ATCP: TCP for Mobile Ad Hoc Networks*

Jian Liu
SUN Microsystems,
Palo Alto, CA 94303

Suresh Singh
Department of Computer Science
Portland State University
Portland, OR 97201

April 4, 2003

To Appear in IEEE J-SAC in 2001

Abstract

Transport connections set up in wireless ad hoc networks are plagued by problems such as high bit error rates (BER), frequent route changes and partitions. If we run TCP over such connections, the throughput of the connection is observed to be extremely poor because TCP treats lost or delayed ACKs as congestion. In this paper we present an approach where we implement a thin layer between IP and standard TCP that corrects these problems and maintains high end-to-end TCP throughput. We have implemented our protocol in FreeBSD and in this paper we present results from extensive experimentation done in an ad hoc network. We show that our solution improves TCP's throughput by a factor of 2 – 3.

1 Introduction

Ad Hoc networks are multi-hop wireless networks consisting of a (large) number of radio-equipped nodes that may be as simple as autonomous (mobile or stationary) sensors to laptops mounted on vehicles or carried by people. These types of networks are useful in any situation where temporary network connectivity is needed, such as in disaster relief or in the battlefield. Recent work has concentrated on developing MAC layer protocols and routing protocols for these types of networks (see, for instance, [9, 10, 8, 11]). In this paper, we turn our attention to the transport layer and implement a solution that enables TCP to function efficiently in the *lossy* and *partition prone* ad hoc networking environment. Before discussing the challenges involved in making TCP perform well in ad hoc networks, however, let us consider some of the idiosyncrasies of this environment.

In addition to a high bit error rate in mobile ad hoc networks, node connectivity tends to change over time. The rate at which the connectivity changes depends on the number of nodes, their velocity, transmission range and obstacles in the environment that may create shadows. There are two effects of this change in node connectivity:

*This work was supported by the NSF under grant number NCR-9706080. The work was done while the authors were at Oregon State University.

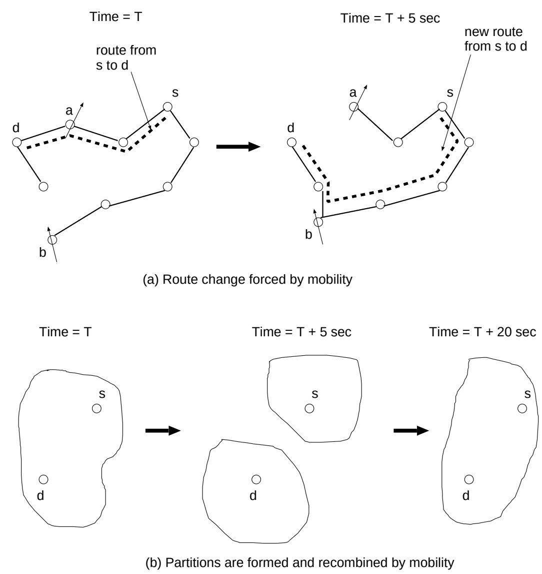

Figure 1: Problems caused due to node mobility.

- Nodes may need to *recompute routes* to some destinations. In Figure 1(a), node s needs to recompute its route to d for an ongoing TCP connection because node a moved out of range of node d.

- It is likely that the ad hoc network may be temporarily *partitioned* due to node mobility. In Figure 1(b), s has an open TCP connection to d. The network gets partitioned at time $T + 5$ causing s and d to lie in different partitions. The network eventually reconnects 15 sec later allowing s and d to continue communicating.

Unfortunately, this change in node connectivity has disastrous consequences for TCP's throughput which can drop to very low levels. This is explained further in the following sections.

1.1 The Problem with TCP in Ad Hoc Networks

TCP is a connection-oriented transport layer protocol that provides reliable, in-order delivery of data to the TCP receiver. If we use TCP without any modification in mobile ad hoc networks, we experience a serious drop in the throughput of the connection. There are several reasons for such a drastic drop in TCP throughput and in this section we examine these reasons in some detail.

The Effect of a High Bit Error Rate (BER)

Bit errors cause packets to get corrupted which result in lost TCP data segments or acknowledgements. When acknowledgements do not arrive at the TCP sender within a short amount of time(the retransmit timeout or RTO), the sender retransmits the segment, exponentially backs off its retransmit timer for the next retransmission, reduces its congestion control window threshold, and *closes its congestion window to one segment.* Repeated errors will ensure that the congestion window at the sender remains small resulting in low throughput (see [1, 3]). It is important to note that error correction may be used to combat high BER, but it will waste valuable wireless bandwidth when correction is not necessary.

The Effect of Route Recomputations

When an old route is no longer available (as in Figure 1(a)), the network layer at the sender attempts to find a new route to the destination (in DSR[8] this is done via route discovery messages while in DSDV[11] table exchanges are triggered that eventually result in a new route being found). It is possible that discovering a new route may take significantly longer than the retransmission timeout interval (RTO) at the sender. As a result, the TCP sender times out, retransmits a packet and invokes congestion control. Thus, when a new route is discovered, the throughput will continue to be small for some time because TCP at the sender grows its congestion window using the slow start and congestion avoidance algorithm. This is clearly undesirable behavior because the TCP connection will be very inefficient. If we imagine a network in which route computations are done frequently (due to high node mobility), the TCP connection will never get an opportunity to transmit at the maximum negotiated rate (i.e., the congestion window will always be significantly smaller than the advertised window size from the receiver).

The Effect of Network Partitions

It is likely that the ad hoc network may periodically get partitioned for several seconds at a time. If the sender and the receiver of a TCP connection lie in different partitions, all the sender's packets get dropped by the network resulting in the sender invoking congestion control. If the partition lasts for a significant amount of time (say several times longer than the RTO), the situation gets even worse because of a phenomena called *serial timeouts*. A serial timeout is a condition wherein multiple consecutive retransmissions of the same segment are transmitted to the receiver while it is disconnected from the sender. All these retransmissions are thus lost. Since the retransmission timer at the sender is doubled with each unsuccessful retransmission attempt (until it reaches 64 sec), several consecutive failures can lead to inactivity lasting one or two *minutes* even when the sender and receiver get reconnected.

The Effect of Multipath Routing

Some routing protocols (such as TORA [10]) maintain multiple routes between source destination pairs, the purpose of which is to minimize the frequency of route recomputation. Unfortunately, this sometimes results in a significant number of out-of-sequence packets arriving at the receiver. The effect of this is that the receiver generates duplicate ACKs which cause the sender (on receipt of three duplicate ACKs) to invoke congestion control.

What does *CWND* really mean in ad hoc networks?

The congestion window in TCP imposes an acceptable data rate for a particular connection based on congestion information that is derived from timeout events as well as from duplicate ACKs. In an ad hoc network, since routes change during the lifetime of a connection, we lose the relationship between the CWND size and the tolerable data rate for the route. In other words, the CWND as computed for one route may be too large for a newer route, resulting in network congestion when the sender transmits at the full rate allowed by the old CWND.

1.2 Our Approach

The approach we propose in this paper utilizes network layer feedback (from intermediate hops) to put the TCP sender into either a persist state, congestion control state or retransmit state. Thus, when the network is partitioned, the TCP sender is put into persist mode so that it does not needlessly transmit and retransmit packets. On the other hand, when packets are lost due to error (as opposed to congestion), the TCP sender retransmits packets without invoking congestion control. Finally, when the network is truly congested, the TCP sender invokes congestion control normally.

