TCP Performance Improvement Considering ACK Loss in Ad Hoc Networks

Dongkyun Kim and Hongseok Yoo

Abstract: **In mobile ad hoc networks, packet loss is unavoidable due to MAC contention, link failure or the inherent characteristics of wireless link. Since TCP relies on the timely reception of TCP ACK packets to progress the transmission of the TCP DATA packets, ACK loss obviously affects the performance due to two main problems: (a) Frequent occurrence of spurious retransmissions caused by timeout events and (b) impairment of the fast retransmit mechanism caused by the lack of a sufficient number of duplicate ACK packets. In particular, since most reactive routing protocols force the packets buffered over a path to be discarded while performing a route recovery, the performance degradation becomes more serious due to such ACK loss. In this paper, therefore, TCP with two piggybacking schemes (called TCP-pgy) is proposed in order to resolve the above-mentioned problems over reactive routing protocols. Through extensive simulations using the ns-2 simulator, we prove that our proposed schemes contribute to TCP performance improvements.**

Index Terms: **Fast retransmit, mobile ad hoc network (MANET), reactive routing protocol, spurious retransmission, TCP ACK loss.**

I. INTRODUCTION

A mobile ad hoc network (MANET) is a wireless network where all nomadic nodes with a fixed radio range are able to communicate with each other without relying on network infrastructure. Since packet forwarding and routing is done via intermediate nodes, the MANET working group in IETF [1] has standardized ad hoc on-demand distance vector (AODV) [2] and optimized link state routing (OLSR) [3] as its reactive and proactive routing protocols, respectively. Moreover, the working group is currently trying to standardize dynamic Manet on-demand (DYMO) [4] as its generalized reactive routing protocol. In proactive protocols, routing information to all possible destinations in the network is maintained so that a packet can be transmitted over an already-made routing path. In reactive protocols, a routing path is acquired in an on-demand manner when a source desires to send packets to a destination. In addition, a hybrid routing protocol like zone routing protocol (ZRP) [5] has been proposed in order to support a large-scale, ad hoc network.

In addition to the network layer protocol, a transport protocol is also needed to provide end-to-end reliability between the source and destination nodes. Thus, since transmission control protocol (TCP) is widely used with the Internet and smooth integration with the fixed Internet is required, it is considered to be a good candidate for transport protocol. Standard TCP that is tailored for the fixed Internet, however, cannot be directly applied to MANETs because it does not differentiate between packet loss caused by route failure and that caused by network congestion. Therefore, a great deal of research has been conducted to improve TCP performance.

Most research, however, has never considered the possibility that the TCP ACK packets can be easily lost while being propagated to a TCP sender, due to wireless channel contention or channel interference. Since the timely arrival of ACK packets allows a TCP sender to transmit more TCP DATA packets (through the so-called "ACK-clocking" technique), the absence of ACK packets (i.e., timeout events) causes the corresponding TCP DATA packets to be retransmitted unnecessarily, even when they have arrived at the destination node successfully (called "spurious retransmissions"). Therefore, more traffic injection, through spurious retransmissions, will aggravate channel contention or interference, thus resulting in performance degradation of all TCP flows in the network.

This paper therefore contributes to improving TCP performance in MANETs through two schemes: (a) Reducing spurious retransmissions (S1 scheme) and (b) retransmitting a lost TCP DATA packet in a timely manner (S2 scheme). Both S1 and S2 schemes utilize piggybacking techniques of certain control information onto transmitted packets. Hence, TCP with these two schemes is called TCP-piggyback in this paper (abbreviated by TCP-pgy).

To reduce spurious retransmissions, it is important to inform a TCP sender of the sequence number expected by the TCP receiver (i.e., ACK sequence). To support this, the S1 scheme utilizes the underlying on-demand routing protocols such as AODV, dynamic source routing (DSR), and DYMO [1].

In these routing protocols, if packet loss occurs over a wireless link, the intermediate node which has experienced the loss will notify the packet's source node of a link failure. Thereafter, the source node has to acquire a new path from itself to the destination node. Therefore, when a TCP ACK packet is lost, the network layer of the TCP receiver will be notified of the link breakage. It will try to acquire a new path toward the TCP sender. At this time, a sequence number that the TCP receiver expects to receive is piggybacked onto routing control packets. Therefore, since the TCP sender can obtain the sequence number from its network layer, it can avoid spurious retransmissions.

In addition, in the case of duplicate ACK packets, which are needed to invoke the fast retransmit of a lost TCP DATA packet, also being lost, the retransmission should depend on the timeout mechanism at a TCP sender. Therefore, in order to notify the TCP sender of the existence of such loss, and to allow a TCP DATA packet to be retransmitted in a timely way, the S2 scheme piggybacks the occurrence number of duplication onto

Manuscript received September 04, 2006; approved for publication by Jeonghoon Mo, Division III Editor, October 11, 2007.

The authors are with the Kyungpook National University, Daegu, Korea, email: dongkyun@knu.ac.kr, hsyoo@monet.knu.ac.kr.

transmitted duplicate ACK packets. This avoids the dependency on the timeout mechanism in order to retransmit the lost TCP DATA packet.

With the goal of improving TCP performance in MANETs, the proposed schemes will be able to supplement other TCP proposals, which have all ignored ACK loss, to further improve overall performance.

The rest of this paper is organized as follows: In Section II, related work is presented, regarding the improvement of TCP performance in MANETs. TCP-pgy proposed in this paper is introduced along with simulation observations in Section III. In Section IV, the performance of TCP-pgy is evaluated using the ns-2 simulator. Finally, concluding remarks are offered in Section V.

II. RELATED WORK

First, the basic operation of on-demand reactive routing protocols is described in brief. Second, some existing techniques for improving TCP performance in MANETs are explained.

A. On-Demand Reactive Routing Protocols

Most on-demand reactive routing protocols such as AODV, DSR and DYMO consist of two phases: (a) Route discovery and (b) route maintenance. In the route discovery phase, a node (e.g., node A) initiates a route discovery when it wants to send data packets to another node (e.g., node B), but does not have any routing information regarding its destination. Node A floods an RREQ (route request) packet and awaits an RREP (route reply) packet from node B. Through the exchange of RREQ and RREP packets, routing protocols such as AODV and DYMO allow the routing entries for nodes A and B to be created at each intermediate node over a path between the two nodes. DSR, however, enables node A to become aware of the list of intermediate nodes visited toward node B and to transmit packets using the source routing technique. For the reverse path (from node B to A), a new route discovery may be initiated in the DSR protocol.

In the route maintenance phase, when an intermediate node cannot forward a packet to a next-hop node due to a link breakage, the node sends an RERR (route error) packet to the source node of the packet (e.g., packet p). During the propagation of the RERR packet, all buffered packets at each intermediate node are dropped and discarded. When the RERR packet reaches the source node of the packet p, the node will discover a new route between itself and the destination for the packet p.

