An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges

Ka-Cheong Leung, *Member*, *IEEE*, Victor O.K. Li, *Fellow*, *IEEE*, and Daiqin Yang, *Student Member*, *IEEE*

Abstract—Transmission Control Protocol (TCP) is the most popular transport layer protocol for the Internet. Due to various reasons, such as multipath routing, route fluttering, and retransmissions, packets belonging to the same flow may arrive out of order at a destination. Such packet reordering violates the design principles of some traffic control mechanisms in TCP and, thus, poses performance problems. In this paper, we provide a comprehensive and in-depth survey on recent research on packet reordering in TCP. The causes and problems for packet reordering are discussed. Various representative algorithms are examined and compared by computer simulations. The ported program codes and simulation scripts are available for download. Some open questions are discussed to stimulate further research in this area.

Index Terms—Computer simulations of TCP, congestion control, flow control, Internet, packet reordering, Transmission Control Protocol (TCP).

1 Introduction

The Internet provides a convenient and cost-effective communication platform for electronic commerce, education, and entertainment. The success of the Internet arises from its capabilities to support survivable, robust, and reliable end-to-end data transfer services for a myriad of applications running over a set of end-systems. The Internet is originated from the Advanced Research Projects Agency Network (ARPANET) designed to support survivable military communications. Currently, Transmission Control Protocol (TCP) [1] is the most popular transport layer protocol for point-to-point, connection-oriented, in-order, reliable data transfer in the Internet. TCP is the de facto standard for Internet-based commercial communication networks.

1.1 Introduction to TCP

TCP is a byte-stream protocol; its flow control and acknowledgement are based on byte number rather than packet number [2]. However, the smallest unit of data transmitted in the Internet is a data segment or packet, each identified by a data octet number. When a destination receives a data segment, it acknowledges the receipt of the segment by issuing an acknowledgement (ACK) with the next expected data octet number. The time elapsed between when a data segment is sent and when an ACK for the segment is received is known as the round-trip time (RTT) of the communication between the source and the destination, which is the sum of the propagation, transmission, queueing, and processing delays at each hop of the communication, and the time taken to process a received segment and generate an ACK for the segment at the destination.

The flow control mechanism used by TCP is a credit allocation scheme. To avoid overwhelming its buffer space, a destination advertises to the associated source the size of a window (advertised window) which indicates the number of data bytes beyond the acknowledged data the source can send to the destination. This information is included in the header of each TCP (data or control) segment sent to the source. Suppose the source knows that, based on ACK(s) received, Byte x is the last data byte received by the destination. The source can send data up to Byte $x + W$, where W is the size of the advertised window. The scenario of the source's sequence number space is exhibited in Fig. 1.

1.2 Congestion Control Operations of TCP

To achieve good performance, it is necessary to control network congestion so that the number of packets within the Internet is below the level at which the network performance drops significantly. Various congestion control measures [3] have been implemented in TCP to limit the sending rate of data entering the Internet by regulating the size of the congestion window *cwnd*, the number of unacknowledged segments allowed to be sent. These measures include slow start, congestion avoidance, fast retransmit, and fast recovery. When a new connection is established, TCP sets *cwnd* to one. In slow start, the value of *cwnd* is incremented by one each time an ACK is received until it reaches the slow start threshold, *ssthresh*.

• *The authors are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong (SAR), China. E-mail: {kcleung, vli, dqyang}@eee.hku.hk.*

Manuscript received 9 Aug. 2005; revised 13 May 2006; accepted 31 May 2006; published online 9 Jan. 2007.
Recommended for acceptance by J. Hou.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0362-0805.
Digital Object Identifier no. 10.1109/TPDS.2007.1011.

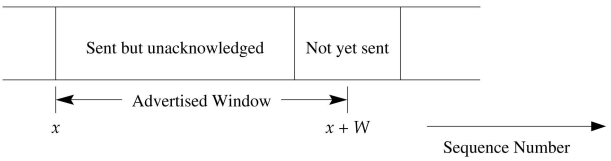

Fig. 1. An illustration of the source sequence number space and advertised window.

TCP uses segment loss as an indicator of network congestion. To characterize a segment as being lost in transit, a source has to wait long enough without receiving an ACK for the segment. Therefore, a retransmission timer is associated with each transmitted segment and a timer timeout signals a segment loss. The retransmission timeout period (RTO) is determined by the sum of the smoothed exponentially weighted moving average and a multiple of the mean deviation of RTT [4]. When a timeout occurs, *ssthresh* is set to half of the amount of outstanding data sent to the network. The slow start process is performed starting with *cwnd* equal to one until *cwnd* approaches *ssthresh*. The congestion avoidance phase is then carried out where *cwnd* is increased by one for each RTT.

When the data octet number of an arriving segment is greater than the expected one, the destination finds a gap in the sequence number space (known as a sequence hole) and thus immediately sends out a duplicate ACK, i.e., an ACK with the same next expected data octet number in the cumulative acknowledgement field,[1] to the source. If the communication channel is an in-order channel, the reception of a duplicate ACK implies the loss of a segment. When the source receives three duplicate ACKs, fast retransmit is triggered such that the inferred loss segment is retransmitted before the expiration of the retransmission timer.

Fast recovery works as a companion of fast retransmit. A fast retransmission suggests the presence of mild network congestion. *ssthresh* is set to half of the amount of outstanding data sent to the network. Since the reception of a duplicate ACK indicates the departure of a segment from the network, *cwnd* is set to the sum of *ssthresh* and the number of duplicate ACKs received. When an ACK for a new segment arrives, *cwnd* is reset to *ssthresh* and then congestion avoidance takes place.

Packet reordering refers to the network behavior where the relative order of some packets in the same flow[2] is altered when these packets are transported in the network. In other words, the receiving order of a flow of packets (or segments) differs from its sending order. Recent studies [5], [6] show that packet reordering is not a rare event. The presence of persistent and substantial packet reordering violates the in-order or near in-order channel assumption made in the design principles of some traffic control mechanisms in TCP. This can result in a substantial degradation in application throughput and network performance [7].

1. A cumulative ACK is an ACK that uses the cumulative ACK field in the TCP header to acknowledge all in-sequence data received by the destination.

2. The term "flow" is used in a very general manner here. A flow can correspond to a stream of packets originating from one end system and departing at another. On the other hand, a flow can be a stream of packets arriving at and leaving from a switch buffer.

1.3 Organization of the Paper

The objective of this paper is to present to the readers a clear overview of TCP for packet reordering. The rest of the paper is organized as follows: In Section 2, we characterize the causes of packet reordering. Section 3 identifies the problems packet reordering introduces on TCP. Sections 4, 5, and 6 provide a taxonomy and a survey of existing solutions to the problems. A performance study of the surveyed algorithms is presented in Section 7. Section 8 explores some open research issues and challenges. Finally, we summarize and conclude the paper in Section 9.

2 Causes of Packet Reordering

There are five major causes of packet reordering: packet-level multipath routing, route fluttering, inherent parallelism in modern high-speed routers, link-layer retransmissions, and router forwarding lulls.

- Packet-Level Multipath Routing: Multipath routing [8], [9] is a load balancing traffic engineering technique to spread the traffic load across the network in order to alleviate network congestion. It has been shown [10], [11] that multipath routing balances the load significantly better than single-path routing and provides better performance in congestion and capacity over wired/wireless networks. Packet-level multipath routing allows packets of the same traffic flow to be forwarded over multiple routes to a destination so as to achieve load balancing in packet-switching networks. This functionality is supported by overlay networks. However, these packets may be reordered on arrival at the destination due to the differences in path delays.
- Route Fluttering: Routing fluttering is a network phenomenon in which the forwarding path to a certain destination oscillates among a set of available routes to that destination. This results from route instability due to shaky links, and heavy loads or bursty traffic where the link cost metrics used in the routing algorithms are related to delays or congestion experienced over the network links. This also results in topological changes in the wireless environment. For example, mobile ad hoc networks are associated with no fixed infrastructure and every mobile node can be a source, a destination, or a router. Similar to packet-level multipath routing, route fluttering causes packets to be forwarded on different paths and arrive at a destination out of order.
- Inherent Parallelism in Modern High-Speed Routers: Modern routers support packet striping so that packets of the same traffic flow can be forwarded over lower-capacity, but much cheaper multiple parallel links connecting to the next-hop router for that flow. To switch packets at high speed, this router is generally work conserving so that its outgoing ports connecting to a certain next-hop router are idle only when there is no outstanding packets to be forwarded to that router. Since packets may be of different sizes and the links can be of different bandwidths, packets may take dramatically different times to transmit, and hence arrive at the neighboring router in a different order from they are sent.

