

Evaluating Different Techniques to Improve TCP Performance over Wireless Ad Hoc Networks

Vinay Sridhara {vsridhar@usc.edu}
Nagendra Subramanya {nsubrama@usc.edu}

Abstract

Wireless channel bandwidth is scarce and also varies significantly during transmission due to mobility or some obstacle. Base TCP does not distinguish between congestion and packet losses due to transmission errors, which are very prevalent in wireless networks. Mobility of nodes results in frequent route re-computations and sometimes network partitions. Discovering new routes takes significantly longer timer than the RTO interval at the sender. As a result, the TCP sender times out, retransmits the packet and invokes congestion control. If route re-computations happen frequently (due to high mobility), the sender TCP will never get an opportunity to transmit at the maximum negotiated rate (because CWND will be significantly smaller than the receiver's advertised window size).

ATCP suggests a solution to improve the performance of TCP in wireless ad hoc networks. But, they tested their implementation without using wireless cards and any ad hoc routing protocols. When ATCP was implemented and tested using wireless cards and ad hoc routing protocols, there was not much performance improvement. But, ATCP performs well when the network is partitioned. In this paper we discuss why ATCP does not improve TCP's performance in certain cases and discuss possible solutions.

1. Introduction

Wireless channel bandwidth is scarce and also varies significantly during transmission due to mobility or some obstacle. The bandwidth actually decreases in slabs and not gradually when the mobile hosts move farther away. This is generally done to increase the availability and reliability of the connection. Base TCP does not distinguish between congestion and packet losses due to transmission errors (high bit error rates), which are very prevalent in wireless networks. Mobility of nodes results in frequent route re-computations and sometimes network partitions. Discovering new routes takes significantly longer timer than the RTO interval at the sender. As a result, the TCP sender times out, retransmits the packet and invokes congestion control. If route re-computations happen frequently (due to high mobility), the sender TCP will never get an opportunity to transmit at the maximum negotiated rate (because CWND will be significantly smaller than the receiver's advertised window size). It is also likely that an ad hoc network gets periodically partitioned and the sender and receiver of a connection lie in different partitions. Then all the sender's packets get dropped resulting in the sender invoking congestion control. If the situation persists for a few seconds, there could be multiple RTO expirations and the RTO may reach its upper bound. Finally when the receiver and sender get connected, it could take several seconds before there is some kind of transmission. In ad hoc networks, some routing protocols like TORA and DSR, maintain multiple routes between the sender and receiver in order to reduce the frequency of route re-computation. This sometimes may result in a number of out-of-sequence packets arriving at the receiver. In response, the receiver generates duplicate ACKs that may cause the sender to invoke congestion control (on receiving the third duplicate ACK).

Some of the solutions to improve the performance of TCP over wireless ad hoc networks are TCP-F, ELFN and ATCP. In the following sections we will provide brief details about three ad hoc routing protocols [DSDV, DSR, AODV], TCP-F and ELFN.

2. Ad Hoc Routing Protocols

This section provides a brief insight into few ad hoc routing protocols – DSDV, DSR and AODV. Since we used AODV as the routing protocol in our experiments, this section should be useful to understand few issues that we will be discussing later.

2.1 DSDV (Destination-Sequenced Distance Vector)

DSDV is a hop-by-hop distance vector routing protocol requiring each node to periodically broadcast routing updates. The key advantage of DSDV over traditional distance vector protocols is that it guarantees loop-freedom. Each DSDV node maintains a routing table listing the “next hop” for each reachable destination. DSDV tags each route with a sequence number and considers a route **R** more favorable than **R'** if **R** has a greater sequence number, or if the two routes have equal sequence numbers but **R** has a lower metric. Each node in the network advertises a monotonically increasing *even sequence number* for itself. When a node **B** decides that its route to a destination **D** has broken, it advertises the route to **D** with an infinite metric and a sequence number one greater than its sequence number for the route that has broken (making an odd sequence number). This causes any node **A** routing packets through **B** to incorporate the infinite-metric route into its routing table until node **A** hears a route to **D** with a higher sequence number.

Link layer breakage detection of the 802.11 MAC protocol may be used in obtaining the DSDV data. If a neighbor **N** of a node **A** detects that its link to **A** is broken, it will broadcast a triggered route update containing an infinite metric for **A**. The sequence number in this triggered update will be one greater than the last sequence number broadcast by **A**, and therefore is the highest sequence number existing anywhere in the network for **A**. Each node that hears this update will record an infinite metric for destination **A** and will propagate the information further. This renders node **A** unreachable from all nodes in the network until **A** broadcasts a newer sequence number in a periodic update. **A** will send this update as soon as it learns of the infinite metric being propagated for it, but large numbers of packets can be dropped in the meantime.

2.2 DSR

DSR uses source routing rather than hop-by-hop routing, with each packet to be routed carrying in its header the complete, ordered list of nodes through which the packet must pass. The key advantage of source routing is that intermediate nodes do not need to maintain up-to-date routing information in order to route the packets they forward, since the packets themselves already contain all the routing decisions. This fact, coupled with the on-demand nature of the protocol, eliminates the need for the periodic route advertisement and neighbor detection packets present in other protocols.

The DSR protocol consists of two mechanisms: Route Discovery and Route Maintenance. Route Discovery is the mechanism by which a node **S** wishing to send a packet to a destination **D** obtains a source route to **D**. To perform a Route Discovery, the source node **S** broadcasts a ROUTE REQUEST packet that is flooded through the network in a controlled

manner and is answered by a ROUTE REPLY packet from either the destination node or another node that knows a route to the destination. To reduce the cost of route discovery, each node maintains a cache of source routes it has learned or overheard, which it aggressively uses to limit the frequency and propagation of ROUTE REQUESTs. Route maintenance is the mechanism by which a packet's sender **S** detects if the network topology has changed such that it can no longer use its route to the destination **D** because two nodes listed in the route have moved out of range of each other. When Route maintenance indicates a source route is broken, **S** is notified with a ROUTE ERROR packet. The sender **S** can then attempt to use any other route to **D** already in its cache or can invoke Route Discovery again to find a new route.

2.3 AODV

AODV is essentially a combination of both DSR and DSDV. It borrows the basic on-demand mechanism of Route Discovery and Route Maintenance from DSR, plus the use of hop-by-hop routing, sequence numbers, and periodic beacons from DSDV.

When a node **S** needs a route to some destination **D**, it broadcasts a ROUTE REQUEST message to its neighbors, including the last known sequence number for that destination. The ROUTE REQUEST is flooded in a controlled manner through the network until it reaches a node that has a route to the destination. Each node that forwards the ROUTE REQUEST creates a reverse route for itself back to node **S**. When the ROUTE REQUEST reaches a node with a route to **D**, that node generates a ROUTE REPLY that contains the number of hops necessary to reach **D** and the sequence number for **D** most recently seen by the node generating the REPLY. Each node that participates in forwarding this REPLY back toward the originator of the ROUTE REQUEST (node **S**) creates a forward route to **D**. The state created in each node along the path from **S** to **D** is hop-by-hop state; i.e., each node remembers only the next hop and not the entire route, as would be done in source routing.

In order to maintain routes, AODV normally requires that each node periodically transmit a HELLO message, with a default rate of once per second. Failure to receive three consecutive HELLO messages from a neighbor is taken as an indication that the link to the neighbor in question is down. Alternatively, the AODV specification briefly suggests that a node may use physical layer or link layer methods to detect link breakages to nodes that it considers neighbors. When a link goes down, any upstream node that has recently forwarded packets to a destination using that link is notified via an UNSOLICITED ROUTE REPLY containing an infinite metric for that destination. Upon receipt of such a ROUTE REPLY, a node must acquire a new route to the destination using Route Discovery as described above.

3. Feedback based TCP Scheme

In [TCP-F], a feedback-based scheme is used by which the source is informed of a route failure so that it does not misinterpret it as congestion and get into the congestion control phase. According to the authors, existing feedback-based schemes for TCP, like ECN and EBSN are not applicable as the backbone in ad hoc networks is not reliable. Instead, they suggest a scheme where they notify the source using IP messages, viz., Route Failure Notification (RFN) and Route Reestablishment Notification (RRN).

The protocol is as follows. Suppose the source mobile host (MH) is involved in a bulk data transfer to a destination MH. When the network layer of an intermediate MH (failure point,

FP) detects a route failure due to the mobility of the next MH along that route, it explicitly sends an RFN packet to the source and records this event. Each intermediate node that receives the RFN packet invalidates the particular route and prevents incoming packets intended for the destination from passing through that route. If the intermediate node knows of an alternate route to the destination, this alternate route will be used to support further communication, and the RFN is discarded. Otherwise, the intermediate node records the event and simply propagates the RFN toward the source.

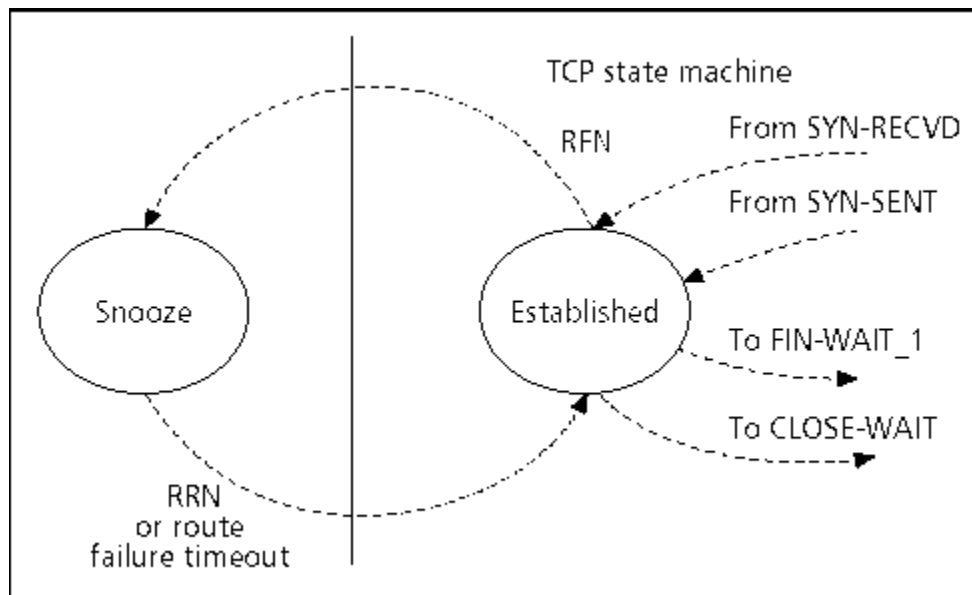


Fig. 1. TCP-F State Machine

On receiving the RFN, the source goes into a snooze state and performs the following:

- Stops sending further new packets and retransmissions
- Invalidates existing timers, freezes congestion window and other state variables.
- Starts a route failure timer that corresponds to a worst-case route reestablishment time (that depends on the underlying routing protocol).

The source then goes into a “snooze” state until it receives a route reestablishment notification (RRN) packet indicating the restoration of the route or the route failure timer times out.

When an intermediate node that had previously forwarded an RFN to the source learns about a new route to the destination, it sends an RRN packet to the source. This intermediate node discards all further RRNs it receives for the same source-destination pair. Other nodes that receive the RRN forward it towards the source.

Once the source receives the RRN, it moves back to active state and flushes all the unacknowledged packets in its current window.