B. Works to Improve TCP Performance in MANETs

Mechanisms for improving TCP performance in MANETs have been proposed because standard TCP itself, which is tailored for the fixed Internet, cannot distinguish packet loss caused by route failure from that brought about by network congestion. Thus, it tends to drop its throughput by reducing its congestion window size unnecessarily even when packet loss has occurred due to route failure, rather than network congestion. To avoid such throughput degradation, these approaches can be categorized into two classes: (a) Preserving the end-to-end semantics of TCP and (b) violating the end-to-end semantics of TCP and requiring the intervention of intermediate nodes.

In the former approaches [8], [9], the TCP sender and receiver attempt to differentiate between network congestion and route failure without the intervention of intermediate nodes. In the latter schemes [10], [11], however, some messages, such as explicit route failure and route recovery notification, or explicit congestion notification, from any intermediate node that has detected route failure or congestion, are used to allow a TCP sender to be notified of different situations and to manage its congestion window size accordingly. Since the latter approaches require the participation of intermediate nodes, a great deal of modification to the existing routing and transport protocols is unavoidable. All approaches have focused on how to differentiate between route failure and network congestion. In addition, some approaches [13], [14] have been proposed to use the delayed ACK technique efficiently, or to restrict the congestion window to a small size in order to avoid excessive channel contention and interference.

On the other hand, all of the aforementioned approaches have not taken into consideration the fact that such ACK loss affects TCP performance, namely, the importance of TCP ACK packets (see Section III-A). Thus, this paper attempts to address this problem and propose solutions for improving TCP performance by considering this ACK loss.

III. PROPOSED SCHEMES

A. Motivation

Using the ns-2 simulator [7], we first investigated the amount of ACK loss using string topologies with a single TCP flow according to hop distance of the flow, where the distance between direct neighbor nodes, the transmission range of nodes and their interference ranges were 200, 250, and 550 meters, respectively. The first and last nodes in each string topology were set to source and destination nodes for FTP traffic, with a duration of 600 seconds. The underlying routing and MAC protocols were set to AODV and IEEE 802.11b for simulation purposes, respectively. The average results were obtained for 100 runs.

In string topologies without node mobility, packet loss is a result of the inefficiency of the 802.11 MAC protocol, which is caused by capture effects or hidden terminal problem, not link failures. Since the 802.11 MAC protocol, however, is known to perform quite well within a 3-hop distances [13], our simulations used exclude connections with less than 3-hop distance. For the same reason, we also exclude those with less than a 3-hop distance in Section IV-B.

As shown in Fig. 1(a), although a difference in the amount of TCP DATA and TCP ACK losses can be seen, we should note that ACK loss occurs as much as DATA loss, regardless of hop distance. In particular, for cases of more than 10 hops, ACK loss is more significant than DATA loss. Since the TCP ACK packets compete with TCP DATA packets for access to their shared wireless medium, more TCP DATA packets occupy the network than the TCP ACK packets because the size of the TCP ACK packets is much smaller. Thus, the medium is loaded more with TCP DATA packets, which result in the significant amount of ACK loss [16].

Also, the amount of ACK losses using a 7-hop string topology was investigated with multiple flows, not a single flow. In

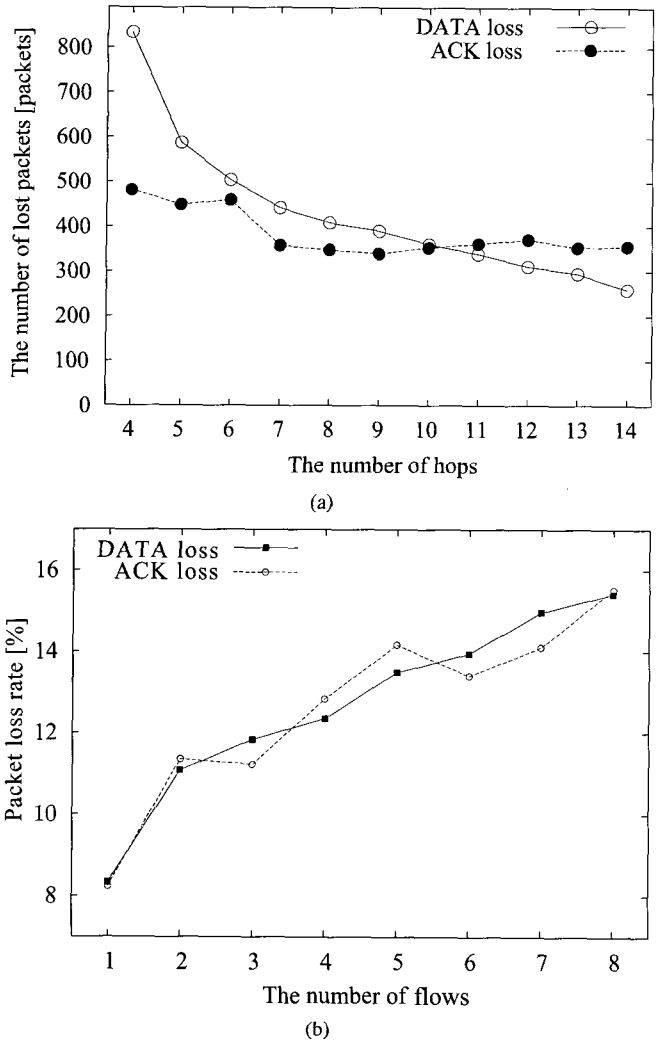

Fig. 1. The loss of TCP packets; (a) packet loss of a single flow, (b) packet loss of multiple flows.

Fig. 1(b), we can see that both DATA and ACK losses are similar and they grow as the number of TCP flows increases. Since multiple TCP flows create much contention and interference, more losses can be observed.

TCP uses the DATA-ACK exchange in order to provide the end-to-end data reliability. If TCP DATA packets have succeeded in reaching a TCP receiver, and their corresponding TCP ACK packets are lost, the absence of these ACK packets will cause a TCP sender to timeout and spuriously retransmit the DATA packets. This results in performance degradation of multiple TCP flows due to contention and collisions.

Also, in most on-demand routing protocols, when an intermediate node fails to forward its ACK packet to a next-hop node, it sends an RERR packet to the TCP receiver. During the propagation of the RERR packet, intermediate nodes discard all ACK packets buffered in their queues. Hence, this will result in excessive spurious retransmissions due to the loss of multiple ACK packets.

In addition, the fast retransmit technique of standard TCP requires three duplicate ACK packets to arrive at the TCP sender for the purpose of retransmitting a lost TCP DATA packet quickly, without relying on the timeout mechanism. Hence, such

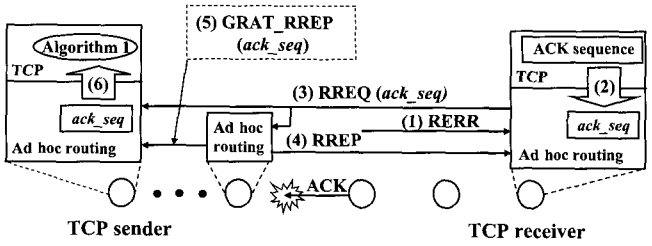

Fig. 2. Route reply generation by an intermediate or destination node.

ACK loss prevents the TCP sender from performing the expected fast retransmit. Consequently, the TCP sender should rely on the timeout mechanism to retransmit lost TCP DATA packets.

