Improve TCP Performance in Ad Hoc Networks

Jianxin Zhou [*] Bingxin Shi Ling Zou Hui Shen

Abstract

Standard TCP misinterpret mobility loss in Ad hoc network as congestion loss, thus, it reduce the TCP performance by invoking unnecessary congestion control action. In this paper, we propose two approaches, simELFN (an variation of TCP-ELFN) and TCP-FSR (an variation of TCP-F). They can distinguish the essence of packet loss and avoid multiple consecutive *dupACK*s. Analyses and simulations show that they can achieve better TCP performance in Ad hoc network.

Keywords*: Ad hoc Networks; simELFN; TCP-FSR; TCP performance.*

1 Introduction

Ad hoc network is a novel wireless network model, which is an autonomous system of a group of mobile routers, and associated hosts, connected by wireless links. The union of these wireless links can form an arbitrary graph[1]. Nodes within each other's radio range can communicate with each other directly via wireless links, while those far apart can talk to each other in a multi-hop routing fashion by using other nodes as relays. With the popularity of mobile computing device and wireless network, the research of ad hoc network has attracted much attention recently. Most works were focused on the development of routing protocols because of the importance of 'routing', and not much attention was paid to the improvement of TCP performance over Ad hoc network, albeit in fact, it may enhance the network performance dramatically, which will be shown latter in this paper.

TCP, which provides application layer with reliable connection-oriented packet transmission service over an unreliable underlying IP layer, is a vital component of the Internet protocol suite. Presently, TCP used in Internet, which is referred as standard TCP[1] in this paper, is designed for wireline network. It assumes that all packet losses are due to the network congestion, and immediately invokes congestion control action to alleviate the congestion by reducing sending rate when it detects a packet loss. In other words, standard TCP cannot distinguish congestion from packet loss due to transmission errors or route failures. Standard TCP can work efficiently in wireline network since the latter situations happen very rarely there.

Nevertheless, on the other hand, in an Ad hoc network, data packet losses happen frequently. This is partially due to its using error-prone wireless links as transmission medium. The effect of these packet losses can be reduced by using reliable link layer protocols [2]. However, there is another reason of packet loses which is much harder to deal with: the route failure, which occurs frequently and unpredictably during the lifetime of a transport session, due to the relative motion of nodes in Ad hoc network. During the period of route failure, none of the data packets can reach

[*] Dept. of Electronics & Information Eng., HuaZhong University of Science and Technology, Wuhan, People's Republic of China, 430074 Email: socall92@hotmail.com

[1] In this paper, 'standard tcp' is Reno.

the destination through the existing route, and may result in packet loss. Thus, the TCP source may take the following actions[3]:

1. Retransmit unacknowledged packets upon timing out.
2. Invoke congestion control action that include exponentially backoff the retransmission timers and immediate shrinking of the congestion window size, thus reducing the transmission rate.
3. Enter a slow start recovery state to ensure that the network congestion has reduced before resuming packet transmission at the normal rate.

But if these packet losses are due to route failure, previous actions are undesirable for the following reasons:

1. When there is no route available, there is no need to retransmit packets that will anyway not reach the destination.
2. Packet retransmission wastes precious battery power and scarce bandwidth of mobile node.
3. In the period immediately following the re-establishment of the route, the throughput will be unnecessarily low as a result of the slow start recovery mechanism even though there is actually no congestion in the network.

Most of the current approaches for improving TCP performance, such as TCP-F[3] and TCP-ELFN[4], etc., use feedback from network/link layer of intermediate mobile nodes to reduce the effect on TCP performance by route failure. The network/link layer at an intermediate mobile node sends explicitly a RFN (Route Failure Notification) packet to the TCP source as soon as it detects a route failure. On receiving RFN, the TCP source will freeze TCP transmission and keep the current TCP state according to the RFN; and the TCP source will resume TCP transmission from the original state before route failure after detecting route re-establishment. However, they may cause multiple consecutive *dupACKs*(duplicated Acknowledgements) in the period immediately following the route re-establishment. Consequently, TCP source invoke ‘Fast Retransmit/Fast Recovery’[5] action, reducing the TCP performance needlessly. The approaches proposed by this paper, simELFN and TCP-FSR, aim at avoiding these multiple consecutive *dupACKs*.

2 Assumptions

For the forthcoming discussions, we make the following assumptions:

1 simELFN and TCP-FSR chooses DSR as its routing protocols. Though they are applicable upon any ‘on-demand’ routing protocols, most former works, such as TCP-F and TCP-ELFN, are based on DSR. To ensure fair comparison with former approaches, we choose DSR as routing protocol in this paper too.

2 DSR is disabled to reply route request from route cache. This assumption bases on the result of the simulation, by Holland etc. [4], that it will reduce TCP performance when reply request from route cache for the effect of stale route information in route cache.

3 RFN and RRN (Route Re-establishment Notification) are assumed never be lost when there is a route to their destination. This paper focuses on the effect of TCP performance by route failure in Ad hoc network, but much less attention on transmission errors.