The use of multiple inexpensive application specific integrated circuits (ASICs) for Internet Protocol (IP) forwarding gives rise to an opportunity to speed up the port forwarding speed. Thus, even when there is only a single outgoing port connecting to the next-hop router, packets processed by different ASICs can be reordered [5].

- Link-Layer Retransmissions: Link-layer retransmission mechanisms [12] have been proposed to efficiently recover transmission losses due to high channel error rates in wireless networks. Such retransmitted packets are sent only after the losses are detected. These packets may then be interspersed with other packets belonging to the same traffic flow.
- Router Forwarding Lulls: Some routers can pause its forwarding activity for buffered packets when it processes a routing update. These buffered packets are interspersed with new arrivals, thus causing packet reordering [6].

To summarize, there is a correlation between the causes and characteristics of packet reordering. Packet-level multipath routing and router fluttering induce packet reordering due to the differences in path delays. The inherent parallelism in modern high-speed routers produces packet reordering because of the differences in queueing and/or transmission times. Link-layer retransmissions incur packet reordering since retransmitted packets are associated with an additional round-trip time over a link. Router forwarding lulls induce packet reordering due to interspersion of buffered packets with new arrivals when processing a routing update. Although they may all lead to persistent and substantial packet reordering, the different causes of packet reordering pose a challenge in the design of an efficient and effective solution that is able to solve different types of packet reordering.

3 Impact of Packet Reordering on TCP

TCP relies on the use of a cumulative ACK to announce the receipt of segment(s). The pace at which a source receives ACKs drives how fast it can inject TCP segments into the network and its associated destination. With persistent and substantial packet reordering, TCP spuriously retransmits segments, keeps its congestion window unnecessarily small, loses its ACK-clocking, and understates the estimated RTT (and, thus, RTO) [5]. These will be described in detail next:

- Spurious Segment Retransmissions: Packet reordering causes the starting data octet number of some arriving segments to differ from the ones expected by a destination. In other words, the destination finds a sequence hole upon segment reception. It then generates duplicate ACKs and sends them to its associated source. When the source receives three such duplicate ACKs consecutively, an inferred loss segment (although there is actually no loss) is retransmitted. Persistent and substantial packet reordering often causes some TCP segments to be retransmitted spuriously and unnecessarily, leading to classical congestion collapse [13].
- Keeping Congestion Window Unnecessarily Small: Fast recovery is always triggered with fast retransmit. A spurious fast retransmission not only generates additional yet unnecessary workload to the network and a destination, but also halves the congestion window. Thus, the congestion window is kept small relative to the available bandwidth of its transmission path, with persistent and substantial packet reordering.
- Loss of ACK-Clocking: Packet reordering causes not only data segments, but also ACKs to arrive at a destination out of order. The former phenomenon is called forward-path reordering, while the latter is known as reverse-path reordering [5]. An illustration of forward-path reordering and reverse-path reordering is shown in Fig. 2. Suppose segments are sent from the source in the order S1, S2, S3, but Segment S1 arrives after Segment S2 at the destination. This represents a forward-path reordering. ACK A1 arrives after ACKs A2 and A3 at the source. This depicts a reverse-path reordering.

  ACK-clocking or self clocking refers to the property that the receiver can generate ACKs no faster than data segments can get through the network [14]. For forward-path reordering, an ACK for several new segments, which follows a number of duplicate ACKs, can in turn allow a source to inject several pending segments into the networks. Even when there is no data segment being reordered, disordered ACKs lead to a source transmitting several segments together rather than one or two segments per ACK. This causes loss of its ACK-clocking and far more bursty traffic, which may lead to transient network congestion and congestion collapse from undelivered packets [13].
- Understating Estimated RTT and RTO: Whenever a segment is retransmitted, a source cannot determine whether a received ACK is for the first transmission or the retransmission of the segment. Karn's algorithm [15] alleviates the problem by discarding all measured RTT samples until an ACK acknowledges a segment that has not been retransmitted. Since a fast retransmission is likely to correspond to a segment that experiences a longer path delay, the use of Karn's algorithm results in a sampling bias against long RTT samples [16]. With persistent and substantial packet reordering, these samples would be discarded. The estimated RTT and RTO are therefore understated.

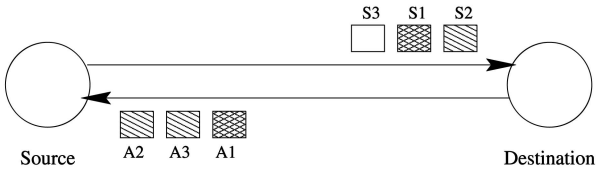

Fig. 2. An illustration of forward-path reordering and reverse-path reordering.

4 Taxonomy of Reordering Solutions for TCP

For an in-order network environment, a destination receives segments in the same order as they are sent. The destination

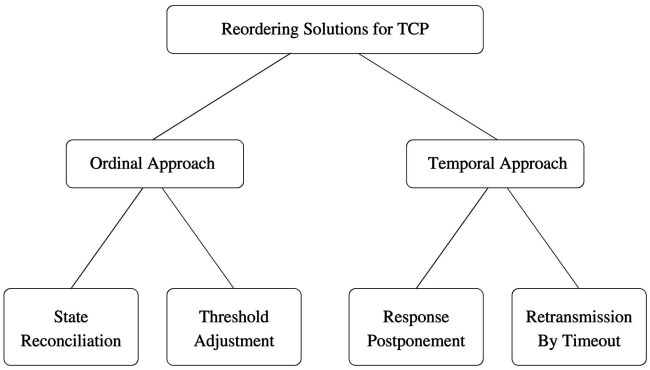

Fig. 3. The taxonomy of reordering solutions for TCP.

realizes the occurrence of a dropped packet when an unexpected segment arrives. It can then embed the information of missing segments into the subsequent ACKs to a source so that the source can retransmit the lost segments to the destination.

However, a general network environment can reorder packets, in addition to dropping packets. Even when an arriving segment is a newly received segment and it is not the expected one, a destination cannot determine whether the expected segment has been dropped in the network or it is simply reordered. The source and the destination have to gather additional information from the network to distinguish between these two possibilities.

In the following two sections, we survey the solutions proposed to date for packet reordering in TCP. The discussed algorithms can be implemented in a TCP client to generate congestion responses when packet reordering occurs, and/or in a participating router to report packet dropping information to a TCP client. The taxonomy used in our survey is depicted in Fig. 3. We categorize the reordering solutions for TCP into two different strategies, namely, the ordinal approach and the temporal approach. The ordinal approach is a collection of methods that process the ordering information of segments and ACKs received so as to infer and generate more appropriate congestion response when packet reordering occurs. The ordinal approach can be further divided into the algorithms for state reconciliation, and the algorithms for threshold adjustment. For state reconciliation, a TCP client finds out which segment or ACK has been reordered. It then reacts, say, by recovering past congestion responses and/or disabling future congestion responses for a time period. For threshold adjustment, a TCP client searches for an appropriate duplicate acknowledgement threshold, *dupthresh*, to proactively avoid, whenever possible, triggering a spurious fast retransmission and fast recovery as well as a retransmission timeout at the same time.

The temporal approach represents a group of techniques that proactively avoid triggering spurious congestion responses by deferring them for a time period. These responses will be carried out only when the corresponding timer fires. The temporal approach can be further classified into the algorithms for response postponement and the algorithms for retransmission by timeout. For response postponement, a TCP client delays triggering a congestion response for a time period. During the time period, the response will be cancelled whenever an event initiating the response is inferred not to be caused by congestion. For retransmission by timeout, a TCP client generates an appropriate congestion response only when a certain timer expires.

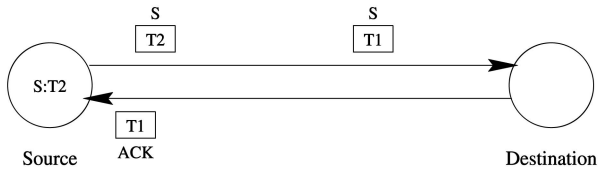

Fig. 4. An illustration of the Eifel algorithm.