In our implementation, we did not modify standard TCP itself because we want to maintain compatibility with the standard TCP/IP suite. Therefore, to implement our solution, we insert a thin layer called ATCP (Ad hoc TCP) between IP and TCP that listens to the network state information provided by ECN (Explicit Congestion Notification) messages [5, 12, 6][1] and by ICMP "Destination Unreachable" messages and then puts TCP at the sender into the appropriate state. Thus, on receipt of a "Destination Unreachable" message, TCP state at the sender is frozen (the sender enters the *persist* state) until a new route is found ensuring that the sender does not invoke congestion control. Furthermore, the sender does not send packets into the network during the period when no route exists between the source and destination.

We use ECN as a mechanism by which the sender is notified of impending network congestion along the route followed by the TCP connection. On receipt of an ECN, the sender invokes congestion control without waiting for a timeout event (which may be caused, more often than not, due to lost packets)[2].

Thus, the benefits our solution are:

- Standard TCP/IP is unmodified.
- ATCP is invisible to TCP and therefore nodes with and without ATCP can interoperate. The only drawback is that nodes without ATCP will see all the performance problems associated with running TCP over ad hoc networks.
- ATCP does not interfere with TCP's functioning in cases where the TCP connection is between a node in the wireline network and another in the wireless ad hoc network.

The remainder of this paper is organized as follows. In the next section we discuss the applicability, to ad hoc wireless networks, of proposed solutions for improving TCP performance for cellular networks. Section 3 presents our protocol design in detail and section 4 discusses some implementation issues. We study its performance in section 5 and summarize our work in section 6.

2 Literature Review

Many papers (see [1, 2, 3, 13]) have been written proposing methods for improving TCP performance in *cellular* networks where the last link is the only wireless link in the system. The typical solution

[1]As of this writing, ECN was actively debated on mailing lists, see: http://www-nrg.ee.lbl.gov/ecn-arch/

[2]Our decision to use ECN was prompted by the discussions within the Internet community surrounding the use of ECN in the wired Internet. Most recently, [12] was proposed as a RFC 2481 (Request For Comment) to the IETF (Internet Engineering Task Force). As of this writing, ECN is in the Experimental stage within IETF and it is beeing submitted to advance as Proposed Standard.

used in these various approaches is to *split* the connection in two at the base station. The base station then retransmits packets to the mobile node in order to prevent the TCP sender located in the wireline network from invoking congestion control. This approach makes sense because the base station typically knows the state of the wireless link and can make intelligent decisions regarding the state of the TCP connection. In an ad hoc network, on the other hand, the TCP connection traverses multiple wireless links. Thus, solutions based on using the base station to "fix things" cannot work particularly well.

[7] investigates the impact of link breakage on TCP performance in ad hoc networks. They use DSR (Dynamic Source Routing [8]) as the underlying routing protocol (simulated in NS2). DSR is an on-demand routing protocol where a sender finds a route to the destination by flooding *route request* packets. DSR's performance is optimized by allowing intermediate nodes to respond to route request packets using cached routes. Unfortunately, if the cached information maintained at a intermediate node is stale, the time it takes to find a new route can be very long (several seconds). Thus, TCP running on top of DSR sees very poor throughput. The paper proposes the use of Explicit Link Failure Notification (ELFN) to improve TCP performance. Here, the TCP sender is notified that a link has failed and it disables its retransmission timer and enters a stand-by mode. In stand-by mode, the TCP sender periodically sends a packet in its congestion window to the destination. When an ACK is received, TCP leaves the stand-by mode, restores its retransmission timers, and resumes transmission as normal.

Finally, [4] discusses a scheme similar to [7] for improving TCP performance in ad hoc networks in the presence of failures. Here the router detecting a failed route generates a Route Failure Notification (RFN) packet towards the source. The TCP source that receives this packet enters a *snooze* state which is very similar to TCP's persist state. When the route is re-established, a Route Re-establishment Notification (RRN) is sent to the source by any router on the previous route that detected the new route. This packet removes the source from the snooze state. In this method the source continues using the old congestion window size for the new route. This is a problem because the congestion window size is route specific (since it seeks to approximate the available bandwidth). [4] also does not consider the effects of congestion, out-of-order packets, and bit error.

Our approach differs significantly from the above proposal in many ways. First, the above approach does not deal with the high loss environment present in ad hoc networks. ATCP, on the other hand, treats loss due to packet loss and loss due to congestion differently. Second, ATCP ensures that the congestion window is recomputed after every new route recomputation. [7] and [4] continue to use the old CWND which could lead to congestion. Finally, the above approach does not take into account the possibility of a large number of out of order packets (if, for instance, the underlying routing protocol was TORA [10] and not DSR [8]). Thus, we believe that our proposed approach is far more comprehensive in that it accounts for all possible sources of inefficiency in TCP.

We summarize the differences between ATCP and the approaches used in [7, 4] in Table 1. As we can see, ATCP provides a comprehensive solution to the problem of implementing TCP in ad hoc networks.

3 Design of ATCP

Our goal in designing ATCP was to provide a complete solution to the problem of running TCP over multihop wireless networks. Specifically, we wanted to design a protocol that has the following

Circumstance	*ATCP*	*[7]*	*[4]*
Packet Loss due to high BER	ATCP Retransmits, TCP does not invoke congestion control (CC)	Not Handled	Not Handled
Route Changes	ICMP "Destination Unreachable" puts sender in *persist* until new route found	ELFN freezes sender state	RRN freezes sender state
Network Partition	As above	As above	As above
Packet Reordering	ATCP reorders packets so TCP does not generate duplicates	Not handled	Not handled
Congestion	ECN used to quickly notify sender of congestion. Sender invokes CC.	Not Handled	Not Handled
CWND	Reset for each new route	Old CWND used	Old CWND used

Table 1: Summary of Differences.

characteristics:

1. *Improve TCP performance* for connections set up in ad hoc wireless networks. As we discussed in section 1.1, TCP performance is affected by the problems of high BER and disconnections due to route recomputation or partition. In each of these cases, the TCP sender mistakenly invokes congestion control. The appropriate behavior in these cases ought to be:
   - *High BER:* Simply retransmit lost packets without shrinking the congestion window.
   - *Delays due to route recomputation:* Sender should stop transmitting and resume when a new route has been found.
   - *Transient partition:* As above, the sender should stop transmitting (because we do not want to flood the network with packets that cannot be delivered anyway) until it is reconnected to the receiver.
   - *Multipath Routing* In this case, when TCP at the sender receives duplicate ACKs, it should not invoke congestion control because multipath routing shuffles the order in which packets are received.
2. *Maintain TCP's congestion control behavior.* This is an important goal because if losses are caused due to network congestion, we do not want the TCP sender to assume that these losses were due to high BER and continue transmitting. In this case, we *want* TCP to shrink its congestion window in response to losses and invoke slow start.
3. *Appropriate CWND behavior* When there is a change in the route (e.g., a reconnection after a brief partition), the congestion window should be recomputed.
4. *Maintain end-to-end TCP semantics.* We believe that it is critical to maintain end-to-end TCP semantics in order to ensure that applications do not crash.
5. *Be compatible with standard TCP.* This is necessary because we cannot assume that all machines deployed in an ad hoc network will have ATCP installed. Thus, machines with or without ATCP should be able to set up normal TCP connections with machines that may or may not have ATCP. Furthermore, *applications* running at machines with ATCP *should not be aware of ATCP's presence.*

Sometimes, it is likely that an ad hoc network may be connected to wireline networks through access points. In such situations, the sender or receiver of a TCP connection may lie in the wireline network with the other end-point in the ad hoc network. It is important to ensure that TCP connections work normally in these cases as well. Our approach to the problem of improving TCP's performance while *maintaining compatibility* is to introduce a thin layer between TCP and IP called ATCP(see Figure 3). The ATCP layer at the sender monitors TCP state and spoofs TCP in a way to ensure that the behavior discussed above is achieved. We discuss this in more detail in the next section.