3 Related researches

3.1 DSR routing protocol

DSR[6] is an 'on-demand' routing protocol developed by CMU researchers. It employs source routing wherein the source determines the complete sequence of nodes through which a packet is to be routed. Whenever a TCP source has a packet to transmit, it checks its route cache for a route to the destination. In case no route is found, then a 'route request' broadcast is initiated. On receiving this request, each node again broadcasts this request by appending its address to the request packet until this packet reaches the destination. Then the destination sends a route reply to the source containing the route from the source to the destination. When the route reply reaches the source, a connection is established and all subsequent packets contain the complete route in the packet header. DSR will broadcasts 'route error' message when it detects a route failure due to the motion of downstream neighbor. A packet which has no suitable route to it's destination temporarily will be reserved in send_buf during the 'route discovery' period until it is sent out or dropped by DSR for its remaining in send_buf longer than SEND_TIMEOUT.

3.2 TCP-ELFN

TCP-ELFN[4] (Explicit Link Failure Notification) is a technique based on feedback. The objective is to provide the TCP source with information about link and route failures so that it can avoid responding to the failures as if congestion happened. *TCP-ELFN* is based upon DSR routing protocol. To implement *ELFN* message, the route failure message of DSR was modified to carry a payload similar to the '*host unreachable*' ICMP message. When a TCP source receives an *ELFN* message, it freezes its retransmission timers and enters a '*stand-by*' mode. A packet is sent periodically to probe the network to see if a route has been established. After receiving a new *ACK* (imply the route has been reconstructed), the source leaves '*stand-by*' mode, restores its retransmission timers and continues to transmit packet as normal.

3.3 TCP-F

TCP-F[3] relies on the network layer at intermediate nodes to detect the route failures. In TCP-F, a TCP source can be in two states, 'active' state and 'snooze' state. Transmission is controlled by the standard TCP when TCP source is in 'active' state. The intermediate node will explicitly sends the TCP source a Route Failure Notification (RFN) if it detects a link failure. After receiving the RFN, the TCP source will go into the 'snooze' state by stopping sending any further packet and freezing the value of TCP state variables such as retransmission timer and congestion window size. The TCP source remains in the 'snooze' state until it is notified of the restoration of the route through a Route Re-establishment Notification (RRN) from an intermediate node and then goes back to 'active' state.

3.4 ECP-ELFN

ECP-ELFN[7] is an variation of TCP-ELFN. It uses the feedback to notify source route failure too. But the congestion control mechanism of ECP-ELFN is based on hop-by-hop rate control, not base on window control, which make it is top-priority when a transmission require little jitter of sending rate. However, as other rate-based congestion control algorithms, ECP-ELFN is slow-responsive to network congestion, so it has more possibility to cause congestion or worsen the degree of network congestion. Because of different mechanisms, in simulations of this paper, we didn't compare the performance of ECP-ELFN with our approaches.

4 simELFN

4.1 Description of simELFN

simELFN utilizes network layer feedback too. It can based upon any ‘on-demand’ routing protocol (In this paper, we use DSR as example). In *simELFN*, The TCP source can be in two state, ‘norma*l*’ and ‘froze’. In ‘normal’ state, TCP source’s behavior is the same as standard TCP; in ‘froze’ state, TCP source’s retransmit timers and *cwnd* is frozen and it stops sending out packet. In order to avoid *dupACKs* by ‘probe’ packets, *simELFN* doesn’t send ‘probe’ packet periodically in ‘froze’ mode. *simELFN* operation can be described as follows:

step 1: TCP source is in ‘normal’ state. It transmits packets as standard TCP.

step 2: While TCP source receives a *ELFN* packet with ‘host unreachable’ message, it enter into ‘froze’ state from ‘normal’, freezes retransmit timers and *cwnd*, and invokes a *frozeTimer*. The timeout of timer is:

$$\text{SEND\_TIMEOUT} - \alpha * \text{RTT} \qquad\qquad 0 \le \alpha \le 1, \ \text{ is a constant;}$$

The value of timeout is set as SEND_TIMEOUT-α*RTT, so TCP can immediately enforcedly defrost when all flying packets and *ACK*s are dropped by DSR agent from it’s *send_buf*.

step 3: TCP source remains in ‘froze’ state until it receives a new *ACK*, which imply route has been reconstructed. Then, TCP source is defrosted, enters ‘normal’ state from ‘froze’. Return to step 1, transmit packet as normal, and cancel the *frozeTimer*.

Since any TCP source state variables (such as *cwnd*, *ssthresh*, *t_seqno_*, and *last_ack*, etc) is not modified during TCP source is in ‘froze’ state, it can return the original state (*slow start* or *congestion avoidance*) before route failure when TCP is resumed.