5 Ordinal Approach

We describe the existing algorithms that can be classified as an ordinal approach in this section. We present these algorithms, compare them, and discuss their strengths and weaknesses. Refer to Table 2 for an overview of the surveyed schemes. Algorithms are referred to by the authors' last names unless they have been named.

5.1 State Reconciliation

5.1.1 Eifel Algorithm

Ludwig and Katz proposed the Eifel algorithm [17] to eliminate the retransmission ambiguity and solve the performance problems caused by spurious retransmissions. A source uses the TCP timestamp option [18] to insert the current timestamp into the header of each outgoing segment to a destination. When the destination sends ACKs, it includes the corresponding timestamps into the ACKs. To eliminate the retransmission ambiguity, the source always stores the timestamp of the first retransmission of a segment. When the first ACK for the retransmitted segment arrives, the source compares the timestamp of that ACK with the stored timestamp. If the stored timestamp is greater, the retransmission is considered spurious.

Fig. 4 illustrates how the Eifel algorithm works. When the source sends Segment S the first time at Time T1, it inserts the current timestamp T1 into the header of the segment. At Time T2, the source initiates a congestion response by retransmitting Segment S. The original segment differs with the retransmitted one as the latter one contains a timestamp T2 instead of T1. When the destination receives the original Segment S first, it sends an ACK with the timestamp of the segment, i.e., T1. When the ACK for the segment arrives, the source finds that the echoed timestamp, T1, is smaller than the stored one, T2. The retransmission is hence identified as spurious.

To solve the problems caused by spurious retransmissions, a source also stores the current values of the slow start threshold, *ssthresh*, and the size of the congestion window, *cwnd*, when a segment is retransmitted the first time. When a detected spurious retransmission has resulted in a single retransmission of the oldest outstanding segment, the source restores *ssthresh* and *cwnd* to the stored values. It has been shown [17] that this technique is simple and effective in improving TCP performance with forward-path reordering. However, bursts of TCP segments may be

injected into the network when the state is restored. Besides, the scheme does not work when the original and retransmitted segments are reordered.

5.1.2 TCP-DOOR

Wang and Zhang developed TCP with detection of out-of-order and response (TCP-DOOR) [19], which can be considered as an extension of [17]. The out-of-order events are deemed to imply route changes in the networks, which happen frequently in mobile ad hoc networks. The TCP packet sequence number and ACK duplication sequence number, or current timestamps, are inserted into each data and ACK segment, respectively, to detect reordered data and ACK packets. When out-of-order events are detected, a source can either temporarily disable congestion control or perform recovery during congestion avoidance. By temporarily disabling congestion control, the source will maintain its state variable constant for a time period, say t_1 seconds, after detecting an out-of-order event. By instant recovery during congestion avoidance, the source recovers immediately to the state before the congestion response, which has been invoked within t_2 seconds ago.

However, TCP-DOOR does not distinguish between forward-path reordering or reverse-path reordering. The responses are suitable to alleviate some performance problems caused by forward-path reordering. They do not help reduce bursty traffic, and in fact exaggerate network congestion under reverse-path reordering. Besides, TCP-DOOR does not perform well in a congested network environment with substantial persistent packet reordering. It disables congestion control for a time period every time an out-of-order event is detected, which may lead to congestion collapse from undelivered packets [13].

5.1.3 DSACK TCP

Floyd et al. discussed the use of duplicate selective acknowledgement (DSACK) [20] to detect segment reordering and retract the associated spurious congestion response. DSACK is an extension of the selective acknowledgement (SACK) option [21] for TCP. It aims to use the SACK option for duplicate segments. The first block of the SACK option field is used to report the sequence numbers[3] of a received duplicate segment which has triggered the ACK. When congestion is detected, *cwnd* is saved before reduction. When a source finds that it has made a spurious congestion response based on the arrival of a DSACK, it performs slow start to increase the current *cwnd* to the stored *cwnd* before congestion avoidance. By performing slow start during state restoration, it allows TCP to reacquire ACK-clocking and avoid injecting traffic bursts into the network.

Fig. 5 shows how DSACK is used to detect packet reordering. Suppose Segment S1 is reordered such that it arrives after Segment S4 at the destination. The last acknowledged segment is Segment S0. In this case, the destination sends out three duplicate ACKs A1, A2, and A3 (with the same cumulative ACK for Segment S0, although the SACK option fields differ) to the source so that Segment S1 is retransmitted (assuming *dupthresh* is three). When the destination receives the retransmitted Segment S1, it sends a duplicate ACK A5 for Segment S4, but the first block of the SACK option field acknowledges an arrival of a duplicate Segment S1. The source then knows that Segment S1 has been retransmitted spuriously due to packet reordering.

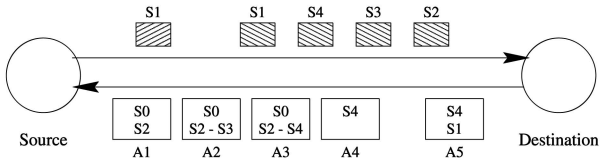

Fig. 5. An example of detecting packet reordering through DSACK for TCP.

This method can be easily coupled with a scheme using the DSACK information to adjust *dupthresh* to proactively prevent triggering spurious congestion responses in the future. The Blanton-Allman algorithm [22], RR-TCP [16], and the Leung-Ma Algorithm [23], have adopted this technique for detection of forward-path reordering and state reconciliation.

5.2 Threshold Adjustment

5.2.1 Lee-Park-Choi Algorithm (Sender-Side Solution)

Lee et al. [24] proposed a sender-side solution to improve TCP performance for forward-path reordering over multiple paths. *dupthresh* is set to increase logarithmically with the number of paths used. Thus, a source has to receive a larger number of duplicate ACKs before a congestion response is triggered, when more paths are used concurrently to transmit a single TCP flow.

However, when packet-level multipath routing is used for data transmission, the level of packet reordering may depend on the differences in path delays and how the packets belonging to a single flow are distributed to these paths [25]. Hence, there exists no direct correlation between *dupthresh* and the number of participating paths.

5.2.2 Blanton-Allman Algorithms

Blanton and Allman [22] proposed three alternatives to dynamically adjust *dupthresh*. The first alternative, denoted as Blanton-Allman:INC, is to increase *dupthresh* by some constant every time a spurious fast retransmission is detected. The second alternative, denoted as Blanton-Allman:AVG, is to increase *dupthresh* by taking the average of the current *dupthresh* and the number of duplicate ACKs required to disambiguate reordering from loss when a spurious fast retransmission is detected. The third alternative, denoted as Blanton-Allman:EWMA, is to assign *dupthresh* to an exponentially weighted moving average (EWMA) of the length of perceived reordering events. For all these algorithms, *dupthresh* is reset to three upon the expiration of the retransmission timer in order to reduce future costly retransmission timer expirations. The authors also extended the limited transmit algorithm [26], which allows a source to send a new segment upon the receipt of the first two duplicate ACKs, so that a new segment could be sent on every two duplicate ACKs received afterward. This helps to maintain ACK-clocking and avoids injecting

3. Due to possible repacketization of a segment, the first sequence number of the segment and the sequence number immediately following the last sequence number of the segment are reported in the block.

traffic bursts when an ACK for a new segment arrives. Furthermore, they employed the approach proposed in [20] by using DSACK information to detect forward-path reordering and state reconciliation.

Their simulation results [22] showed that, when compared with the default *dupthresh* of three, the proposed techniques improved connection throughput and reduced the number of unnecessary retransmissions. However, their algorithms have three major shortcomings. First, the adjustment of *dupthresh* for some proposed algorithms is not adaptive enough to the dynamic behavior of the reordering events. For example, it takes 17 detected spurious fast retransmissions for *dupthresh* to grow from 3 to 20, when the first algorithm with the increment being set to one is used. Second, there is no adaptive mechanism to reduce *dupthresh* dynamically, except for the third algorithm. These algorithms thus fail to adapt a suitable *dupthresh* that strikes a balance between the cost of a spurious fast retransmission and that of a retransmission timeout expiration. They are also unable to search for an appropriate but reduced value of *dupthresh* when the extent of reordering decreases. Third, the reset of *dupthresh* to three upon the expiration of the retransmission timer destroys all historical information about the level of forward-path reordering in the networks. It takes another time period to allow *dupthresh* to increase to the desired value.

