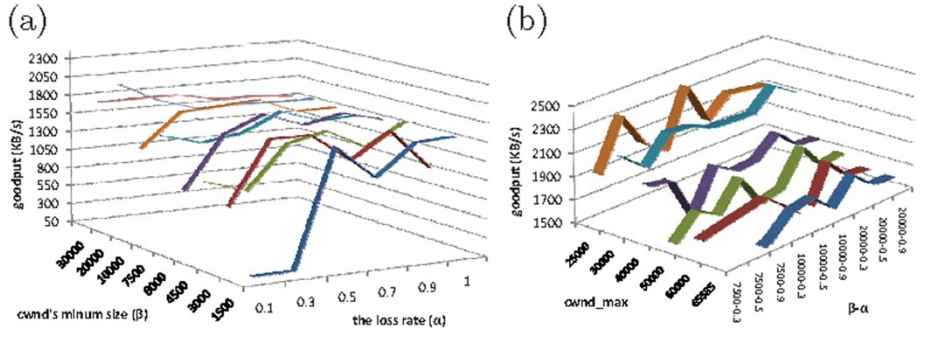

Figure 6 Testing values for α and β.

the combinations of α and β. After that, in Figure 6b, we tested various values for *cwnd_max* with the strong candidates of α and β. The result shows that the performance is the best when α, β, and *cwnd_max* is 0.5%, 10000 bytes, and 25000 bytes, respectively. Namely, while *cwnd* decreases linearly as the loss rate increases based on the equation, these parameters indicate that (1) if the loss rate is higher than zero, *cwnd* cannot be higher than 25000, and (2) if the loss rate is higher than 0.5%, *cwnd* cannot be smaller than 10000. The following results in this article are based on these values.

Comparison of responding behaviors to FRRs/RTOs

Using a simple scenario, we observed if our scheme responds to packet losses as we intended, and also compared our scheme's response with conventional TCP's response. In the scenario, only one TCP connection flows from the source to the destination in the simple topology. Thus, packets are lost only due to wireless transmission errors and the loss rate is about 5%.

In Figure 7a,b, we compared the variations of *cwnd* between TCP and our scheme during the go-back-*N* retransmissions after an RTO expires in the scenario. In Figure 7a, TCP's *cwnd* becomes one segment (1024) at 54.70 s when an RTO is triggered, and *ssthresh* reduces into 5120. Whenever an ACK is received, *cwnd* increases gradually, and it is finally set to 6480 at 54.87 s when the retransmissions end. On the other hand, our algorithm's *ssthresh* becomes 132811 at 54.70 s by reflecting the available bandwidth. And, its *cwnd* becomes about 11024 at 54.75 s by reflecting the available bandwidth and the loss rate. When the retransmissions end, *cwnd* of our scheme is set to 10000 at 54.82 s.

Table 2 Comparison of TCP and the proposed scheme

Comparison	RTOs	FRRs	Retransmitted packets	Goodput
TCP	56	82	711	607057
Dynamic Responding	32	82	419	899390

During the go-back-*N* retransmissions of the RTO, TCP retransmitted 13 packets and also our scheme retransmitted 13 packets. We checked how much time it took to retransmit the packets in each scheme. In case of TCP, it took 0.158 s to retransmit 13 packets, and in case of our scheme it took 0.114 s to retransmit the same number of packets. This shows that our scheme recovered from the loss faster than TCP by avoiding reducing unconditionally *cwnd* and *ssthresh*.

Figure 7c,d shows the differences between TCP's response and our scheme's response when an FRR is triggered. In each graph, the variations of *cwnd*/*ssthresh* are represented. In Figure 7c, TCP sends the lost packet, which triggered the FRR, at 27.44 s, and it increases *cwnd* by one segment whenever a duplicate ACK is received until 27.49 s. When a new ACK is received, *cwnd* is set to the halved value (5245) at 27.57 s.