Known from 3.1 section, *DSR* Agent will reserve the packet without route in *send_buf* as long as SEND_TIMEOUT, so the packet or *ACK* can be transmitted correctly after route reconstruction if the route failure period is shorter than SEND_TIMEOUT.

step 4: The *frozeTimer* is timeout (TCP still be ‘*froze*’ state), then TCP is defrosted enforcedly, enter ‘normal’ state from ‘froze’. And return to step 1.

This step is an important part of *simELFN*, it guarantee TCP from being locked in ‘froze’ state. If route failure period is longer than SEND_TIMEOUT, all flying packets and *ACKs* will be dropped by *DSR* agent as TCP remain in ‘froze’ state. Without this step, TCP source will wait an *ACK* to leave ‘froze’ state and resume the normal TCP transmission, while TCP receiver will not send a new ACK until it receive a packet. So TCP will be locked in ‘froze’ state even route has been reconstructed under this scenario .

Fig. 1 gives a finite state machine of TCP in *simELFN*. Contents in the box is the same as standard TCP.

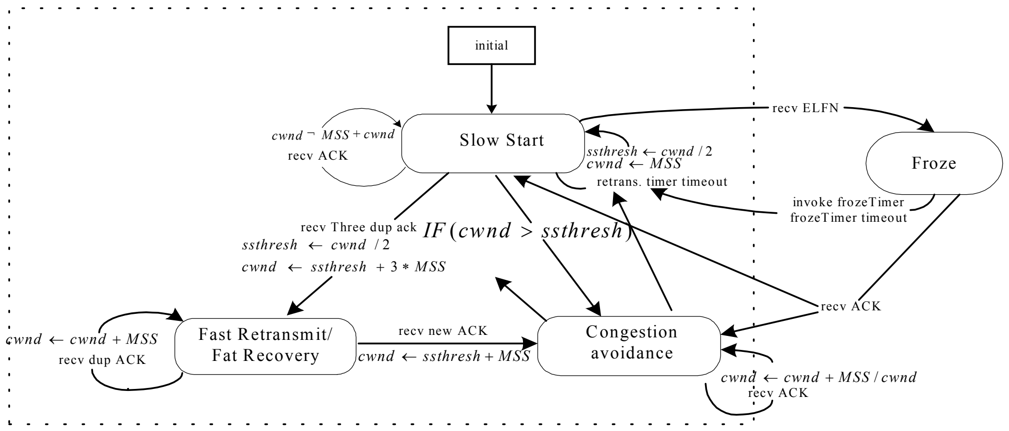


Fig.1: finite state machine of TCP in simELFN

4.2 Analyses of simELFN

TCP-ELFN can distinguish the essence of packet loss through explicit feedback. It will freeze TCP if the packet loss is caused by route failure. And it can resume TCP transmission from the original state, not always from 'slow start', after route reconstruction, which make TCP utilizes bandwidth efficiently and improve TCP performance. However, the 'probe' packet pumped by TCP source during route failure period may cause multiple consecutive *dupACK*s after route reconstruction, which will make TCP invokes '*Fast Retransmit/Fast Recovery*' action and decrease the *ssthresh* and *cwnd*. It certainly will reduce TCP performance, which is not our wishes.

Like *TCP-ELFN*, *simELFN* can distinguish essence of packet loss through explicit feedback, avoid invoking unnecessary congestion control action, and it will avoid causing multiple consecutive *dupACK*s like *TCP-ELFN*, i.e., avoid invoking '*Fast Retransmit/Fast Recovery*', so it can improve TCP performance over Ad hoc network more efficient than *TCP-ELFN*. And *simELFN* has the following characters:

1. Simple. TCP source needn't send 'probe' packet during it be in 'froze' state, which can save the energy and bandwidth of mobile nodes.
2. Reliable. Timeout of *frozeTimer* is the finally mechanism to resume TCP transmission, which guarantee TCP from being locked in 'froze' state.
3. Efficient. If route failure period is short, performance of *simELFN* is better than *TCP-ELFN* since it is free of multiple consecutive *dupACK*s.

If route failure period is longer than SEND_TIMEOUT, TCP source rely on frozeTimer timeout to resume TCP transmission. *simELFN* is degraded to Reno. In this scenario, performance of *TCP-ELFN* is better than *simELFN*, although route failure period is scarcely longer than SEND_TIMEOUT.

4.3 Simulations of simELFN

Simulation results in this paper are based on Ns2[8]. All mobile nodes use IEEE 820.11 as MAC protocol, and their radio communication range is 250 meters.

The topology for simulating *simELFN* is shown in Fig. 2. There are four mobile nodes, with a TCP connection between node 1 and 4 which are fixed in the whole simulation period. A *FTP* application is on node 1, starting sending data at 10sec. The route between node 1 and 4 is failed and reconstructed due to the movement of node 2 and 3.