3.1 Functioning of the ATCP layer

The ATCP layer is only active at the TCP sender (in a duplex communication, the ATCP layer at both participating nodes will be active). This layer monitors TCP state and the state of the network (based on ECN and ICMP messages) and takes appropriate action. To understand ATCP's behavior, consider Figure 2 which illustrates ATCP's four possible states – *normal, congested, loss* and *disconnected*. When the TCP connection is initially established, ATCP at the sender is in the *normal* state. In this state, ATCP does nothing and is invisible. Let us now examine ATCP's behavior under four circumstances:

- *Lossy Channel:* When the connection from the sender to the receiver is lossy, it is likely that some segments will not arrive at the receiver or may arrive *out-of-order*. Thus, the receiver may generate *duplicate* acknowledgements(ACKs) in response to out of sequence segments. When TCP receives three consecutive duplicate ACKs, it retransmits the offending segment and shrinks the congestion window. It is also possible that due to lost ACKs, the TCP sender's RTO may expire causing it to retransmit one segment and invoke congestion control.

  ATCP in its *normal state* counts the number of duplicate ACKs received for any segment. When it sees that three duplicate ACKs have been received, it *does not forward* the third duplicate ACK but puts TCP in *persist mode*. Similarly, when ATCP sees that TCP's RTO is about to expire, it again puts TCP in *persist mode* (implementation details are discussed in section 4). By doing this, we ensure that the TCP sender does not invoke congestion control because that is the wrong thing to do under these circumstances. After ATCP puts TCP in persist mode, ATCP enters the *loss state*.

  In the *loss state*, ATCP transmits the unacknowledged segments from TCP's send buffer. It maintains its own separate timers to retransmit these segments in the event that ACKs are not forthcoming. Eventually, when a *new* ACK arrives (i.e., an ACK for a previously unacknowledged segment), ATCP forwards that ACK to TCP which also removes TCP from persist mode. ATCP then returns to its *normal state*.

- *Congested:* We assume that when the network detects congestion, the ECN flag is set in ACK and data packets. Let us assume that ATCP receives this message when in its *normal state*. ATCP moves into its *congested state* and does nothing. It ignores any duplicate ACKs that arrive and it also ignores imminent RTO expiration events. In other words, ATCP does *not interfere* with TCP's normal congestion behavior. After TCP transmits a new segment, ATCP returns to its *normal state*.

- *Disconnected:* Node mobility in ad hoc networks causes route recomputation or even temporary network partition. When this happens, we assume that the network generates an ICMP *Destination Unreachable* message in response to a packet transmission. When ATCP receives this message, it puts the TCP sender into persist mode and itself enters the *disconnected state*. TCP periodically generates *probe packets* while in persist mode. When, eventually, the receiver is connected to the sender, it responds to these probe packets with a duplicate ACK (or a data packet). This removes TCP from persist mode and moves ATCP back into *normal state*.

  In order to ensure that TCP does not continue using the old CWND value, ATCP sets TCP's *CWND to one segment* at the time it puts TCP in persist state. The reason for doing this is to force TCP to probe the correct value of CWND to use for the new route.

- *Other Transitions:* Finally, when ATCP is in the *loss state*, reception of an ECN or an ICMP *Source Quench* message will move ATCP into *congested state* and ATCP removes TCP from its persist state. Similarly, reception of an ICMP *Destination Unreachable* message moves ATCP from either the *loss state* or the *congested state* into the *disconnected state* and ATCP moves TCP into persist mode (if it was not already in that state).

- *Effect of Lost Messages:* Note that due to the lossy environment, it is possible that an ECN may not arrive at the sender or, similarly, a "Destination Unreachable" message may be lost. If an ECN message is lost, the TCP sender will continue transmitting packets. However, every subsequent ACK will contain the ECN thus ensuring that the sender will eventually receive the ECN causing it to enter the congestion control state as it is supposed to. Likewise, if there is no route to the destination, the sender will eventually receive a retransmission of the "Destination Unreachable" message causing TCP to be put into the persist state by ATCP. Thus, in all cases of lost messages, ATCP performs correctly.

Let us examine how ATCP changes TCP's behavior under the conditions discussed in section 1.1. Under lossy conditions (due to high BER), ATCP retransmits unacknowledged segments while TCP is put into persist state. Thus, TCP *does not* invoke congestion control. In the event that the source and the destination get disconnected (either for short periods of time while a new route is computed or for longer periods due to partition), TCP is again put into persist mode for the duration of the disconnection and no segments are transmitted by ATCP. When the network is reconnected, TCP automatically comes out of persist mode because the receiver responds to the sender's probe packets. However, the congestion window used in this case is one segment initially. Finally, TCP's congestion behavior is unchanged ensuring that TCP appropriately throttles back its transmission rate when the network is congested.

Finally, we need to make a comment regarding ATCP's behavior when the connection traverses the fixed internet. There are two cases to consider:

- If the fixed internet does implement ECN, ATCP will operate correctly. If, however, the fixed internet does not implement ECN, then we need to *split* the connection at the node that connects the wireless network with the wired internet. Thus, there will be two conjugated TCP connections (this is similar to I-TCP [1] for cellular networks).

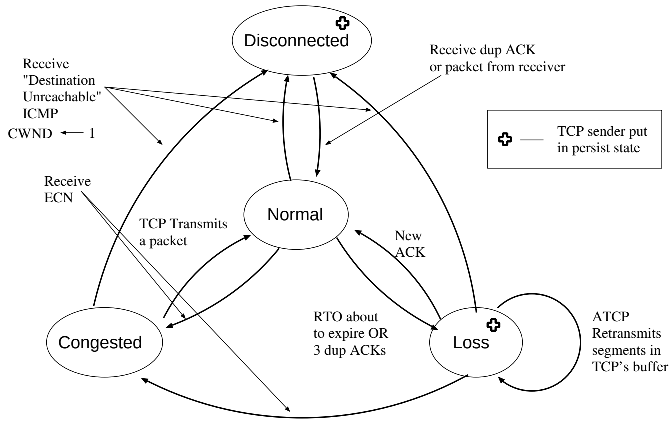

Figure 2: State transition diagram for ATCP at the sender.

3.2 Benefits of our Approach

How does the ATCP meet our design goals outlined earlier in this section? It is clear that the *performance* of ATCP will be better than TCP as we discuss in section 5. Similarly, end-to-end TCP semantics are maintained because ATCP does not generate ACKs on its own. The only time when ATCP inserts itself in the data path is when it is in the *loss state*. Here, ATCP retransmits unacknowledged segments from TCP's buffer. However, even in this case, ATCP forwards the first new ACK to TCP (thus removing TCP from persist mode) and returns itself to the *normal state*. This behavior does not affect end-to-end semantics of the connection.