Our scheme's response is similar with that of TCP until a new ACK is received as shown in Figure 7d. When a new ACK is received, our scheme does not blindly halve *cwnd*, but it updates *cwnd* according to the available bandwidth

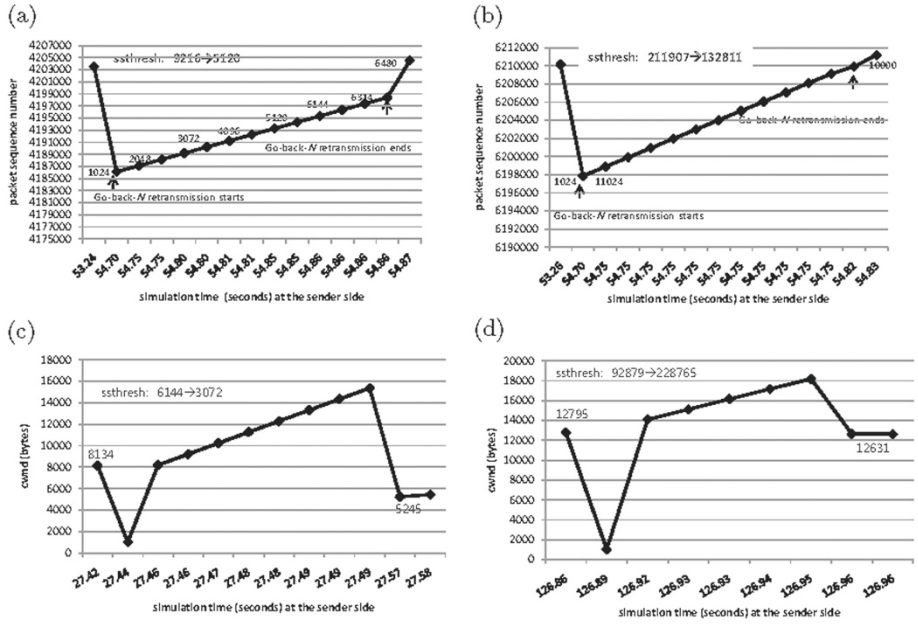

Figure 7 Comparison of response to FRR and RTO.

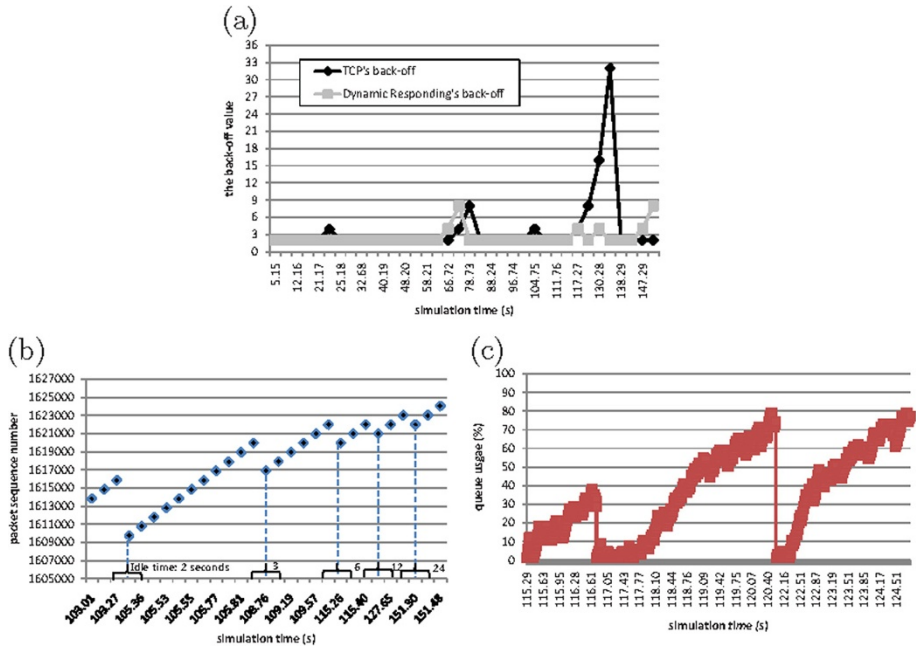

Figure 8 Comparison in the back-off value.

and the loss rate. As a result, our scheme's *cwnd* is similar with the previous *cwnd* (12631).