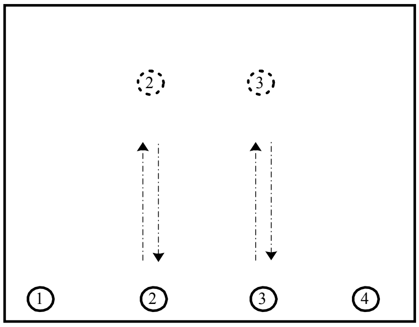
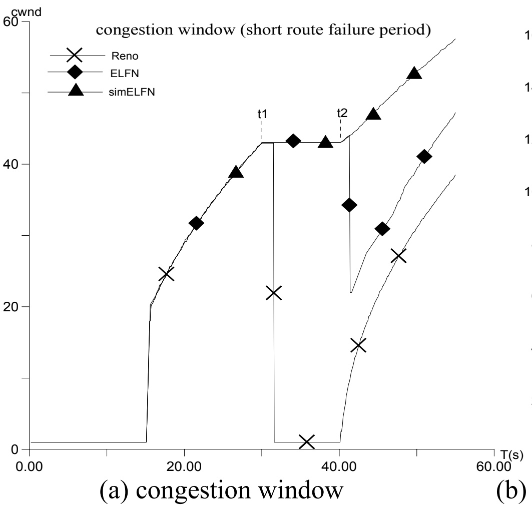
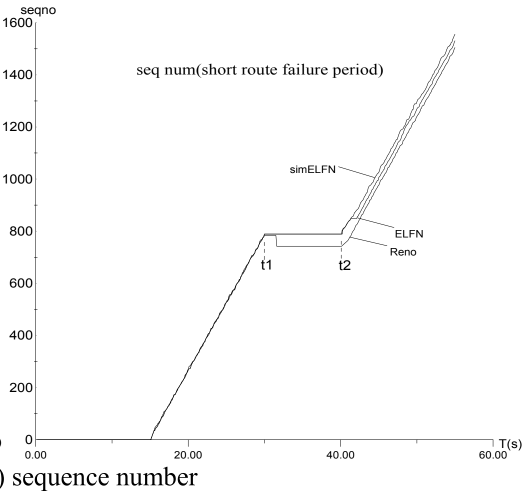


Fig. 2: topology of simulation

Fig. 3 shows the simulating result of short route failure period (shorter than SEND_TIMEOUT - α*RTT), part (a) gives the result of *cwnd* and part (b) gives the result of *seqno*. In Fig. 3, route is failed at **t1**, and is reconstructed at **t2**. From Fig. 3, The result(includes part (a) and (b)) of *Reno*, *TCP-ELFN* and *simELFN* is similar completely before **t1**, which implies that *simELFN* transmits packet as standard TCP when there is no route failure. From part (a), route is failed at **t1**, then *Reno* invokes congestion control action and set *cwnd* to 1. However *TCP-ELFN* and *simELFN* enter 'froze' state after receiving *ELFN* message from intermediate nodes and keep states of TCP connection. When route between node 1 and 4 is reconstructed at **t2**, *Reno* resumes TCP transmission from *Slow Start*, while *TCP-ELFN* and *simELFN* resume TCP from original state before route failure. However, *TCP-ELFN* invokes '*Fast Retransmit/Fast Recovery'* because of multiple consecutive *dupACK*s. From part (b), since it resume TCP transmission from original TCP state and avoid invoking '*Fast Retransmit/Fast Recovery*', *simELFN* gets the best TCP performance. *SimELFN* achieves about 5% improvement based on *Reno*, and 2% improvement based on *TCP-ELFN*. The improvement of TCP will be significant when route is failed frequently.