5.2.3 RR-TCP

Zhang et al. devised the reordering-robust TCP (RR-TCP) [16] as an extension of the Blanton-Allman algorithms [22], but they differ in three ways. First, RR-TCP uses a different mechanism to adjust *dupthresh* dynamically. The authors formulated a combined cost function for retransmission timeouts, spurious fast retransmissions, and limited transmit to adapt the false fast retransmit avoidance ratio (FA radio). The FA ratio, which represents the portion of reordering events to be avoided in order to minimize the cost, can then be used to find the corresponding *dupthresh*. Thus, this provides a mechanism to raise or reduce *dupthresh* dynamically, by changing the FA ratio based on the current network conditions. Second, the authors considered another extended version of the limited transmit algorithm [26]. This extension permits a source to send up to one ACK-clocked additional congestion window's worth of data. Third, the authors suggested an idea to correct the sampling bias against long RTT samples for the RTT and RTO estimations. Instead of skipping the samples for retransmitted segments in the Karn's algorithm [15], an RTT sample is taken for each retransmitted segment by taking it as the average of the RTTs for both the first and the second transmissions of that segment.

The simulation results in [16] showed that RR-TCP could significantly improve TCP performance over reordering networks. When 1-2 percent of segments were randomly selected to experience a longer delay (according to a normal distribution), RR-TCP could improve the connection throughput by more than 50 percent and 150 percent when compared with the Blanton-Allman algorithms [22] (including the time-delayed fast retransmit algorithm) and SACK TCP [27], respectively. However, RR-TCP needs to maintain a reordering histogram to store the reordering information. It is also required to scan and update the histogram for every reordered segment.

5.2.4 Leung-Ma Algorithm

Leung and Ma [23] proposed to improve the TCP robustness to persistent packet reordering by extending the Blanton-Allman algorithms [22]. First, Leung and Ma suggested using an EWMA and the mean deviation of the lengths of the reordering events. By including the mean length deviation, *dupthresh* is selected to avoid triggering a certain portion of spurious fast retransmissions and prevent costly retransmission timer expirations. This shares the same design philosophy as RR-TCP [16] but incurs fewer computational and storage overheads. Second, an upper bound of *dupthresh* was introduced to avoid retransmission timeouts. To avoid the timer expiration for a lost segment, an ACK for the retransmitted segment must be received by a source before the timer fires. The maximum number of duplicate ACKs received before triggering a fast retransmission can then be estimated to satisfy the aforementioned criteria. Third, Leung and Ma also suggested a mechanism to exponentially reduce *dupthresh* for the retransmission timer expiration, since an occurrence of a retransmission timer could imply that *dupthresh* has been too large.

The simulation results in [23] demonstrated that the Leung-Ma algorithm could improve the connection throughput by at least 35 percent and reduce the unnecessary fast retransmissions by 6 percent when compared with the Blanton-Allman algorithms (including the time-delayed fast retransmit algorithm). When compared with RR-TCP, the Leung-Ma algorithm achieved similar performance in terms of connection throughput and unnecessary fast retransmissions, but it takes much fewer computations and storage space.

5.2.5 RN-TCP

Sathiaseelan and Radzik developed the reorder notifying TCP (RN-TCP) [28] to use the information about dropped segments at the routers along the transmission paths connecting a source to a destination. If a router has dropped some data packets belonging to a flow, it is responsible to remember the known maximum and minimum sequence numbers of the dropped packets belonging to that flow. This means that the router has dropped some, if not all, packets of that flow with sequence numbers within this range. When a data packet passes through the router and there is a dropped entry for the flow of that packet, the router inserts a dropped entry into the packet and the dropped entry is then removed from the router. In case the packet contains a dropped entry collected from upstream routers, the router in question will compute an appropriate range of sequence numbers with dropped packets before inserting it into the captioned packet.

A destination sends an ACK for a segment arrival. It may set a "reordered" bit of the ACK depending on whether the gap between the reordered segments are caused by forward-path reordering or not. When the destination receives a data segment, it uses the sequence number of the incoming segment, the dropped entry stored in the incoming segment, and the sequence number of the last received segment in the buffer queue to find out which

segments have been reordered or dropped. If the destination considers the gap between the reordered segments are caused by segment reordering, the "reordered" bit of the corresponding ACK will be set.

A source uses two different rules to trigger a fast retransmission, depending on whether the "reordered" bit is set or not. If the third duplicate ACK has a reordered bit set, the source contemplates that the expected segment has been reordered. The source will trigger a retransmission when it receives k more duplicate ACKs with the "reordered" bit set or a duplicate ACK without the "reordered" bit set. Otherwise, the source retransmits the inferred loss segment immediately.

However, RN-TCP has three shortcomings. First, RN-TCP requires all participating routers to store, compute, and insert the dropping information into forwarded packets. It is not practical in a heterogeneous network environment, such as the Internet, where network devices are maintained by different organizations and they are generally upgraded incrementally. Second, the value of k is dependent on the extents of segment dropping and reordering in the networks. A mechanism is still needed to dynamically determine an appropriate value of k. Third, RN-TCP does not consider its robustness to reverse-path reordering. The sender algorithm is sensitive to the order in which ACKs are received, as different order of these ACK arrivals can trigger different rules for fast retransmissions, thereby impeding the effectiveness of RN-TCP to improve TCP performance for packet reordering.

6 Temporal Approach

We describe the existing algorithms that can be classified as a temporal approach in this section. Refer to Table 2 for an overview of the surveyed schemes. Algorithms are referred to by the authors' last names, unless they have been named.

6.1 Response Postponement

6.1.1 Lee-Park-Choi Algorithm (Receiver-Side Solution)

Lee et al. [24] proposed a receiver-side solution to improve TCP performance for forward-path reordering. A destination sends delayed ACKs for reordered segment arrivals. It generates an ACK at every two segment arrivals or when a delayed ACK timer fires. In addition, a destination sends ACKs immediately for a retransmitted segment so as to avoid retransmission timer expiration.

However, since the time between two successive segment arrivals may depend on the bottleneck link of a participating path, there exists no direct correlation between the bottleneck path bandwidth and the differences in path delays. Thus, the presented response postponement is not adaptive to the extent of packet reordering due to packet-level multipath routing.

6.1.2 Paxson Algorithm

Paxson [6] outlined an idea of introducing an additional waiting time before a destination generates a duplicate ACK upon the advance of a sequence hole. The delay in ACK generation provides an opportunity for the destination to see if sending a duplicate ACK is necessary. However, it does not provide a mechanism to determine how to set a proper time interval to adapt to the level of packet reordering experienced.

6.1.3 Time-Delayed Fast Retransmit Algorithm

Blanton and Allman developed the time-delayed fast retransmit algorithm [22], denoted as Blanton-Allman:DEL, to postpone congestion response with the presence of forward-path reordering. Upon receiving three duplicate ACKs, a source waits for an additional time period before triggering a congestion response. It can be viewed as an extension of [6]. However, they differ as the Paxson algorithm is a receiver-based algorithm, but the time-delayed fast retransmit algorithm is a sender-based algorithm. When an ACK for the inferred loss segment arrives, the pending congestion response is cleared. The time period is increased by some constant each time a spurious retransmission is detected. Thus, this method is similar to the algorithms proposed in [22] to adjust *dupthresh* dynamically, and shares their associated merits and limitations.

6.1.4 TCP-DCR

Bhandarkar and Reddy devised the delayed congestion response TCP (TCP-DCR) [29] to meliorate the TCP robustness to noncongestion events. It advances the time-delayed fast retransmit algorithm [6] by delaying a congestion response for a time interval after the first duplicate ACK is received. The authors suggested to set this interval to one RTT so as to have ample time to deal with forward-path reordering due to link-layer retransmissions for loss recovery. To maintain ACK-clocking, TCP-DCR sends one new data segment upon the receipt of each duplicate ACK.

The simulation results in [29] demonstrated that TCP-DCR performed significantly better than SACK TCP [27]. TCP-DCR achieved 10 times more in connection throughput than SACK TCP when more than 5 percent of packets are delayed according to a normal distribution with negligible congestion loss. However, the chosen bottleneck link delay is at least equal to the highest possible reordered delay for their experiments. This implies that a reordering event is unlikely to last longer than the interval for delaying the congestion response. The suggested interval may not be a proper choice for multipath routing since packets are reordered mainly based on the differences in path delay, while the estimated RTT is a weighted average of RTT based on the traffic distribution to the participating paths. Further study is needed to find a proper choice of the delayed interval for congestion response with the presence of packet reordering.