3.3 Discussion of some Design Decisions

In our design of the ATCP protocol, we rely on explicit notification regarding congestion and disconnections to make our protocol work. This choice, we feel, is justified because of the unique nature of the ad hoc networking environment. In the internet, TCP relies on timeouts or duplicate ACKs to inform it of network congestion. Unfortunately, in the hostile ad hoc networking environment, packet losses are frequently a result of high bit error rates (caused by fading, interference or jamming) or network partition. The correct behavior in the presence of loss due to bit error is to retransmit the lost packets without reducing the transmission rate. In the case of network partition, the correct behavior is to stop all transmission until the network is reconnected (i.e., stop the flow on the periphery of the network). As is clear from our discussion above, ATCP relies on ECN messages to enable it to determine when the network is congested and on "Destination Unreachable" messages to inform it when the network is partitioned or no route exists. Finally, we chose not to use SACKs because the delay*bandwidth product in ad hoc networks is small (i.e., the transmit window sizes are small) and thus SACKs would contribute little in terms of performance while requiring additional processing at the sender and receiver (we have not considered energy consumption in this paper but minimizing the processing involved is important to increase the life of the battery).

4 Implementation of ATCP

We implemented ATCP as a layer between TCP and IP, see Figure 3. Function atcp_input() intercepts every packet IP passes up to TCP. It examines the TCP and IP headers of the packet and finds out to which TCP connection this packet is sent by locating the TCP control block according to the packet's source and destination addresses and port numbers(see Figure 5).

Let us look at ATCP's behavior in the *normal* state. In this state, atcp_input() first checks if the ECN has been set(also see Figure 5). If the ECN bit is set to one(1), atcp_input() sets ATCP state to *congested* and then passes the segment to function tcp_input(). Upon receiving the ECN, TCP will start congestion control algorithms because a network traffic congestion has been detected by a router somewhere between the sender and the receiver. Function atcp_output() takes ATCP back to *normal state* from *congested state* when TCP sends out a packet.

If the ECN flag is not set, atcp_input() counts the number of duplicate ACKs received and puts TCP into persist mode if it has received tcprexmtthresh number of duplicate ACKs (the default value of tcprexmtthresh is 3). ATCP itself enters the *loss* state and processes the segment. Another case in which ATCP enters the *loss* state (and puts TCP in persist) occurs when TCP's RTO is about to expire.

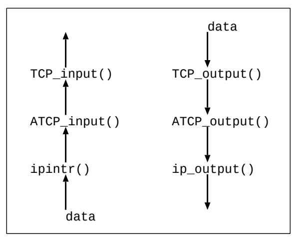

Figure 3: Data flow through the TCP/ATCP/IP stack

ATCP checks TCP/ATCP timers by calling atcp_slowtimo() every 500 milliseconds. Function atcp_slowtimo() is registered in the inetsw struct which also holds atcp_input(), atcp_fasttimo(), etc. In the *normal* state, if atcp_slowtimo() finds out that a timeout is about to happen, it will call function atcp_timers() for processing. atcp_timers() will adjust the retransmission timeout value, stop the RTT timer, call tcp_output() to resend the timed out packet, put TCP into persist mode, and then change ATCP state to *loss* (see Figure 4).

In the *loss* state, ATCP performs packet retransmission on behalf of TCP (see Figure 6). In order to do this correctly, ATCP copies member variable snd_cwnd in TCP's TCP control block into member variable asnd_cwnd in the control block. atcp_output() will use asnd_cwnd instead of snd_cwnd when retransmitting segments. In addition to asnd_cwnd of type u_long, we added at_status of type int and at_timer[TCPT_NTIMERS] of type int in TCP control block. as_status is used to hold ATCP state – *normal*, *loss*, *congested*, and *disconnected*. at_timer is used for ATCP timers (including the retransmit timer used by ATCP when retransmitting packets from TCP's buffer). One other noteworthy feature of the *loss* state is that atcp_output() discards TCP's persist probe packets (unlike the case when ATCP is in the *disconnected* state) in order to maintain the efficiency of the connection.

When receiving a packet from IP, atcp_input() examines if it contains either a new ACK or new data from the other end of the connection. In either case, atcp_input() will take TCP out of persist

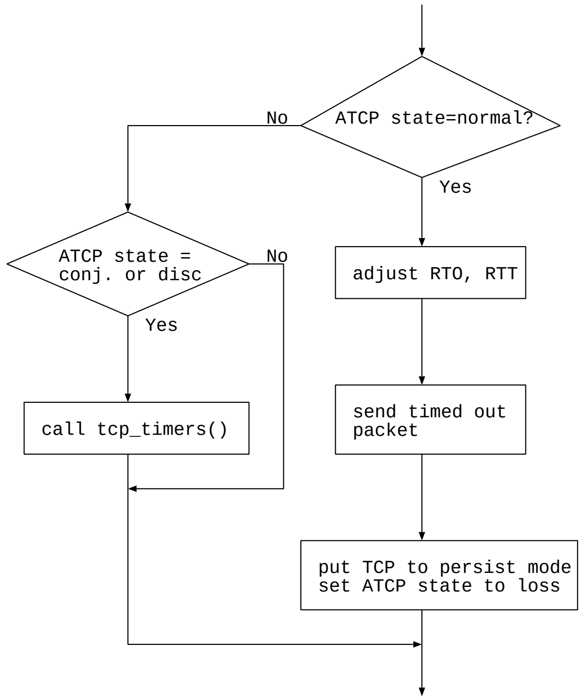

Figure 4: Flowchart for function atcp_timer()

mode and change ATCP state to *normal* from *loss* state. If, after taking TCP out of persist mode, ATCP's congestion window size is greater than TCP's congestion window size, TCP's congestion window size is set to ATCP's congestion window size.

Function atcp_notify() is called when an ICMP_UNREACH_NET icmp message is received. This causes ATCP to change its state to *disconnected* from either of *normal, loss* or *congested.* atcp_notify() stops TCP's retransmission timer, sets TCP's receiver advertised window size to zero(0), starts TCP's persist timer , sets the congestion window to one(1) segment, and puts TCP into persist mode. In *disconnected* state, atcp_output() simply passes TCP persist probes to IP and atcp_input() passes packets from IP to TCP until atcp_output() catches a nonprobing packet from TCP(see Figure 6). Function atcp_output() will put ATCP back into *normal* state upon receiving such a packet from TCP. The fact that TCP sent a non-probe packet implies that TCP came out of persist in response to a packet forwarded by ATCP – thus ATCP does not have to explicitly remove TCP from persist mode.

When TCP comes out of persist mode, it will have a congestion window of just *one segment.* There are a few reasons for doing this. When route failure occurs due to a network partition or if the old route is no longer valid, the network layer will try to find a new route (a process that may take several seconds[3]). If we continue using the previous congestion window for the new route, it may result in congestion at some intermediate hop. It is also possible that the new found route may

[3]When a route is no longer valid, the network layer attempts to find new routes. However, it is possible that some nodes may respond to the request for a new route with stale cached routes. This causes additional delay in some routing protocols such as DSR [8].

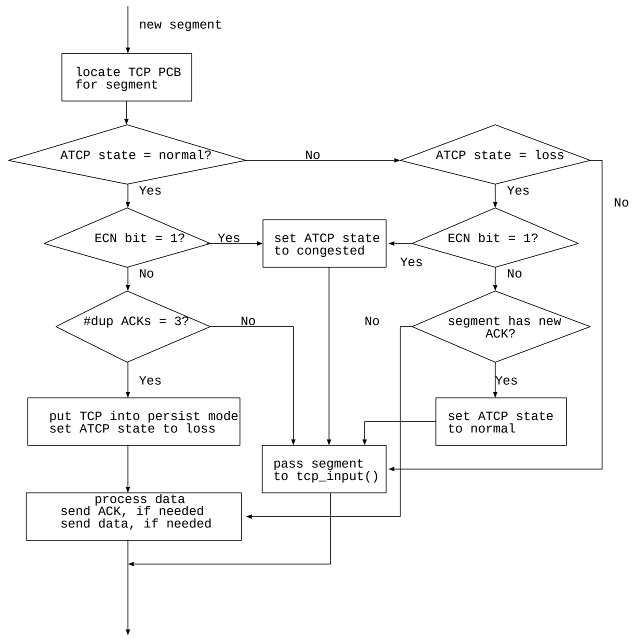

Figure 5: Flowchart for function atcp_input()

go through a almost congested region because not every proposed routing protocol takes traffic into account. Based on these considerations, ATCP sets TCP congestion window to one segment upon Destination Unreachable ICMP messages so that when TCP resumes transmission, it will invoke *slow start* algorithm.

