A SURVEY ON TCP OVER MOBILE AD-HOC NETWORKS

FENG WANG* AND YONGGUANG ZHANG†

Abstract. This chapter is a survey on TCP performance in mobile ad-hoc networks. We first describe the problems of standard TCP in ad-hoc networks, and then present the design space and existing solutions to improve TCP throughput. Particularly, we use a detection-response framework to categorize different approaches and analyze the possible design options.

1. Introduction.

A mobile ad-hoc network (MANET) is a special type of wireless networks. It consists of a collection of "peer" mobile nodes that are capable of communicating with each other without help from a fixed infrastructure. The interconnections between nodes are capable of changing on a continual and arbitrary basis. Nodes within each other's radio range communicate directly via wireless links, while those that are far apart use other nodes as relays in a multi-hop routing fashion. The typical applications of MANETs include conferences or meetings, emergency operations such as disaster rescue, and battlefield communications.

Compared with the traditional wired internet, MANETs have two fundamental differences, its usage of wireless channel and frequent node mobility. Both of these two characteristics have significant impact on the TCP performance.

1.1. Wireless Channel.

In a MANET, all nodes share the wireless medium and communicate through radios. The problem of contention and collision in such wireless networks is much more serious than in the wired environment. Currently, IEEE 802.11 [13] is the *de facto* Medium Access Control (MAC) protocol for Wireless LANs. It coordinates the medium access by Distributed Coordination Function (DCF), also known as CSMA/CA. It can operate in two modes, ad hoc mode and infrastructure mode. A central base station is needed to operate in infrastructure mode, while the ad hoc mode is for the non-centralized type of communication as in MANETs.

However, it is doubted by some researchers [23] whether IEEE 802.11 works well in multi-hop ad hoc networks, particularly when TCP is used as the transport layer protocol. Existing research work [11] has shown that TCP performance decreases drastically as the hop count becomes larger and larger.

Another problem with wireless channel that hurts TCP performance is, the high bit error rate of wireless links. Nowadays, the wired links are already so stable that we can ignore the link errors. But it is not yet the case as far as wireless links are concerned. Designed for wired networks originally, TCP, particularly its Congestion Avoidance and Control [14] mechanism, does not consider link errors as a possible reason for packet errors or losses. This can significantly degrade the performance of TCP over wireless networks including MANETs.

1.2. Node Mobility.

The node mobility we discuss here is on a continual and arbitrary basis, i.e., the nodes can keep moving in an arbitrary speed, toward an arbitrary direction. A typical scenario is people communicating with each other while driving their cars. It is a nature of MANETs.

Due to the limited communication range of radios, such constant node mobility immediately leads to frequent link breakages and therefore route changes. Many ad-hoc routing

*Department of Computer Sciences, The University of Texas at Austin (wangf@cs.utexas.edu).
†HRL Laboratories, LLC, Malibu, California (ygz@hrl.com).

protocols have been designed to establish and maintain routes under mobility. But it is not clear how TCP should react to mobility-induced events, such as route changes. In fact, it has been shown in [12] that the legacy TCP performs poorly under mobility.

In general, a MANET is a very complicated network system. There are many research problems that are worth exploring. In this chapter, we focus on how TCP performs over MANETs and the approaches to improving TCP performace, which is a very important issue since TCP is the mostly used transport layer protocol in current Internet. The performance metric we are particularly interested in is throughput. Other metrics, such as delay and fairness, are also important and beyond the scope of this chapter. In other words, we are more interested in throughput performance of long run TCP sessions such as file transfer applications using FTP on top of TCP. The approaches we study here are also applicable to short TCP sessions, although where typically people are more concerned about the delay performance.

The rest of this chapter is organized as follows. We give an overview in section 2 on the TCP Congestion Control mechanisms. We explain in details in section 3 why TCP performs poorly in MANETs, and discuss the solution space and approaches in sections 4 through 7. Section 8 is a summary of the performance studies of the approaches we cover in this chapter.

2. Overview of TCP Congestion Control. TCP is a sliding window protocl. It is so called 'self-clocking' in that the sender uses acknowledgments as a 'clock' to transmit more packets to the network. To dynamically adapt to the underlying network load, TCP employs a Congestion Control algorithm, which consists of three major components: Additive-Increase Multiplicative-Decrease (AIMD), Slow Start, and Fast Recovery.

The basic idea of TCP Congestion Control is that, whenever a packet is lost (or more precisely, when the TCP sender believes a packet is lost), the sender should slow down the sending rate. This slowdown is achieved by reducing the congestion window size. The rationale behind this is, TCP assumes only congestion causes packet losses. Here the congestion is essentially buffer overflow occuring at intermediate routers. There are two types of events that can indicate packet losses, timeouts and triple duplicate acknowledgements. TCP reacts differently to these two types of loss events.

2.1. AIMD. When TCP does not detect congestion, it conservatively increases the sending rate by increasing the congestion window size. Roughly speaking, after the initial phase, if there is no loss event during one round trip time, the congestion window size is incremented by one packet. By this additive increase, TCP can slowly probe the available bandwidth without injecting too many packets to congest the network.

On the other hand, when a loss event occurs, TCP halves a threshold value that bounds from above the congestion window size. This is again conservative in terms of the load the send can add to the network.

2.2. Slow Start. In the beginning of a TCP connection, the congestion window size is initialized to be 1 segment. This yields a very low initial sending rate. In order to quickly grow to the available bandwidth which could be much larger, TCP sender doubles its congestion window size every RTT time during the initial phase, until the congestion size reaches a pre-determined threshold value. Beyond this threshold value, the window size increase becomes linear. This initial phase is called the slow start phase. In fact, TCP enters slow start not only in the beginning of a connection. TCP will enter slow start whenever a timeout loss event occurs, i.e. the TCP sender will reset the congestion window size to 1 if a timeout event occurs and then grow the window size exponentially until reaching the threshold value.