In Table 2, we compared how many FRR or RTOs are triggered in the scenario in each response. As shown in the table, when every packet loss is responded by the conventional way of TCP, 711 data packets are retransmitted, and 56 RTOs/82 FRRs are triggered. However, when all the packet losses are responded by our scheme, only 419 data packets are retransmitted, and 32 RTOs/82 FRRs are triggered in the same scenario. This table shows that our scheme could reduce the number of the congestion irrelative FRRs/RTOs, and the number of unnecessarily retransmitted packets by responding appropriately to the triggered FRRs/RTOs according to the network conditions. As a result, our scheme's performance reached 148% of TCP's performance in the scenario.

Comparison of back-off mechanism

We compared the variations of the back-off value when each of TCP and our scheme is applied to a scenario in the simple topology. In the scenario, there are 10 connections between the sender and the receiver, and the loss rate is about 10% (the ratio of wireless losses to congestion losses is 3:7). In Figure 8a, TCP's back-off value reaches 32 in the worst case, while our scheme's back-off value reaches 8 in the same scenario. As the back-off value increases, the waiting time for the sender to trigger an RTO exponentially increases.

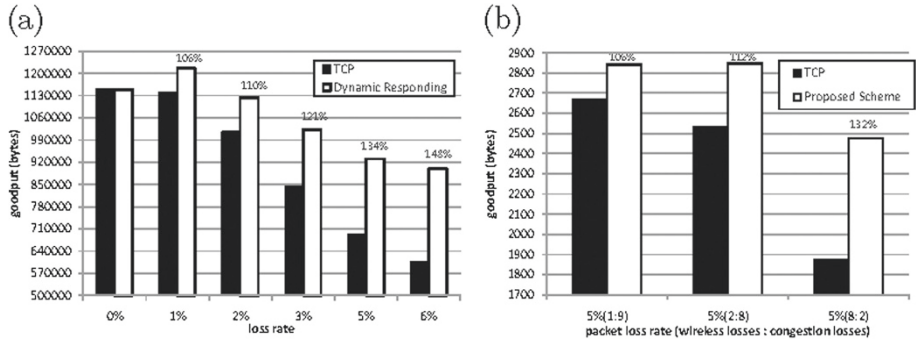

Figure 9 Results in a simple topology.

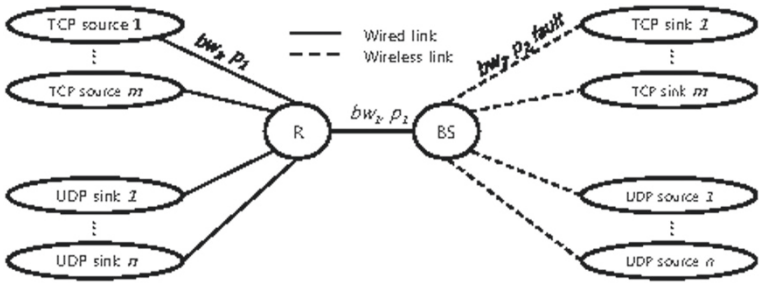

Figure 10 A dumbbell topology.

We checked how much time the sender spent time on waiting to trigger RTOs. Figure 8b shows the packet sequence number around when the back-off value reaches 32 in the scenario. As shown in the figure, successive RTOs are triggered at 105.36, 108.76, 115.26, 127.65, and 151.30. In each RTO, the sender's waiting time increases exponentially like about 2, 3, 6, 12, and 24 s. In other words, a TCP sender wastes 47 s during the time from 103 to 151 s in the scenario.