Hence, when route is reestablished, communication resumes at the same rate as before the route failure occurred. The authors feel that this makes sure that there is no unnecessary loss

of throughput during the route-disrupted period and TCP's congestion control mechanism can take over and adjust to the existing load in the system.

The authors suggest enhancements to this scheme. Suppose the intermediate nodes could buffer the packets (those in flight) to a limited capacity. Then, as the RFN propagates to the source from the FP, all the intermediate nodes can temporarily buffer subsequent packets. If the new route overlaps with the old one, the RRN message can be used to flush out the buffers. Similarly, intermediate nodes may forward the buffered packets to the destination without waiting for an RRN when they learn a new route. This scheme saves packet retransmissions and packet flow resumes much before the source learns about the route reestablishment. Also, the buffering overhead at each node will be low, as the buffering will be shared across the intermediate nodes.

The success of TCP-F critically depends on the ability of the intermediate nodes (acting as routers) to detect route failures and reestablishments, and quickly provide feedback to the source. While sending an RFN, each MH has to record the source id for the corresponding RFN so that it can send an RRN when a route is found, thereby overloading the mobile hosts. Instead, the intermediate nodes could just inform the source about the route failure. The source then could keep sending the zero-window-probe packets to the destination till it receives an ACK from it, confirming the reestablishment of the route. The time interval between the probes should be such that they do not congest the network (as suggested in [ELFN]).

RFN and RRN are IP messages and hence unreliable. In case the source does not receive an RFN, it could misinterpret it as congestion. If an RRN is lost, the source moves to active state only when the route failure timer expires, thereby wasting the bandwidth available once the route is reestablished.

On receiving an RRN, the source starts sending packets at the same rate as before the route failure occurred. This could congest the network, as the connection has not yet gauged the capacity of the network using this new route. Hence, the congestion window and the slow-start threshold must be set to their default initial values for better performance.

The authors acknowledge that this scheme does not work well when there are multiple flows. In a real-life situation, the same route/link could be used by many flows. They suggest that, when the FP receives a packet, it sends an RFN in response to the source (assuming that the packet has source and destination addresses). This also ensures that sources presently not using the route do not know about the route failure.

Hence, when the route is reestablished, communication resumes at the same rate as before the route failure occurred. Though this results in no unnecessary loss of throughput during the route-disrupted period, the TCP must probe the network through this new route.

4. Explicit Link Failure Notification

The results put forward by the authors of [ELFN] are based on their simulations where they used TCP-Reno over IP on an 802.11 wireless network and DSR for routing. Firstly, the authors comment that routing protocol has a very significant impact on TCP performance, especially due to caching and propagation of stale routes. In relatively slowly changing topologies, the inability of the TCP sender's routing protocol to quickly recognize and purge stale route from its cache results in repeated routing failures. They suggest the usage of effective

cache maintenance strategies, including simple techniques like dynamically adjusting the route cache timeout mechanism depending on the observed route failure rate, the use of negative route information or the use of signal strength information. Alternatively, replying to route requests from caches could be turned off. But, this only improves performance in a single flow. In a network with multiple data sources, the additional routing information introduced when replies from caches are disabled could degrade TCP performance. Secondly, the authors suggest that instead of augmenting TCP/IP, it would be better to improve the routing protocols so that mobility is more effectively masked. Regardless of the efficiency and accuracy of the routing protocol, network partitioning and delays will still occur because of mobility, which affect the performance.

The authors answer the issue of link failures due to mobility. They suggest a technique that is similar to TCP-F; instead they use only one message called Explicit Link Failure Notification (ELFN) - an IP-level message. The objective of ELFN is to provide the source with information about the link and route failures so that it does not misinterpret such failures as congestion. As they are using DSR, the route failure message was modified to carry a payload similar to the “host unreachable” ICMP message.

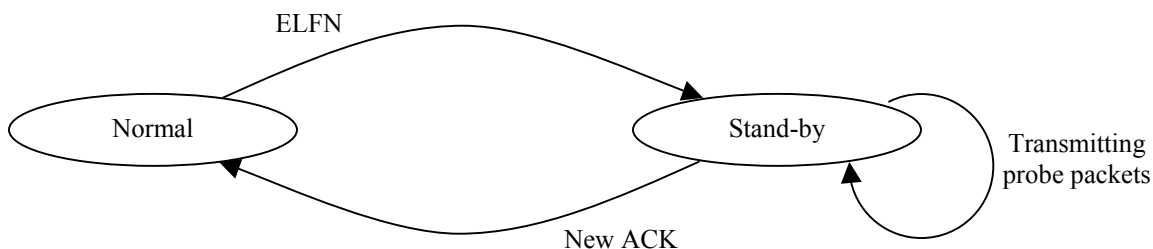


Fig. 2. TCP+ELFN State Machine

The source that receives ELFN disables congestion control mechanisms until the route has been restored, i.e., it disables retransmission timers and enters a “stand-by” mode. In stand-by mode, the source sends probe packets to the network at periodic intervals to see if a route has been established. If an ACK is received, then it leaves “stand-by” mode, restores its retransmission timers and continues as normal.

The authors also suggest some variations to analyze the protocol. In the first variation, the length of the interval between probe packets was varied. They determined that throughput is critically dependent on the time between probe packets. Increasing the interval between probes delayed the route discoveries by the length of that interval. If the interval is too small, the rapid injection of probes into the network will cause congestion and degrade the throughput. The best bet would be to choose the interval that is a function of the RTT.

In the second, the congestion window (CWND) and/or retransmission timeout (RTO) were adjusted after the failed route was restored. When they set CWND to one, it had little impact on the throughput and the authors substantiate it by commenting that the optimal window

(bandwidth-delay product) of the network is relatively small and it takes only a few RTTs to ramp up to the optimal window after route failure. When they set the RTO to its default initial value, it had a significant impact on the throughput. The authors suspect that this could be due to the frequency at which routes break, coupled with ARP's tendency to silently drop packets. When a restored route immediately breaks again and results in a failed ARP lookup, ARP drops the packet and the sender will likely timeout. Given the length of the timeout, it takes few such occurrences to affect the throughput.

Finally, the impact on performance based on the choice of the probe packet was considered – either the first packet in the congestion window or the packet with the lowest sequence number among those signaled as lost in the ELFNs that were received. The presence of different forward and reverse routes equalizes the two methods when only the forward route is disrupted, since those packets that managed to get through before the failure can be acknowledged through the reverse route. In that case, the lowest sequence numbered packet that was lost would also be the first in the congestion window.

According to [ELFN-L], TCP+ELFN is designed mainly for dynamic networks. In static ad hoc networks, if the flows create enough channel contention to cause frequent route error messages, the freezing the network state could lead to reduced throughput.

Even for static networks, routing packets are generated during data transmission as a result of extensive packet delays (may be due to contention caused by same flow) causing nodes to assume link breakages. High mobility of nodes causes frequent route disruptions, resulting in large number of routing packets, which force the TCP packets to reside in the network buffers for significant periods of time, causing many timeouts. Hence, queue management is very crucial for mobile networks because data packets can be significantly delayed or dropped due to changes in network configuration. To reduce queue overflows, congestion control mechanisms with high efficiency and low buffer utilization are needed. And rate control mechanisms achieve these goals. According to [ELFN-L], hop-by-hop rate control mechanisms along with ELFN are better suited for ad hoc networks rather than TCP+ELFN.

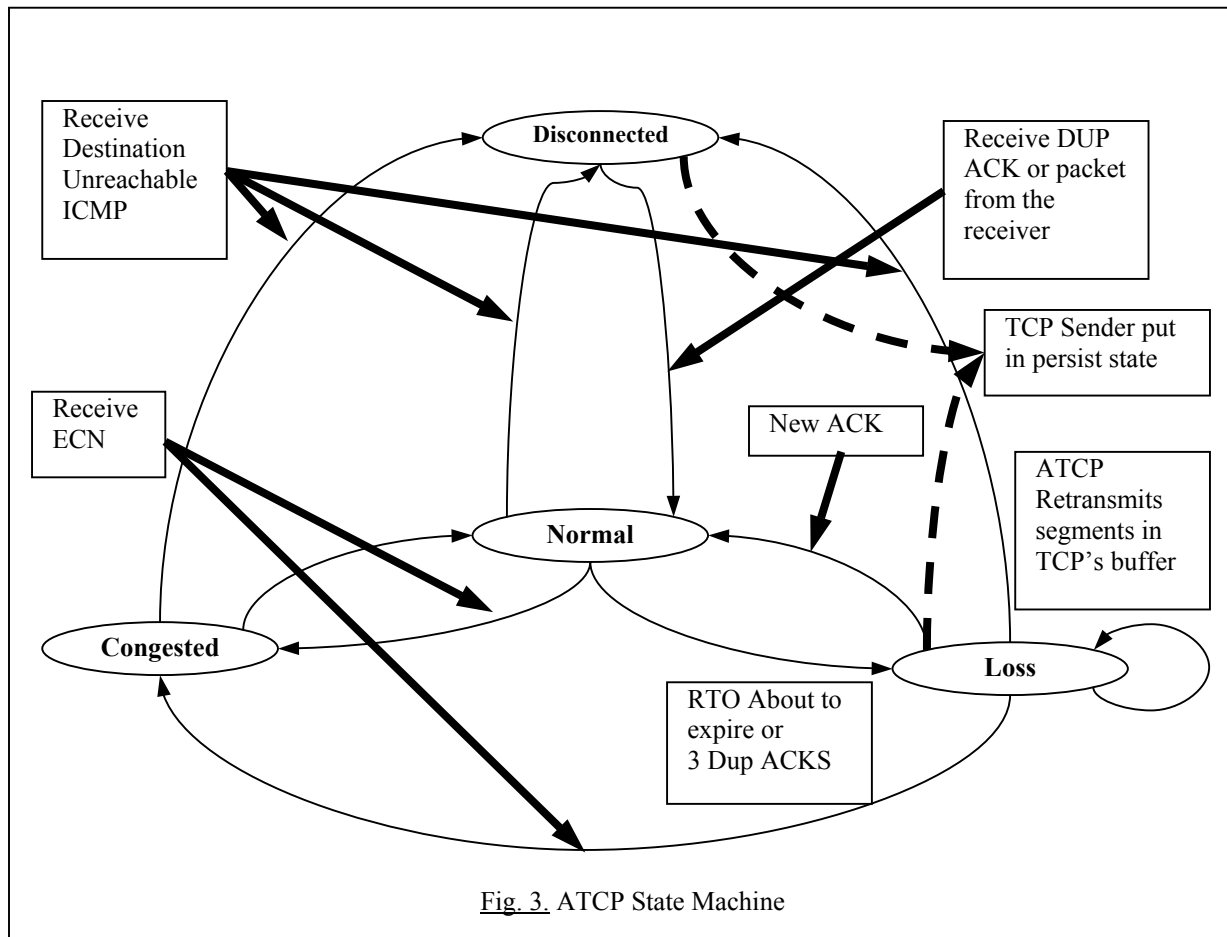
Another problem with ELFN is that it keeps sending the packet that was last transmitted as the probe. Since the size of the TCP packet can be a maximum of 65,535 bytes, this could cause unnecessary congestion at the intermediate routers. Instead of adopting this approach it would be better if smaller probe packets were sent to find if the route has been re-established. Since the power is also limited in the mobile devices, it would consume more power if huge packets were sent out to probe the route re-establishment.