The main goal of our proposed TCP, TCP-pgy, therefore, provides solutions to address the above-mentioned problems. In particular, it also attempts to preserve the end-to-end semantics of TCP without the intervention of intermediate nodes like explicit route or congestion-related notification.

B. Detailed Description

We introduce two schemes of TCP-pgy, which attempt to address the two above-mentioned problems caused by ACK loss. It is assumed that TCP operates over on-demand, reactive routing protocols.

B.1 Mechanism to Reduce Spurious Retransmissions (S1 Scheme)

In most on-demand reactive routing protocols, a path is acquired between the source and destination nodes through the exchange of RREQ and RREP packets. During the flooding of an RREQ packet, an RREP packet can be generated in one of two ways: (a) A destination node only will respond to the RREQ packet by unicasting an RREP packet back to the source, or (b) an intermediate node which knows the path toward the destination is allowed to respond to the RREQ packet. DYMO protocol uses method '(a),' and others such as AODV and DSR use method '(b).' Hence, a solution is proposed for each method, respectively.

B.1.a **Route Reply Generation by an Intermediate or Destination Node.** If a TCP ACK packet is lost over a wireless link, a sending node detecting the loss will transmit an RERR packet to the ad hoc routing entity of the TCP receiver. See (1) in Fig. 2. On receiving the RERR packet, the ad hoc routing entity of the TCP receiver will invoke a new route discovery by flooding an RREQ packet. Before flooding the RREQ packet, the ad hoc routing entity obtains the current sequence (i.e., ACK sequence) expected by the TCP receiver. See (2) in Fig. 2. Hence, the ACK sequence is piggybacked onto the RREQ packet. See (3) in Fig.2.

Next, consider the case where an intermediate node knows of the path toward the destination. This intermediate node is allowed to respond to the RREQ packet by unicasting an RREP packet back to the TCP receiver node (if any). See (4) in Fig. 2. Hence, a mechanism is needed to inform the TCP sender of the ACK sequence.

In AODV, the intermediate node generates a gratuitous RREP

(called GRAT_RREP). This GRAT_RREP is sent toward the TCP sender, in order to refresh the state of nodes along the path. Refer to [2] for details. Therefore, the ACK sequence will be piggybacked onto the GRAT_RREP packet. See (5) in Fig. 2. Although DSR does not use such a GRAT_RREP packet, a new routing control packet can be added to the DSR mechanism through minor modification.

For the purpose of piggybacking the ACK sequence, the RREQ and GRAT_RREP packets require an additional field, called *"ack_seq"* (see Section III-C). On receiving the RREQ or GRAT_RREP packet, the routing entity will toss the sequence number to the TCP sender. See (6) in Fig. 2. Hence, a minimal cross-layering approach is taken between network and transport layers. Here, cross-layering involves the exchange of sequence information between routing and transport layers, with minor modifications to protocol stacks. Once informed of the sequence number from the network layer, a TCP sender executes Algorithm 1 (see below).

Suppose that a TCP sender has received an RREQ or GRAT_RREP packet containing an ACK sequence. The *highest_ack* represents the sequence number of the last packet which was acknowledged by its TCP receiver. Hence, packets with a sequence number greater than the *highest_ack* will be buffered in the queue, until a new cumulative ACK packet arrives.

In standard TCP, when a timeout event occurs, the TCP sender retransmits packets whose sequence is larger than the *highest_ack* by using the go-back-N mechanism. In our scheme, however, if the *ack_seq* is larger than the *highest_ack*, the *highest_ack* is updated to the *ack_seq* (see Algorithm 1).

Algorithm 1 Algorithm to avoid spurious retransmissions at a TCP sender

highest_ack : The sequence number of the last packet which have been successfully acknowledged by the TCP receiver
ack_seq : A cumulative ACK sequence number piggybacked on an RREQ message

```
if (RREQ or GRAT_RREP Received) then
  if (ack_seq > highest_ack) then
    highest_ack = ack_seq;
  end if
end if
```

Hence, the standard TCP sender would have retransmitted the packets between the old *highest_ack* and the new *highest_ack* spuriously. However, our scheme enables the TCP sender to avoid these spurious retransmissions by updating the *highest_ack* to the *ack_seq* before the timeout event occurs.

If a timeout occurs before a routing control packet (such as an RREQ or GRAT_RREP) arrives at the TCP sender, our scheme can still result in spurious retransmissions. Here, further research is required.

B.1.b **Route Reply Generation by Destination Node.** Some protocols (for example, DYMO) do not allow an intermediate node to respond to an RREQ packet even though it knows of a partial path toward the destination node. Hence, there are no GRAT_RREP packets. However, our scheme can still be used to reduce spurious retransmissions. We allow the ACK sequence

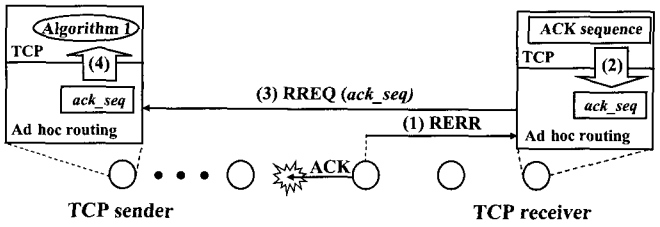

Fig. 3. Route reply generation by destination node.

to be piggybacked onto the flooded RREQ packet. When the RREQ packet arrives at the TCP sender, the TCP sender can be informed of the sequence information and it can then suppress spurious retransmissions (see Fig. 3).

B.2 Mechanism to Address the Impairment of Fast Retransmit (S2 Scheme)

In standard TCP, a TCP sender retransmits each TCP DATA packet if its corresponding ACK packet is not received in a certain period of time called a timeout interval. In particular, a TCP receiver transmits a duplicate ACK packet immediately in the case of an out-of-order TCP DATA packet arriving, implying that a packet loss has occurred. This duplicate ACK packet still has a sequence number of the next expected TCP DATA packet. The TCP fast retransmit technique utilizes these duplicate ACK packets. It aims to retransmit a lost packet more quickly than the timeout-based retransmission. In other words, if three duplicate ACK packets for a lost packet arrive at the TCP sender before a timeout occurs, the lost packet will be retransmitted immediately.

As mentioned before, however, if these duplicate ACK packets are lost, the retransmission of a lost TCP DATA packet ultimately depends on the timeout mechanism due to the lack of the number of ACK packets needed to invoke the fast retransmit. Before retransmitting the lost TCP DATA packet, the TCP sender should stay in an idle state without any other transmission of further TCP DATA packets until the retransmission timer expires.

Hence, although some duplicate ACK packets have been lost, we would like the TCP sender to be informed that the TCP receiver has already transmitted a sufficient number of ACK packets, in order to execute the fast retransmit successfully.