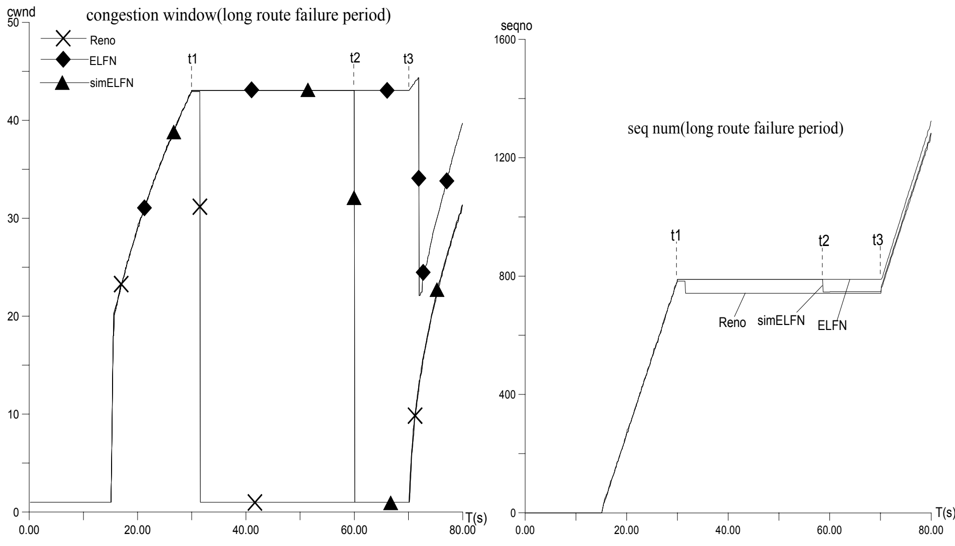
(a) congestion window



(b) sequence number

Fig. 3: simulation result of short route failure period

Fig. 4 shows the simulating results of long route failure period (longer than SEND_TIMEOUT - α*RTT), part (a) gives the result of *cwnd* and part (b) gives the result of *seqno*. In Fig. 4, route is failed at **t1**, *frozeTimer* of *simELFN* timeout at **t2**, and route is reconstructed at **t3**. From part (a), after route is failed at **t1**, *Reno* invokes congestion control action and set *cwnd* to 1. However *TCP-ELFN* and *simELFN* enter 'froze' state. At **t2**, *frozeTimer* timeout, TCP of *simELFN* enter 'normal' from 'froze' state enforcedly, and *simELFN* is degraded to *Reno*. When route between node 1 and 4 is reconstructed at **t3**, *Reno* and *simELFN* resume TCP transmission from *Slow Start*, while *TCP-ELFN* resumes TCP from original state before route failure. part (b) shows that *TCP-ELFN* gets the best TCP performance, *Reno* and *simELFN* get the same TCP performance.



(a) congestion window (b) sequence number

Fig. 4: simulation result of long route failure period

The results of Fig. 3 and Fig. 4 are quite the same as our analyses in section 4.2.

5 TCP-FSR

5.1 Limitations of TCP-F

Based on Fig. 5, the limitations of TCP-F are described in this section. Because of the random motion of its nodes, Ad hoc network is often suffered from route failure, and sometimes even multiple link failures occur simultaneously along a route.

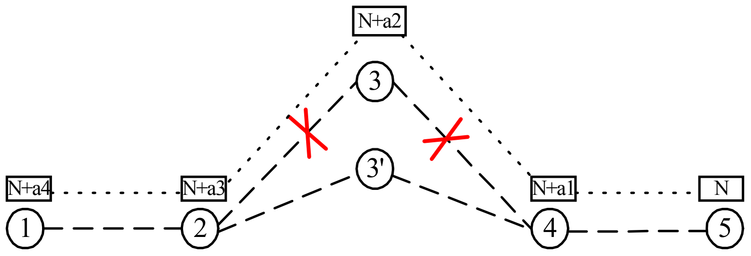


Fig. 5: Topology of network

In Fig: Route between 1 and 5 is failed at node 2-3 and node 3-4 at ***t1*** due to mobility of node 3

Route between 1 and 5 is re-established at ***t4*** when node 3' come into communication range of node 2 and 4

$\boxed{\text{N}}$ represent packet of sequence number being N

a1<a2<a3<a4; all of them is integer

As shown in Fig. 5, there is a TCP connection between node 1 (TCP source) and 5 (TCP receiver). Route between node 1 and 5 is failed at the links between nodes 2-3 and nodes 3-4 at time **t1** due to the motion of node 3. Then TCP-F will send RFNs back to TCP source respectively from intermediate node 2 and 3, and these RFNs carry information of sequence number being N+a2+y and N+a1+x respectively(assume that node 2 and 3 have sent out the packet of sequence number being N+a2+y-1 and N+a1+x-1 at **t1**). However, only the RFN from node 2 can reach the TCP source, while the one from node 3 is dropped since no route for it to the TCP source. The TCP source enters into ‘snooze’ state as soon as it receive the RFN from node 2 and record N+a2+y as the sequence number of next packet to transmit. After a period of route re-establishment delay, route between node 1 and 5 is re-established at **t4** as node 3’ enter into communication range of node 2 and 4. Since DSR is disabled to reply route request from route cache, route requests are always replied by receiver (node 5). TCP-F will send a RRN to the TCP source from node 5 after it replies route request. TCP source resumes transmission after receiving RRN. However, as mentioned above, since the TCP source only receive the RFN from node 2 after route is failed at **t1**, TCP transmission will be resumed from an incorrect state (packet of sequence number N+a2+y), which will cause multiple consecutive dupACKs of sequence number N+a1+x. Consequently, the TCP source invoke ‘Fast Retransmit/Fast Recovery’ action. Table 1 describes the operations of TCP-F in detail according to time.

From above description, it is clear that TCP-F cannot alleviate satisfactorily the effect on TCP performance by multiple link failures along route at same time in Ad hoc network. So a new approach, TCP-FSR, is worthy to be proposed in this paper for this problem.