5. ATCP

ATCP is a thin layer between IP and TCP that maintains high end-to-end TCP throughput. This thin layer listens to the network state information provided by ECN messages and ICMP “destination unreachable” messages, and puts TCP at the sender in the appropriate state. The ATCP layer is active only at the TCP sender (in a duplex communication, its active at both participating nodes).

When the TCP connection is initially established, ATCP at the sender is in the “normal” state and does nothing. If the connection from the sender to the receiver is lossy, segments may be lost or arrive out-of-order. The receiver generates duplicate ACKs in response. ATCP, in its

“normal state”, counts the number of duplicate ACKs received for any segment and forwards the first two to TCP. On receiving the third duplicate ACK, it is not forwarded and TCP is put to “persist” mode. Similarly, when RTO expires, ATCP moves TCP to “persist” mode. Hence, TCP sender does not invoke congestion control because both these events correspond to lossy links



wherein both data segments and ACKs could be lost. Then ATCP enters “loss” state and retransmits all the unacknowledged segments from the TCP’s buffer, reusing the TCP’s timers. Finally, when a new ACK arrives ATCP forwards it to TCP, which removes it from “persist” mode and ATCP returns to “normal” state.

When the network gets congested, the sender receives ECNs. Then ATCP moves to “congested state” and does not interfere with TCP’s congestion control behavior. If ATCP is in “loss” state, ATCP removes TCP from “persist” mode. ATCP ignores all the duplicate ACKs and RTO expirations. Once TCP transmits a new segment, ATCP returns to “normal” state.

Node mobility could result in either route re-computation or temporary network partition that result in the generation of ICMP “destination unreachable” messages. When the sender receives this message, ATCP (in “normal”, “congested” or “loss” state) puts TCP into “persist” mode, sets TCP’s CWND to one segment and itself into “disconnected” state. Setting CWND to one segment ensures that TCP probes the network to determine the correct value of CWND using the new route. In “persist” mode, TCP periodically generates “probe” packets, with the

interval between them equal to RTO. When the receiver gets connected to the network, it responds to the “probe” packets with an ACK (or a data packet). This removes TCP from “persist” mode and ATCP goes back to “normal” state. We comment on the limitations of ATCP in a later section.

6. Limitations of ATCP

ATCP had an experimental setup consisting of 5 PCs each with 2 Ethernet cards. They did this to simulate a wireless ad hoc network, where in nodes are separated by multiple hops. They simulated the condition of network partition by implementing an IP timer at a host, which on expiring would generate ICMP "host unreachable" messages. They did not use any ad hoc routing protocol. In a typical wireless ad hoc network, each node will have a routing protocol running which finds a route to the destination of a connection. And it is completely dependent on the routing protocol to conclude when the network is partitioned. Further, the routing protocol has to make the decision about how to inform the TCP layer about the network partition. One way is to just drop the packets, since there is no route. But, this decision must be made by TCP. So a better solution would be for the routing protocol to inform the TCP layer about the network partition. And one way to do that is by sending an ICMP "host unreachable" message to the TCP layer at the sender. Hence, one cannot assume that the routing protocol handles network partition very well. A protocol similar to ELFN may have to be used in that case.

ATCP heavily depends on ECN messages to recognize a congested network. Suppose one of the intermediate nodes (say CN) on experiencing congestion sets the CE bit and forwards the packet. There are two scenarios now. Firstly, at the same time there may be a network partition and the source and destination become part of different partitions. Hence, the packet with CE bit set will not reach the destination and thus the source will not receive an ECN. It takes sometime for the routing protocol to decide that there is a network partition and inform the TCP at source about it. During the duration between creation of network partition and the source's TCP being informed about it, re-transmit timer could have fired quite sometimes. According to ATCP, every time the RTO is about to expire, they retransmit all the unacknowledged packets in the TCP's buffer. This definitely congests the already congested network. Such a flow is definitely misbehaving and will affect the throughput of other flows that are forwarded by CN.

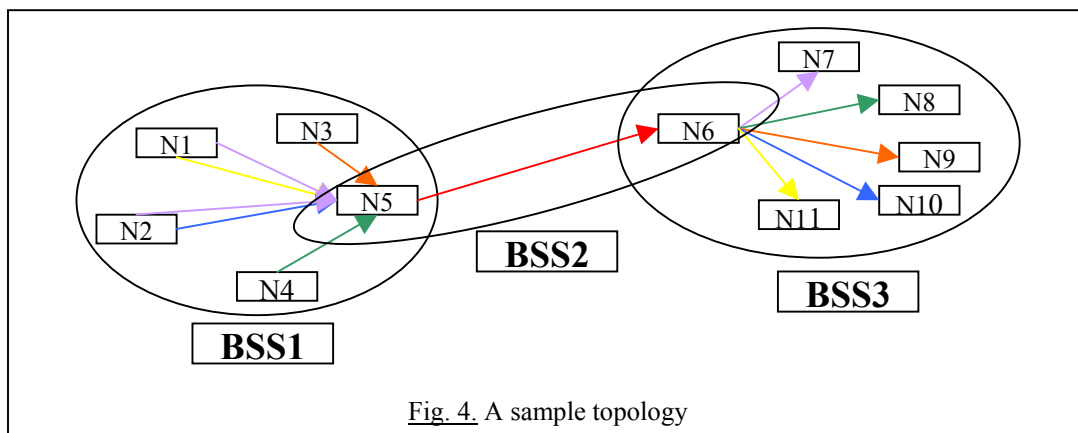


Fig. 4. A sample topology

Secondly, consider the above shown scenario where there are multiple flows that are going through a single bottleneck router (N5 in Fig. 2). When the distance between two nodes changes the data rate drops discretely to a lower value (5.5Mbps, 2Mbps or 1Mbps). Consider the topology as shown in Fig.4.

Wireless links	Data Rate of links in Mbps
N1- N5	5.5
N2- N5	5.5
N3- N5	5.5
N4- N5	5.5
N5- N6	11
N7- N6	5.5
N8- N6	5.5
N9- N6	5.5
N10- N6	5.5
N11- N6	5.5

Table 1. Example Data Rates

Suppose the data rates are as shown in Table 1 and node N5 is using RED algorithm for congestion control and marks a packet with the CE bit in TOS field of the IP header when the queue length equals the maximum threshold value. The congestion windows of the nodes N1, N2, N3 and N4 could have reached very high values and all of them are in congestion avoidance mode. Suppose due to mobility, N6 moves away from N5 and their data rate reduces to 2Mbps. Now there is no mechanism to find this out immediately and take action accordingly. There is a very high possibility that an ECN sent by the receiver will not reach the sender. The packet on which the ECE bit was set could be dropped due to congestion itself. According to ATCP, the sender assumes that the packet loss was due to bit errors and keeps transmitting the packets in the TCP's buffers when the retransmit timer expires. This will increase the congestion severely.

Another issue is that, when ATCP is in "loss" state, all unacknowledged packets in TCP's buffers are retransmitted unnecessarily. Probably the receiver did not receive only one segment and it suffices if only that particular segment is re-transmitted. In an Ad Hoc network, each mobile node is a router and the queue size of the mobile nodes is limited. Hence retransmitting all the packets in the TCP's buffer could also lead to unnecessary congestion in the network. Another drawback is that it consumes more power of all the intermediate nodes unnecessarily.

7. Possible solutions

To handle scenarios when ECNs cannot make it to the sender, we propose an enhancement that times out only a certain number of times and later gives control back to TCP. We maintain a threshold for this purpose. This threshold needs to be fixed by more experiments. Another enhancement is that when ATCP is in the "loss" state, only the lost packet be re-transmitted.

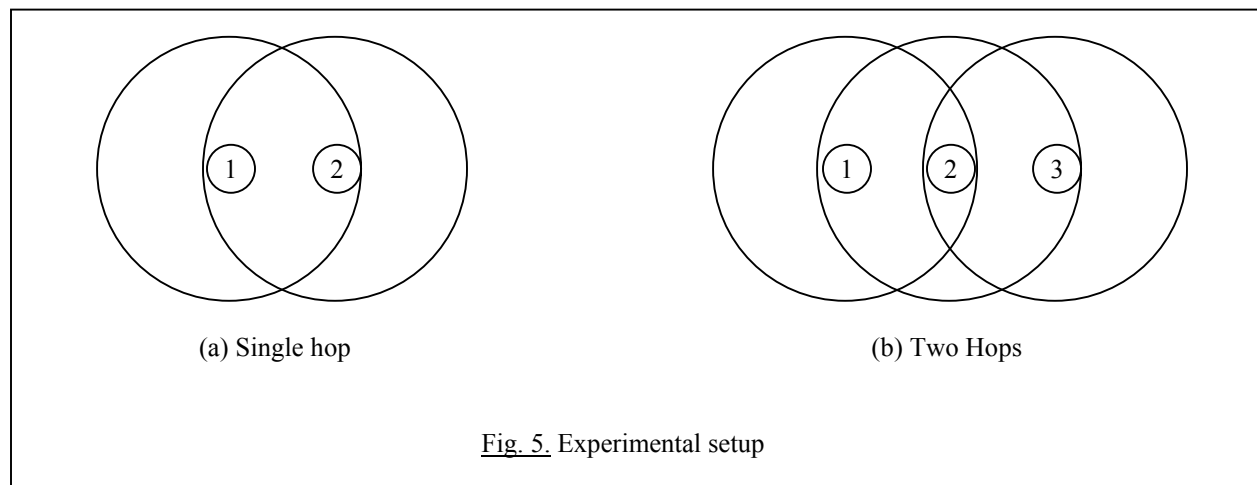
For discovering the route re-establishment faster we propose that Probe timer should back off linearly rather than exponentially. Since the size of the probe packet is just the size of the header it will not really congest the network.

When we detect a packet loss, either due to 3 duplicate acknowledgements or due to the retransmission timer expiration, we transmit only the packet that was lost and not the whole queue. Transmitting the whole queue will unnecessarily overload the network which might actually lead to congestion. Another disadvantage of sending the whole queue is that it will reduce the good put of the TCP. Power in the mobile devices is always limited. Hence transmitting the whole queue of packets will consume the power unnecessarily.

8. Experimental setup

Our experimental setup consisted of 3 wireless nodes (laptops with wireless cards). All the nodes used the AODV (Ad-hoc On-demand Distance Vector) routing protocol. The kernel on all the nodes was Linux with TCP New Reno. All the nodes will also have RED (Random Early Drop) queuing enabled on their wireless interfaces to support ECN.

We modified the New Reno TCP for our experimental procedure in the Linux kernel. We modified the kernel to handle the case of the network partition and the packet losses due to high bit error rates. The ECN was present by default in the kernel that we were using. We conducted our experiments with multiple hops up to 2 and used file sizes of 1MB, 10MB and 30MB. The 802.11b compatible network cards were used so that we could get high bandwidth delay product and hence get good congestion window sizes (we could achieve a maximum congestion window size of 45 at 1448 bytes per packet). We used the topologies as shown in Fig. 5.



Again, in single-hop topology, we conducted trials where both the nodes are stationary and one of them is mobile. The disconnected operation was studied with the single-hop topology wherein one of the nodes goes out range of the other for about one minute and comes back into the stationary node's transmission range.

All the experiments were conducted together between two pairs of machines simultaneously so that the network conditions remained same for the experiments with modified and normal TCP.