To support this goal, our scheme piggybacks the occurrence number of duplication onto transmitted ACK packets. In this scheme, an additional sequence field is also required (called a *dup_count* field in this paper). For example, the first, second and nth duplicate ACK packets have "1," "2," and "n" as the values of the *dup_count* field, respectively. Also, a TCP receiver maintains a variable, *dupacks* in order to count the occurrence of the duplicate ACK packets. In the case of an in-order TCP DATA packet or a TCP DATA packet filling a gap in the receiver's queue arriving, the *dupacks* is initialized to 0. Otherwise, in the case of an out-of-order TCP DATA packet arriving, the *dupacks* variable is increased by 1. Hence, when the TCP receiver transmits its ACK packet, it fills the *dup_count* field in the ACK packet with a current value of the *dupacks* variable, and transmits the ACK packet (see Algorithm 2).

In standard TCP, a TCP sender decides an execution of

the fast retransmit, depending on the number of the duplicate ACK packets. However, in our scheme, the TCP sender executes the fast retransmit if $dup\_count \geq num\_dupacks$, where $num\_dupacks$ is the number of the duplicate ACK packets required to invoke the fast retransmit. (In standard TCP, $num\_dupacks$ is 3.) See Algorithm 2.

Finally, our scheme also conforms to the fast recovery used in TCP variants in order to compensate for the congestion window after the fast retransmit is performed.

Algorithm 2 Mechanism to address impairment of fast retransmit

dupacks : A variable counting the occurrence of duplicate ACK packets at a TCP receiver
num_dupacks : The number of duplicate ACK packets needed to invoke the fast retransmit (default value is 3)
dup_count : A field in the ACK packet representing the occurrence of duplication
seqno : A cumulative ACK sequence contained in a duplicate ACK packet
cwnd : Congestion window

TCP Receiver :
if (**TCP DATA packet** Received) **then**
  if (In-order TCP DATA packet Received) **then**
    if ($dupacks \neq 0$) **then**
      $dupacks = 0$;
    end if
  else
    $dupacks = dupacks + 1$;
  end if
  send an ACK packet with *dup_count* set to *dupacks*
end if

TCP Sender :
if (Duplicate ACK Received) **then**
  if ($dup\_count >= num\_dupacks$) **then**
    $cwnd = cwnd/2$;
    retransmit the lost packet with $seqno + 1$
  end if
end if

Fig. 4 shows an illustrative example of the operation of the S2 scheme. The TCP sender starts its retransmission timer at time t_1 and transmits six TCP DATA packets consecutively. Assume that all TCP DATA packets except the second TCP DATA packet reached the TCP receiver. In this case, the TCP receiver generates four duplicate ACK packets. Without ACK loss, standard TCP would enable the TCP sender to execute the fast retransmit when the third duplicate ACK packet is received (at t_2). However, in the case that the first and second ACK packets are lost, standard TCP cannot successfully perform the fast retransmit, but retransmits the lost packet after the timer expires (at t_3). Our S2 scheme, however, is able to execute the timely fast retransmit by examining the *dup_count* field in the received packet (at t_2).

C. Discussion on Cross-Layering and Modification of the Packet Format

In TCP-pgy, the S1 scheme requires the interactions between routing and transport layers and the inclusion of the *ack_seq* into the routing control packets (like RREQ or GRAT_RREP). Also, the S2 scheme needs some additional field of *dup_count* in the transport message format.

Compared to other cross-layering approaches to improve TCP performance [15], the S1 scheme is able to achieve this goal with minor modification to protocol stacks. Also, the IETF working group has standardized a Generalized MANET Packet/Message Format (packetBB) [17] that routing protocols conform to. Hence, the *ack_seq* field needed to suppress spurious retransmission can be easily added into the RREQ or GRAT_RREP packet.

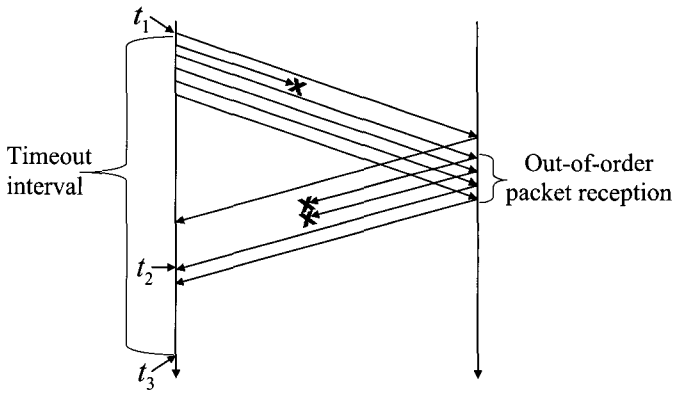

Fig. 4. An illustrative example of the operation of the S2 scheme.

Finally, the *dup_count* field, which is needed to implement the S2 scheme mentioned in Section III-B.2, can be defined by using the option field or the 6-bit reserved field in the TCP header.

IV. PERFORMANCE EVALUATION

We implemented TCP-pgy by modifying standard TCP, TCP-NewReno to have the S1 and S2 schemes and compared TCP-pgy with TCP-NewReno. The S1 and S2 schemes, however, can also be applied to any TCP variants [18]–[20] where the absence of ACK incurs retransmissions and the fast retransmit mechanism is needed.

We evaluated our TCP-pgy performance from two points of view; the intra-flow performance and the inter-flow performance. In particular, since the reduction of spurious retransmissions affects the performance of all cross TCP flows in the network, we also measured the total average throughput of all TCP flows, which is called the average aggregate throughput (denoted by AAT). In this simulation study, three performance metrics of interests are: (a) Throughput of a single flow, (b) the number of retransmissions, and (c) AAT.

A. Simulation Set-Up

Fig. 5 shows network topologies tested in these simulations. For the purpose of measuring the intra-flow performance, string topologies were tested. Next, grid topologies were set up to evaluate the inter-flow performance. Additionally, in order to investigate the effect of node mobility on the inter-flow performance, dynamic networks with random topologies were tested.

In string and grid topologies, the distance between direct neighbor nodes, the transmission range of nodes and their interference ranges were 200, 250, and 550 meters, respectively. In random topologies, 50 nodes were initially located at random positions over a space of 1500 m $\times$ 300 m. The well-known random way point model was used. The pause time was set to 0 and maximum speeds were varied. In addition, TCP senders and receivers were statically placed at the edges of the simu-

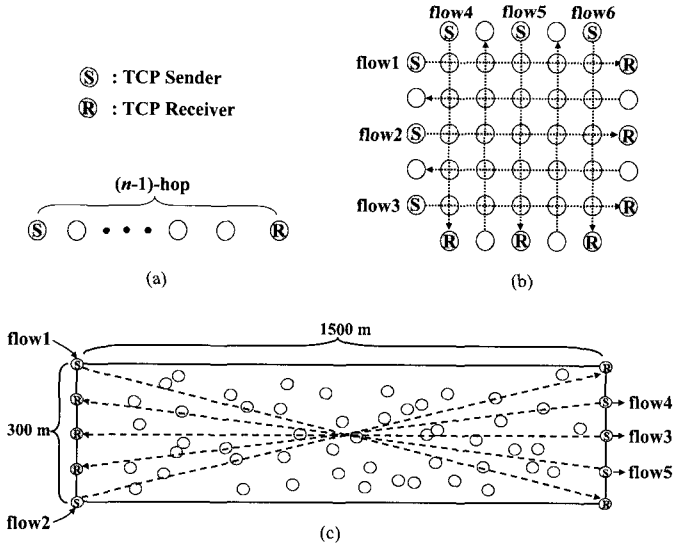

Fig. 5. Simulation topologies; (a) string topology, (b) grid topology, (c) random topology.