Such back-off mechanism is necessary only when congestion is heavy. Otherwise, it just wastes time and bandwidth. We checked the queue usage to see if there was heavy congestion when the back-off value reached 32. Figure 8c shows the queue usage during the time from 115 to 124 s before the last successive RTO is triggered. As shown in the figure there is no congestion until 119 s. After 119 s, the queue usage is getting full and then it is almost empty at about 120 s. During the idle time of a TCP sender assuming that the network is severely congested, the queue usage is not always high, and it changes dynamically by other TCP connections. Consequently, such long back-off value was unnecessary to a TCP sender.

Our scheme does not increase blindly the back-off value when successive RTOs are triggered. It increases the back-off value only when the network is assumed to be congested. For this, our scheme checks if *BDP* is larger than the halved value of *cwnd_max* as we mentioned before. If it is, it initializes the back-off value assuming the network is not congested. This is why the back-off value of our scheme is lower than that of TCP in Figure 8a. As a result, our scheme can remove unnecessary idle time of the sender and it outperforms TCP significantly.

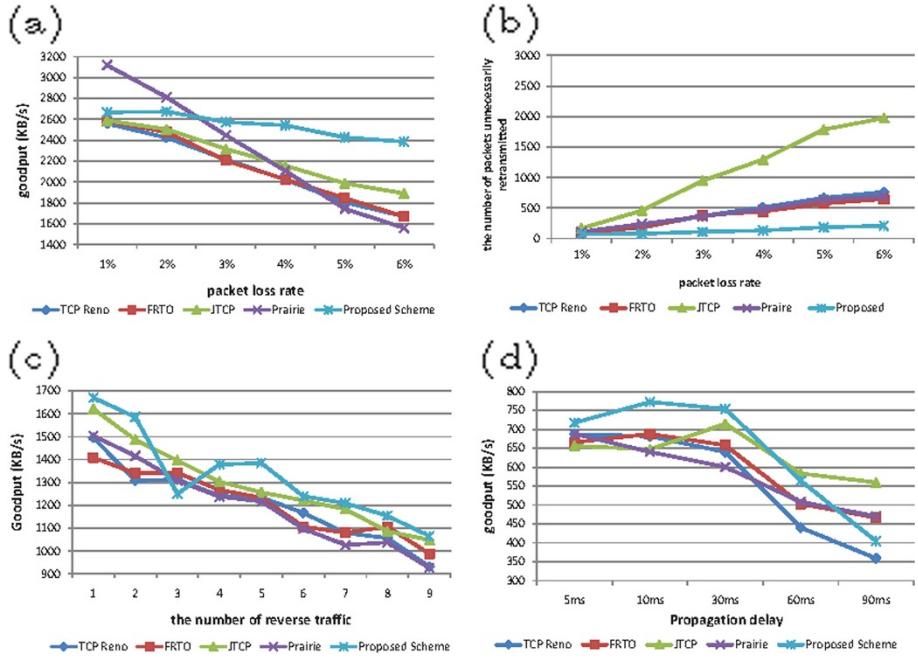

Figure 11 Performance comparison.

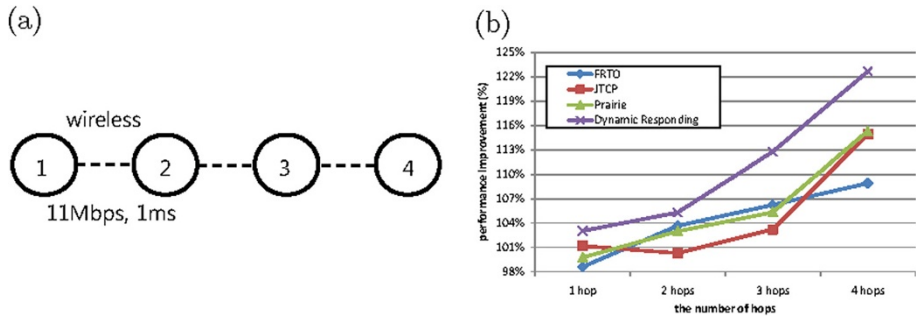

Figure 12 Results in a chain topology.