6.2 Retransmission by Timeout

6.2.1 TCP-PR

Bohacek et al. proposed TCP for persistent packet reordering (TCP-PR) [30] to devise the RTO timer to enhance TCP performance for persistent packet reordering. Instead of keeping track of the EWMA of the mean RTT, TCP-PR utilized a nonsmoothed, exponentially weighted maximum possible RTT. By doing so, spikes in RTT can be promptly reflected in the estimated RTT for some time. When a segment drop is detected, *cwnd* is set to half of *cwnd* at the

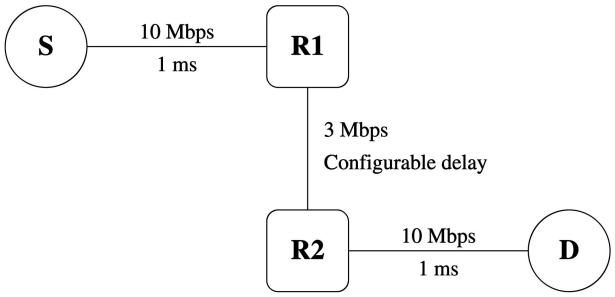

Fig. 6. The network topology used in the simulation study.

TABLE 1
Simulation Parameters

Parameter	Value
Mean of *R1-R2* path delay	$(200\tau + 50)$ ms, $\tau \in [0, 2]$
Standard deviation of path delay	$\frac{200\tau}{3}$ ms, $\tau \in [0, 2]$
Inter-switching time, δ	50 ms or 250 ms
Buffer size in a router	300 segments
Maximum *cwnd*	500 segments
Segment size	1500 bytes

time the segment has been sent. Congestion avoidance is then carried out. Subsequent occasional segment drops detected in the same congestion window will not cause any further reduction of *cwnd* to avoid over-reaction to congestion. When more than half of a congestion window's worth of segments are inferred to be lost, *cwnd* is set to one and the slow start process is performed.

The major advantage of TCP-PR is that the new RTT and RTO estimators are very effective in shielding the effect of packet reordering due to differences in path delays, since they are devised from the sampled maximum possible RTT. Another merit for TCP-PR is that it is able to maintain ACK-clocking with the presence of packet reordering. The reason is that a pending segment can be sent so long as the number of outstanding segments to be acknowledged is less than the size of the congestion window.

There are two limitations for TCP-PR. First, in order to maintain a constant scaling (which is less than one) of the RTT spikes per RTT, the scaling factor is raised to the factor of $\frac{1}{cwnd}$. However, this makes TCP-PR computationally expensive since a series of exponentiation computations have to be performed on every ACK arrival. Second, the proposed RTT estimator may be overly sensitive to spikes in RTT. A sudden high RTT samples, which can be caused by routing loops due to topological changes, can greatly enlarge the estimated RTT for some time. This can substantially defer segment retransmissions if they are indeed dropped during that time.

7 Performance Evaluation

In this section, we present our simulation results and compare the surveyed algorithms. Section 7.1 discusses the experimental setup for our simulation study. Simulation results are exhibited in Section 7.2. Section 7.3 gives further discussion and comparison of these algorithms.

7.1 Experimental Setup

The network topology used for the study is shown in Fig. 6. It involves two end-systems (S and D) and two routers ($R1$ and $R2$). The path between $R1$ and $R2$ models the underlying network path connecting $R1$ and $R2$. A transmission path usually consists of multiple hops. It has been shown [31] that the average hop-count for an Internet path is 16.2. The central limit theorem [32] suggests that the end-to-end delay over a multihop path, which is the sum of a large number of independent hop-delays, is approximately normally distributed. To simulate packet reordering (such as those caused by route fluttering), we repeatedly change the $R1 - R2$ path delay according to a truncated normal distribution such that every path delay sample is at least 50 ms. The mean and standard deviation of the path delay are $(200\tau + 50)$ ms and $\frac{200\tau}{3}$ ms, respectively, where τ is the path delay factor ranging from 0 to 2 in our study. A larger τ will induce more variation in the path delay, thereby increasing the degree of packet reordering.

The time interval between two successive changes on the path delay, denoted as the interswitching time, dictates the frequency of the reordering events. In our simulation study, the interswitching time, δ, is exponentially distributed with mean 50 ms or 250 ms. The smaller the interswitching time is, the more frequently reordering events are produced, and vice versa.

The simulation parameters are summarized in Table 1. Our simulation study has been performed using the Network Simulator (ns) Version 2.29 [33]. Except for the Lee-Park-Choi algorithms [24] and the Paxson algorithm [6], the program codes of all algorithms under study are ported to the same version of the simulator for fair comparison. The Lee-Park-Choi algorithms are engineered to packet-level multipath routing, whereas the Paxson algorithm is merely an outline of thought. Hence, they are not considered for this simulation study. The ported program codes and simulation scripts can be obtained at http://www.eee.hku.hk/~kcleung/research/TCP_reordering_survey.html.

A single, long-lived TCP flow from S to D is simulated for 1,100 seconds. We take the goodput of a flow, which represents the rate of useful data (that can be acknowledged cumulatively) delivered to the destination successfully, as the performance metric of the surveyed algorithms. For each simulation run, the statistics for computing the performance metric are collected after the trial period of the first 100 simulated seconds. A total of 30 runs have been done to compute an average value of the performance metric, and a 95 percent confidence interval for each average value of the metric is also calculated. The quality of an algorithm depends on how well the goodput of the flow can be sustained with various degrees of packet reordering.

7.2 Simulation Results

The results are provided in two sets. The first set examines the effect of forward-path reordering on the performance of TCP implemented with various surveyed schemes. The reverse path from $R2$ to $R1$ is an in-order channel with a constant delay of 50 ms, i.e., $\tau = 0$. Fig. 7 shows the goodput of the flow when the path delay factor, τ, varies between 0 and 2. Except for TCP-PR [30], the goodput generally drops as τ increases from 0 to 2. A larger value of τ implies a larger mean and

TABLE 2
An Overview of Surveyed Reordering Solutions for TCP

Algorithms	Solution Strategy	Devices Involved			Additional Information Needed			Reduction of Spurious Retransmissions	Maintaining ACK–Clocking	Sustaining Larger Congestion Window	Fairer Estimation of RTT/RTO
		Source	Destination	Router	DSACK	Reordered Bit	Timestamp / Sequence Number				
Blanton–Allman Algorithms	Threshold Adjustment	√			√			√	√	√	
	Response Postponement	√			√			√	√	√	
DSACK TCP	State Reconciliation	√			√						
Eifel Algorithm	State Reconciliation	√					√				
Lee–Park–Choi Algorithms	Threshold Adjustment	√						√		√	
	Response Postponement		√								
Leung–Ma Algorithm	Threshold Adjustment	√			√			√	√	√	
Paxson Algorithm	Response Postponement		√					√		√	
RN–TCP	Threshold Adjustment	√	√	√		√		√		√	
RR–TCP	Threshold Adjustment	√			√			√	√	√	√
TCP–DCR	Response Postponement	√						√	√	√	
TCP–DOOR	State Reconciliation	√	√				√	√			
TCP–PR	Retransmission by Timeout	√						√	√	√	√

standard deviation of the path delay. This results in a higher degree of packet reordering. Hence, it is more likely to trigger spurious fast retransmissions. In addition, the goodput ordinarily soars when δ rises from 50 ms to 250 ms, because the reordering events occur less often.

The algorithms for threshold adjustment and those for the temporal approach generally perform better than those for state reconciliation. The latter class of algorithms is only able to recover the congestion state just before a congestion response is taken. Hence, these algorithms do not alleviate performance problems due to persistent and substantial segment reordering. By suspending the congestion response for a certain time period instead of initiating state recovery upon detecting a spurious fast retransmission, TCP-DOOR [19] outperforms DSACK TCP [20], the Eifel algorithm [17], and SACK TCP [27] by at least 89.4 percent in connection goodput when τ is two.