Table 1. Simulation parameters.

Parameter Types	Value
Simulation time	600 s
Data Packet Size	1040 Bytes
MAC protocol	IEEE 802.11
Data Rate	2 Mbps
IFQ length	50
Routing protocol	AODV
Source traffic	FTP

lated space [21] (see Fig. 5(c)). Table 1 shows the simulation parameters commonly used in network topologies.

B. Intra-Flow Performance

In this section, we investigated the effect of the S1 and S2 schemes on the performance of a single TCP flow. In the string topology, throughput performance with various hop-distance from 4 to 14 was measured. As shown in Fig. 6, as the hop-distance increases, the throughput decreases because the probability of a packet surviving to be transmitted to the destination is lower due to channel interference or contention. Our schemes, however, improved the throughput performance regardless of hop-distance.

In the static string topology, link failure caused by node mobility cannot be expected. The topology, however, suffers from a situation where two neighbor nodes cannot communicate due to the capture effect of the wireless channel or hidden terminal problem (this situation is called "false link failure") [13], [21].

The ACK loss caused by the false link failure prevents the TCP receiver from sending TCP ACK packets until a routing layer at the destination node acquires a new path toward the source node. Hence, the TCP sender will experience a timeout event, which will bring about spurious retransmissions.

Furthermore, even if the TCP receiver has sent a sufficient number of duplicate TCP ACK packets to execute the fast retransmit, it is possible that such ACK loss makes the TCP sender

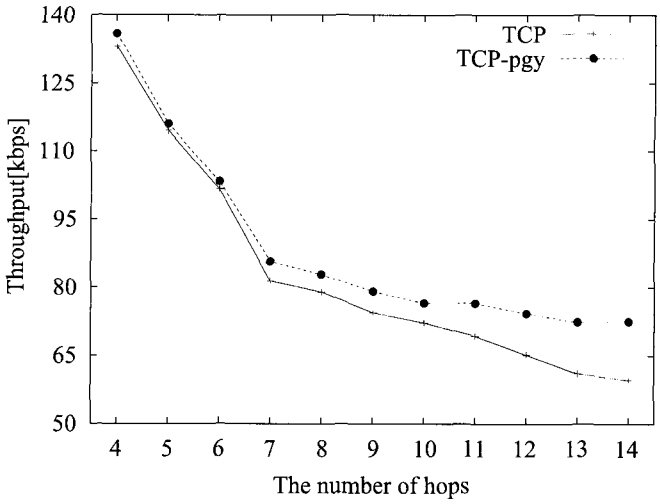

Fig. 6. Throughput in string topologies.

miss the chance to succeed in performing the fast retransmit. It requires the TCP sender to stay in an idle state until the retransmission timer expires. Only after the expiration, the TCP sender will be able to retransmit the lost TCP DATA packet.

In the S2 scheme, when the TCP sender receives a duplicate ACK packet, the TCP sender decides an execution of the fast retransmit, depending on the value of *dup_count* field. Therefore, despite the occurrence of such ACK loss, the S2 scheme allows the timely fast retransmit to be executed. It enables the TCP sender to avoid staying in the idle state until the retransmission timer expires.

In addition, in spite of the existence of ACK loss, the S1 scheme allows the TCP sender to be informed of the latest cumulative ACK sequence through the cross layer technique mentioned before. Hence, in the case that a timeout has occurred at the TCP sender, the spurious retransmissions of the packets whose sequence is less than the ACK sequence updated by the S1 scheme can be avoided.

The spurious retransmission obviously affects the degradation of end-to-end throughput performance in a wireless, multihop network with the constrained spatial reuse. Thus, TCP-pgy achieved throughput improvement by reducing the number of spurious retransmissions, as shown in Fig. 7. In addition, since connections with longer hop-distance need more time to send a packet to the destination and excessive spurious retransmissions require a great deal of wireless channel interference or contention during the journey to the destination, better throughput can be obtained by reducing the number of spurious retransmissions for long hop-distance connections, rather than reducing them for short hop-distance ones. In particular, TCP-pgy improved throughput performance up to 21.6% at a 14-hop distance.

Here, from a trace file produced by the ns-2 simulator, we investigated how each of the S1 and S2 schemes in TCP-pgy could contribute to performance improvements. Fig. 8(a) shows the time sequence diagram where the standard TCP sender performs spurious retransmissions. After TCP ACK packets, with sequence numbers 14 and 15, were lost due to MAC contention, the TCP sender experienced a timeout due to the elapsed route recovery time and it retransmitted the TCP DATA packet with

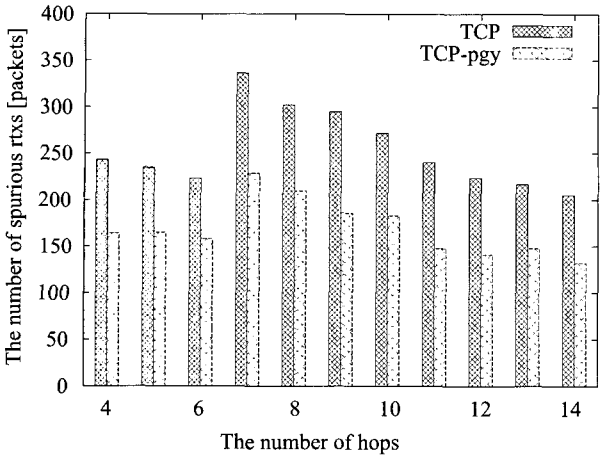

Fig. 7. The number of spurious retransmissions in string topologies.

the sequence number 14. After the route was recovered, the TCP ACK packets (16-19), that had not been transmitted to the TCP sender due to the absence of an available path, arrived at the TCP sender. Thus, according to the go-back-N mechanism in standard TCP, eight TCP DATA packets, that the TCP receiver had previously received, were retransmitted spuriously.

In contrast, Fig. 8(b) shows the time sequence diagram where the S1 scheme is applied. After TCP ACK packets with sequence numbers 14 and 15 were lost, the RREQ packet, which was transmitted by the TCP receiver, piggybacked a cumulative ACK packet with the sequence number 23. Thus, the TCP sender was notified of the sequence number and such a spurious retransmission was avoided.

After a route is recovered, the continuously-arriving ACK packets allow the standard TCP sender to transmit its TCP DATA packets aggressively with an increasing congestion window size, according to the slow start mechanism. Therefore, a packet burst can occur, which causes packet loss due to excessive MAC contention [22]. Hence, in the S1 scheme, new TCP DATA packets are transmitted from the instant when an ACK packet, with a greater sequence number than the ACK sequence piggybacked onto the RREQ packet, arrives.

Next, Fig. 9(b) shows a time-sequence diagram where the S2 scheme is able to avoid a timeout by performing the timely fast retransmit despite the absence of a sufficient number of ACK packets.