The AODV routing protocol we used would declare a route to be stale when there was inactivity on a link for 15000ms.

9. Results

TCP-Type	Number of Hops	Disconnection	Mobility	Transfer File Size	Transfer Time
Base TCP	1	NO	NO	1MB	1.69s
Modified TCP	1	NO	NO	1MB	1.47s
Base TCP	1	NO	NO	10MB	21.3s
Modified TCP	1	NO	NO	10MB	17s
Base TCP	1	NO	NO	30MB	60s
Modified TCP	1	NO	NO	30MB	54.7s
Base TCP	1	NO	YES	30MB	153s
Modified TCP	1	NO	YES	30MB	92.1s
Base TCP	2	NO	YES	1MB	30.8s
Modified TCP	2	NO	YES	1MB	15.4s
Base TCP	1	YES	YES	30MB	198s
Modified TCP	1	YES	YES	30MB	151s

Table 2. Comparison of Transfer time times

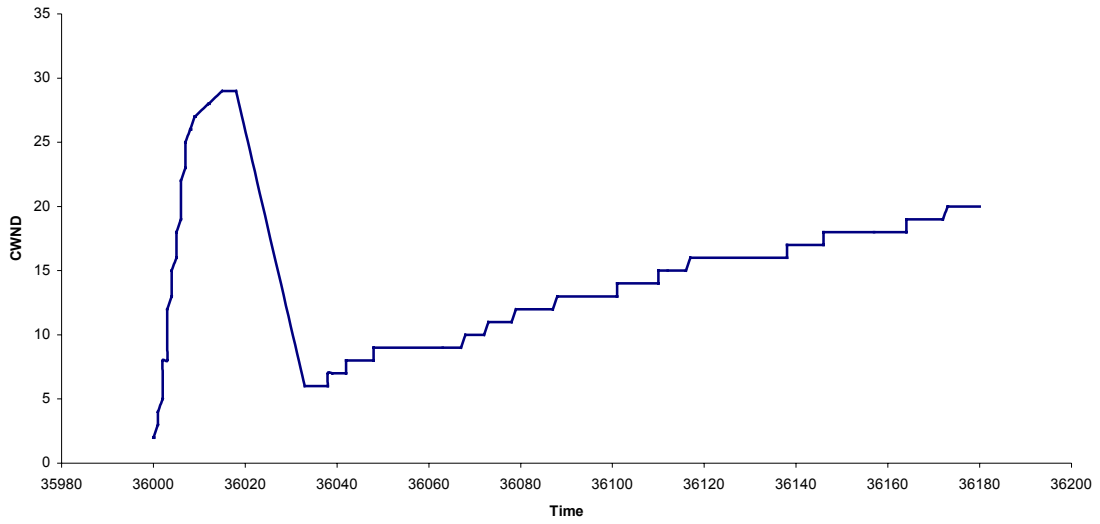
The table above summarizes the results of all the trials executed in our setup. It can be seen that there is an improvement of 25 – 50%.

In all the following graphs, we see that CWND reaches its maximum value of 45 and stays there in case of our modified TCP. But in case of Base TCP, CWND fluctuates and does not stabilize even though there is no congestion.

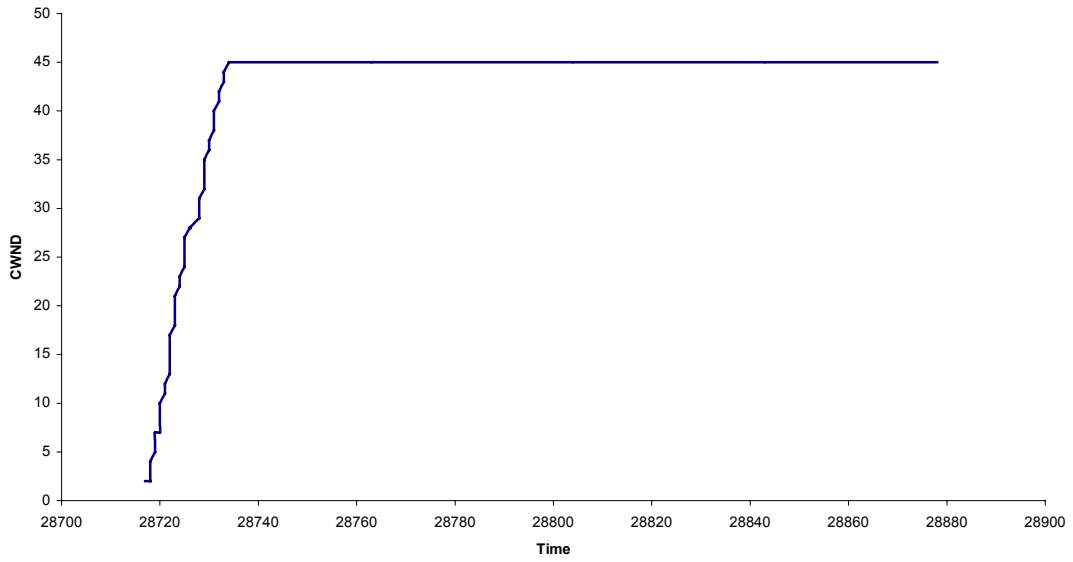
The performance improvement is very obvious in the case when a node is mobile and in the case when a node gets disconnected from the other. When the node is mobile, lot of packets are lost in the medium because network conditions keep varying, thereby creating lot of holes in the receiver’s buffer. This results in the receiver keeping all the packets in the buffer till the holes get filled on receiving retransmissions. In this case, the receiver initiates flow control by shrinking the advertised window, which increases the transfer time. This is compounded with fluctuating congestion window in case of Base TCP increasing the transfer time further.

In case of two-hop topology, one of the end nodes was stationary and the other mobile with respect to the intermediate node (router). Hence, the router would detect link failures very frequently, thus causing the congestion window to fluctuate very much in case of Base TCP. But, in the modified TCP, congestion window stabilizes to the maximum value. There are a lot of retransmissions due to timeouts and duplicate acknowledgements. Hence, the rugged nature of the sequence number/time curve for our modified TCP.

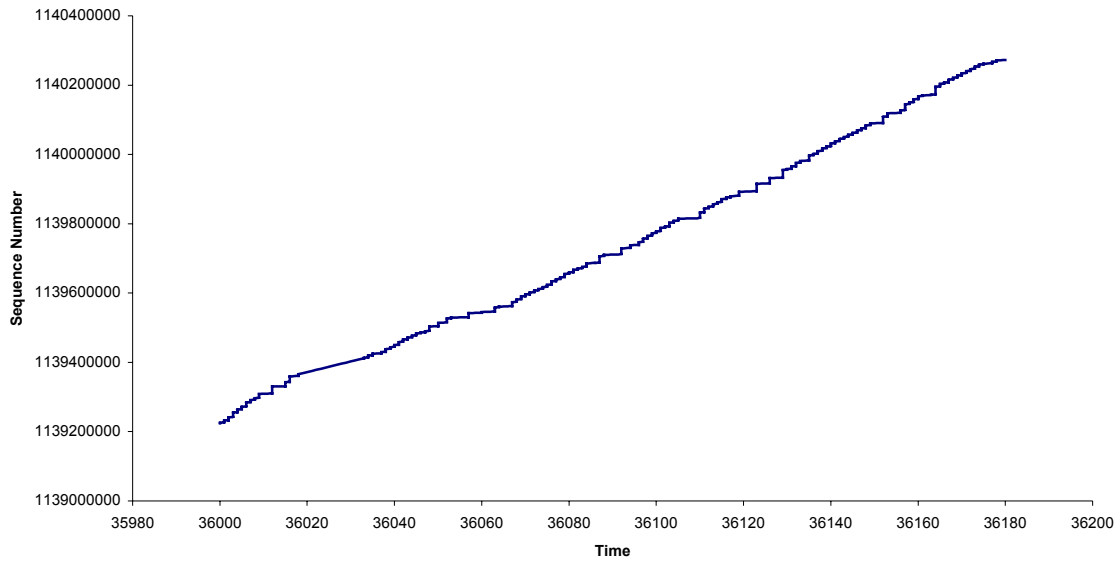
**Congestion Window vs Time for a 1MB data transfer without mobility
in Base TCP**



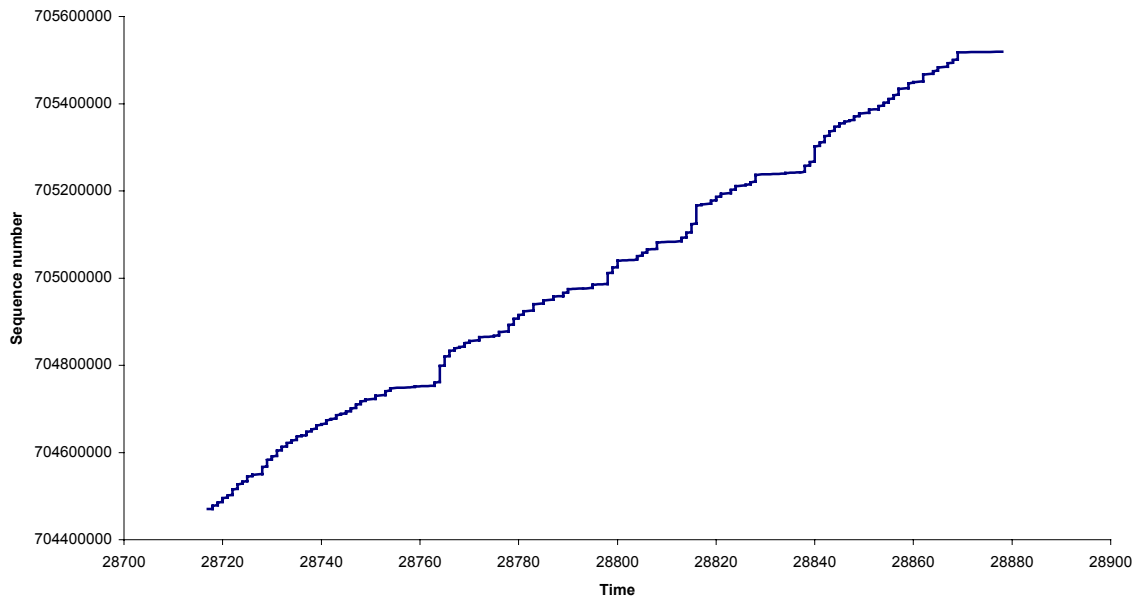
Congestion Window vs Time for a 1MB data transfer without mobility