We note that the overhead of implementing ATCP consists of the following:

- ATCP timers including timers for fast timeout and retransmission timers.
- Data structures including CWND size, status control blocks, and one additional control block to maintain the current ATCP state information.
- Functions including atcp_input(), atcp_output(), atcp_slowtimo(), atcp_timers(), atcp_fasttimo, and atcp_notify().

In all, ATCP code is approximately 2000 lines long with the bulk of the contribution coming from atcp_input() and atcp_output().

5 Performance Study

We implemented ATCP in the FreeBSD kernel and in this section we discuss the performance of our implementation. Our goal in running the various experiments was to examine ATCP's performance in the presence of bit error, network partition and congestion. Specific questions we looked at included:

1. What is the effect of high bit error on ATCP performance? How does ATCP perform when the wireless bandwidth is low? How does ATCP performance scale with varying RTT values?

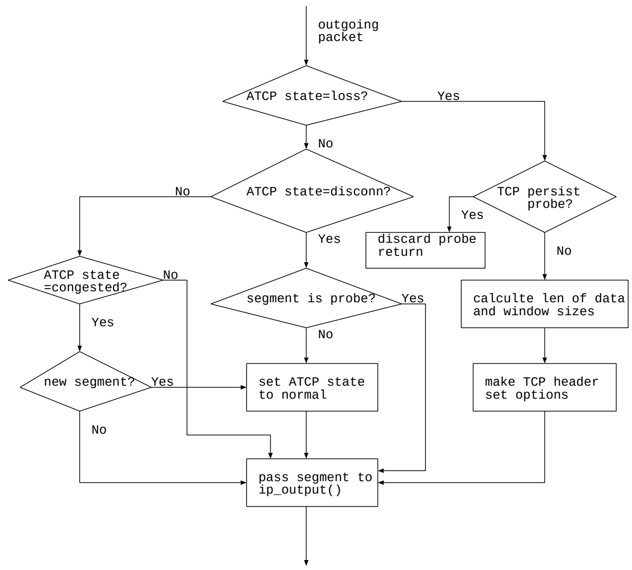

Figure 6: Flowchart for function atcp_output()

2. How does multipath routing affect ATCP's performance in relation to TCP?

3. Does ATCP perform correct congestion control when network congestion occurs?

4. How does ATCP perform, in relation to plain TCP, in ad hoc networks where there are frequent short disconnections? How does ATCP deal with cases where network partitions occur during file transfer?

In order to evaluate the performance of our protocol, we used an experimental testbed consisting of five pentium PCs each of which had two ethernet cards. This gives us a four hop network where the traffic in each hop is isolated from the other hops. To model the lossy and low-bandwidth nature of the wireless links, we emulated, in IP, a 32Kbps channel over each hop[4]. We modified the IP code as follows. All calls to ip_output() are intercepted and then, based on the link speed and packet size (including TCP and IP headers), a timer is set to go off each time a packet can be sent on the wireless link. At each timeout one packet is removed from a link queue and ip_output() is called normally. In addition to the link bandwidth, the modified IP code also allowed us to introduce bit errors in the packets during transmission. We used a bit error rate of 10^{-5} for *all* experiments. We also introduced hop-by-hop delays by the simple mechanism of delaying ip_input() by some amount of time at each hop. For instance, to have a 20ms average delay on a link, we generate a uniform random number between 10ms and 30ms. That number is then converted into an integer that specifies a timeout value. Thus, ip_input() is called when this timer expires. Network partition occurs at an intermediate hop in our setup. This host periodically thinks that its next hop is no longer valid (this is again implemented by using a timer in IP) thus causing ICMP to generate the appropriate host unreachable message. Network congestion is made to occur at some intermediate

[4]Emulating the wireless link in this manner gives us precise control over the available wireless bandwidth, disconnection events, etc.

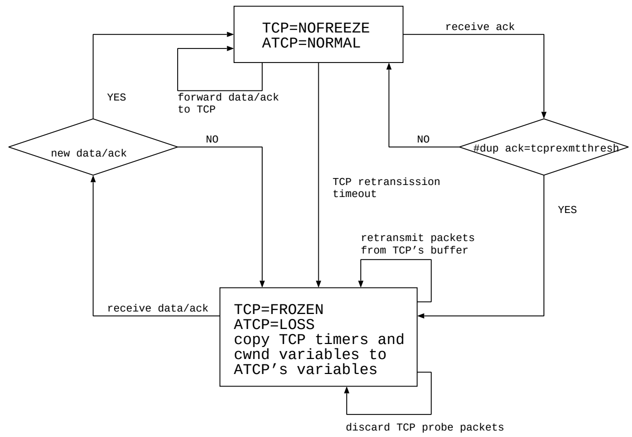

Figure 7: Flowchart for ATCP transition between *normal* and *loss* states.

host as well by flooding that host with spurious packets (generated by a process running on that host). This results in the generation of an explicit congestion notification (ECN).

For each data point in our graphs, we use twenty (20) measurements and compute 90% confidence intervals (that are also plotted).

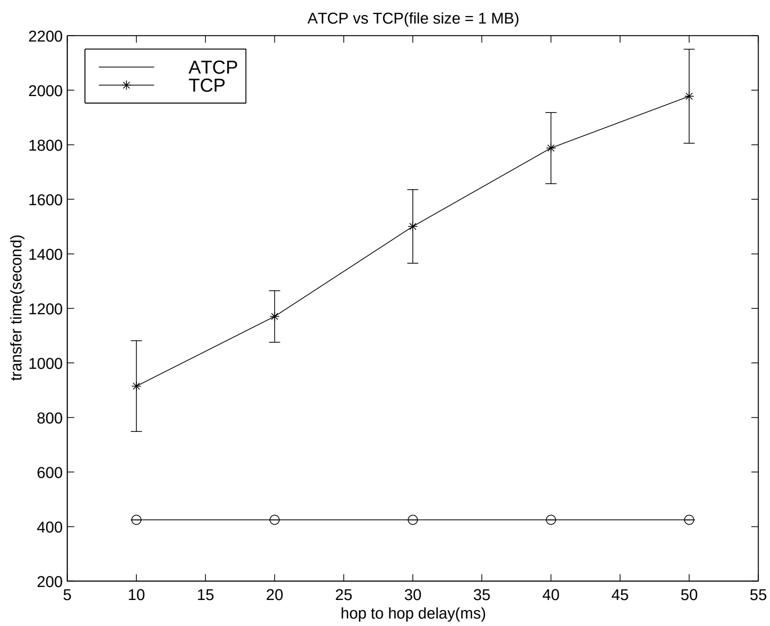

Figure 8: ATCP and TCP performance in the presence of bit error only.