After TCP DATA packets with sequence numbers 993, 994, and 995 were lost, the TCP receiver sent three duplicate ACK packets with sequence number 992. The TCP sender, however, succeeded in receiving only one duplicate ACK packet due to ACK losses. In this case, standard TCP had to depend on the timeout event in order to retransmit a lost packet due to the lack of the second and third duplicate ACK packets (around 107.2s in Fig. 9(a). The sequence number, *dup_count* included in the ACK packets, however, allowed the TCP sender to avoid the timeout by executing the fast retransmit immediately (around 106.7s in Fig. 9(b)).

C. Inter-Flow Performance

In the previous section, the performance of a single TCP flow in the network was investigated. In particular, the reduction of

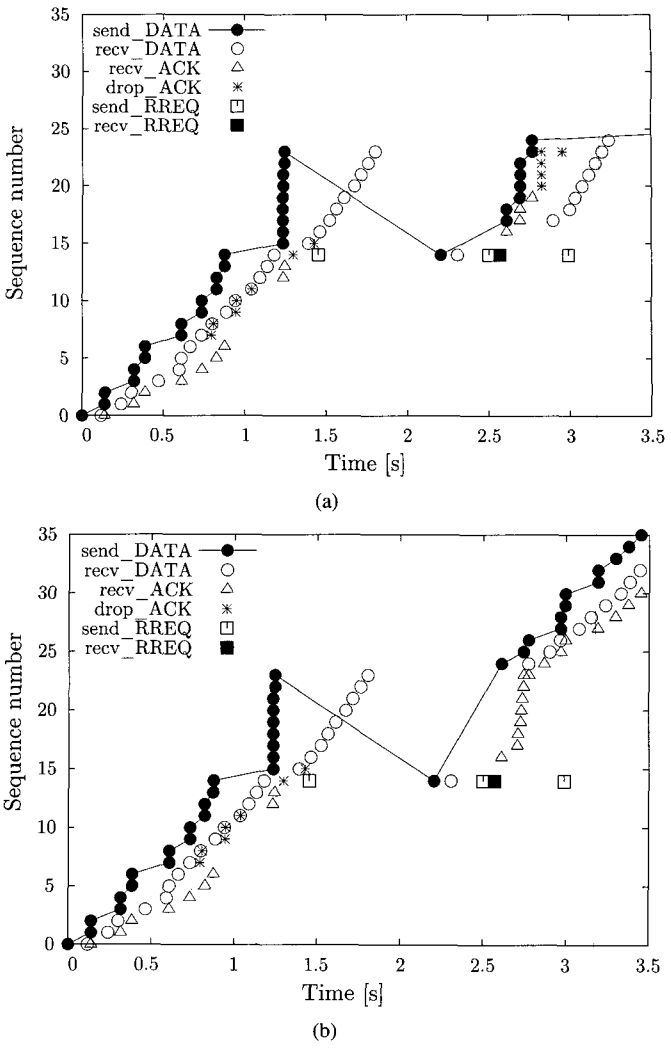

Fig. 8. Time-sequence diagram to illustrate the S1 scheme's contribution; (a) spurious retransmissions in standard TCP, (b) spurious retransmissions in TCP-pgy.

spurious retransmission from a given TCP flow can relieve other cross TCP flows from the inter-flow interference. Hence, using the grid topology, we conducted the performance measurement of the cross TCP flows in this section. In addition, the effect of node mobility on the performance of TCP-pgy was examined, using random topologies.

C.1 Grid Topology

We measured the aggregate throughput over a different number of TCP flows in the grid network topology. Various cross traffic scenarios of two flows (flow 1 and flow 2), four flows (flow 1 to flow 4), six flows (flow 1 to flow 6), eight flows (flow 1 to flow 8), and ten flows (flow 1 to flow 10) were simulated (see Fig .5(b)).

With more cross traffic created as the number of TCP flows increases, greater interferences, and ACK losses occur due to false link failures. As the number of cross TCP flows grows in the network, AATs of both TCP-pgy and standard TCP increase, as shown in Fig. 10. However, compared to standard TCP, TCP-pgy achieves higher AAT due to the reduction of spurious retransmissions through the S1 scheme and the reduction

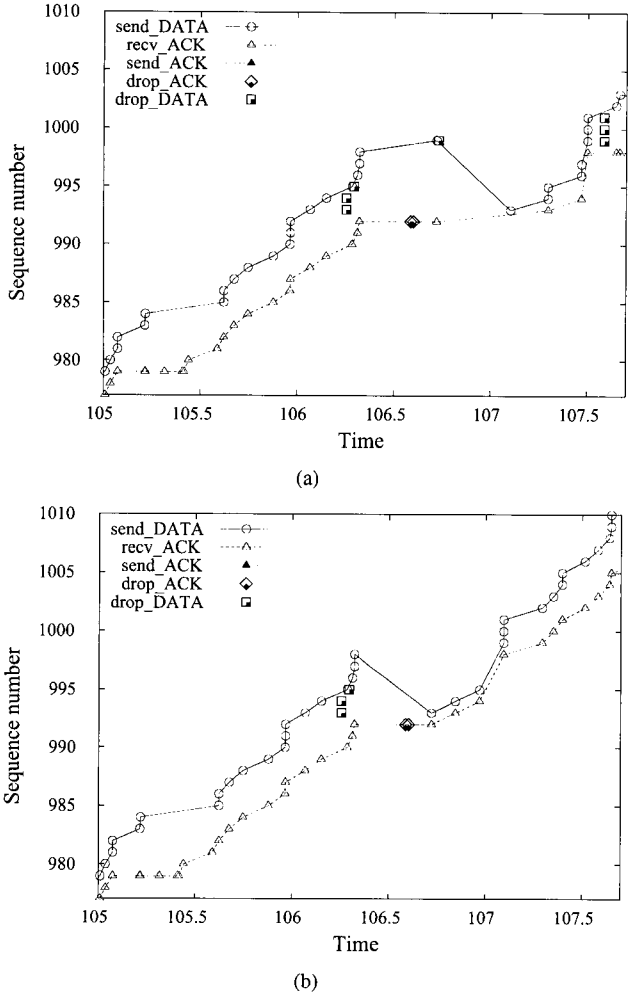

Fig. 9. Time-sequence diagram to illustrate the S2 scheme's contribution; (a) retransmission in standard TCP, (b) retransmission in TCP-pgy.

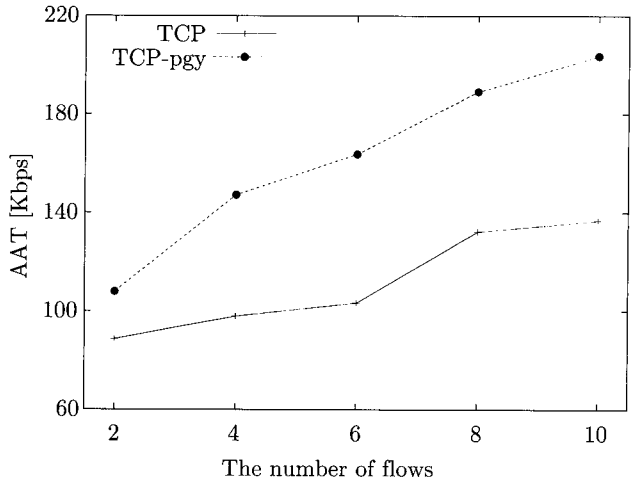

Fig. 10. Comparison of average aggregate throughput in grid topologies. AAT: Average aggregate throughput.