2.3. Fast Recovery. Slow start is the reaction of TCP to timeout events. When there are triple duplicate acknowledgments, TCP sender simply halves its current congestion window size and then increases it linearly. This reaction to triple duplicate acknowledgments without slow start is called fast recovery.

3. Problems of TCP in MANETs. As reviewed in section 2, TCP assumes that network congestion has happened whenever a packet is lost. It then invokes appropriate congestion control actions including window size reduction. Although this assumption is reasonable for wired networks, it is questionable for wireless networks especially MANETs. Other than congestion (i.e. buffer overflow), possible causes of packet losses in MANETs include, wireless link errors, MAC layer losses due to channel contention, and link breakages due to node mobility. All those causes that are not related to congestion can result in unnecessary congestion control, which will degrade the TCP performance.

3.1. Wireless Link Errors. Wireless links posses high bit error rates that can not be ignored. But TCP interprets packet losses caused by bit errors as congestion. As a result, its performance suffers in wireless networks when TCP unnecessarily invokes congestion control, causing reduction in throughput and link utilization.

There has been much research work (e.g., [1, 2]) in improving TCP performance over wireless links. The Snoop Protocol [3] is a typical approach to solving the problem in cellular networks. In this protocol, a snoop agent is located at the base station, which monitors the packets that pass through the TCP connection in both directions. It caches the TCP segments the acknowledgements of which have not been received. If a loss event happens for a cached packet, the snoop agent can retransmit the packet directly without having to wait for the retransmission from the sender.

However, most of the proposed mechanisms are designed for infrastructure-based networks and depend on the base stations in distinguishing the error losses from congestion losses. Since mobile ad-hoc networks do not have such infrastructure, these mechanisms cannot apply.

Fortunately, this issue is not that serious when link layer acknowledgments are used as in IEEE 802.11. Now, if a packet is corrupted due to wireless link errors, the MAC receiver will not acknowledge the original packet. After a certain timeout without receiving the acknowledgement from the receiver, the MAC sender will retransmit the packet. Since this is handled at the MAC layer, the wireless link errors are transparent to TCP.

3.2. MAC Layer Losses due to Channel Contention. In a shared wireless medium, due to the channel contention among all nodes, it is not guaranteed that a packet sent out by node A will be received successfully by its destined neighbor B. If RTS/CTS handshake is required, it is even not guaranteed that node A can have the right to send a packet unless it successfully sends out an RTS message and receices the corresponding CTS message from the receiver B. In IEEE 802.11 standard, it specifies that, if a node does not receive a CTS after sending or retransmitting 4 RTS messages, or does not receive a link layer acknowledgement after sending or retransmitting 7 DATA messages, the node determines the link is broken and should drop the DATA packet it tries to transmit.

This type of packet loss has nothing to do with buffer overflow. It can happen even there is only one packet in the buffer. A recent research work [10] has shown that, in general, buffer overflows are rare, while most packet drops experienced by TCP are due to link contentions. It is certainly a wrong reaction to invoke congestion control when such MAC layer losses occur.

3.3. Link Breakages due to Mobility. In addition to all links being wireless, frequent route disruption events – route failures and route changes – due to mobility can cause serious problems to TCP as well. Given a source node S, a destination node D, and a pre-established route from S to D which traverses an intermediate node I, a route failure occurs (at node I) if the packets arrive I but the routing protocol can not find any valid route to D (even though there may exist such a route). On the other hand, a route change occurs if the pre-established path from I to D is broken but the routing protocol successfully finds another route and replaces the broken one. We examine below what impact route failures and route changes have on TCP.

First, route failures can cause packet drops at the intermediate nodes, which will be interpreted as congestion loss. When forwarding packets along a pre-established route, if the downstream neighbor of an intermediate node moves out of its transmission range, a link breakage occurs. The outstanding packets have to be queued at this node. A properly designed routing protocol will try to establish another route to the destination. But if no other routes can be found (e.g., in case of network partition), those queued packets have to be dropped. Consequently, if the sender does not receive the acknowledgments of these packets within an estimated amount of time, a timeout event happens and TCP enters the slow-start process as if congestion occurred.

Second, even if the underlying routing protocol can replace the broken routes in a timely manner, route changes can introduce frequent out-of-order packet delivery, which will also confuse the current TCP control mechanism. Using the above example, assume another route is established from S to D after the downstream link from I is broken (the new route may or may not go through I). It is possible that the end-to-end delay on the new route is shorter than the old route, so that a later packet traversing the new route arrives earlier than a packet being forwarded through the old route. This results in an out-of-order packet delivery. The cumulative acknowledgement mechanism of TCP will generate duplicate ACKs before receiving the expected packet in sequence. If the sender receives three of such duplicate ACKs, TCP also presumes the network is congested and invokes fast retransmission.

In this chapter we mainly cover the second and third problems, especially the third problem. We next describe the state-of-the-art approaches to improving the performance of TCP over MANETs.

4. Design Space and Existing Solutions. To overcome the drawbacks of the standard TCP in MANETs, the first step is to distinguish between the real congestion and those other causes, such as link contention and route changes, which would result in the same loss events of timeout and/or 3 dup-ACKs.

4.1. Distinguishing Link Contention from Buffer Overflow. Fu et al has shown in [10] that, in MANETs, there exists an optimal window size W^*, at which TCP achieves highest throughput. This optimal value depends on the network topology and traffic pattern. The existence of such optimal window size is because at most $h/4$ nodes can transmit concurrently on a h-hop chain due to the interference among neighboring nodes. But TCP normally grows its window size W_{avg} much larger than this optimal value.