Performance evaluation

We evaluated the performance of our scheme under various topologies, and compared it with the previous studies such as F-RTO [14,15], JTCP [9], and Prairie [18]. F-RTO is one of previous schemes to detect spurious RTOs, JTCP is one of previous schemes to detect wireless FRRs/RTOs, and Prairie is one of previous schemes to respond FRRs/RTOs according to available bandwidth.

Figure 9 shows the results in the simple topology. The graph in Figure 9a is from the scenarios when all FRRs/RTOs are triggered regardless of congestion. In the graph, we compared the two performances of TCP Reno and our scheme according to the loss rate. As shown in the graph, there is no difference when the loss rate is 0%. However, as the loss rate increases, our scheme outperforms significantly TCP. For example, when the loss rate is 1%, our scheme's performance is 106% of TCP's. And, when the loss rate is 6%, our scheme's performance is 148% of TCP's.

In Figure 9b, we checked the performance variations of our scheme according to the ratio of wireless losses to congestion losses. The graph is from the scenarios whose loss rate is 5% and the ratio of wireless losses to congestion losses is 1:9, 2:8, or 8:2. It shows that the performance enhancement is significantly different according to the ratio of wireless losses to congestion losses even though the loss rate is the same (5%). For example, when the ratio is 1:9 (congestion losses are much higher), the enhancement is 6%. However, when the ratio is 8:2 (wireless losses are much higher), the enhancement is about 32%. This observation shows that our scheme's performance is better as the ratio of the congestion irrelative FRRs/RTOs increases.

We also evaluated the performance of our scheme in a more complex dumbbell topology shown in Figure 10. In the figure, TCP traffics flow from left to right while UDP traffics flow from right to left. TCP's packet sizes are fixed to 1000 bytes (UDP's 512 bytes), and TCP receivers always follow the delayed ACKs algorithm as well as the Nagle algorithm. We designed about several hundreds of scenarios by changing the values, n, m, the bandwidth (bw_1, bw_2), the propagation delay (p_1, p_2), and the wireless fault rate (*fault*) (n ranging from 1 to 12, m from 1 to 10, bw_1 is one of 10/100 Mbps, bw_2 is one of 1/2/5/11 Mbps, p_1 is one of 5/10/30/60/90 ms, p_2 is 1 ms, wireless transmission errors (*fault*) occurs one of 0/5/15/30/45/60/75 times during a 200-s simulation).

In Figure 11a, b, c, d, we measured and compared the performances of the schemes according to the loss rate, the number of reverse traffic, and the propagation delay. In Figure 11a, we can see that Prairie's performance severely degrades as the loss rate increases while our scheme

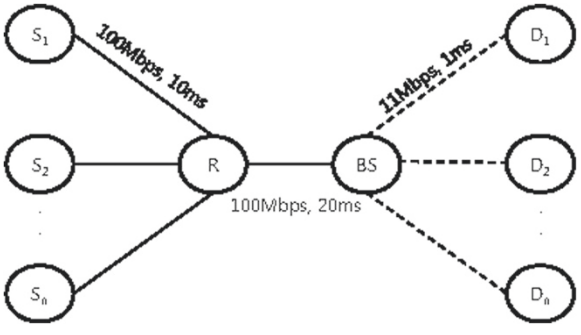

Figure 13 A simple dumbbell topology.