Table 1: comparison of operations of TCP-F and TCP-FSR

Time	TCP-F	TCP-FSR	Note
t1	Links of node 2-3 and node 3-4 are failed due to mobility of node 3		
t2 = t1+ $\Delta T1$	Node 2 and 3 detect route failure, and send RFNs back to TCP source. These RFNs carry the information of sequence number of being transmitted packet (being N+a1+x and N+a2+y respectively)		$\Delta T1$ <<RTT; N+a1<N+a1+x<N+a2<N+a 2+y<N+a3
t3 = t2+ $\Delta T2$	TCP source receives RFN from node 2, then freeze TCP and keep the N+a2+y as the sequence number of the next packet		$\Delta T2 \leq RTT / 2$
t4 = t3+ $\Delta T3$	Node 3' come into communication range of node 2 and 4. Then, route between node 1 and 5 is re-established. After replying route request, node 5 will		$\Delta T3$ is route re-establish delay, it is undetermined. 'route reply' and RRN are sent by receiver (node 5) since DSR is disabled to reply route request from route cache
	Send RRN to TCP source	send RRN to TCP receiver	
t5 = t4+ $\Delta T4$		After receiving RRN, TCP receiver will send the ACK of the newest data packet (sequence number is N+a1+x) to the TCP source	Since RRN is transmitted only between different layer at the same node (node 5) , $\Delta T4$ is almost ignorable.
t6 = t5+ $\Delta T5$	The TCP source receive RRN, then TCP resume transmission from packet of sequence number N+a2+y		$\Delta T5 \approx RTT / 2$. since $\Delta T4$ is almost ignorable, the TCP source of TCP-FSR can receive new ACK at t6 after route re-establish.
		The TCP source receive the new ACK after route re-establishment. Then, TCP resume transmission from packet of sequence number N+a1+x	
T7 = t6+ $\Delta T6$	TCP source receive multiple consecutive dupACKs of sequence number N+a1+x, then it invoke 'Fast Retransmit/Fast Recovery', and retransmit from packet of sequence number being N+a1+x		$\Delta T6$ =RTT
T8 = t7+ $\Delta T7$	After 'Fast Recovery', TCP source enter 'Congestion Avoidance' phase, reducing congestion window size. And continue TCP transmission from packet of sequence number N+a1+x+J		$\Delta T7$ =RTT J is the number of dupACKs received by TCP source during 'Fast Recovery' phase.

5.2 Description of TCP-FSR

In TCP-FSR, TCP source can also be in ‘froze’ or ‘normal’ state. When in ‘normal’ state, TCP source is controlled by the standard TCP too, and transmit data packet as standard TCP. While in ‘froze’ state, TCP source stop sending any further packet and freeze the state of retransmit timer and congestion window size. Its operations can be described as follows:

Step 1: TCP source is in ‘normal’ state when there is no route failure. It transmit packet as standard TCP.

Step 2: When a node detects a route failure due to relative motion of downstream nodes, TCP-FSR send a RFN which carry information of sequence number of the last packet being transmitted before route failure (As in Fig-5, sequence number carried by RFN of node 2 and 3 are N+a2+y and N+a1+x respectively) to TCP source.

Step 3: On receiving the RFN; If it is in ‘normal’ state, the TCP source enters into ‘froze’ state and executes following actions:

I. Completely stops sending further packets(new or retransmissions);
II. Freezes retransmit timer and value of congestion window size;
III. Keeps current value of some TCP states, such as, setting t_seqno_ as sequence number carried by the RFN;
IV. Starts a routes failure timer which deals with the worst case of route re-establishment. The timeout value of this timer depends on the underlying routing protocol. Since TCP-FSR is based on DSR protocol, the timeout of route failure timer is set as the SEND_TIMEOUT which is the longest period of time of DSR agent keeping packet without route to destination.

If it is in ‘froze’ state, then the TCP source update the value of TCP states;

Step 4: TCP-FSR sends a RRN to TCP receiver after route is re-established.

Step 5: On receiving the RRN, the TCP receiver sends TCP source the latest ACK immediately.

Step 6: If TCP source receive an ACK in ‘froze’ state, which implies the re-establishment of route, it will unfreeze and enter into ‘normal’ state. Back to step 1 and continue to transmit packet as standard TCP from the TCP state kept by step 3, and cancel route failure timer.

Step 7: If route failure timer expires while TCP source is still in ‘froze’ state, TCP source will unfreeze forcibly, entering into ‘normal’ state and letting congestion control mechanism of standard TCP to handle this route failure. Back to step 1.

Table 1 gives the operations of TCP-FSR according to time.

Fig-6 shows the schematic figure of state machine of TCP-FSR.