In contrary to the intuition, this larger window size does not increase TCP throughput, but decreases the throughput instead. This can be explained by the link contention. Since now there are more than W^* packets in the congestion window, and these packets are distributed among the intermediate nodes on the path from source to destination, most nodes have non-empty buffers. Although the buffers are not full, the nodes will try to contend for the wireless channel under the random access control of IEEE 802.11. This eventually generates more link contention among nodes compared with if there are only W^* packets in the window, and in turn reduces TCP throughput due to the reduced spatial reuse.

Two link layer techniques are proposed in [10] to improve TCP performance, Distributed Link RED (LRED) and Adaptive Pacing. LRED assigns a link drop probability to the head-of-line packet in the buffer based on the average number of link layer transmission retries of recent outgoing packets. In IEEE 802.11 link layer, the packets are dropped with probability 1 after exceeding the retry thresholds. In LRED, this drop probability is tuned and adaptive to the recent link contention.

Adaptive Pacing is also a link layer approach. It tries to solve the exposed receiver problem, by adding an additional back-off period of a packet transmission time. The purpose is to reduce the link contention caused by exposed receivers.

4.2. Distinguishing Link Breakages from Congestion. In order to discriminate the packet losses due to link breakages from those caused by buffer overflow, we need to answer the following two questions:

- How can the TCP sender/receiver *detect* the route disruptions?
- How should the TCP sender/receiver *respond* to the route disruptions?

First, to alleviate the performance degradation in MANETs, it is desirable for TCP to be able to distinguish between network congestion and non-relevant events such as route failures and route changes. It is not sufficient to determine from the superficies of packet losses or duplicate ACKs. More detailed detection is necessary in order to find out the real cause behind. This can be done by analyzing the feedback from network or transport layer. The network layer feedback includes the information provided by the underlying ad hoc routing protocols, such as the routing error messages. The transport layer feedback is the information generated by and accessible to the transport protocol, such as the timing and sequence of TCP packets.

Second, appropriate reactions other than congestion control should be taken as soon as the TCP sender discovers that such superficies is not induced by congestion, but by route disruptions. The effectiveness of any performance enhancement approaches highly depends on the timeliness of responses to the route disruptions. Intuitively, the ideal solution to the route disruption problem is for the TCP to freeze its current state (e.g., to freeze the congestion window size and RTO, etc.) as soon as the route breaks (i.e., the sender stop sending more packets) and resume as soon as a new route is found [5, 12, 18]. However, this requires instant notification or feedback from the network layer at the intermediate nodes to the transport layer at the TCP senders. Such a feedback system can be difficult to implement and expensive to operate. Other simpler, less expensive, but still effective alternatives are fixing the RTO, temporarily disabling congestion control, or instantly recovering during congestion avoidance.

Recently, there have been several approaches proposed to address this problem, all fitting in the detection-response framework, i.e., consisting of both the detection and response parts. We can categorize all existing approaches according to the different detection and response mechanisms of them. We will discuss each of them in the following sections.

5. Detection by Network Layer Feedback. Route changes are triggered by link breakages at some intermediate nodes (possibly the sender itself). Detecting these link breakages is a basic requirement for any ad-hoc routing protocol. If the intermediate nodes, where the breakages happen, can convey this information back to the sender, the TCP controller at the sender will be able to detect the event. We call this a network layer feedback mechanism. The majority of the existing approaches employ this detection mechanism, namely TCP-F (TCP-Feedback) [5], ELFN (Explicit Link Failure Notification) [12], ATCP (Ad hoc TCP) [18], and TCP-BuS [17].

5.1. TCP-F. TCP-F (TCP-Feedback) [5] relies on the network layer at intermediate nodes to detect the route failures due to the mobility of downstream neighbors along the route. TCP-F puts the TCP sender in one of the two states: active state and snooze state. In the active state, TCP sender follows the standard TCP behavior. As soon as an intermediate node detects a link failure, it explicitly sends a route failure notification (RFN) packet to the sender and records this event. After receiving the RFN, the sender enters the snooze state, stops sending further packets and freezes the values of state variables such as retransmission timer and congestion window size. The sender remains in the snooze state until the intermediate node notifies it of the restoration of the route through a route reestablishment notification (RRN) packet. Then it enters the active state again.

Thee link breakage is first detected at the intermediate node. The TCP sender can not detect it until a special RFN packet arrives from the failure point. Similarly, the detection of restoration depends on the special RRN packets from some intermediate nodes. The RFN and RRN packets are transported to the sender by TCP. There is a potential danger of malfunction if the RFN or RRN packets are lost.

5.2. ELFN. ELFN (Explicit Link Failure Notification) [12] is another technique based on feedback. The objective is to provide the TCP sender with information about link and route failures so that it can avoid taking congestion control actions. ELFN is based upon DSR [16] routing protocol. To implement ELFN message, the route error message of DSR was modified to carry a payload similar to the "host unreachable" ICMP message. When a TCP sender receives an ELFN, it disables its retransmission timers and enters a "stand-by" mode, which is similar to the snooze state of TCP-F. Instead of using an explicit notice to signal that a route has been reestablished, a packet is sent periodically to probe the network to see if a route has been established. After finding a new route, the sender leaves the stand-by mode, restores its retransmission timers and continues as normal.

Compared with TCP-F, the detection in ELFN is achieved with the help of modified route error messages, which are forwarded under the control of the routing protocol. The detection is moved up to the transport layer only at the TCP sender. No extra packets, such as RFN and RRN packets, are necessary.