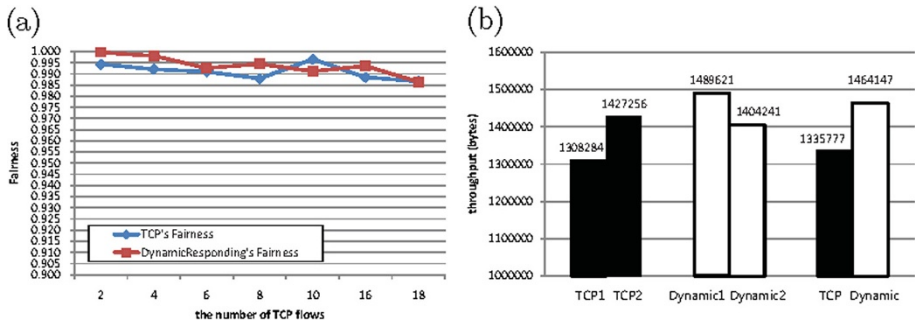

Figure 14 Fairness and friendliness.

shows the highest performance. And we counted the number of unnecessarily retransmitted packets according to the loss rate in Figure 11b. As shown in the graph, our scheme has the lowest unnecessary retransmission while the unnecessary retransmission in previous schemes increases gradually according to the loss rate. Even when the reverse traffic/the propagation delay increases, our scheme shows better performance compared to the other schemes as shown in Figure 11c, d. This is because our scheme has an ability to adjust the transmission rate dynamically and appropriately according to the network conditions.

Lastly, we evaluated our scheme under a multi-hop wireless network. Figure 12a shows a 4-hop chain topology of IEEE 802.11b wireless nodes. The bandwidth of the wireless link is 11 Mbps with 1-ms propagation delay. In this topology, we observed if the performance enhancement of our scheme degrades or not as the number of hops increases.

Figure 12b shows the result in the chain topology. The graph shows that our scheme's performance is the highest among the previous studies, and its performance increases according to the number of hops. This is because as the number of hops increases, the ratio of the congestion irrelative FRRs/RTOs increases due to sudden delay or high wireless errors and our scheme has advantages to handle such FRRs/RTOs.

Throughout the three different topologies, we evaluated our scheme and compared it with the previous studies. The results consistently showed that our scheme has the highest performance among the previous studies, and its performance is better as the ratio of the congestion irrelative FRRs/RTOs increases.

Fairness and friendliness

In order to assess the ability of our scheme in allowing a fair distribution of bandwidth, we also evaluated our scheme in terms of fairness and friendliness. The network topology used in this simulation experiment is the single-bottleneck dumbbell topology as shown in Figure 13.

Fairness is the bandwidth allocation measure for the multiple connections of the "SAME" TCP. To evaluate the fairness, we use the fairness index mentioned in [23]. The fairness index is calculated using the following formula.

$$\text{Fairness} = \frac{(\sum x_i)^2}{n(\sum x_i^2)} \tag{6}$$

where x_i is the throughput of the ith connection, and n is the number of the same TCP connections. The fairness index ranges from $1/n$ to 1.0; a perfectly fair bandwidth allocation results in a fairness index of 1, and if all bandwidth are consumed by one connection, it results in $1/n$.

TCP Reno is well known to get fair network capacity. In this experiment, we measured the fairness of our scheme and compared it with that of TCP Reno. Figure 14a shows the fairness of TCP Reno and that of our scheme according to the number of the same connections. For example, when two TCP Reno connections are running from the source to the destination in Figure 13, the fairness of two TCP Reno connections is about 0.995. And, when two connections of our scheme are running, our scheme's fairness is about 1. Although the fairness index of our scheme tends to fluctuate according to the number of connections, it is very similar with TCP's fairness index and the values are higher than 0.986 even when many connections compete one another. It means that our scheme as well as TCP achieves satisfactory fairness index.

