Performance evaluation of TCP algorithms in multi-hop wireless packet networks‡

Shugong Xu*† and Tarek Saadawi
Department of Electrical Engineering
City University of New York
City College
CUNY Convent Ave at 138th St
New York
NY 10031
U.S.A.

Summary
Wireless packet ad hoc networks are characterized by multi-hop wireless connectivity and limited bandwidth competed among neighboring nodes. In this paper, we investigate and evaluate the performance of several prevalent TCP algorithms in this kind of network over the wireless LAN standard IEEE 802.11 MAC layer. After extensively comparing the existing TCP versions (including Tahoe, Reno, New Reno, Sack and Vegas) in simulations, we show that, in most cases, the Vegas version works best. We reveal the reason why other TCP versions perform worse than Vegas and show a method to avoid this by tuning a TCP parameter—maximum window size. Furthermore, we investigate the performance of these TCP algorithms when they run with the delayed acknowledgment (DA) option defined in IETF RFC 1122, which allows the TCP receiver to transmit an ACK for every two incoming packets. We show that the TCP connection can gain 15 to 32 per cent good-put improvement by using the DA option. For all the TCP versions investigated in this work, the simulation results show that with the maximum window size set at approximately 4, TCP connections perform best and then all these TCP variants differ little in performance. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS
TCP
802.11
multi-hop
wireless ad hoc networks

Published online: 5 December 2001

*Correspondence to: Shugong Xu, Dept. of Electrical Engineering, City University of New York, City College, CUNY, Convent Ave at 138th St, New York, NY 10031, U.S.A.
†E-mail: xu001@ieee.org
‡Prepared through collaborative participation in the Advanced Telecommunications & Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

Copyright © 2001 John Wiley & Sons, Ltd.
1. Introduction

In wireless mobile ad hoc networks, nodes may communicate with each other through multi-hop wireless links. There is no stationary infrastructure such as a base station. Each node in the network also acts as a router, forwarding data packets for other nodes. There are many challenges in the design of this kind of network, such as dynamic routing. In fact, much of the current research for mobile ad hoc networks investigates routing protocols [1].

However, in this paper, we focus on the performance of TCP within this kind of network and examine the problems caused by multi-hop connectivity. We do not address the link failure problem which is caused by mobility. Our target network is a wireless multi-hop network—the basis of the wireless mobile ad hoc networks.

Earlier research on wireless systems shows that TCP connections over wireless links are frequently plagued by problems such as high bit error rates, frequent disconnections of the mobile host, and low wireless bandwidth that may change dynamically. Much research has since focused on mechanisms to improve TCP performance in cellular wireless systems (e.g. References [2,3]). Further studies have addressed other network problems that negatively affect TCP performance, such as bandwidth asymmetry and large round-trip times, which are prevalent in satellite networks (e.g. Reference [4]). In other words, these studies proposed schemes to solve the one-hop wireless link problem. More recently, several researchers examined the performance of TCP on multi-hop networks [5–7]. The interaction between MAC and TCP layers was investigated in Reference [5], while the effects of link breakage were studied in References [6] and [7]. This problem was also addressed in Reference [4]. These will be discussed in more detail in the ‘related works’ section.

The primary objectives of this paper are:

- to investigate which TCP algorithm works best in multi-hop networks;
- to adjust the parameters in those TCP algorithms to fit this network environment; and
- to evaluate the impact of the delayed acknowledgment (DA) option defined in IETF RFC 1122.

The rest of the paper is organized as follows: Section 2 gives a brief introduction to TCP algorithms used in this research. The simulation environment and methodology we used for our simulation are reported in Section 3. Section 4 provides our simulation results without DA option. The impact of DA option will be shown in Section 5. Some discussion and related works are presented in Section 6. Finally, Section 7 concludes the paper based on the simulation results and analysis.

2. TCP Algorithms

The Transmission Control Protocol (TCP) is the prevalent reliable transport protocol used in the Internet today. It is a window-based ACK-clocked flow control protocol. It uses an additive-increase/multiplicative-decrease strategy for changing its windows according to network conditions. Starting from one packet (or a larger value in some TCP versions), the window is increased exponentially by one packet for every non-duplicate ACK until the estimate of network capacity is reached. This is the slow start (SS) phase, and the capacity estimate is called the slow start threshold. Once this threshold is reached, the source (sender) switches to a slower rate of increase in the window by one packet for every window’s worth of ACKs. This phase, called congestion avoidance (CA), aims to slowly probe the network for any extra bandwidth. The window increase will stop when it reaches the maximum TCP window size, which is defined when the connection starts. Otherwise, the window increase is interrupted when a loss is detected. Either the expiration of a retransmission timer or the receipt of three duplicate ACKs (fast retransmit) could result in such a loss detection. The source supposes that the network is in congestion and sets its estimate of the capacity to half the current window. Since non-negligible loss may not result from congestion in wireless mobile networks, TCP encounters problems in such an environment.