5.3. ATCP. ATCP (Ad hoc TCP) [18] utilizes network layer feedback too. It relies on the network to generate appropriate ICMP host unreachable messages and propagate them back to the TCP sender. In addition to the route failures, ATCP also tries to deal with the problem of high bit error rate. The TCP sender can be put into a persist state, congestion control state or retransmit state. ATCP inserts a thin layer between TCP and IP. It listens to the network state information by monitoring ECN (Explicit Congestion Notification) messages [8] and ICMP "Destination Unreachable" messages, and then puts TCP at the sender into the appropriate state. On receiving a "Destination Unreachable" message, the sender enters the persist state. The TCP at the sender is frozen and no packets are sent until a new route is found, so the sender does not invoke congestion control. The

ECN is used as a mechanism to explicitly notify the sender the network congestion along the route being used. On receipt of an ECN, congestion control is invoked without waiting for a timeout event. If a packet loss happens and the ECN flag is not set, ATCP assumes the loss is due to bit errors and simply retransmits the lost packet.

5.4. TCP-BuS. TCP-BuS [17] is similar to TCP-F in detection mechanisms. Two control messages (ERDN and ERSN) related to route maintenance are introduced to notify the TCP sender of route failures and route reestablishment. These indicators are used to differentiate between network congestion and route failures as a result of node movement. ERDN (Explicit Route Disconnection Notification) message is generated at an intermediate node upon detection of a route disconnection, and is propagated toward the sender. After receiving an ERDN message, the sender stops transmission. Similarly, after discovering a new partial path from the failed node to the destination, the failed node returns an ERSN (Explicit Route Successful Notification) message back to the sender. On receiving ERSN message, the sender resumes data transmission.

TCP-BuS considers the problem of reliable transmission of control messages. If a node A reliably sends an ERDN message to its upstream node B, the ERDN message subsequently forwarded by node B can be overheard by A (assuming same transmission ranges of A and B). Thus, if a node has sent an ERDN message but can not overhear any ERDN message relayed by its upstream node during a certain period, it concludes the ERDN message is lost and retransmits it. The reliable transmission of ERSN is similar.

To summarize, these mechanisms all rely on the intermediate nodes, where the route failures are detected, to send some control messages to notify the TCP sender. We categorize and call them the *network layer feedback* mechanisms.

6. Detection by Transport Layer Feedback. It is also possible to detect route disruptions by looking at the timing and sequence information of TCP packets. We call this the transport layer feedback mechanism. There are two known approaches which employ this detection mechanism: the consecutive timeouts heuristic [7] and TCP-DOOR [22].

6.1. Consecutive Timeouts Heuristic. Since routes are likely to be broken frequently in mobile environments, routing algorithms for MANETs are designed to repair broken routes quickly. To take advantage of this capacity, it is reasonable to let a TCP sender retransmit the unacknowledged packets at periodic intervals rather than having to wait increasingly long periods of time between retransmissions.

The fixed RTO [7] technique adopts this idea and does not rely on the feedback from lower layers. In fact, a heuristic is employed to distinguish between route failures and congestion. When timeouts occur consecutively, the sender assumes a route failure has happened rather than network congestion. Here, the detection of route failures is achieved when the TCP sender encounters the second successive retransmission timeout. This detection may be not correct. But presumably, the probability of route failures is higher than congestion when consecutive timeouts occur, especially in highly dynamic environments.

6.2. Detection through Out-of-Order Packet Delivery. In an ideal TCP session, the packets sent by one end should arrive at the other end in sequence and in order. However, there are two cases in which this ideal order can be violated. One case is retransmissions due to packet losses – a previously transmitted sequence number has to be repeated later. We call this an Out-of-Sequence event. The other case is Out-of-Order (OOO). It happens when a packet sent earlier arrives later than a subsequent packet.

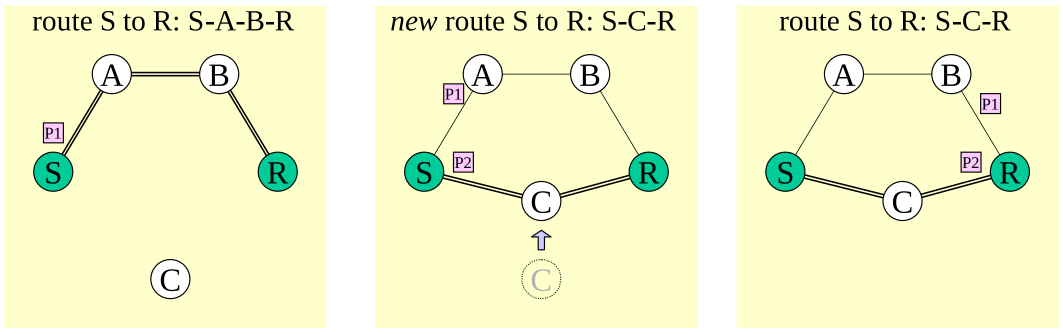


FIG. 6.1. *A Possible Case of Route Change*

OOO often implies route changes in the network. Consider two hosts communicating over a fixed route in the network. Assuming the FIFO queues are used at every node, all packets should be received in the same order as they are sent. However, if the route changes during the communication session so that a later packet P2 takes a different path than an earlier packet P1 (see Fig. 8.1), and if the new route taken by P2 is "faster" or "shorter" than the one taken by P1, P2 can arrive before P1, therefore OOO delivery can happen.

While we have largely ignored the route change events in a wired network because they are such an infrequent event, we can no longer do so with mobile ad-hoc networks. In mobile ad-hoc networks, route changes can happen frequently and multiple times during a TCP session, especially in a highly dynamic environment. Ignoring OOO delivery can have serious performance implication in mobile ad-hoc networks.