5.1 Loss Case

The first experiments we ran did not include partition or congestion events. The connection was only subjected to bit error that occurred at a BER of 10^{-5} at each hop. We measured the time taken to transfer a 1MByte file when using plain TCP and when using ATCP. In Figure 8 we plot the transfer time (in seconds) on the y-axis and the mean hop-by-hop delay on the x-axis. It is interesting to note that the time taken by TCP to transfer the file increases almost linearly from 900s to 1900s with increasing hop-by-hop delays. On the other hand, the time taken by ATCP is almost constant at approximately 425s. It is instructive to perform a rough computation to explain the ∼425s transfer time for ATCP. At a BER of 10^{-5}, we have a end-to-end probability of packet loss of approximately 3.2% (100 byte packets). Since the raw bandwidth of the connection is 32Kbps, we get an upperbound of 31Kbps for the actual bandwidth. This bandwidth is shared by the data packets as well as by ACK traffic (data is only transferred in one direction). Thus, the bandwidth available for data is 22.1 Kbps (data size is 100 bytes and ACKs are 40 bytes long). Thus, in the absence of any timer or other protocol overhead, it should take approximately 361 seconds to transfer the 1MByte file. It is interesting to note that ATCP is fairly close to this limit.

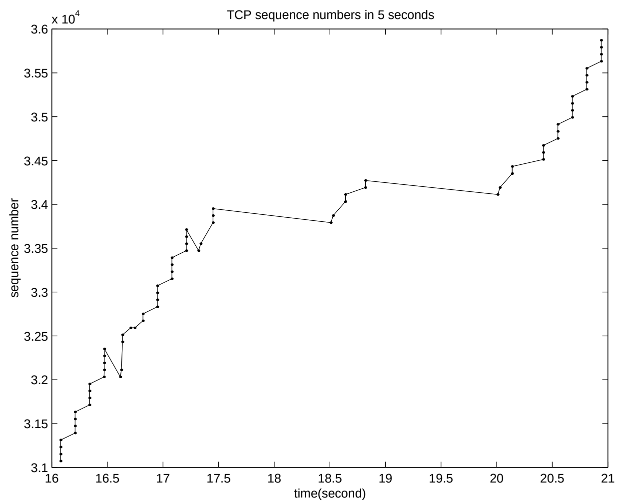

Figure 9: TCP trace in the presence of bit error only.

The difference in behavior between TCP and ATCP is illustrated in the sequence number versus time plots shown in Figure 9 and Figure 10. In the case of TCP, lost packets or ACKs result in TCP timeout (the long almost horizontal lines between 17.5s to 18.5s and between 19s and 20s) and retransmission. In addition, three duplicate ACKs result in retransmission as well (times 16.5s and again at time 17.25s where the curve drops sharply). In all of these cases, TCP shrinks its congestion window thus resulting in low throughput (see Figure 11). In the case of ATCP, on the other hand, we never see a TCP timeout. This is because ATCP puts TCP into persist mode and retransmits the unacknowledged packet. Likewise, we observe that ATCP retransmits packets upon seeing three duplicate ACKs (see times 19.45s and 19.6s where the curve drops sharply).

This dramatic difference in performance between TCP and ATCP can be explained by the fact that TCP invokes congestion control frequently during the experiment because of lost packets or duplicate ACKs. TCP uses slow start to increase its transmit window. ATCP, on the other hand, puts the TCP sender in persist mode and retransmits the packet whose retransmit timer was about to expire. Figures 12 and 11 illustrate the typical behavior of the congestion window for TCP and ATCP (these graphs are snapshots of a random time period and are not related to Figures 9 and 10 directly). In these graphs we plot the congestion window size for consecutive packet transmissions.

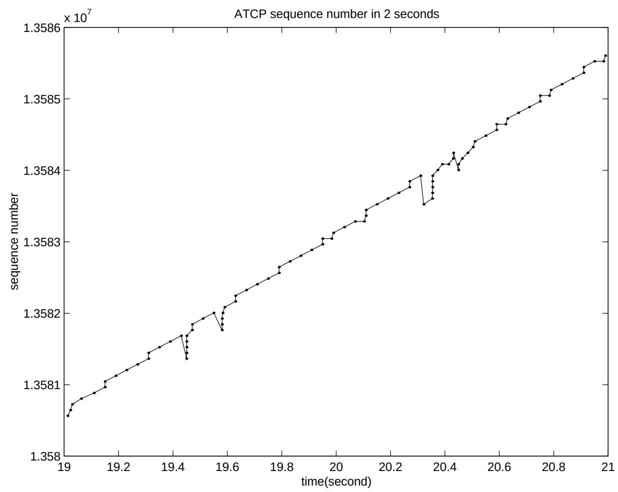

Figure 10: ATCP trace in the presence of bit error only.

TCP's congestion window never really has an opportunity to grow in size because losses due to bit error result in congestion control[5]. ATCP's congestion window, on the other hand, never shrinks. This accounts for the dramatic difference in TCP and ATCP performance illustrated in Figure 8. Finally, the linear increase in transfer time for plain TCP with increasing RTT is explained by the fact that TCP's congestion window remains small making TCP behave almost like a stop-and-wait protocol. Thus, as the RTT increases, so does the transfer time.

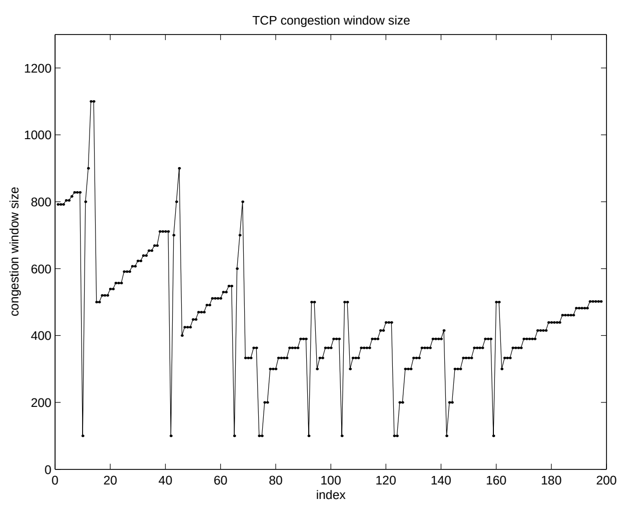

Figure 11: TCP congestion windows in the presence of bit error only.

[5]It is noteworthy that in Figures 11 and 12, we see the effect of timeouts (where the congestion window shrinks to 100 bytes) as well as the effects of duplicate ACKs (where the window is cut in half and grows linearly from that point on) on TCP's congestion window.

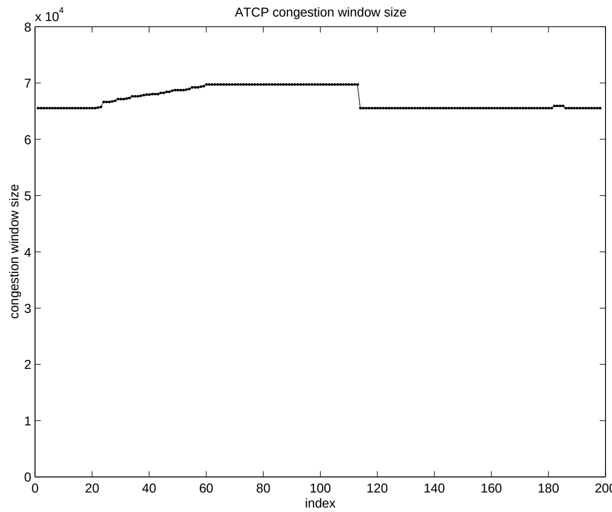

Figure 12: ATCP congestion windows in the presence of bit error only.