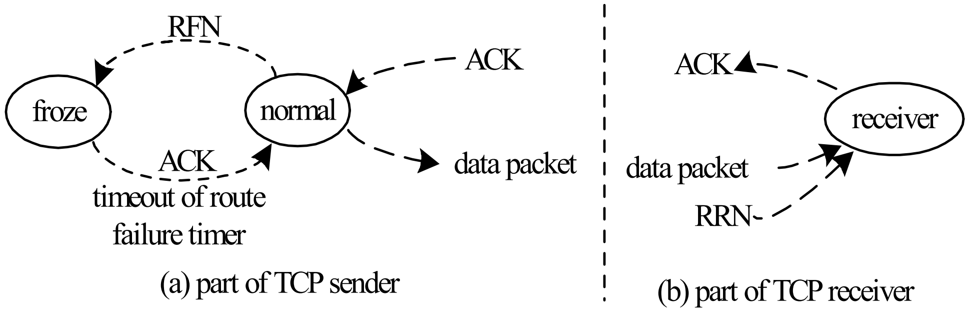


Fig. 6: schematic Figure of state machine of TCP-FSR

5.3 Analyses of TCP-FSR

Known from Table-1 and above discussions, TCP-FSR is much more efficient than TCP-F in improving TCP performance. It takes TCP-FSR a shorter time, about 2*RTT, than TCP-F to resume TCP transmission as normal after route re-establishment when multiple link failures are occurred along the route. And TCP-FSR avoids invoking ‘Fast Retransmit/Fast Recovery’ in the period immediately following route re-establishment. However, these advantages are based on the assumptions of section 2 and some special scenarios. Now, we will compare the performance of TCP-F and TCP-FSR under other situations:

- DSR doesn’t be disabled to reply route request from route cache. In this case, route request may be replied by intermediate node (not always by TCP receiver), and similarly, RRN may be sent out by intermediate node. Therefore, the $\Delta T4$ (transmission delay of RRN), which is between 0 and RTT/2 and thus with an average of RTT/4, in Table-1 can not be ignored anymore. Consequently, TCP-FSR will be about 7/4*RTT (2*RTT-1/4*RTT) earlier than TCP-F to resume data transmission as normal. But it will incur some undesired effect on TCP performance by stale route information in route cache.
- There is only one link failure along the route. In this case, TCP-F can resume TCP transmission as normal at **t6** in Table-1. The performance of TCP-FSR is same as TCP-F’s.
- Route re-establishment delay is longer than SEND_TIMEOUT. In this case, the route failure timer expires. Then, TCP-F and TCP-FSR let congestion control mechanism of standard TCP to handle this route failure. The performances of TCP-F and TCP-FSR are same as standard TCP’s;

There is no route failure. The behaviors of these three techniques are same under this situation and therefore they have equivalent TCP performances.

5.4 Simulations of TCP-FSR

Experiment 1: The network topology of experiment 1 is shown in Fig. 5. There are 5 mobile nodes in a 500 meters square. Among them, node 1, 2, 4 and 5 are fixed during the whole simulation, while node 3 can move lengthways. There is a TCP connection between node 1 and 5, with node 1 being the TCP source and node 5 being the TCP receiver. A FTP application on node 1 start to transfer data to node 5 at 10sec. From 30sec, we introduce 5 times route failures by moving node 3, and let the route persist shortly each time after it is re-established. The result of this simulation is given in Fig. 7, with part (a) showing the result of congestion window size and part (b) showing the result of sequence number.