Packet losses caused by route changes are not related to network congestion. It is therefore not advisable to invoke congestion control to slow down the sending rate. Unfortunately in TCP, if a data packet is delivered to the receiver three packets later than a subsequent one, a triple-duplicate-ack condition will happen and the sender will halve its congestion window size, halve the slow start threshold, and reset the the retransmission timer (RTO). It can be worse, if the data packet is delivered so out-of-order that the sender times out when waiting for the ACK, which will result in a slow start that is more detrimental to TCP throughput. If such OOO events happen multiple times in one TCP session, the TCP throughput can become deteriorated.

By detecting these OOO events and responding appropriately, we can improve the throughput. Although an OOO delivery may not directly result in triple duplicate ACKs or timeout, the sender still can learn about the current network state of route changes, and possible performance improvement action can be taken to avoid unnecessary TCP slowdown.

In a TCP session, OOO can happen in either direction. Both streams of data packets and ACK packets can be delivered out of order. Accordingly, OOO detection should be carried out by both ends: sender detecting Out-of-Order ACK packets and receiver detecting Out-of-Order data packets.

6.2.1. Sender Detecting Out-of-Order ACK Packets. Every TCP ACK packet carries a sequence number indicating the highest data segment number that the receiver has received consecutively so far. Since there are no retransmissions of an ACK packet (duplicate ACK is not a retransmission), the sequence number of an ACK sent later is impossible to be less than any ACK sent earlier. This non-decreasing property of ACK sequence numbers makes it simple for the sender to detect OOO delivery of non-duplicate

ACK packets: whenever the sender receives an ACK packet, it can compare the sequence number it carries with the one in the previous ACK, and if the current sequence number is smaller, the sender can surely declare OOO.

Detecting OOO delivery of duplicate ACK packets requires additional ordering information, because otherwise two duplicate ACK packets would have the identical content. To achieve this, we propose to add to the TCP ACK header a one-byte TCP option, called *ACK duplication sequence number* (ADSN). When TCP receiver sends first ACK packet for a data segment, the ADSN option is set to zero. Whenever it sends out a duplicate ACK for the same sequence number, it increments the ADSN number. Under the extremely rare condition that ADSN ever reaches 256, it will be wrapped back to zero. This way, each duplicate ACK will carry a different ADSN field and the TCP sender can compare this field to detect an OOO delivery.

6.2.2. Receiver Detecting Out-of-Order Data Packets. The method of comparing sequence numbers does not apply to detecting OOO data packets, because TCP retransmission can cause a data packet with lower sequence number to arrive after one with higher sequence number. To reliably detect OOO data packets, we must include strict ordering information in the TCP data packet. One way to achieve this is to add to the TCP header a two-byte TCP option, called *TCP packet sequence number* (TPSN). Starting from zero, and incremented with every data packet sent (including retransmissions), this TPSN records the exact order of the data packet stream. This is different from the normal TCP sequence number because the latter refers only to the data segment stream – a retransmitted packet always carries the old data segment sequence number. Similar to TCP sequence number wrapping, TPSN wraps around 2^{16} as well. With TPSN option, the TCP receiver can detect OOO reliably.

Another method that can facilitate OOO detection without needing a new TCP option is the use of TCP timestamp option [15]. When this option is used, the TCP sender records the precise time each packet is sent in the TCP packet header. Since TCP timestamp option is already used in many TCP algorithms that operate on fine-grain timers, when it is available, the TCP receiver can compare the timestamp in each packet with the previous one to detect OOO.

7. Response Mechanisms. When route failure or route change events are detected, TCP can respond accordingly to avoid the performance degradation. In case of route failures, congestion control by slow-start does not help to make better use of the network, but in fact potentially reduce the performance by backing off the retransmission timer. The optimal reaction should be to freeze TCP, i.e., the sender stops sending more packets. For route changes, it depends on the link utilization of the newly established routes whether or not the congestion control is appropriate. If the new routes are less utilized, it is too pessimistic to reduce the congestion window or slow down the sending rate. Instead, a better choice could be to disable these congestion control actions in a certain period of time. Another possibility is, if the congestion control has been invoked, to recover instantly from a previous state.

Since TCP sender bears the burden of congestion control, the response actions should mainly take place at the sender. Therefore, if TCP receiver or some intermediate node detects such events, it should notify the sender, for example, by setting a special flag bit in the TCP ACK packet. Once the TCP sender receives such notification, or it detects such events directly from the ACK stream, it can accordingly take the following response actions: freezing TCP, fixing the RTO, temporarily disabling congestion control, or instantly recovering during congestion avoidance.

7.1. Freezing TCP. Freezing TCP mainly involves two steps:

1. Completely Stop sending further packets (new or retransmissions).
2. Freeze all timers, the sending window of packets, and the values of state variables such as RTO and cwnd.

All the four approaches based on network layer feedback (TCP-F, ELFN, ATCP, TCP-BuS) employ this response mechanism. The detailed operations are slightly different, particularly how to recover from the frozen state. We next describe several design options in this mechanism.

When an intermediate node detects a link breakage (this can be done at the MAC layer as seen in section 3.2), the first design choice to make is, how to propagate the link breakage information to the TCP sender. One option is to take advantage of the route error messages of underlying routing protocol, as in ELFN. The other option is to send a special notification packet to the sender, as in TCP-F and TCP-BuS. Furthermore, during the propagation process from the failure node to the sender, each intermediate node may attempt to repair the route locally and stop the propagation if the repair is successful.