5.2 Congestion Case

In the next set of experiments, we introduced periodic *congestion* in the network every 5 seconds. In order to congest the intermediate node, a local source dumps packets into ip_output() for a period of 200ms. In Figure 13 we plot the transfer time for a 1MByte file as a function of mean hop-by-hop delay (the bit error rate is 10^{-5}). We notice that the file transfer time for TCP increases from about 1200s to almost 3000s while ATCP's file transfer time increases from approximately 460s to about 500s. Again, we can perform some rough calculations to determine the minimum time it takes to transfer the file in the presence of congestion. As before, it takes a minimum of 361s to transfer the file in the absence of congestion. Since congestion occurs every 5s for a period of 200ms, we will have approximately 80 congestion events during the file transfer each lasting 200ms. Thus, the additional file transfer time (assuming no data can be transferred during congestion) is 16s for a total time of 377s. As before, it is interesting to note that ATCP's file transfer time is quite close to this minimum.

There are a couple of reasons for the difference in performance between ATCP and TCP. First, the number of timeout events in TCP is high because of the high bit error as well as because of loss due to congestion. Thus, TCP does not get much of an opportunity to grow its congestion window. ATCP, on the other hand, defers to TCP's congestion control *only when* it receives an ECN message. In other cases, it enters the loss state and retransmits the lost packets from TCP's buffer. The slight increase in transfer time for ATCP as a function of hop-by-hop delay is caused because the round trip time affects the rate at which the congestion window can be grown. The effect of this is more pronounced in the case of TCP because TCP invokes congestion control very often.

5.3 Partition Case

In this section, we consider the case when the network suffers periodic partitions. For our experiments, a network partition occurs every five(5) minutes (at an intermediate node), and the

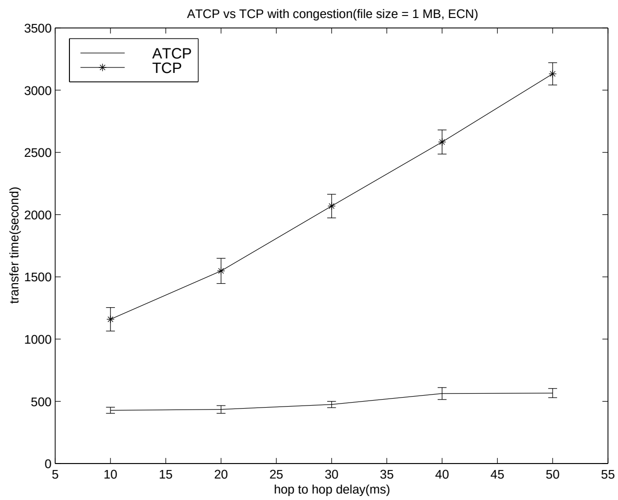

Figure 13: ATCP and TCP performance in the presence of bit error and congestion.

partition lasts for 1 minute. Figure 14 plots the file transfer time for a 1MByte file as a function of hop-by-hop delay. The transfer time for ATCP is almost constant at a little over 500s while TCP's file transfer time increases with hop-by-hop delay. It is easy to explain ATCP's time if we refer back to Figure 8. The transfer time for ATCP in the presence of loss only is about 425s. If the network gets partitioned every 5 minutes (i.e., every 300 sec) for one minute, we expect the transfer time to increase by at least the length of the partition, which is 60s. Note that ATCP puts TCP into persist mode upon receiving the ICMP destination unreachable message. In persist mode, TCP generates probe packets at exponentially increasing intervals (starting at 2s) up to a maximum interval of 60s. The effect of this behavior is that the sender does not realize that the network is connected until it sends out the next probe packet. In the worst case, this may happen anywhere from 32s to 60s after reconnection! This brings the total transfer time for ATCP to 425s + 60s (partition time) + 32s (time to realize the network is no longer partitioned) = 517s. TCP's poor behavior is caused because of the high error (see Figure 8) as well as serial timeout behavior when the network is partitioned.

We also plot the delays for ATCP and TCP for transferring a 200 KByte file when network partitions happen every 10 minutes and last for five(5) minutes(see Figure 15). Our purpose here is to investigate TCP and ATCP performance when the network experiences long network partitions. As in the short network partition case, delays for ATCP almost remain a constant while delays for ATCP grow linearly as the hop-to-hop delay increases.

5.4 Packet Reordering

Packet reordering may happen when there are *multiple routes* available from the source to the destination or when *route recomputing* occurs. When a router has more than one outgoing interface that leads to the same destination, it can distribute incoming packets among those different interfaces provided that the packets are going to that same destination. These packets may, therefore, arrive at the destination out of order because they have taken different routes. Another reason for packet reordering is route recomputation. This happens when a router fails to locating an outgoing route

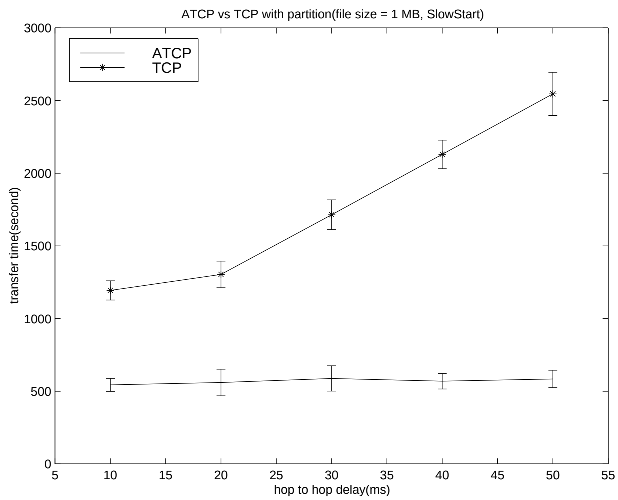

Figure 14: ATCP and TCP performance in the presence of bit error and partition.

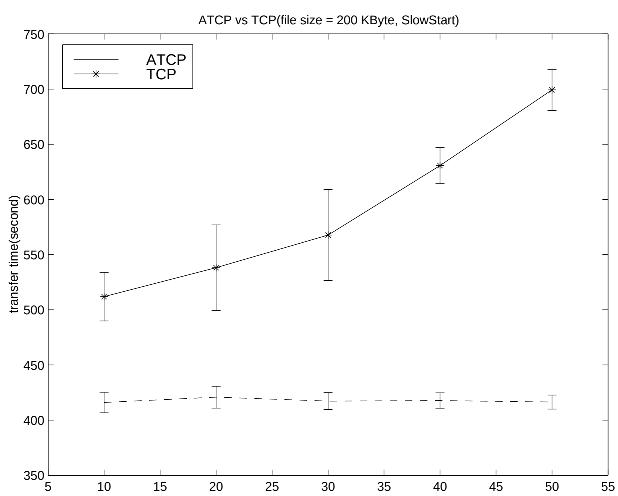

Figure 15: ATCP and TCP performance in the presence of bit error and larger partition.

to forward a packet. In ad hoc wireless networks, route failure occurs frequently. Packets in the previous route and those that take the new route may reach their destination in a different order, see Figure 16[6].

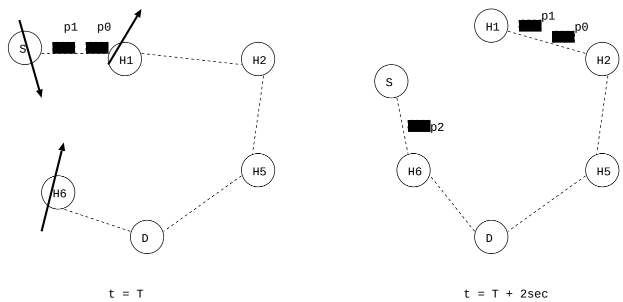

Figure 16: Route recomputation causes packet reordering.