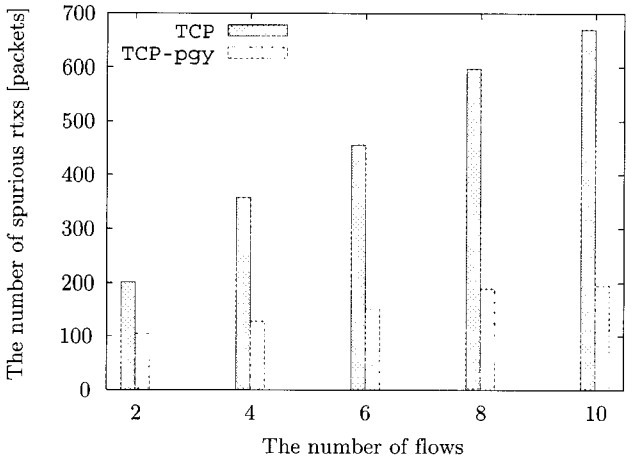

Fig. 11. Comparison of the number of spurious retransmissions in grid topologies.

of the period of idle time spent by the TCP sender through the S2 scheme. In particular, TCP-pgy achieved as high as 58.4% performance improvement in the six-flow case.

With reference to Fig. 11, in standard TCP, when the number of TCP flows increases, TCP senders experience a larger number of timeouts. This results in significant spurious retransmissions. However, TCP-pgy yields a lower number of spurious retransmissions for all cross traffic scenarios. As the number of TCP flows is increased from 2 to 10, the number of spurious retransmissions increases by 232.8% and 87.3% for standard TCP and TCP-pgy, respectively. In particular, TCP-pgy reduces the number of spurious retransmissions by 70.8% in the ten-flow case, when compared to standard TCP.

C.2 Random MANET Topology

In this simulation, random topologies with node mobility are considered with evaluation of TCP-pgy according to speed of nodes and the number of TCP flows. Similar to the previous simulations using grid topologies, two cross traffic scenarios of three-flow (flow 1, flow 2, and flow 3) and five-flow (flow 1 to flow 5) cases were tested (see Fig. 5(c)).

From Fig. 12(a) and 12(b), as mobility increases in both cross traffic scenarios, each AAT of both TCPs decreases. This is because too much time is spent in recovering from frequent route breakage. However, TCP-pgy shows better performance than standard TCP regardless of node mobility. Similar to static networks, such ACK loss occurs frequently in dynamic networks, which is caused by link failure. Hence, as mentioned before, the S2 scheme contributes to reducing spurious retransmissions of the TCP sender and the idle time of the TCP sender can be reduced through the S2 scheme, which enables a lost TCP packet to be retransmitted in a timely manner. The reduced number of spurious retransmissions and the reduced amount of idle time at the TCP sender are in proportion to throughput improvements.

From the bar graph as shown in Fig. 12(a), we observe that as node mobility increases, the frequency of spurious retransmissions decreases. This is because high mobility causes greater "actual link failures" than "false link failures" [15]. Hence, due to TCP DATA losses, a TCP sender will finally experience timeouts and retransmit the lost TCP DATA packets. Although these retransmissions are not spurious, they are indispensable. If the difference in the number of spurious retransmissions between TCP-pgy and standard TCP is low, TCP-pgy cannot show significant throughput improvements.

Next, consider, however, the case with five cross TCP flows.

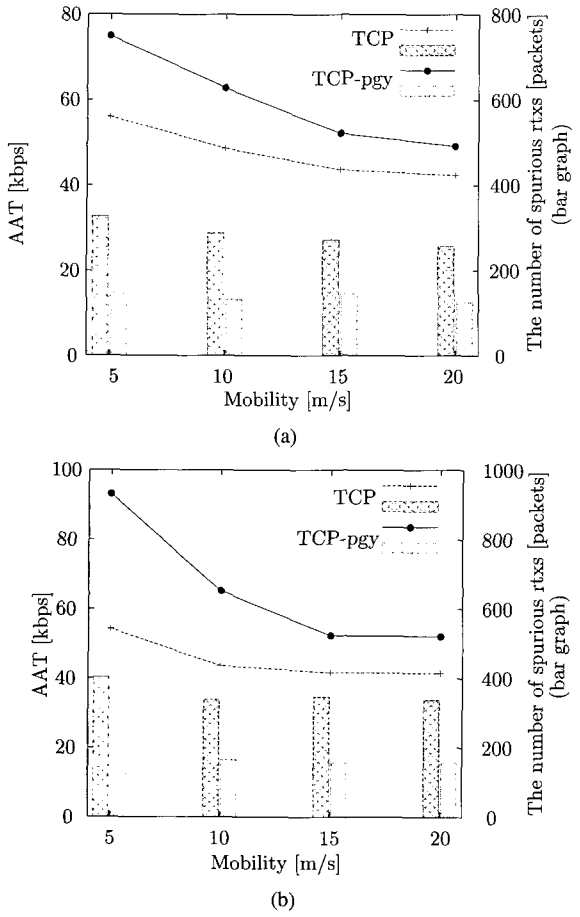

Fig. 12. Comparison of average aggregate throughput of multiple TCP flows in random topologies. AAT: Average aggregate throughput; (a) three flows (flow1, flow2, and flow3), (b) five flows (flow1~flow5).

As shown in Fig. 12(b), as the number of cross TCP flows is increased, TCP-pgy achieves significant performance improvements (in terms of aggregate throughput of all TCP flows), when compared to the three-flow case. Since much cross traffic increases channel contention and interferences (regardless of node mobility), the occurrence of spurious retransmissions due to false link failures also increase as shown in Fig. 12(b). Hence, the reduction of such spurious retransmissions in TCP-pgy contributes to improving TCP throughput performance. In particular, TCP-pgy achieved as high as 71.5% throughput performance improvement at the mobility of 5 m/s.

Also, the TCP AIMD (Additive Increase Multiplicative Decrease) mechanism allows the TCP sender to transmit its TCP DATA packets by increasing its congestion window aggressively until it detects packet loss. Hence, in static networks without any mobility-driven link failure, packet loss mainly occurs due to excessive MAC contention which is caused by network overload. In other words, since the TCP sender has already increased its congestion window, consecutive TCP DATA packets must have been transmitted from the TCP sender. Thus, a sufficient number of TCP ACK packets are transmitted to the TCP receiver, so that the fast retransmit for a lost packet can be executed despite the occurrence of packet loss. In dynamic networks with node mobility, however, packet loss caused by route breakage due to node mobility, prevents the TCP sender from allowing a large congestion window to occur. Thus, the TCP sender cannot expect a sufficient number of TCP ACK packets.