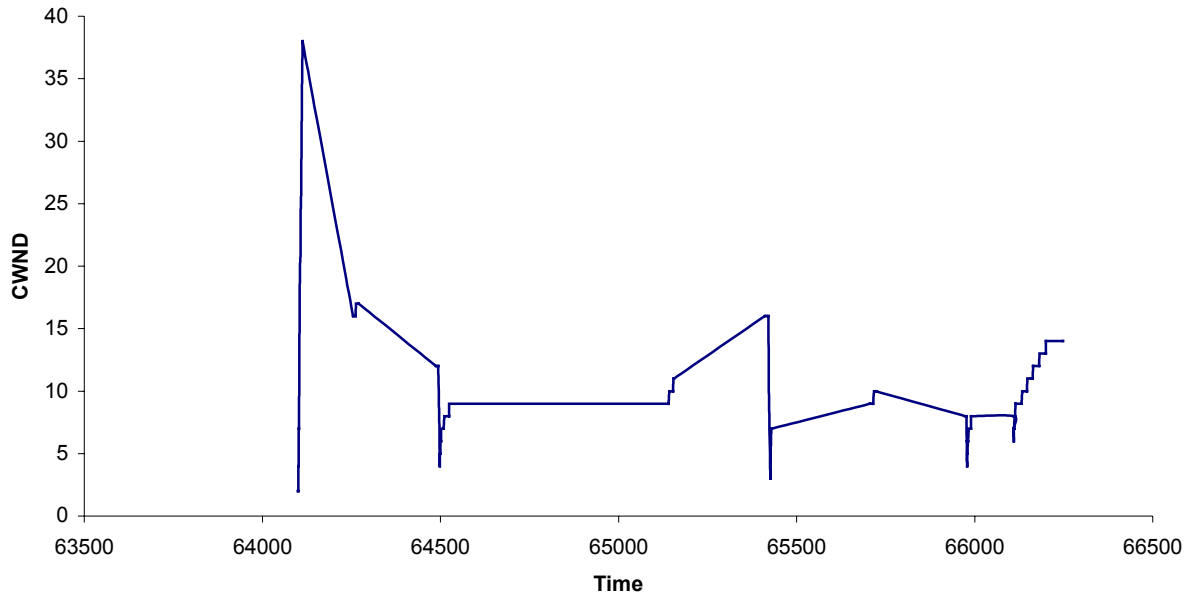
**Sequence Number vs Time for a 1MB data transfer without mobility
in Base TCP**



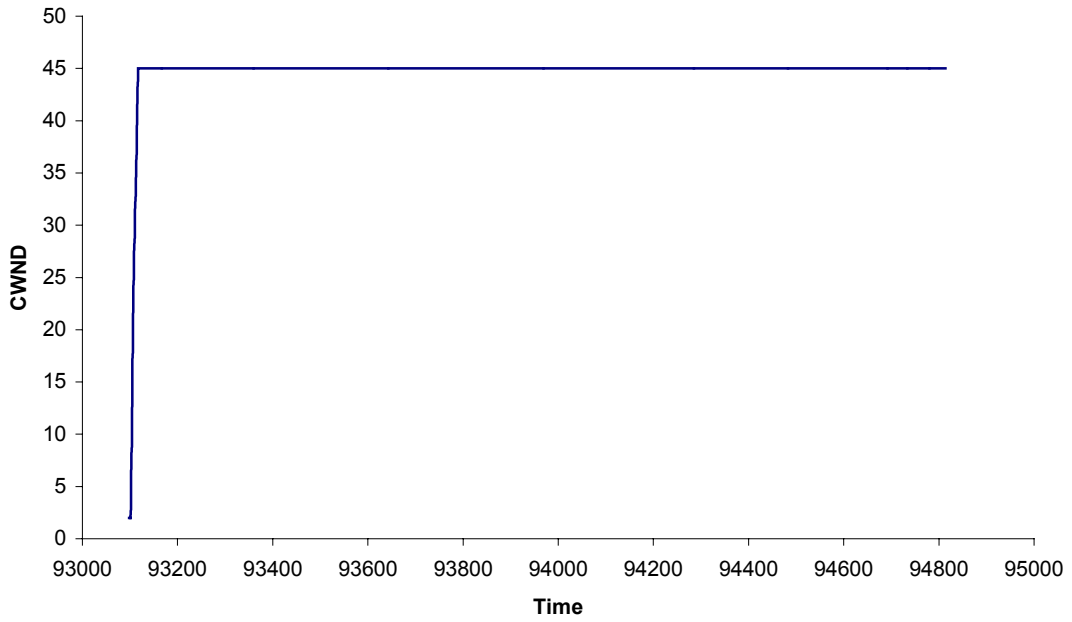
Sequence Number vs Time for a 1MB data transfer without mobility



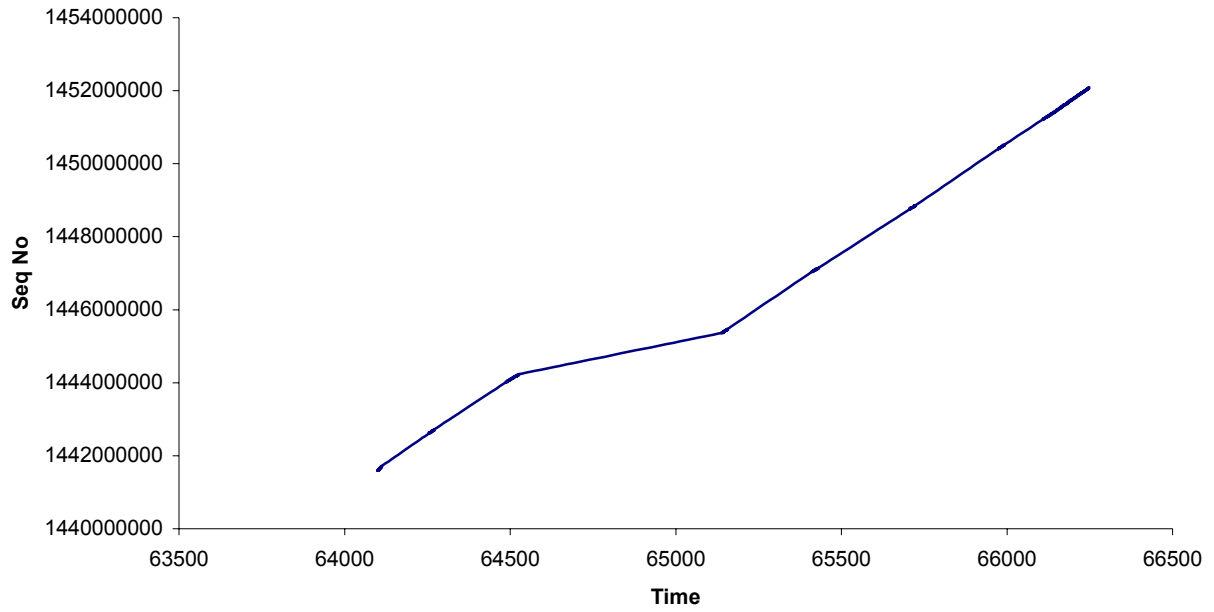
**Congestion Window vs Time for a 10MB data transfer without mobility
in Base TCP**



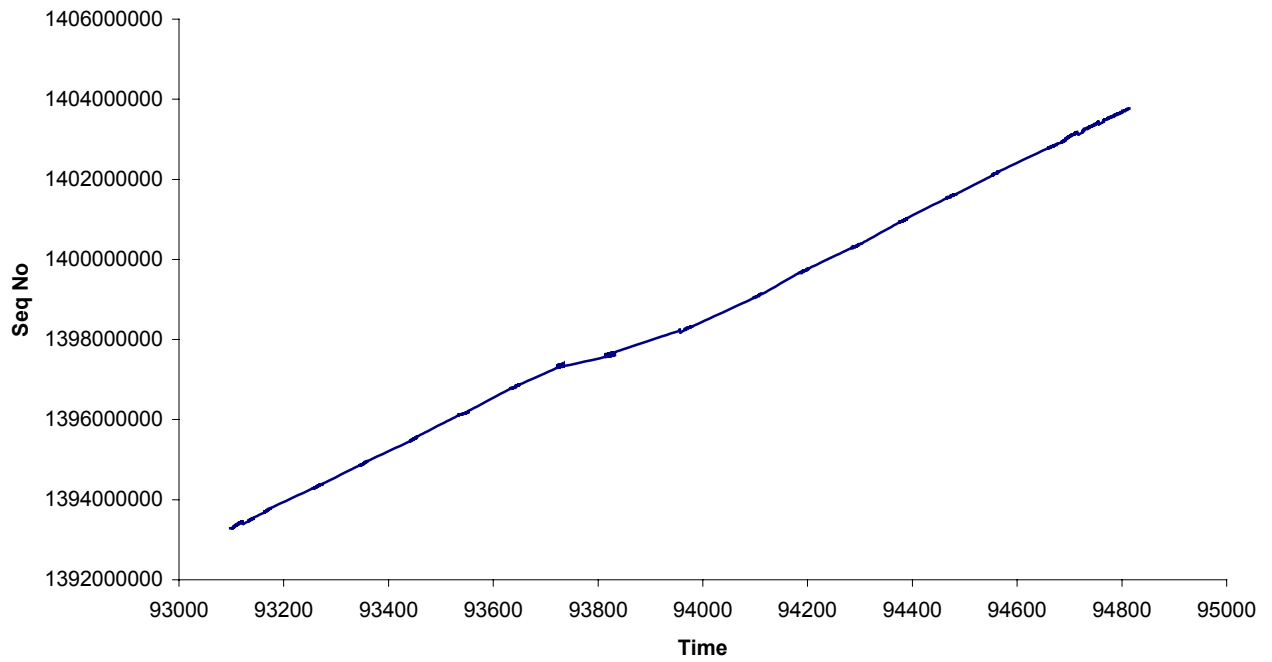
Congestion Window vs Time for a 10MB data transfer without mobility



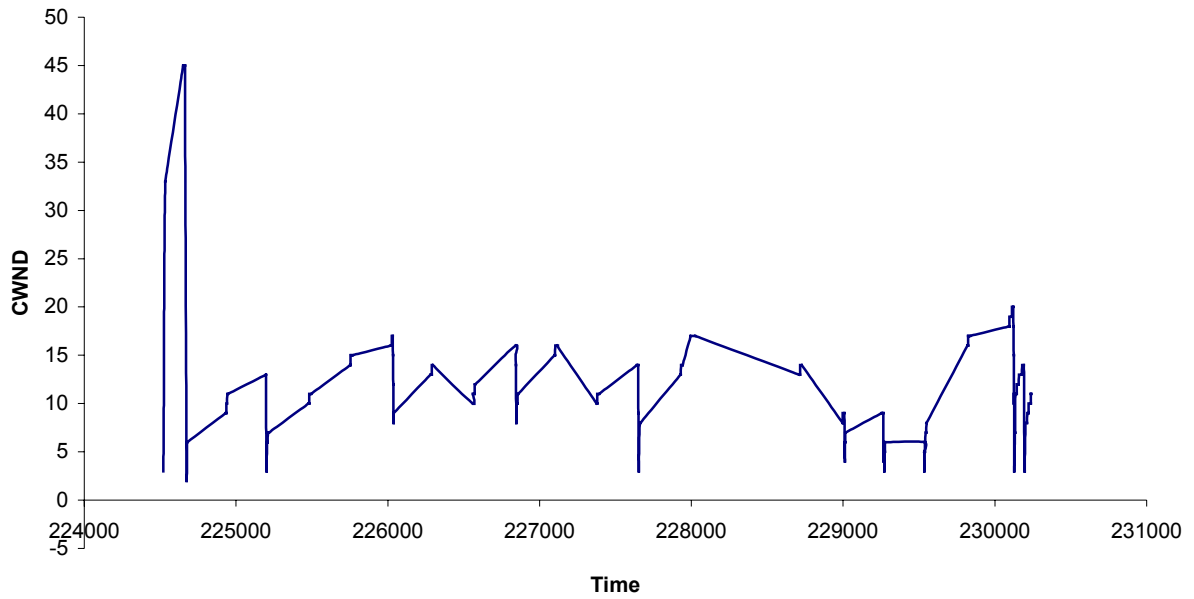
Sequence number vs Time for a 10MB data transfer without mobility
in Base TCP