The algorithms for threshold adjustment and those for the temporal approach can help TCP reduce spurious retransmissions due to segment reordering, thereby maintaining a larger congestion window and sustaining a higher connection goodput. The Leung-Ma algorithm [23], RR-TCP [16], and TCP-DCR [29] give similar performance. They outperform the Blanton-Allman algorithms [22] and RN-TCP [28] when τ is larger than a certain value, say, one, since they provide effective mechanisms to either dynamically adapt to the reordering conditions in the network or wait sufficiently long to avoid triggering fast retransmissions unnecessarily. TCP-PR sustains a good connection goodput at various levels of packet reordering, since its RTT and RTO estimators are very effective in shielding the effect of packet reordering.

The second set investigates the effect of reverse-path reordering on TCP performance. As in the first set, the forward path from $R1$ to $R2$ is an in-order channel with a constant delay of 50 ms so that no packet reordering is possible in the forward path. Fig. 8 exhibits the goodput of the flow when the path delay factor, τ, varies between 0 and 2. Except for TCP-PR, the connection goodput falls as τ increases from 0 to 2. All algorithms except TCP-PR perform more or less the same. When τ is large, TCP-PR can still maintain a high connection goodput since its RTT and RTO estimators are highly effective in covering any adverse effect due to packet reordering. However, other surveyed algorithms are not very effective in maintaining the connection goodput with the presence of reverse-path reordering, because reordered ACKs can lead to the loss of ACK-clocking and burst injection.

7.3 Further Discussion and Comparison

We summarize the properties of the presented algorithms in Table 2. These properties have already been discussed in the previous sections. For the ordinal approach, we find that the algorithms for state reconciliation are only able to recover the congestion state just before a congestion response is taken. These algorithms alone are therefore not adequate at alleviating performance problems due to persistent and substantial packet reordering, but they can be complemented by the algorithms for threshold adjustment. The latter class of algorithms can help TCP reduce spurious retransmissions, maintain ACK-clocking with the use of an extended limited transmit algorithm for segment reordering, and sustain a larger congestion window.

For the temporal approach, we discover that the algorithms for response postponement are able to reduce spurious retransmissions and, thus, maintain a larger congestion window, but all except TCP-DCR fail to clock out traffic during the deferment of congestion response. We note that the algorithm for retransmission by timeout is able to resolve all four problems stated in Section 3 for packet reordering, except for the computational and stability issues mentioned when the algorithm was presented.

Nevertheless, none of the existing algorithms attempts to resolve the potential loss of ACK-clocking due to reverse-path reordering. In addition, none of the existing algorithms

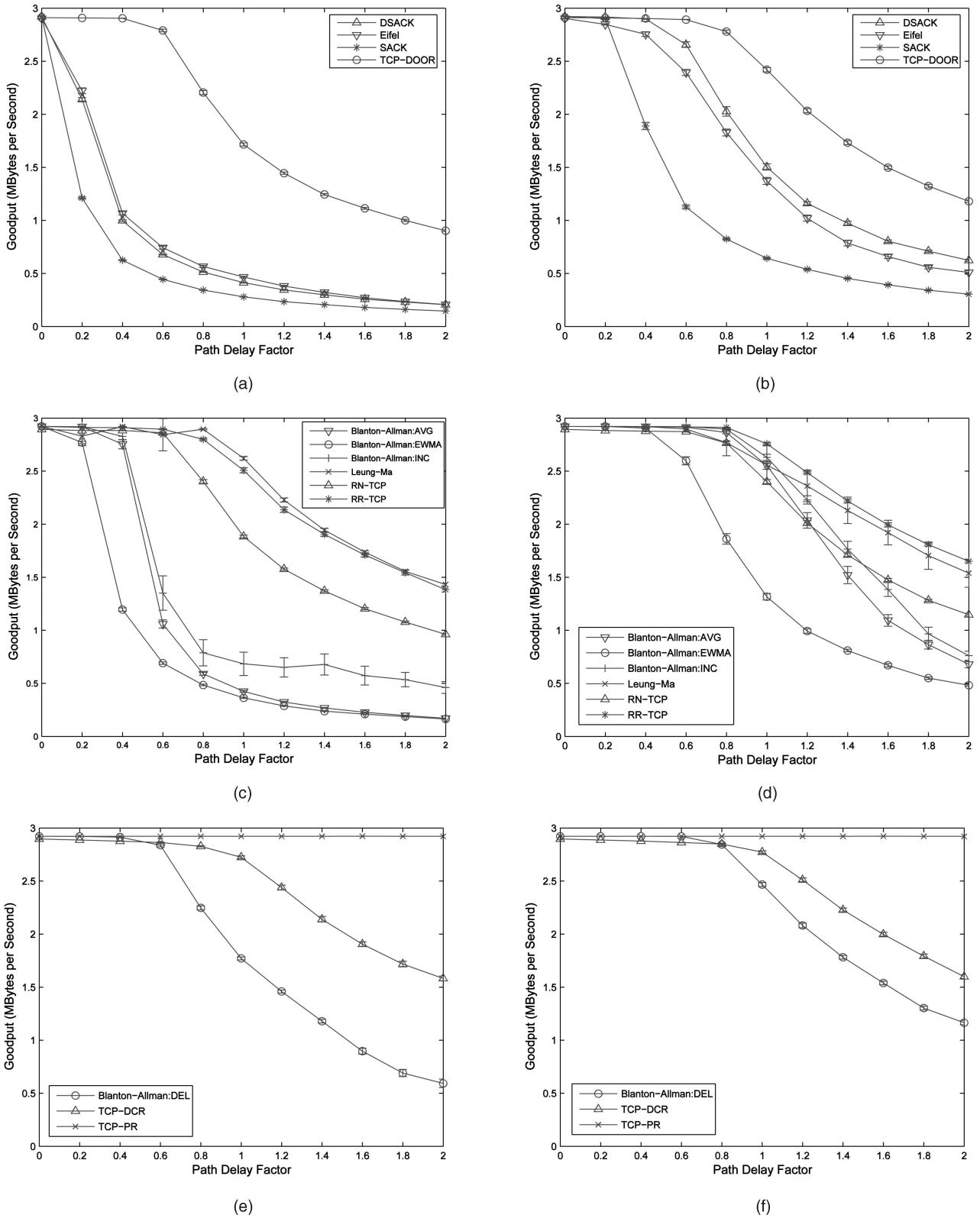

Fig. 7. Goodput against path delay factor for various settings on forward-path reordering. (a) State reconciliation with $\delta = 50$ ms. (b) State reconciliation with $\delta = 250$ ms. (c) Threshold adjustment with $\delta = 50$ ms. (d) Threshold adjustment with $\delta = 250$ ms. (e) Temporal approach with $\delta = 50$ ms. (f) Temporal approach with $\delta = 250$ ms.

studies whether the current approach for the RTT and RTO estimations is still effective for retransmission timer management when a traffic flow is forwarded over multiple paths. The only exception is TCP-PR which applies an alternative approach to estimate the maximum possible RTT instead of the mean and deviation of RTT. In summary,