In accordance with [23], if a TCP sender uses a small congestion window, the TCP sender should perform the second duplicate ACK-based fast retransmit, not the third duplicate ACK. Using a small congestion window implies the small number of TCP DATA packets which the TCP sender can transmit consecutively. It indicates that when a loss of a TCP DATA packet occurs, the number of TCP DATA packets, which will arrive at a TCP receiver consecutively after the lost packet, is small. It results in the absence of a sufficient number of ACK packets to invoke the fast retransmit. However, the S2 scheme decides an execution of the fast retransmit, depending on the *dup_count* field in a received duplicate ACK packet, not the number of the received duplicate ACK packets. Consequently, the S2 scheme can accomplish the timely fast retransmit, despite the existence of such ACK loss. In this sense, the S2 scheme contributes to greater performance improvement in dynamic networks than in static networks.

V. CONCLUSION

This paper introduced a new TCP with two piggybacking schemes (called TCP-pgy) to resolve problems associated with TCP ACK loss in order to improve TCP performance over reactive routing protocols. First, in order to reduce the number of spurious retransmissions which occur if a retransmission timer expires due to the loss of an ACK packet, a cumulative ACK sequence number of the TCP connection is piggybacked onto routing control packets, from which the TCP sender can be notified of the sequence and can avoid spurious retransmission. The sequence number is included in the routing control packets by using the packetBB structure, as defined by IETF. Second, with an addition of a counter field in duplicate ACK packets transmitted by the TCP receiver, we also avoided performance degradation caused by impairment of the fast retransmit technique, due to the absence of a sufficient number of duplicate ACK packets. The field is defined using the TCP option or a 6-bit reserved field.

Through extensive simulations using the ns-2 simulator, we observed significant throughput performance improvements in static networks and dynamic networks. These two schemes can be applied to any TCP variant which uses retransmission mechanisms such as a timeout or the fast retransmit, based on the reception of the ACK packets. The schemes will be able to contribute to overall TCP performance improvements, together with other TCP proposals, in order to cope with other factors causing TCP performance degradation.

REFERENCES

[1] Internet Engineering Task Force: MANET working group charter. [Online]. Available: http://www.ietf.org/html.charters/manet-charter.html

[2] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," RFC 3561, July 2003.

[3] T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, "Optimized link state routing protocol (OLSR)," IETF RFC 3626, Oct. 2003.

[4] I. Chakeres and C. Perkins,"Dynamic MANET on-demand (DYMO) routing," IETF Internet-Draft, draft-ietf-manet-dymo-05.txt, June 2006.

[5] Z.J. Haas, M.R Pearlman, P. Samar, "The zone routing protocol (ZRP) for

ad hoc networks," IETF Internet Draft, draft-ietf-manet-zone-zrp-04.txt, July 2002.
[6] D. B. Johnson, D. A. Maltz, and Y.-C. Hu,"Dynamic source routing protocol for mobile ad hoc networks (DSR)," Internet Draft (work-in-progress), draft-ietf-manetdsr-10.txt, July 2004.
[7] K. Fall and K. Varadhan, "ns notes and documents," The VINT Project, UC Berkeley, LBL, USC/ISI, and Zerox PARC.
[8] T. Dyer and R. Boppana, "A comparison of TCP performance over three routing protocols for mobile ad hoc networks," in *Proc. ACM MOBIHOC 2001*, Long Beach, CA, USA, Oct. 2001, pp. 56–66.
[9] F. Wang and Y. Zhang, "Improving TCP performance over mobile ad hoc networks with out-of-order detection and response," in *Proc. ACM MOBIHOC 2002*, Lausanne, Switzerland, June 2002, pp. 217–225.
[10] D. Kim, C. Toh, and Y. Choi, "TCP-BuS: Improving TCP performance in wireless ad hoc networks," *J. Commun. Networks*, vol. 3, no. 2, pp. 175–186, June 2001.
[11] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 7, pp. 1300–1315, July 2001.
[12] K. Nahm, A. Helmy, and C. J. Kuo, "TCP over multihop 802.11 networks: Issues and performance enhancement," in *Proc. ACM MobiHoc 2005*, Urbana-Chanpaign, IL, May 2005.
[13] R. de Oliveira, T. Braun, "A dynamic adaptive acknowledgment strategy for TCP over multihop wireless networks," in *Proc. IEEE INFOCOM 2005*, Miami, USA, vol. 3, Mar. 2005, pp. 1863–1874.
[14] K. Chen, Y. Xue, and K. Nahrstedt, "On setting tcp's congestion window limit in mobile ad hoc networks," in *Proc. IEEE ICC 2003*, Anchorage, Alaska, May 2003.
[15] A. A. Hanball, E. Altman, and P. Nain, "A survey of TCP over ad hoc networks," *IEEE Commun. Surveys Tuts.*, vol. 7, no. 3, no. 3, pp. 22–36, 2005.
[16] R. de Oliveira, T. Braun, "TCP in wireless mobile ad hoc networks," Tech. Report IAM-02-003, university of Bern, Switzerland, July 2002.
[17] T. Clausen, C. Dearlove, J. Dean and C. Adjih, "Generalized MANET Packet/Message Format," IETF Internet-Draft, draft-ietf-manet-packetbb-02.txt, July 2006.
[18] L. S. Brakmo, L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE J. Sel. Areas Commun.*, vol. 13, pp. 1465–1480, Oct. 1995.
[19] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in Proc. ACM MobiCom, 2001, pp. 287–297.
[20] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP performance," *IEEE Trans. Mobile Comput.*, vol. 4, no. 2, pp. 209–221, 2005.
[21] F. Klemm, Z. Ye, S. V. Krishnamurthy, S. K. Tripathi, "Improving TCP performance in ad hoc networks using signal strength based link management," *Elsevier Ad Hoc Networks J.*, vol. 3, no. 2, pp. 175–186, Jan. 2005.
[22] S. M. ElRakabawy, A. Klemm, C. Lindemann:, "TCP with adaptive pacing for multihop wireless networks," in *Proc. ACM MobiHoc 2005*, Urbana-Chanpaign, IL, May 2005.
[23] S. floyd, "Limited slow-start for tcp with large congestion windows," RFC 3742, IETF Network Working Group. Mar. 2004.

Dongkyun Kim is a professor with the Department of Computer Engineering, Kyungpook National University, Daegu, Korea. He received the B.S. degree at Kyungpook National University. He also obtained the M.S. and Ph.D. degrees at Seoul National University, Korea. He was a visiting researcher at Georgia Institute of Technology. He also performed a postdoctorate program at University of California Santa Cruz. He has been a TPC member of several IEEE conferences. He received the best paper award from the Korean Federation of Science and Technology Societies, 2002. His research interests are ad hoc network, sensor network and wireless LAN, etc.

Hongseok Yoo received the B.S. and M.S. degrees in the School of Computer Engineering, Kyungpook National University, Daegu, Korea, in 2005 and 2007, respectively. He performed an internship at Daegu Gyeongbuk Institute of Science and Technology (DGIST), Daegu, Korea in 2007. He is currently a Ph.D student with the Graduate School of Electrical Engineering and Computer Science, Kyungpook National University, Korea. His research interests include protocol design and implementation in mobile ad hoc networks.