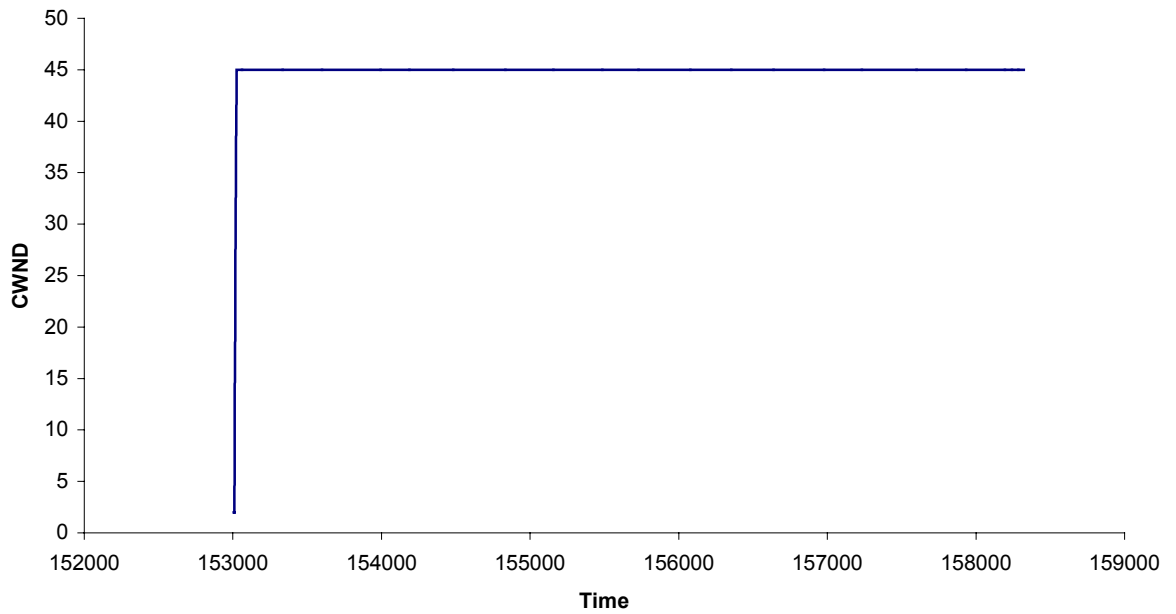
Sequence number vs Time for a 10MB data transfer without mobility



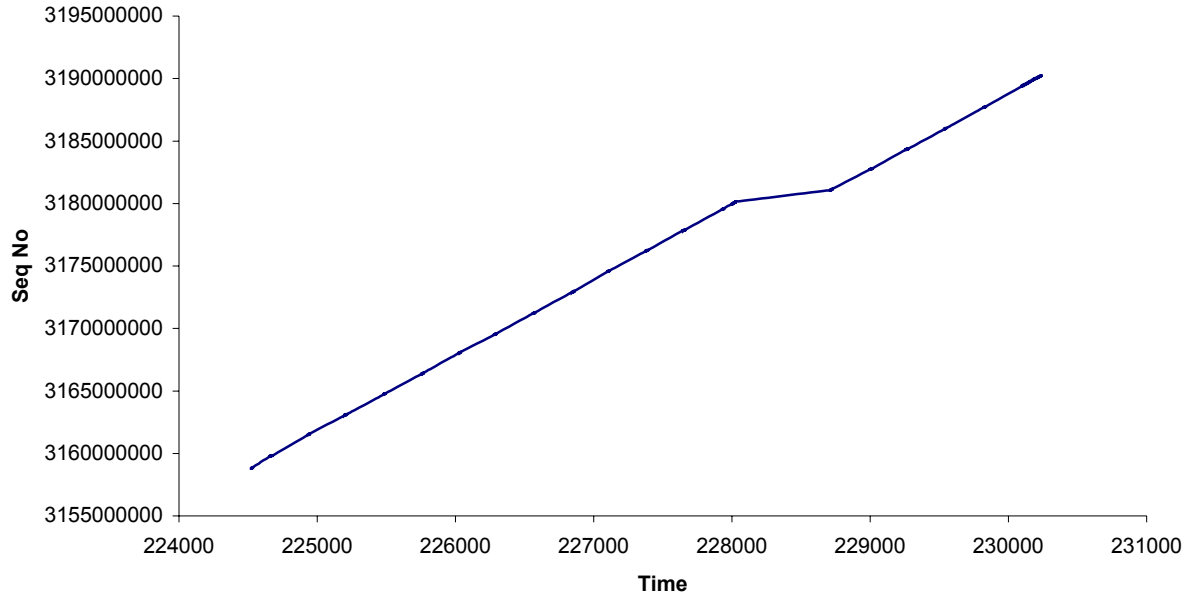
**Congestion Window vs Time for a 30MB data transfer without mobility
in Base TCP**



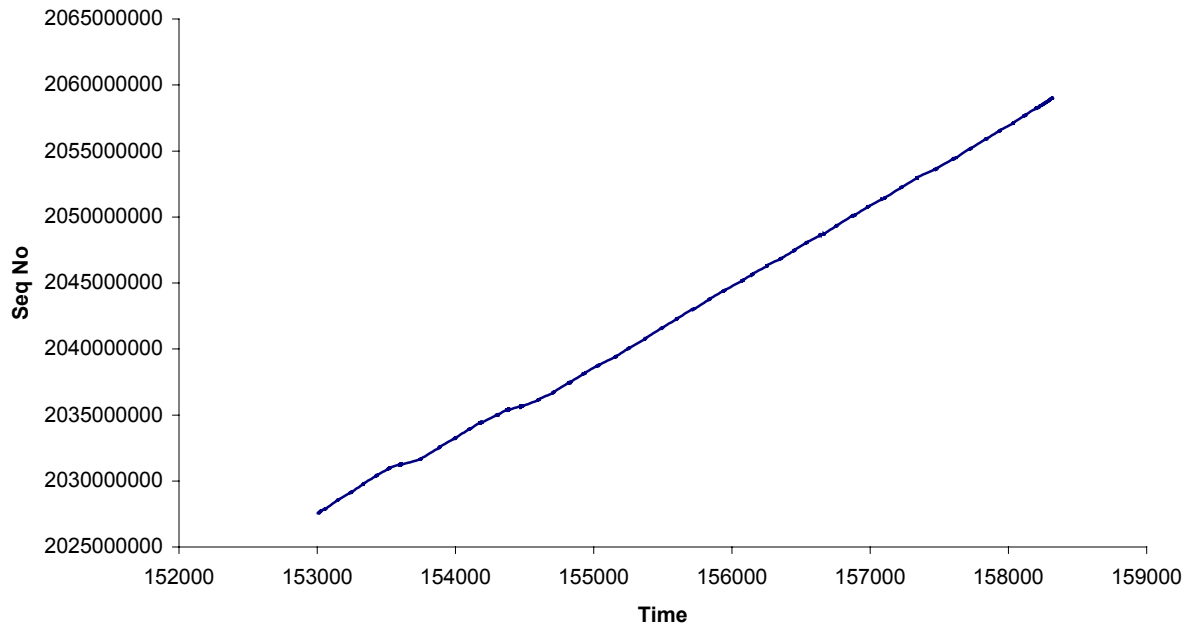
Congestion Window vs Time for a 30MB data transfer without mobility



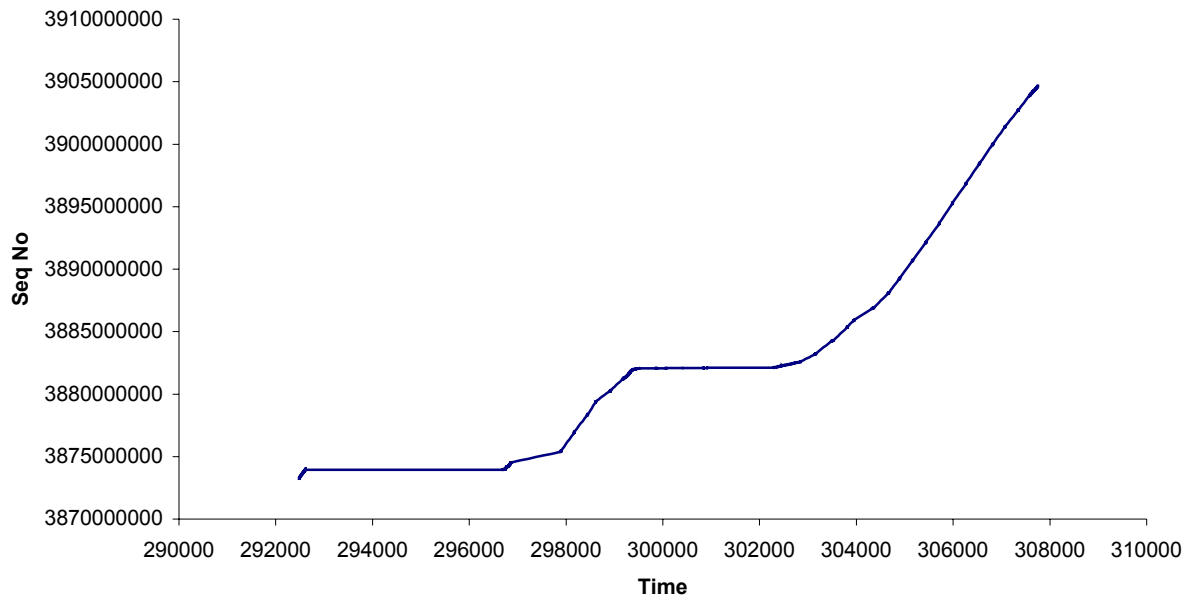
**Sequence Number vs Time for a 30MB data transfer without mobility
in Base TCP**



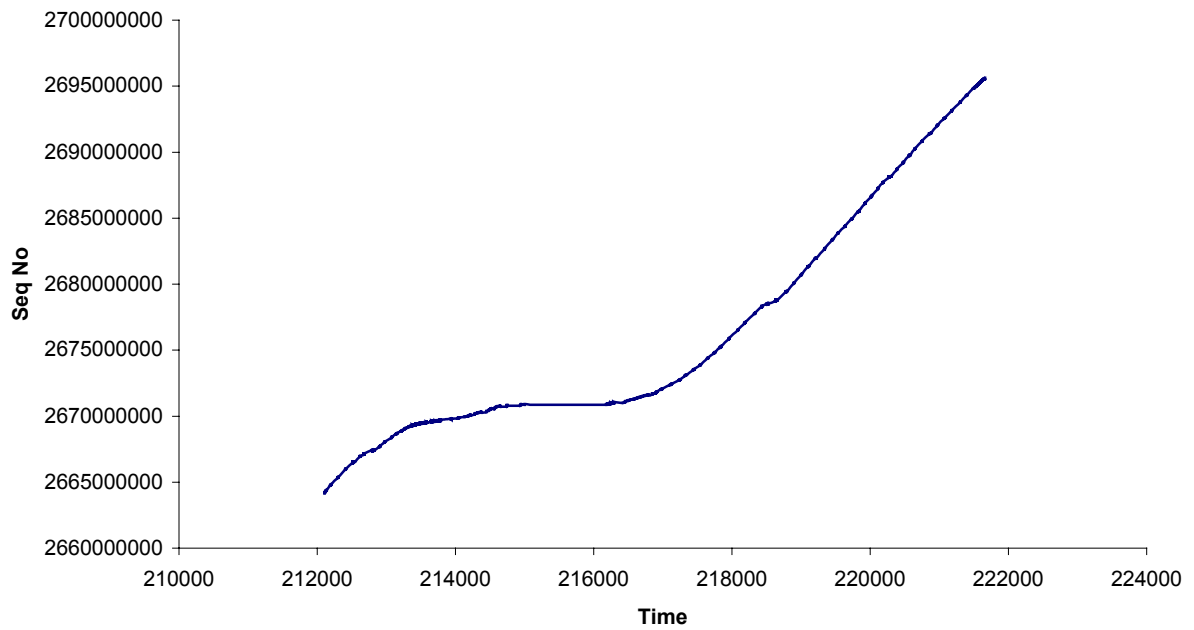
Sequence Number vs Time for a 30MB data transfer without mobility