In our experiment, we simulate packet reordering as follows. We set a timer to expire every 25 seconds on one of the intermediate hosts. The next four(4) packets are then inserted into the front of the packet queue when the timer expires. Figure 17 plots the transfer time needed by TCP and ATCP for a 1-MByte file. TCP needs much more time to transfer the same amount of data than ATCP does. The amount of time needed for ATCP remains almost a constant around 425 seconds while the transfer time for TCP increases (approximately) linearly from around 940s to 2010s as the hop-to-hop delay increases from 10ms to 50 ms. The reason for this difference is that ATCP puts the TCP sender into persist mode when it receives three consecutive duplicate acknowledgements. On the other hand, TCP will start congestion control algorithms resulting in a substantially smaller congestion window and slow congestion window growth.

5.5 Putting the pieces together

Finally, we wanted to compare the performance of ATCP and TCP in a network which experienced *all of the effects of network partition, multipath routing, congestion and bit error, together.* We continue to use 10^{-5} for the bit error rate. Network partition occurs, as before, every 5 minutes and lasts for 1 minute. Interfering traffic is generated at a rate of 8 packets/sec in one experiment and 16 packets/sec in another (100 byte packets) by each intermediate hop with the exception of the sender and receiver.

After recovering from a network partition in a real network, we expect the round trip time (rtt) as well as the bandwidth to change. To simulate the change in bandwidth, we modify the bandwidth at one intermediate hop each time a partition ends. The bandwidth at that hop can be either 16 kbps, 24 kbps or 32 kbps. If the bandwidth before partition was 32 kbps, after reconnection, either of 16 or 24 kbps is selected randomly, and so on. To simulate the change in rtt values, we do the same thing as for the bandwidth. Say the hop delay of the chosen hop was in the range [10-30] ms before partition. After partition, the delay for that hop is randomly set to lie in either a [60-80]

[6]Current routing protocols for ad hoc networks take a substantial amount of time to find new routes. Consequently the packets that have been sent before the rerouting may have ample time to reach their destination before a new route is found. So we believe that packet retransmission because of packet loss due to link error is the major reason for packet reordering, not route recomputation.

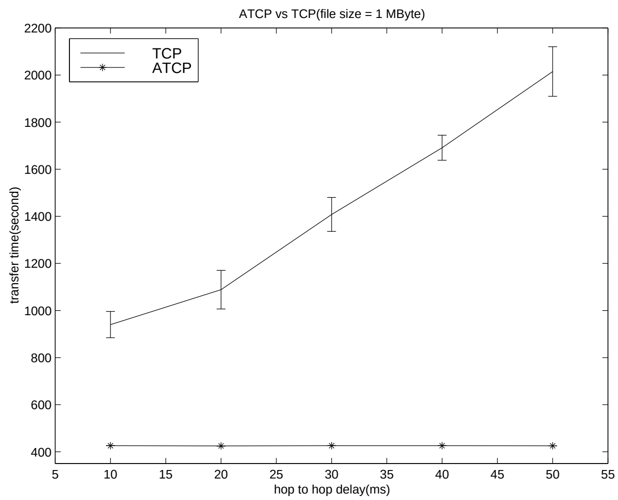

Figure 17: ATCP and TCP performance in the presence of bit error and packet reordering.

ms range or in a [100-120] ms range. Thus, the bandwidth as well as the rtt changes after each partition.

Figure 18 illustrates the performance of TCP and ATCP for transferring a 1MB file for two values of interfering traffic. We see a *1/3rd* reduction in transfer time for ATCP as compared with TCP in both cases. The reasons for this have been discussed earlier but it highlights the effectiveness of our solution.

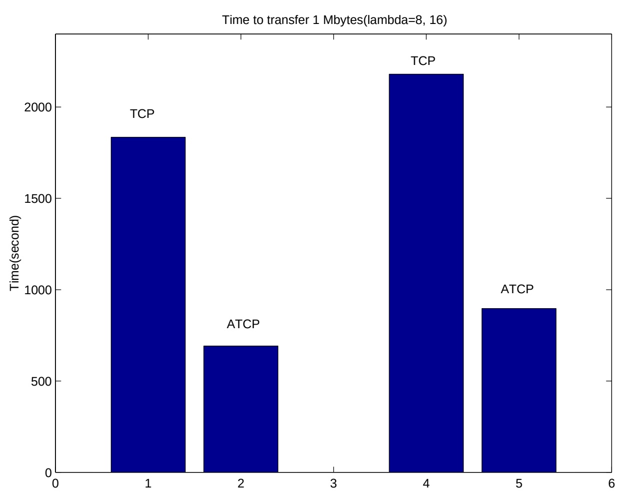

Figure 18: TCP and ATCP transfer time for 1 MB data in the general case.

6 Conclusions

In this paper we presented a solution to the problem of running TCP in ad hoc wireless networks. Our solution is to implement a thin layer between IP and TCP (called ATCP) that ensures correct

TCP behavior while maintaining high throughput. This is done by putting TCP into persist mode when the network is disconnected or when there are losses due to high bit error. The highlights of ATCP are:

1. End-to-end TCP semantics are maintained.
2. ATCP is transparent which means that nodes with and without ATCP can set up TCP connections normally.
3. ATCP's performance is almost ideal as measured by the time to transfer large files.
4. ATCP does not interfere with TCP's congestion control behavior when there is network congestion.

We believe that our solution is almost ideal for ad hoc networks as demonstrated by the performance results shown above.

References

[1] A. Bakre and B. R. Badrinath,"I-TCP: Indirect TCP for Mobile Hosts", *Proceedings of the 15th International Conference on Distributed Computing Systems, Vancouver, Canada,* June 1995, pp. 136–143.

[2] H. Balakrishnan, S. Seshan, and Randy Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks", *Wireless Networks*, Vol. 1, No. 4, December (1995).

[3] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks", *ACM Computer Communication Review* , Vol. 27(5), 1997, pp. 19-43.

[4] K. Chandran, S. Raghunathan, S. Venkatesan and R. Prakash, "A Feedback-based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks", *Proceedings 18th Intl. Conf. on Distributed Computing Systems (ICDCS'98)*, Amsterdam, The Netherlands, May 26 – 29, 1998.

[5] S. Floyd, "TCP and Explicit Congestion Notification", *ACM Computer Communications Review*, Vol. 24(5), October 1994, pp. 10 – 23.

[6] Jamal Hadi Salim and Uvaiz Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, July 2000.

[7] G. Holland and N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks", *Proceedings ACM Mobile Communications Conference (Mobicom'99)*, Seattle, WA, August 15 – 20, 1999.

[8] David B. Johnson and David A. Maltz. "Dynamic source routing in ad hoc wireless networks", In Tomasz Imielinski and Henry F. Korth, editors, Mobile Computing, pages 153–181. Kluwer Academic Publishing, 1996.

[9] P. Karn,"MACA – a New Channel Access Method for Packet Radio", in *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pp. 134-140, 1990.

[10] V. D. Park and M. S. Corson,"A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks", *Proc. IEEE INFOCOM'97*, Kobe, Japan, (1997).

[11] Charles E. Perkins and Pravin Bhagwat, "Routing over multi-hop wireless network of mobile computers", In Tomasz Imielinski and Henry F. Korth, editors, Mobile Computing, pages 183–205. Kluwer Academic Publishing, 1996.

[12] K. K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP". RFC 2481, January 1999 (Status: Experimental).

[13] R. Yavatkar and N. Bhagawat,"Improving End-to-End Performance of TCP over Mobile Internetworks", *IEEE 1994 Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA*, (1994).