To check if our scheme is able to coexist friendly with other TCP connections such as TCP Reno, we did

Table 3 Friendliness between two different TCP variants

TCP source	Proposed scheme source	TCP average throughput	Proposed scheme average throughput
1	9	296610	309328
3	7	284701	317354
5	5	286520	327523
7	3	284258	359688
9	1	298804	391007

some simulations. First, we evaluated how one connection of our scheme influences on one TCP Reno connection. Figure 14b shows the result. "*TCP1 TCP2*" shows the throughput of each of two TCP Reno connections when the two Reno connections compete with each other. For example, the throughput of one TCP connection is 1,308,284 Bytes, and the throughput of the other TCP connection is 1,427,256 Bytes. "*Dynamic1 Dynamic2*" shows the throughput of each connection of our scheme when two connections of our scheme compete with each other. "*TCP Dynamic*" shows the throughput of each when one TCP Reno competes with one connection of our scheme.

If we compare the throughput of *TCP1* in "*TCP1 TCP2*" with that of *TCP* in "*TCP Dynamic*", we can see that TCP Reno's performance does not degrade due to the connection of our scheme. It means that our scheme is very friendly to TCP Reno while it outperforms TCP Reno.

Using more connections, we checked if the connections of our scheme can coexist with TCP Reno in a friendly way. Table 3 shows another result when ten connections of TCP Reno and our scheme are running from the source to the destination in Figure 13. The first and the second columns in the table show the number of TCP Reno connections and that of our scheme's connections, respectively, among the ten connections. The third and the forth columns show the average throughput of each scheme. For example, when three TCP Reno connections compete with seven connections of our scheme, the average throughput of TCP Reno is 284,701 Bytes, and that of our scheme is 317,354 Bytes. In this table, we can also see that our scheme is friendly to TCP Reno while it outperforms TCP Reno.

Conclusion

In wireless networks, TCP's performance significantly degrades since its congestion control algorithms (FRRs and RTOs) are often triggered regardless of congestion. To avoid such performance degradation, we proposed an enhanced TCP to dynamically adjust the transmission rate based on the available bandwidth and the loss rate. When an FRR/RTO is triggered, our scheme adjusts the transmission rate in proportion to the available bandwidth, and also re-adjusts it in inverse proportion to the loss rate. By doing so, our scheme has significant effects to avoid the performance degradation caused by the congestion irrelative FRRs/RTOs.

Throughout the extensive experiments, we showed that our scheme significantly outperforms previous studies while it maintains the fair and friendly behavior to other TCP connections. Specially, as the number of the congestion irrelative FRRs/RTOs increases, our scheme showed better performance compared to the other schemes. Since our scheme requires only sender-side modification of TCP, it does not need any support either from the receiver side or from the intermediate nodes. Thus, it is convenient to deploy our scheme in wireless networks to improve TCP's performance.

Competing interest

The authors declare that they have no competing interests.

Acknowledgments

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).

Author details

[1] Department of Computer Engineering, Dong-A University, Busan, South Korea. [2] Department of Computer Engineering, Pusan National University, Busan, South Korea.