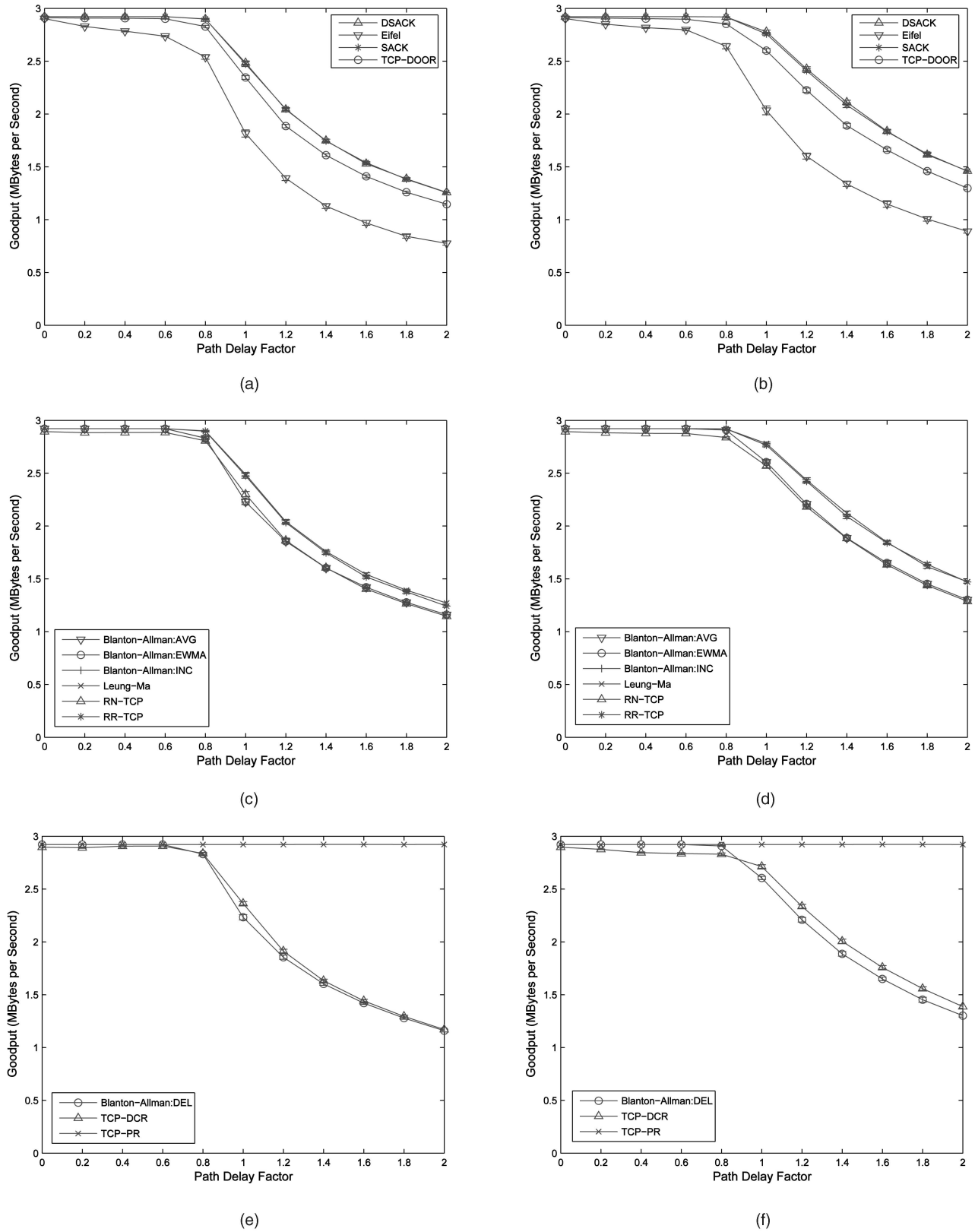

Fig. 8. Goodput against path delay factor for various settings on reverse-path reordering. (a) State reconciliation with $\delta = 50$ ms. (b) State reconciliation with $\delta = 250$ ms. (c) Threshold adjustment with $\delta = 50$ ms. (d) Threshold adjustment with $\delta = 250$ ms. (e) Temporal approach with $\delta = 50$ ms. (f) Temporal approach with $\delta = 250$ ms.

these issues need to be considered to resolve the packet reordering problems for TCP.

We believe the following is desirable in a good reordering algorithm for TCP:

1. Operate as a sender-based algorithm so as to yield high interoperability.
2. Achieve high connection throughput.
3. Minimize spurious segment retransmissions and avoid retransmissions by timeouts.
4. Maintain ACK-clocking and avoid injecting traffic bursts into the network.
5. Manage the retransmission timer effectively so that it can adapt to various network conditions, including packet reordering due to packet-level multipath routing.
6. Achieve low algorithm complexity.

8 Open Research Issues

Although the existing algorithms provide some possible solutions to packet reordering in TCP, some issues have not been discussed in the literature and are potential research topics. They are listed as follows.

8.1 Integrated Solution for All Types of Noncongestion Loss

Packet reordering is merely one type of noncongestion loss TCP has to deal with. When a TCP connection is established over some wireless networks, it has to deal with other types of noncongestion packet loss, including the transmission loss over wireless links and disconnection loss due to host or network mobility. There have been some research in enhancing TCP for such noncongestion wireless loss [29], [34], [35], [36], [37], [38]. TCP-DCR [29] is so far the only work that deals with performance problems due to both packet reordering and reliable link-layer retransmissions. However, all other proposed techniques for dealing with noncongestion wireless loss are mainly based on the standard TCP protocol and do not generally take packet reordering into account. This means that it might not be possible to have an effective solution for dealing with both noncongestion wireless loss and packet reordering by simply bundling the proposed techniques separately designed for each. Hence, further study is needed to devise an integrated solution for TCP that can solve all types of noncongestion loss for wired/wireless networks with packet reordering.

8.2 Improved Selective Acknowledgement Mechanism

The DSACK option [20] allows TCP to acknowledge out-of-order data and duplicate segments. Unlike the standard cumulative ACK and SACK [21], a notification for a duplicate segment is sent only once. Thus, this is not a robust mechanism to report the receipt of a duplicate segment as an ACK can easily get corrupted and lost in an error-prone network.

In order to improve its robustness, this information should be sent repeatedly, say at least k times, in some subsequent ACKs until a destination considers that the information has been received, with an acceptable confidence, by the source. Ways to properly formulate the scheme to choose an appropriate value of k requires further study.

8.3 Quantitative Assessment on Causes of Packet Reordering

Recent studies [5], [6] have discussed the occurrence and causes of packet reordering. Substantial quantitative results have been provided to justify that packet reordering occurs normally in packet-switching networks. However, the discussion about the causes of packet reordering was somewhat qualitative, without any empirical results to infer which causes are more likely to happen and their impact to packet reordering. Thus, a quantitative assessment on the causes of packet reordering warren to be investigated further.

9 Conclusion

In this paper, we have presented a comprehensive and in-depth survey of current research on packet reordering in TCP. Packet-level multipath routing, route fluttering, inherent parallelism in modern high-speed routers, link-layer retransmissions, and router forwarding lulls are major causes of packet reordering. With persistent and substantial packet reordering, TCP spuriously retransmits segments, keeps its congestion window unnecessarily small, loses its ACK-clocking, and understates the estimated RTT (and, thus, RTO).

Existing algorithms were categorized into two different strategies, namely, the ordinal approach and the temporal approach. The ordinal approach is a collection of methods that process the ordering information of segments and ACKs received so as to infer and generate more appropriate congestion response with the presence of packet reordering. We found that an algorithm for threshold adjustment complemented with state reconciliation can form an effective solution to deal with packet reordering for TCP.

The temporal approach represents a group of techniques that avoid triggering spurious congestion responses by deferring them for a time period. We noted that the algorithms for response postponement could lose ACK-clocking and inject bursty traffic into the network. The algorithm for retransmission by timeout was found to be effective to deal with packet reordering, except for the computational and stability issues involved.

We also proposed some future research directions, including the need of a mechanism to resolve the potential loss of ACK-clocking due to reverse-path reordering, improved retransmission timer management, the development of an integrated solution for all types of noncongestion loss, the formulation of an improved selective acknowledgement mechanism, and the quantitative assessment on the causes of packet reordering.

Acknowledgments

This research is supported in part by the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99). The authors would like to thank Sumitha Bhandarkar, Ethan Blanton, Chansook Lim, Arjuna Sathiaseelan, Morten Schläger, Feng Wang, and Ming Zhang for releasing their ns source codes to the authors' simulation study. The authors would also like to express their gratitude to the anonymous reviewers for their valuable comments and suggestions which assisted them in improving the quality of the paper.

REFERENCES

[1] J. Postel, *Transmission Control Protocol,* RFC 793, Protocol Specification, DARPA Internet Program, Sept. 1981.

[2] D.D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *ACM SIGCOMM Computer Comm. Rev.,* vol. 18, no. 4, pp. 106-114, Aug. 1988.

[3] M. Allman, V. Paxson, and W. Stevens, *TCP Congestion Control,* IETF RFC 2581, Apr. 1999.

[4] V. Paxson and M. Allman, *Computing TCP's Retransmission Timer,* IETF RFC 2988, Nov. 2000.

[5] J. Bennett, C. Partridge, and N. Shectman, "Packet Reordering is Not Pathological Network Behavior," *IEEE/ACM Trans. Networking,* vol. 7, no. 6, pp. 789-798, Dec. 1999.

[6] V. Paxson, "End-to-End Internet Packet Dynamics," *IEEE/ACM Trans. Networking,* vol. 7, no. 3, pp. 277-292, June 1999.

[7] M. Laor and L. Gendel, "The Effect of Packet Reordering in a Backbone Link on Application Throughput," *IEEE Network,* vol. 16, no. 5, pp. 28-36, Sept./Oct. 2002.