Upon receiving the link breakage notification from the failure node, the TCP sender should immediately stop sending more packets even if the congestion window allows to do so, since the route being used is broken. Accordingly, the sender should also stop the retransmission timers, freeze related state variables, especially RTO and cwnd. The TCP state of the sender is kept frozen until a new valid route is established. The period of the sender being frozen may be long or short, which is dependent on the network topology and nodes movement.

For the TCP sender to leave the frozen state, it can wait for another notification from an intermediate node that a new valid route is established. Or more actively, it can probe the network periodically with a short HELLO-type message until it learns of a valid route. TCP-F and TCP-BuS use the former method, while ELFN uses probing.

However, after the TCP sender leaves the frozen state and resumes the transmission, it is not a simple decision to make what values the state variables should take. Intuitively, the state variables can continue with their frozen values. An alternative is, as proposed in ATCP, to assign 0 to TCP's receiver advertised window and assign 1 segment to congestion window size. The argument is that, it may result in congestion at some intermediate node if using the previous congestion window for the new route. Hence when TCP resumes transmission, it will enter slow start process.

The advantage of freezing TCP is, the sender can react very quickly to the link failure (almost in real-time, i.e., as soon as it is detected), and therefore minimizes the number of packet losses and subsequent delays. This benefits from the timely network layer feedback.

However, the disadvantage of relying on network layer feedback is, it results in a reinvention of feedback support in each new ad hoc routing protocol. On the contrary, transport layer feedback results in a reusable mechanism for all ad hoc routing protocols. Additionally, the reliable transmission of the feedback to the TCP sender further complicates the approaches based on network layer information. Next, we describe three response mechanisms that do no rely on the network layer feedback and hence avoid these problems.

7.2. Fixed RTO. Fixed RTO [7] is a very simple responding mechanism, originally coming from the consecutive timeouts heuristic. If the sender encounters two consecutive retransmission timeouts, it assumes some events other than congestion happen. Then the value of retransmission timeout is fixed, without incurring exponential backoff. The RTO remains fixed until the route is re-established and the retransmitted packet is acknowledged.

This simple technique is particularly effective when network partition happens. Without fixing the RTO, it will become longer and longer exponentially, which implies that the chance to probe a valid route is smaller and smaller.

An improved approach is, not only to fix the RTO, but also to reset it to the initial value which is a short time period. In other words, it is better to probe the network frequently after a network partition is believed to have happened in order to avoid wasting time idling.

7.3. Disabling Congestion Control. This response mechanism is from TCP-DOOR [22]. Since we have mentioned that OOO is likely caused by route changes and not by congestion, one way to avoid the undesired congestion control effects is for the TCP sender to temporarily disable congestion control action whenever an OOO condition is detected. That is, for a time period T_1 after detecting an OOO, TCP sender will keep its state variables constant, such as the retransmission timer (RTO) and the congestion window size. Setting the length for this disabled period is a tradeoff.

As we have noticed earlier, this is not guaranteed to be appropriate. If the newly established routes after the link breakages are also much utilized or even more congested than the old routes before the breakages, congestion control actually should not be disabled. To be even more conservative, slow start may be invoked as suggested in ATCP. In general, if the traffic in a MANET is not heavy, disabling congestion control can improve the TCP performance (as proven by the supportive results later); otherwise, congestion control or even slow start should be invoked.

7.4. Instant Recovery. This is also from TCP-DOOR [22]. The detection of OOO condition implies that a route change event has just happened. Therefore, if during the past time period T_2 the TCP sender has suffered from “congestion symptoms” (such as gross timeout or three duplicate ACKs) and entered the congestion avoidance state (such as halving its window size), it should recover immediately to the state before such congestion avoidance action was invoked. (A similar idea was first suggested by Sally Floyd [9].) The rational for this is the following. If the congestion indication event was a gross timeout, it was likely caused by the temporary disruption during some route change. The fact that the TCP sender has begun receiving ACK means that the disruption was over and the TCP should quickly resume the pre-route-change state, without going through slow start or linear window recovery. Similarly, if congestion avoidance was triggered by a three-duplicate-ACK event, these duplicate ACKs were likely caused by OOO deliveries or temporary route disruption, not by congestion losses. And likewise, this condition would have passed and TCP should resume its previous state.

8. A Summary of Performance Studies. We have seen several approaches in previous three sections, all of which fit in the detection-response framework very well. Although it is by no means a complete list of existing approaches, we believe they are representative with respect to possible design choices. Table 8.1 summarizes the detection and response mechanisms of these approaches.

Following is a summary of the performance studies of these approaches. Our purpose is not to compare them among each other. Rather we want to briefly show how well each approach performs in terms of improvement over standard TCP. The results are cited from the papers where these approaches are proposed.

8.1. TCP-F. In [5], the authors simulated a single unidirectional bulk transfer session. The network was viewed as a *black box.* There were two simulation parameters, failure rate

TABLE 8.1
Detection and Response Mechanisms of Each Approach

Approach	*Detection*	*Response*
TCP-F	Network Feedback	Freezing TCP
ELFN	Network Feedback	Freezing TCP
ATCP	Network Feedback	Freezing TCP
TCP-BuS	Network Feedback	Freezing TCP
Fixed RTO	Consecutive Timeouts	Fixing RTO
TCP-DOOR	Out-of-Order Delivery	Disablling CC + Instant Recovery

(number of route failures in the total simulation time) and route re-establishment delay (RRD). This is a very simplified simulation model. Figure 8.1 shows the throughput of TCP-F vs. TCP under different RRDs when there were 10 route failures in 100 seconds of simulation time. TCP-F outperformed standard TCP when RRD was long.