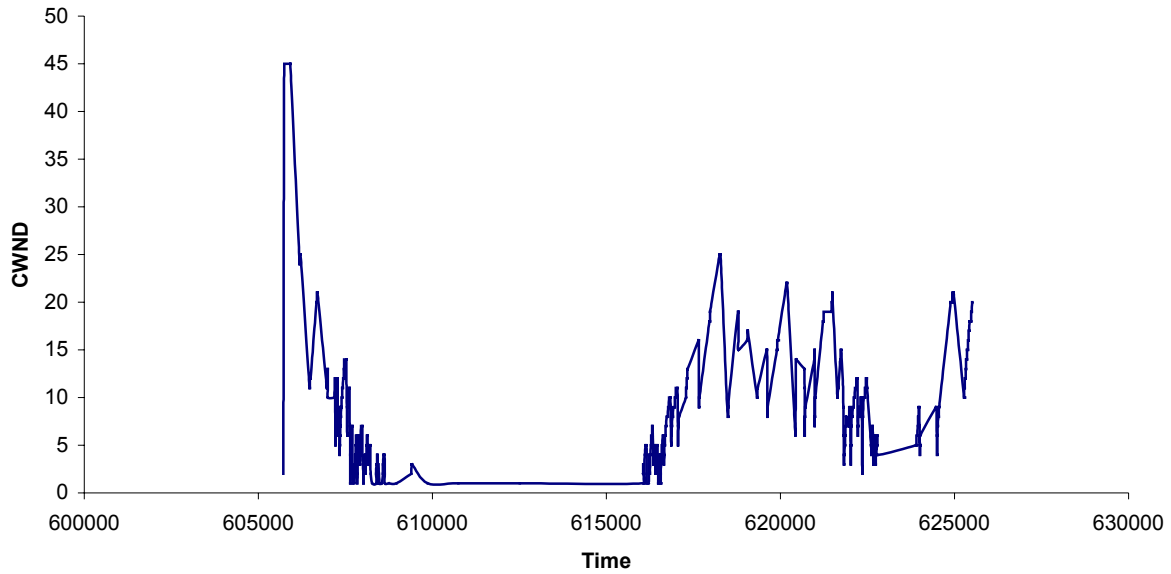
**Sequence Number vs Time for a 30MB data transfer with mobility
in Base TCP**



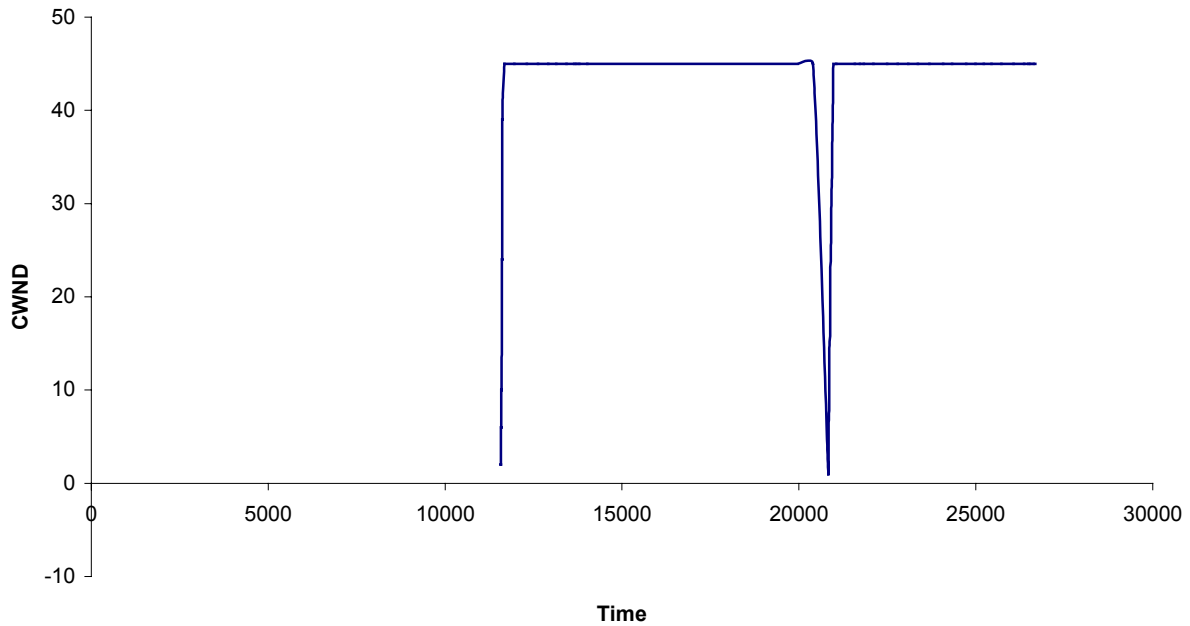
Sequence Number vs Time for 30MB with mobility



**CWND v/s Time for 30 MB, Disconnected for 1 minute
in Base TCP**



CWND v/s Time for 30MB file, Disconnected for 1 minute

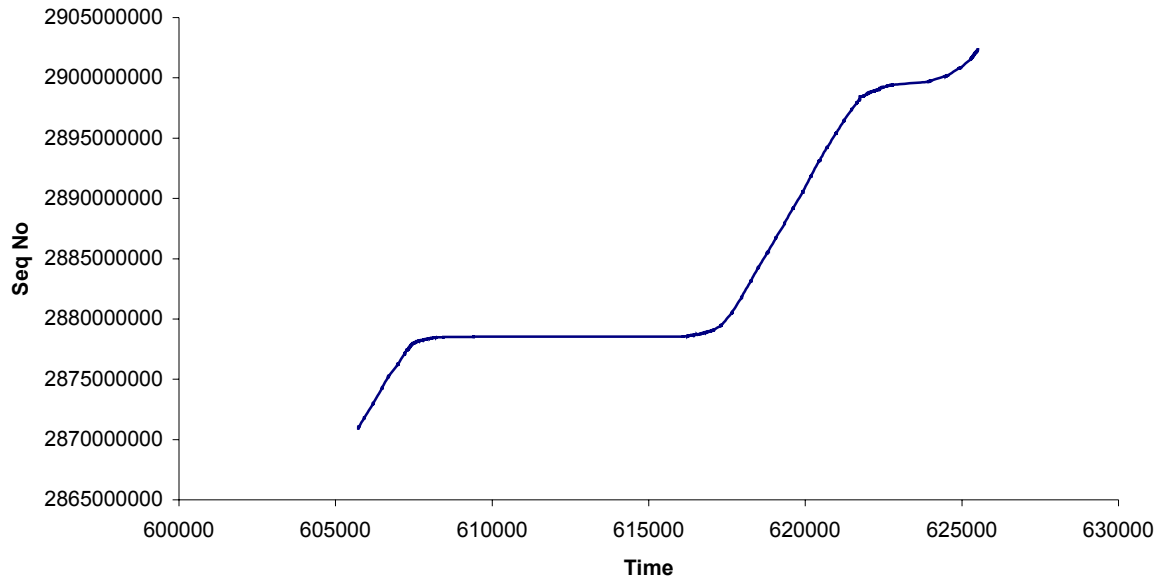


In case of Base TCP, when the nodes get disconnected for about a minute, the RTO goes to a very high value before the receiver gets reconnected to the sender. Hence, there is no data transfer for few seconds even after the nodes get connected. While the nodes were out of range,

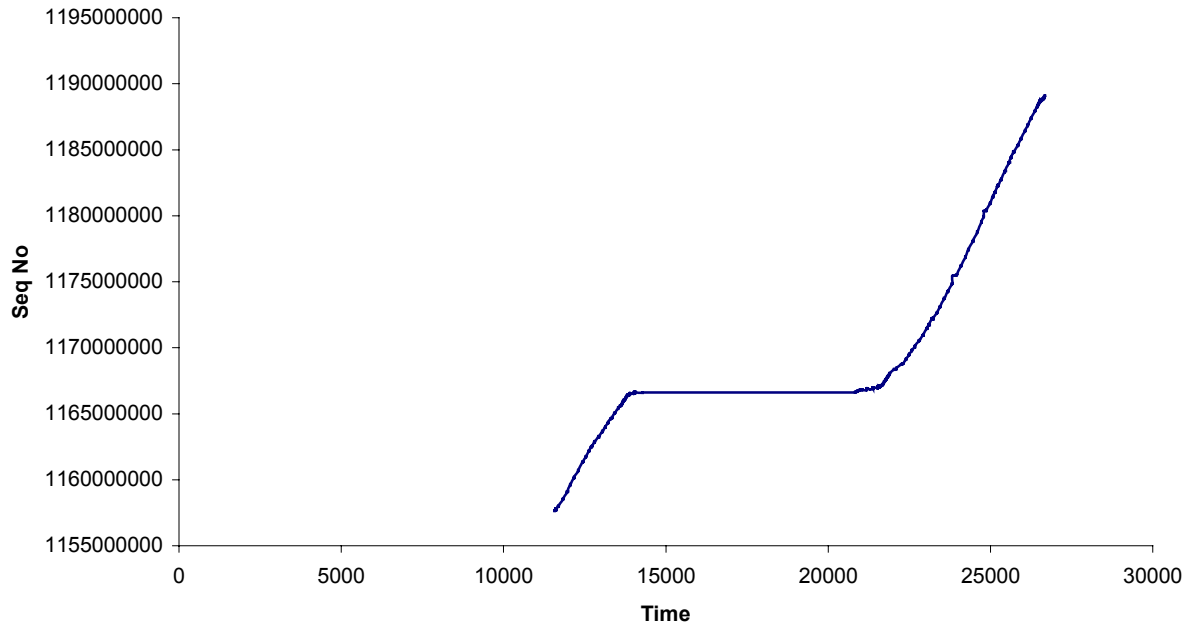
there was a lull. Once, they get re-connected, slow start springs into action and there is exponential increase in data flow.

In case of our modified TCP, CWND is maintained at a stable value till the receiver gets reconnected to the sender. Once the nodes get re-connected, the CWND is reset to 1, so that the network is probed again. The value of CWND recovers very soon once the nodes get re-connected, when compared to base TCP. The RTO does not shoot up, but is gradually increased with time as each probe is sent out. Once, the receiver gets back in range and receives a probe, an ACK is sent. Hence, the lull-period in the data transfer is considerably reduced. The data transfer starts much earlier in comparison to base TCP once the nodes are back in each other's range.