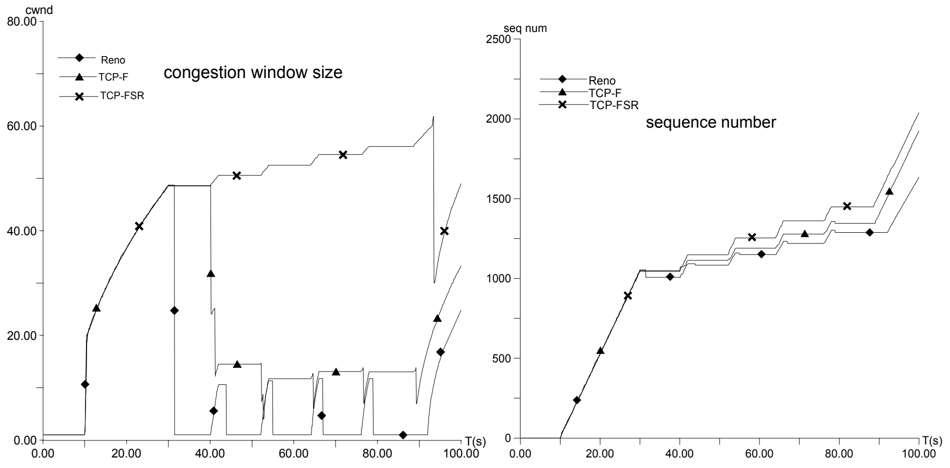


Fig. 7: (a) congestion window size (b) sequence number

From Fig. 7, it is clear that the results in congestion window size and sequence number of Reno, TCP-F and TCP-FSR are almost same before 30sec, which indicates their performance are almost same when there is no route failure. While, there is much difference among their performance since the route is failed. In the period from the first route failure to the re-establishment of last one, Reno always stays in ‘slow start’ phase, which reduces TCP performance dramatically. What is more, since Reno back off RTT value exponentially when re-transmission timer expires, it is the last one to resume TCP transmission when route is re-established after the last route re-establishment, as shown in Fig. 7 (a). On the other hand, by entering ‘froze’ state and keeping TCP state when route is failed, TCP-F and TCP-FSR can resume TCP transmission when the route is re-established from the original state before route failure. Nevertheless, due to multiple consecutive dupACKs, TCP-F invokes ‘Fast Retransmit/Fast Recovery’ in the period immediately following the route re-establishment, which lower its performance significantly. However, since it resume TCP transmission from original state before route failure and avoid invoking ‘Fast Retransmit/Fast Recovery’, TCP-FSR achieves the best TCP performance. Quantitatively, from part(b), wee can see that the sequence numbers of Reno, TCP-F and TCP-FSR are 1054, 1052 and 1052 respectively at 30sec, and they are 1453, 1922 and 2039 at the end of simulation, which means TCP-FSR achieve 40% and 6% improvement based on Reno and TCP-F respectively in whole simulation period. If we just consider their performance after route failure (30sec), then Reno, TCP-F and TCP-FSR transmitted 399, 870 and 985 packets from 30sec to the end of simulation respectively, which implies TCP-FSR achieve 150% and 15% improvement of TCP performance based Reno and TCP-F respectively. The improvement of TCP performance by TCP-FSR is obvious and significant.

Experiment 2: Network topology of experiment 2 consists of 30 nodes in a 1500m by 300m rectangular. The nodes move randomly. This experiment is aimed to investigate the correlation between TCP performance and node’s mean speed, since the motion of nodes is known as the main reason of route failure. Using the scenario generator tool of Ns, setdest, 50 different movement scenarios are generated respectively for cases with node’s mean speed of 2m/s, 5m/s, 10m/s, 15m/s, 20m/s and 30m/s. The result of averaged throughput is showed in Fig. 8. Apparently, TCP-FSR achieves the highest averaged throughput. And throughputs of Reno, TCP-F and TCP-FSR drop as mean speed is increased, but the decrease is much slight after mean speed reach 20m/s.

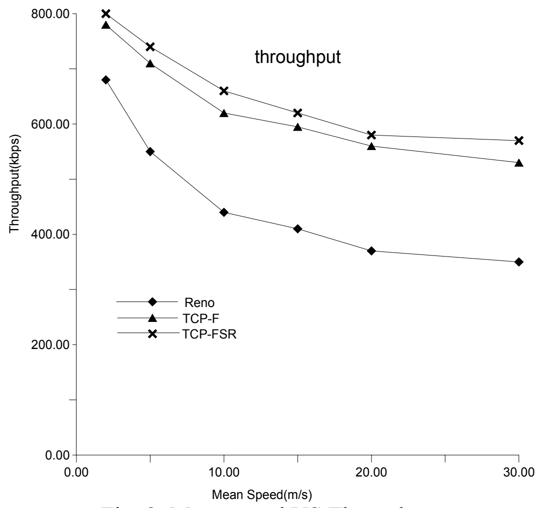


Fig. 8: Mean speed VS Throughput

6 Conclusions

In this paper, we propose two window-based TCP congestion control approachess, simELFN and TCP-FSR, to improve TCP performance in Ad hoc networks. They can distinguish the essence of packet loss and avoid causing multiple consecutive *dupACK*s, achieving better performance than TCP-ELFN and TCP-FSR respectively, and both of them get a higher performance than Reno. Simulation results prove their validity.

There are still some open issues of TCP performance in Ad hoc networks. For example, how to deal with the loss of the explicit notification packets (such as RFN, RRN, ELFN), how to re-compute TCP state when reconstructed route is different from the original one, and rate-based congestion control mechanism in paper [7], using heuristic way to distinguish essence of packet loss in paper [9], i.e. without using explicit feedback notification, etc.

References

[1] http://www.ietf.org/html.charters/manet-charter.html

[2] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, “A comparison of mechanisms for improving TCP performance over wireless links,” in ACM SIGCOMM, Stanford, CA, Aug. 1996

[3] K. Chandran, S. Raghunathan, S. Venkatesan and R. Prakash, "A feedback based scheme for improving TCP performance in ad-hoc wireless networks", International Conf. Distributed Computing Systems, May 1998

[4] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks", Mobicomm'99, Aug. 1999

[5] M. Allman, V. Paxson, W. Stevens. “TCP Congestion Control”. RFC2581. Apr. 1999.

[6] D. Johnson, D.A. Maltz, J. Broch, “The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (draft-ietf-manet-dsr-06.txt),” Mobile Ad-hoc Network(MANET) Working Group, IETF, Nov 2001

[7] J.P. Monks, P. Sinha, and V. Bharghavan, "Enhancements and Limitations of TCP-ELFN for Ad Hoc Networks", Proceedings of The 7th International Workshop on Mobile Multimedia Communications (MoMuC 2000) , Oct., 2000

[8] K. Fall, K. Varadhan, *ns* Notes and Documents, VINT project, http://www.isi.edu/nsnam/ns/ Feb. 2000.

[9] F. Wang and Y. G. Zhang, “Improving TCP performance over Mobile Ad-Hoc Networks with Detection of Out-of-Order and Response (DOOR)”, Proceedings of Mobihoc’02, June. 2002