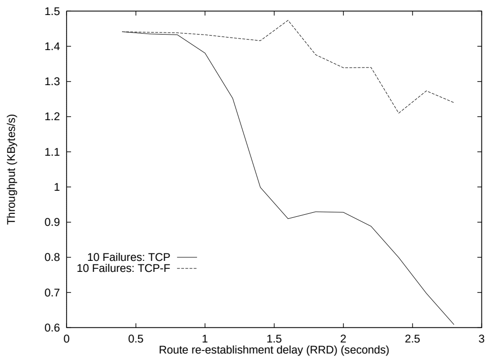


FIG. 8.1. *Throughput of TCP-F vs. TCP for 10 failures with data rate of 12.8 Kbps [5]*

8.2. ELFN. In [12], the authors introduced a notion of *expected throughput*, which is an upper bound of TCP throughput and used as a comparison basis to measure the performance of ELFN. The simulation was done by using *ns* simulator. There were 30 mobile nodes in a 1500mx300m simulated area. The nodes followed the *random waypoint* mobility model with 0 pause time. Figure 8.2 shows the throughput of ELFN and base TCP as a percentage of the expected throughput under several mean speeds. Five probing intervals were simulated (recall that in ELFN, TCP sender probes the network in order to leave the frozen state). We can see that, with a reasonable value (i.e. neither too small nor too large) of probing interval, ELFN outperformed base TCP.

8.3. ATCP. In [18], ATCP was implemented in FreeBSD kernel. Its performance was studied using an exprimental testbed. The testbed consisted of five PCs which formed a four hop network. A bit error rate of 10^{-5} was used for all experiments. To emulate network partitions, a node was configured to periodically generate ICMP host unreachable messages. Packet reordering was simulated by simply manipulating the order of the packets in the queue. Figure 8.3 gives a comparison between ATCP and TCP when there were both bit errors and packet reordering. Instead of throughtput, the data being compared was the transfer time of a 1MB file.

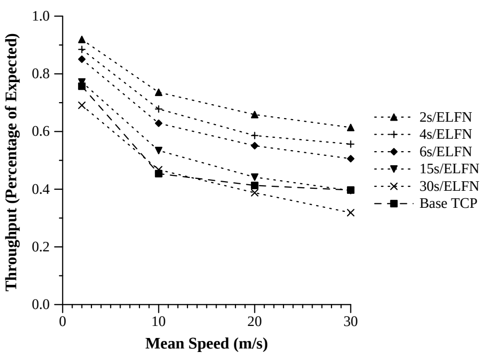


FIG. 8.2. *Throughput of ELFN vs. base TCP [12]*

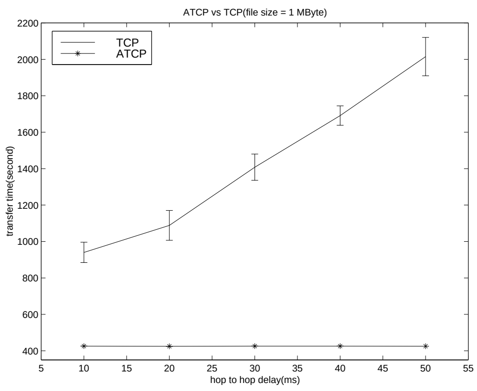


FIG. 8.3. *File Transfer Time of ATCP vs. TCP [18]*

8.4. TCP-BuS. In [17], the authors studied the performance of TCP-BuS using their own simulator which only implemented routing and transport protocols. The topology was simply a chain of 20 nodes, which were separated from one another by 50 meters. One link was randomly chosen to be disconnected in order to simulate route failures. In figure 8.4, the delivery ratios of TCP-BuS, TCP-F and base TCP were copmpared. TCP-BuS outperformed both TCP-F and base TCP in terms of delivery ratios for all frequencies of route failures.

8.5. Fixed RTO. The simulator used in [7] was *ns2*. The authors simulated a network of 50 nodes over a 1000mx1000m area. The mobility pattern was based on *random waypoint* model with 0 pause time. The mean node speed was 10 m/s. Besides TCP connections, there was background traffic of 10 or 40 constant bit rate (CBR) flows. It was reported that fixed-RTO yielded about 70% gain in DSR throughput when there were 10 CBR flows. In case of AODV routing protocol, the gain was about 8% for 10 CBR flows and increased to 48% for 40 CBR connections.

8.6. TCP-DOOR. The performance study in [22] was also done in *ns2* simulator. The network consisted of 20 nodes in a 1000mx750m area. Again, the nodes moved freely according to *random waypoint* pattern with 0 pause time. The moving speed was randomly chosen between 0 and 10m/s. Both non-congested (without background traffic) and

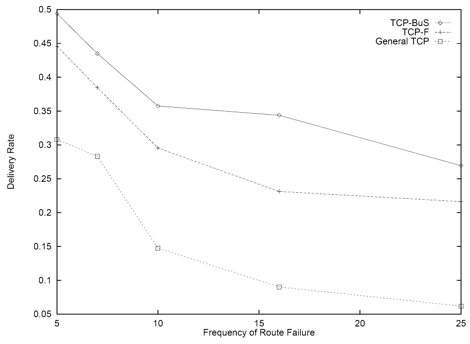
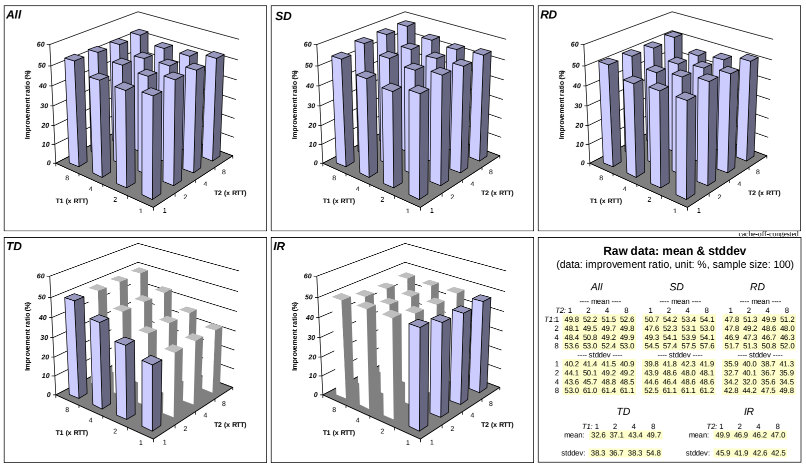


FIG. 8.4. *Delivery Ratios of TCP-BuS, TCP-F and base TCP [17]*

congested (with CBR flows as background traffic) conditions were studied.