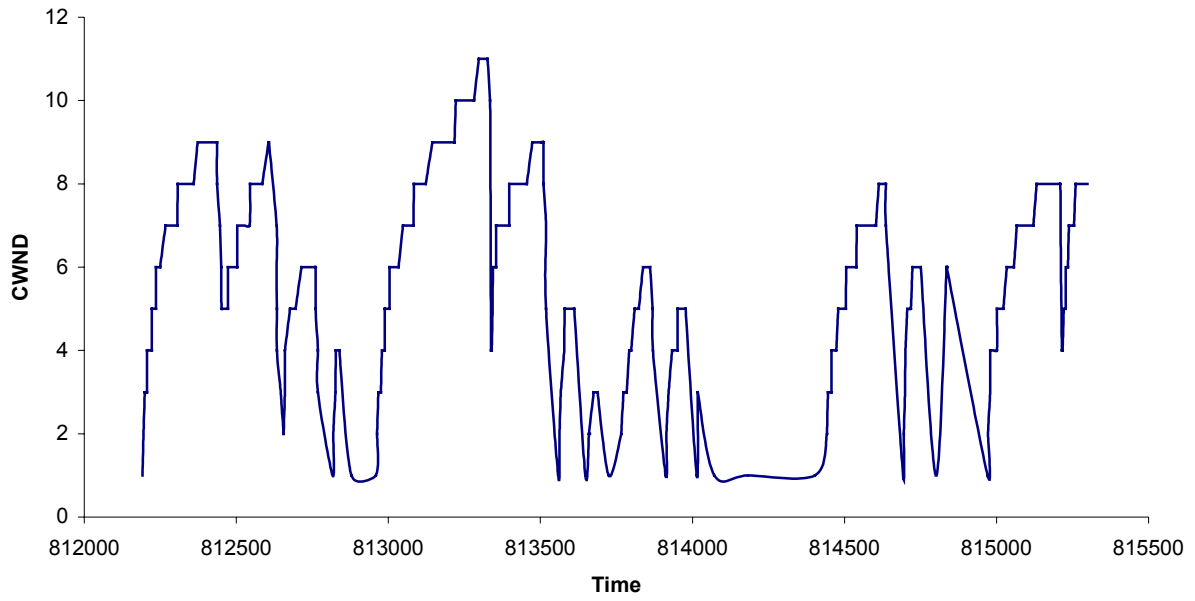
Sequence Number v/s Time for 30 MB, Disconnected for 1 minute
in Base TCP



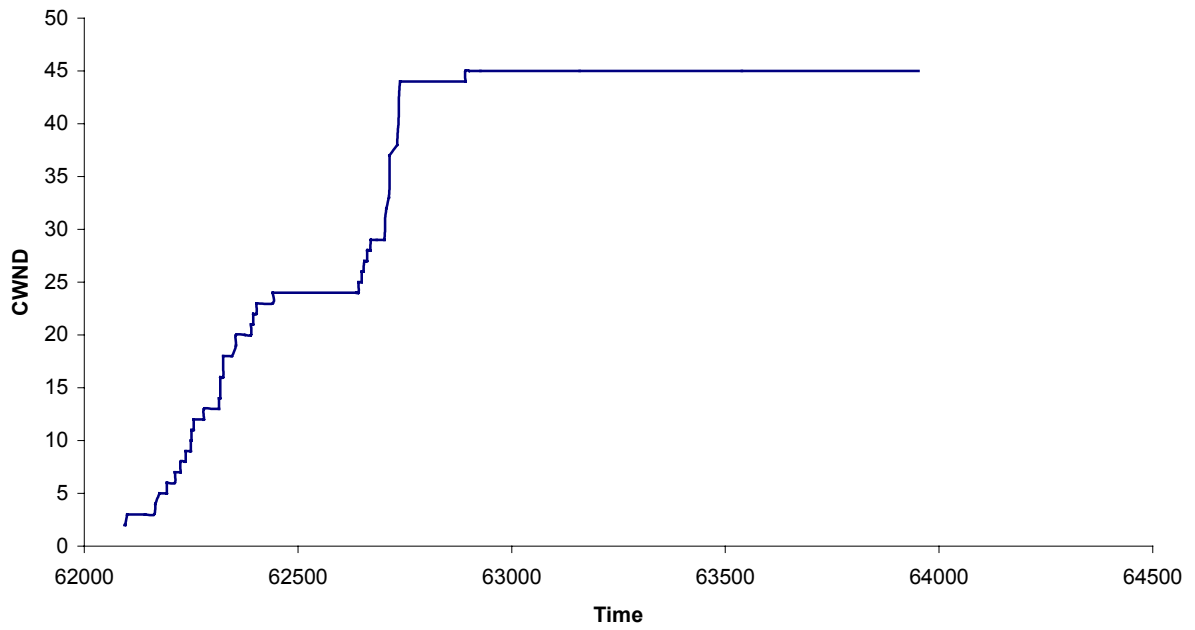
Sequence Number v/s Time for 30MB file, Disconnected for 1 minute



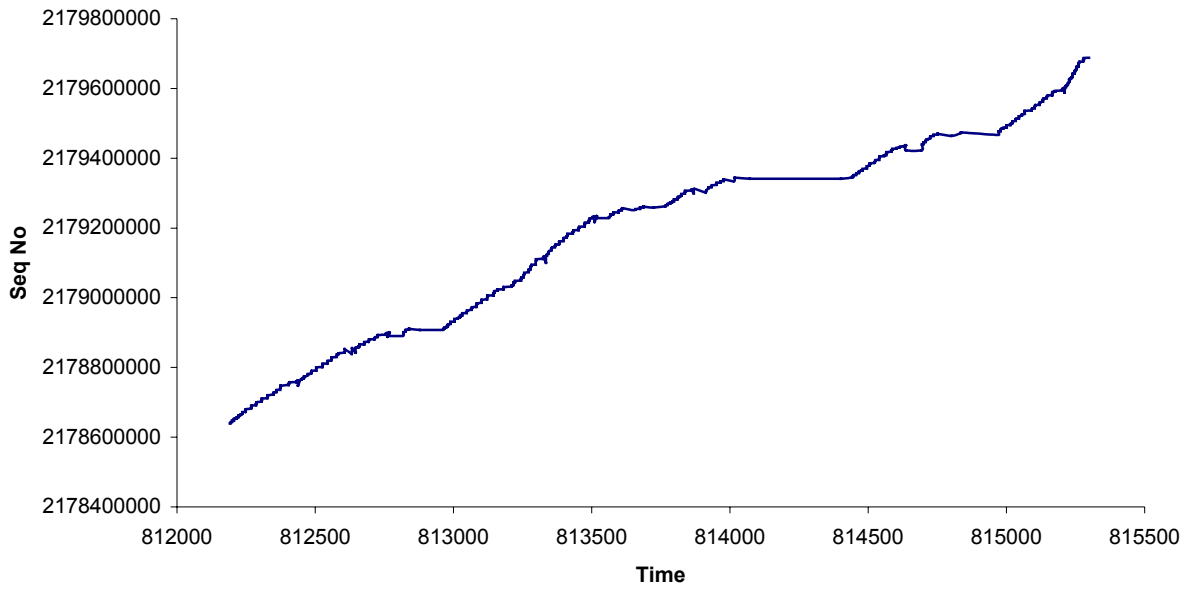
**Congestion Window vs Time for 1MB with 2hops
in Base TCP**



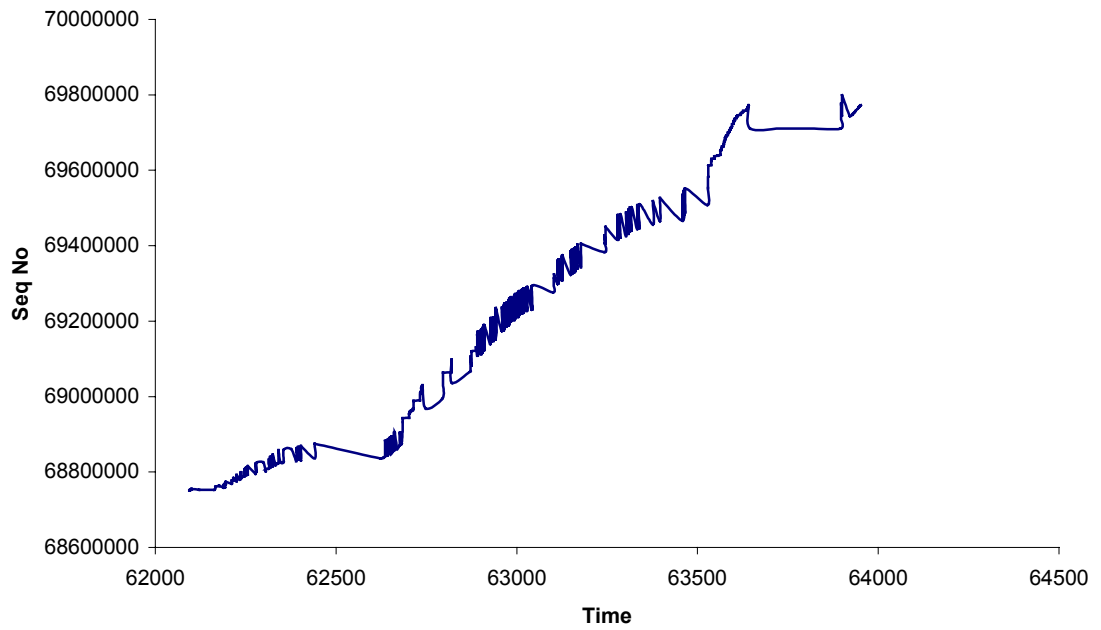
Congestion Window vs Time for 1MB with 2hops



Sequence Number vs Time for 1MB with 2hops
in Base TCP



Sequence Numbers vs Time for 1MB with 2hops



10. Future work and Conclusion

During our experiments we found that lot of packets are lost in a single window resulting in a burst of duplicate acknowledgements arriving at the sender. TCP New Reno can handle only a few losses in a window without too much performance degradation. In order to handle the huge number of losses in a single transmission window, TCP SACK would be a good choice. By modifying TCP SACK the way TCP New Reno was modified by us would improve the TCP performance further.

Some enhancements in the routing protocol will result in better performance in case of data transfers involving multiple hops.

References

- [ATCP] Jian Liu and Suresh Singh. "ATCP: TCP for Mobile Ad Hoc Networks", IEEE Journal on Selected Areas in Communications, Vol. 19, No. 7, July 2001
- [ELFN] Gavin Holland and Nitin Vaidya. "Analysis of TCP Performance over Mobile Ad-Hoc Networks", in Proceedings of IEEE/ACM MOBICOM, August 1999
- [ELFN-L] J. Monks, P. Sinha and V. Bhargavan. "Limitations of TCP-ELFN for Ad-Hoc Networks", MOMUC, Tokyo, Japan, October 2000
- [TCP-F] Chandran K., Raghunathan S., Venkatesan S. and Prakash R. "A feedback-based scheme for improving TCP performance in ad hoc wireless networks", IEEE Personal Communications, February 2001
- [Ad Hoc] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva. "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", in Proceedings of Mobicom 1998
- [DSR] D.B. Johnson and D.A. Maltz. "Dynamic Source Routing in ad-hoc wireless networks", in *Mobile Computing*, 1996
- [AODV] Charles E. Perkins and Elizabeth M. Royer. "Ad hoc On-Demand Distance Vector Routing." Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999