[8] K.-C. Leung and V.O.K. Li, "Generalized Load Sharing for Packet-Switching Networks I: Theory and Packet-Based Algorithm," *IEEE Trans. Parallel and Distributed Systems,* vol. 17, no. 7, pp. 694-702, July 2006.

[9] N.F. Maxemchuk, "Dispersity Routing in High-Speed Networks," *Computer Networks and ISDN Systems,* vol. 25, no. 6, pp. 645-661, Jan. 1993.

[10] S.-J. Lee and M. Gerla, "AODV-BR: Backup Routing in Ad Hoc Networks," *Proc. IEEE Wireless Communications and Networking Conf. (WCNC '00),* vol. 3, pp. 1311-1316, Sept. 2000.

[11] P.P. Pham and S. Perreau, "Performance Analysis of Reactive Shortest Path and Multi-Path Routing Mechanism with Load Balance," *Proc. IEEE INFOCOM '03,* vol. 1, pp. 251-259, 2003.

[12] F. Hu and N.K. Sharma, "Enhancing Wireless Internet Performance," *IEEE Comm. Surveys and Tutorials,* vol. 4, no. 1, pp. 2-15, Dec. 2002.

[13] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Trans. Networking,* vol. 7, no. 4, pp. 458-472, Aug. 1999.

[14] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM Computer Comm. Rev.,* vol. 18, no. 4, pp. 314-329, Aug. 1988.

[15] P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *ACM Trans. Computer Systems,* vol. 9, no. 4, pp. 364-373, Nov. 1991.

[16] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK," *Proc. IEEE Int'l Conf. Network Protocols (ICNP '03),* pp. 95-106, Nov. 2003.

[17] R. Ludwig and R.H. Katz, "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions," *ACM SIGCOMM Computer Comm. Rev.,* vol. 30, no. 1, pp. 30-36, Jan. 2000.

[18] V. Jacobson, R. Braden, and D. Borman, *TCP Extensions for High Performance,* IETF RFC 1323, May 1992.

[19] F. Wang and Y. Zhang, "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response," *Proc. ACM MOBIHOC '02,* pp. 217-225, June 2002.

[20] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, *An Extension to the Selective Acknowledgement (SACK) Option for TCP,* IETF RFC 2883, July 2000.

[21] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, *TCP Selective Acknowledgment Options,* IETF RFC 2018, Oct. 1996.

[22] E. Blanton and M. Allman, "On Making TCP More Robust to Packet Reordering," *ACM SIGCOMM Computer Comm. Rev.,* vol. 32, no. 1, pp. 20-30, Jan. 2002.

[23] K.-C. Leung and C. Ma, "Enhancing TCP Performance to Persistent Packet Reordering," *J. Comm. and Networks,* vol. 7, no. 3, pp. 385-393, Sept. 2005.

[24] Y. Lee, I. Park, and Y. Choi, "Improving TCP Performance in Multipath Packet Forwarding Networks," *J. Comm. and Networks,* vol. 4, no. 2, pp. 148-157, June 2002.

[25] K.-C. Leung and V.O.K. Li, "Flow Assignment and Packet Scheduling for Multipath Routing," *J. Comm. and Networks,* vol. 5, no. 3, pp. 230-239, Sept. 2003.

[26] M. Allman, H. Balakrishnan, and S. Floyd, *Enhancing TCP's Loss Recovery Using Limited Transmit,* IETF RFC 3042, Network Working Group, Jan. 2001.

[27] K. Fall and S. Floyd, "Simulation-Based Comparisons of Tahoe, Reno, and SACK TCP," *ACM SIGCOMM Computer Comm. Rev.,* vol. 26, no. 3, pp. 5-21, July 1996.

[28] A. Sathiaseelan and T. Radzik, "Improving the Performance of TCP in the Case of Packet Reordering," *Lecture Notes in Computer Science,* vol. 3079, pp. 63-73, June/July 2004.

[29] S. Bhandarkar and A.L.N. Reddy, "TCP-DCR: Making TCP Robust to Non-Congestion Events," *Lecture Notes in Computer Science,* vol. 3042, pp. 712-724, May 2004.

[30] S. Bohacek, J.P. Hespanha, J. Lee, C. Lim, and K. Obraczka, "A New TCP for Persistent Packet Reordering," *IEEE/ACM Trans. Networking,* vol. 14, no. 2, pp. 369-382, Apr. 2006.

[31] W. Theilmann and K. Rothermel, "Dynamic Distance Maps of the Internet," *Proc. IEEE INFOCOM '00,* vol. 1, pp. 275-284, Mar. 2000.

[32] W.W. Hines and D.C. Montgomery, *Probability and Statistics in Engineering and Management Science,* third ed. John Wiley & Sons, 1990.

[33] K. Fall and K. Varadhan, "The *ns* Manual (Formerly *ns* Notes and Documentation)," *The VINT Project,* May 2006.

[34] T. Goff, J. Moronski, D.S. Phatak, and V. Gupta, "Freeze-TCP: A True End-to-End TCP Enhancement Mechanism for Mobile Environments," *Proc. IEEE INFOCOM '00,* vol. 3, pp. 1537-1545, Mar. 2000.

[35] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," *IEEE J. Selected Areas in Comm.,* vol. 19, no. 7, pp. 1300-1315, July 2001.

[36] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang, "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks," *Wireless Networks,* vol. 8, no. 5, pp. 467-479, 2002.

[37] C.P. Fu and S.C. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," *IEEE J. Selected Areas in Comm.,* vol. 21, no. 2, pp. 216-228, Feb. 2003.

[38] S. Biaz and N.H. Vaidya, "'De-Randomizing' Congestion Losses to Improve TCP Performance over Wired-Wireless Networks," *IEEE/ACM Trans. Networking,* vol. 13, no. 3, pp. 596-608, June 2005.

Ka-Cheong Leung (S'95-M'01) received the BEng degree in computer science from the Hong Kong University of Science and Technology, Hong Kong, in 1994, and the MSc degree in electrical engineering (computer networks) and the PhD degree in computer engineering from the University of Southern California, Los Angeles, in 1997 and 2000, respectively. He worked as a senior research engineer at the Nokia Research Center, Nokia Inc., Irving, Texas, from 2001 to 2002. He was an assistant professor in the Department of Computer Science at Texas Tech University, Lubbock, Texas, between 2002 and 2005. Since June 2005, he has been with the University of Hong Kong, Hong Kong, where he is a visiting assistant professor in the Department of Electrical and Electronic Engineering. His research interests include wireless packet scheduling, routing, congestion control, and quality of service guarantees in high-speed communication networks, content distribution, high-performance computing, and parallel applications. He is a member of the IEEE.

Victor O.K. Li (S'80-M'81-SM'86-F'92) received the SB, SM, EE, and ScD degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, Massachusetts in 1977, 1979, 1980, and 1981, respectively. He joined the University of Southern California (USC), Los Angeles in February 1981, and became a professor of electrical engineering and director of the USC Communication Sciences Institute. Since September 1997, he has been with the University of Hong Kong, Hong Kong, where he is Chair Professor of Information Engineering in the Department of Electrical and Electronic Engineering. He also served as managing director of Versitech Ltd., the technology transfer and commercial arm of the university, and on various corporate boards. His research is in information technology, including all-optical networks, wireless networks, and Internet technologies and applications. He is a principal investigator in the area of excellence in information technology funded by the Hong Kong Government. He is now serving as an editor of *ACM/Springer Wireless Networks* and *IEEE Communications Surveys and Tutorials*. He has received numerous awards, including, most recently, the Changjiang Chair Professorship from the Ministry of Education, China, UK Royal Academy of Engineering Senior Visiting Fellowship in Communications, the Outstanding Researcher Award of the University of Hong Kong, the Croucher Foundation Senior Research Fellowship, and the Order of the Bronze Bauhinia Star, Government of HKSAR, China. He was elected an IEEE fellow in 1992. He is also a fellow of the HKIE and the IAE.

Daiqin Yang (S'04) received the BE and ME degrees from the Department of Electronic and Information Engineering, Huazhong University of Science and Technology, Wuhan, China, in 1999 and 2002, respectively. She is currently pursuing the PhD degree in the Department of Electrical and Electronic Engineering, the University of Hong Kong, Hong Kong. Her current research interests include scheduling and flow control in the Internet and wireless networks. She is a student member of the IEEE.