Received: 14 November 2011 Accepted: 9 August 2012
Published: 25 September 2012

References

1. GR Wright, WR Stevens, *TCP/IP illustrated, Volume 2, The Implementation* (Addison Wesley, Reading, MA, 1995), pp. 891–1025
2. M Allman, V Paxson, W Stevens, TCP congestion control, RFC2581 (1999), http://tools.ietf.org/html/rfc2581. Accessed 29 August 2012
3. WR Stevens, *TCP/IP Illustrated, Volume 1, The Protocols* (Addison Wesley, Reading, MA, 1994), pp. 223–274
4. V Paxson, M Allman, Computing TCP's retransmission timer, RFC2998 (2000), http://tools.ietf.org/html/rfc2988. Accessed 29 August 2012
5. A Gurtov, in *Proc. IFIP TC6/WG6.8 Working Conference on Emerging Personal Wireless Communications,* vol. 195 Effect of delays on TCP performance. (Lappeenranta, Finland, August 2001), pp. 87–108
6. A Gurtov, R Ludwig, in *Proc. 22th Joint conference of the IEEE Computer and Communications,* vol. 3 Responding to spurious timeouts in TCP. (San Francisco, USA, March 2003), pp. 2312–2322
7. J Wang, M Zhou, Y Li, in *Proc. of 7th IEEE International Conference on High Speed Networks and Multimedia Communications,* LNCS vol.3079, Toulouse, France Survey on the end-to-end internet delay measurements. (Springer-Verlag, Toulouse, 2004), pp. 155–166
8. G Yang, R Wang, M Gerla, MY Sanadidi, TCPW bulk repeat. Comput. Commun. **28**(5), 507–518 (2005)
9. EH-K Wu, M-Z Chen, JTCP: jitterbased TCP for heterogeneous wireless networks. IEEE J. Sel. Areas Commun. **22**(4), 757–766 (2004)
10. C-H Lim, J-W Jang, Robust end-to-end loss differentiation scheme for transport control protocol over wired/wireless networks. IET Commun. **2**(2), 284–291 (2008)
11. R Ludwig, RH Katz, The Eifel algorithm: making TCP robust against spurious retransmissions. ACM SIGCOMM Comput. Commun. Rev. **30**(1), 30–36 (2000)
12. R Ludwig, M Meyer, The Eifel detection algorithm for TCP, RFC3522 (2003), http://tools.ietf.org/html/rfc3522. Accessed 29 August 2012
13. R Ludwig, A Gurtov, The Eifel response algorithm for TCP, RFC4015 (2005), http://tools.ietf.org/html/rfc4015. Accessed 29 August 2012
14. P Sarolahti, M Kojo, K Raatikainen, F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts. ACM SIGCOMM Comput. Commun. Rev. **33**(2), 51–63 (2003)
15. P Sarolahti, M Kojo, Forward RTO-recovery (F-RTO): an algorithm for detecting spurious retransmission timeouts with TCP and the stream control transmission protocol (SCTP), RFC4138 (2005), http://tools.ietf.org/html/rfc4138. Accessed 29 August 2012
16. K Tan, Q Zhang, W Zhu, in *Proc. IEEE Global Telecommunications Conference,* vol. 6 STODER: a robust and efficient algorithm for handling spurious retransmit timeouts in TCP. (St. Louis, USA, December 2005), pp. 3692–3696
17. SCALABLE Network Technologies: PRODUCTS QualNet, http://http://www.scalable-networks.com/. Accessed 29 August 2012

18. N Parvez, E Hossain, TCP Prairie: a sender-only TCP modification based on adaptive bandwidth estimation in wired-wireless networks. Comput. Commun. **28**(2), 246–256 (2005)
19. L-P Tung, W-K Shih, T-C Cho, YS Sun, MC Chen, TCP throughput enhancement over wireless mesh networks. IEEE Commun. Mag. **45**(11), 64–70 (2007)
20. Z Fu, P Zerfos, H Luo, S Lu, L Zhang, M Gerla, in *22th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM2003),* vol. 3 The impact of multihop wireless channel on TCP throughput and loss. (San Francisco, USA, March 2003), pp. 1744–1753
21. M Hassan, R Jain, *High Performance TCP/IP Networking: Concepts Issues, and Solutions* (Pearson/Prentice Hall, New Jersey, USA, 2004), pp.125–151
22. YJ Zhu, L Jacob, On making TCP robust against spurious retransmissions. Comput. Commun. **28**(1), 25–36 (2005)
23. A Aggarwal, S Savage, T Anderson, in *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000),* vol. 3 Understanding the performance of TCP pacing. (Tel Aviv, Israel, March 2000), pp. 1157–1165

doi:10.1186/1687-1499-2012-304
Cite this article as: Park *et al.*: TCP's dynamic adjustment of transmission rate to packet losses in wireless networks. *EURASIP Journal on Wireless Communications and Networking* 2012 **2012**:304.