Raw data: mean & stddev
(data: improvement ratio, unit: %, sample size: 100)

All, *SD*, *RD* — mean

T1 \ *T2:*	1	2	4	8	1	2	4	8	1	2	4	8
1	49.8	52.2	51.5	52.6	50.7	54.2	53.4	54.1	47.8	51.3	49.9	51.2
2	48.1	49.5	49.7	49.8	47.6	52.3	53.1	53.0	47.8	49.2	48.6	48.0
4	48.4	50.8	49.2	49.9	49.3	54.1	53.9	54.1	46.9	47.3	46.7	46.3
8	53.6	53.0	52.4	53.0	54.5	57.4	57.5	57.6	51.7	51.3	50.8	52.0

---- stddev ----

	1	2	4	8	1	2	4	8	1	2	4	8
1	40.2	41.4	41.5	40.9	39.8	41.8	42.3	41.9	35.9	40.0	38.7	41.3
2	44.1	50.1	49.2	49.2	43.9	48.6	48.0	48.1	32.7	40.1	36.7	35.9
4	43.6	45.7	48.8	48.5	44.6	46.4	48.6	48.6	34.2	32.0	35.6	34.5
8	53.0	61.0	61.4	61.1	52.5	61.1	61.1	61.2	42.8	44.2	47.5	49.8

TD

T1:	1	2	4	8
mean:	32.6	37.1	43.4	49.7
stddev:	38.3	36.7	38.3	54.8

IR

T2:	1	2	4	8
mean:	49.9	46.9	46.2	47.0
stddev:	45.9	41.9	42.6	42.5

FIG. 8.5. *DSR-Cache-off with Congested condition [22]*

In the simulation, DSR was used as the underlying routing protocol. In order to eliminate the effect of DSR route cache and isolate the performance gain of TCP-DOOR, the route cache was turned off in one set of the simulations. The improvement ratio of TCP-DOOR over base TCP is shown in figure 8.5 for the case of congested condition and the route cache being turned off. The average improvement ratio was about 50%.

REFERENCES

[1] H. Balakrishnan and R. Katz. Explicit loss notification and wireless web performance. In *Proceedings of IEEE Globecom - Internet Mini-Conference*, Sidney, Australia, Nov. 1998.
[2] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of ACM SIGCOMM'96*, Stanford, California, Aug. 1996.
[3] H. Balakrishnan, S. Seshan, and R. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 1(4), Dec. 1995.
[4] L. Breslau, et.al. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[5] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A feedback-based scheme for improving TCP performance in ad hoc wireless networks. *IEEE Personal Communications Magazine*, 8(1):34–39, Feb. 2001.

[6] CMU Monarch Group. CMU monarch extensions to the NS-2 simulator. URL: http://monarch.cs.cmu.edu/cmu-ns.html.

[7] T. Dyer and R. Boppana. A comparison of TCP performance over three routing protocols for mobile ad hoc networks. In *Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc'01)*, Long Beach, California, Oct. 2001.

[8] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):8–23, Oct. 1994.

[9] S. Floyd. Re: TCP and out-of-order delivery. Message ID <199902030027.QAA06775@owl.ee.lbl.gov> to the end-to-end-interest mailing list, Feb. 1999.

[10] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, M. Gerla. The Impact of Multihop Wireless Channel on TCP Throughput and Loss. In *Proceedings of IEEE Infocom 2003.*

[11] M. Gerla, K. Tang, and R. Bagrodia. TCP performance in wireless multihop networks. In *Proceedings of IEEE WMCSA 1999.*

[12] G. Holland and N. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *Proceedings of ACM MobiCom'99*, Seattle, Washington, Aug. 1999.

[13] IEEE 802.11 WG. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specification. Standard, IEEE, Aug. 1999.

[14] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM'88*, Stanford, California, Aug. 1988.

[15] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. IETF RFC 1323, May 1992.

[16] D. Johnson, D. Maltz, Y.-C. Hu, and J. Jetcheva. The dynamic source routing protocol for mobile ad hoc networks (DSR). IETF Internet-Draft, draft-ietf-manet-dsr-06.txt, work in progress, Nov. 2001.

[17] D. Kim, C.-K. Toh, and Y. Choi. TCP-BuS: Improving TCP Performance in Wireless Ad Hoc Networks. *Journal of Communications and Networks*, Vol. 3, No. 2. Jun. 2001.

[18] J. Liu and S. Singh. ATCP: TCP for mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 19(7):1300–1315, July 2001.

[19] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. IETF RFC 2018, Oct. 1996.

[20] C. Perkins, A. Myles, and T. Watson. Mobile IP. In *Proceedings of International Telecommunications Symposium*, 1992.

[21] C. Perkins and E. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, Feb. 1999.

[22] F. Wang and Y. Zhang. Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response. In *Proceedings of MobiHoc'02*, Lausaen, Swizterland, Jun. 2002.

[23] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks? IEEE Communi.Magazine, pp.130-137, June 2001.