Here we do not address the modified TCP schemes which were proposed for the one-hop wireless environment, such as I-TCP, M-TCP, and SNOOP, etc. [2,3]. In fact, since they assume that there is no congestion loss in the wireless side and no non-congestion loss in the wire-line side, they are not suitable in multi-hop environments. In this paper, we investigate the TCP algorithms commonly used in the Internet. They include Tahoe, Reno, New Reno, Sack, and Vegas. One of the differences between these TCP versions lies in their methods of recovering from packet loss which supposedly comes from network congestion. In general, they include the sender and
the receiver. The sender side algorithms investigated in this paper include:

- Tahoe—performs basic slow start and congestion avoidance and fast retransmit.
- Reno—very similar to the Tahoe TCP, except it also includes fast recovery.
- NewReno—based on the Reno TCP, but modifies the action taken when receiving new ACKs. In order to exit fast recovery, the sender must receive an ACK for the highest sequence number sent.
- Vegas—including a modified retransmission strategy (compared to Reno) that is based on fine-grained measurements of the round-trip time (RTT) as well as new mechanisms for congestion detection during SS and CA phase.
- Sack—implements selective repeat, based on selective ACKs provided by the receiver.

The TCP receiver transmits acknowledgments (ACKs) to the sender as data segments arrive. This provides reliability, as the sender retransmits any segments that are not acknowledged by the receiver. RFC 1122 defines an optional delayed acknowledgment mechanism. DA allows a TCP receiver to refrain from transmitting an ACK for every incoming segment. However, the receiver must send an ACK for every second full-sized segment. In addition, an ACK cannot be delayed for more than 100 ms while waiting for a second full-sized segment to arrive. And the receiver must send an ACK for any out-of-order packet. Since ACKs are cumulative, using delayed ACKs has little impact on transmission reliability. Furthermore, DA conserve resources by decreasing the load on the network and the machines that must generate and process these segments. The details of above algorithms can be found in Reference [8] and the references therein.

3. Simulation Environment and Methodology

The results reported in this paper are based on simulations using the NS2 network simulator from Lawrence Berkeley National Laboratory (LBNL) [9], with extensions from the MONARCH project at Carnegie Mellon [10]. The extensions include a set of mobile ad hoc network routing protocols and an implementation of BSD’s ARP protocol, as well as an 802.11 MAC layer and two radio propagation models.

For more information about this software, we refer the reader to References [9] and [10]. The link layer of the simulator implements the complete IEEE 802.11 standard Medium Access Control (MAC) protocol Distributed Coordination Function (DCF) in order to accurately model the contention of nodes for the wireless medium. DCF is designed to use both physical carrier sense and virtual carrier sense mechanisms to reduce the probability of collisions due to hidden terminals. The transmission of each unicast packet is preceded by a Request-to-Send/Clear-to-Send (RTS/CTS) exchange that reserves the wireless channel for transmission of a data packet. Each correctly received unicast packet is followed by an Acknowledgment (ACK) to the sender, which retransmits the packet a limited number of times until this ACK is received. All nodes communicate with identical, half-duplex, wireless radios that are modeled after the commercially available 802.11-based WaveLan wireless radios which have a bandwidth of 2 Mbps and a nominal transmission radius of 250 m.

With a few exceptions, we adhered to the simulations parameters used in Reference [10]. The following is the description of our simulation setup. Each node has a queue (called IFQ) for packets awaiting transmission by the network interface, which holds up to 50 packets and is managed in a drop-tail fashion. DSR routing protocol was used. The TCP packet size was 512 bytes unless otherwise indicated. Note that the current version NS2 software can only support a fixed size TCP packet in each simulation. That is why we can use the packet index as the TCP sequence number in the next section.

We consider one type of network topology: a string topology with 8 nodes (0 through 7) as shown in Figure 1. The same topology was also used in Reference [5]. The distance between any two neighboring nodes is equal to 200 m, which allows a node to connect solely to its neighboring nodes. In other words, only those nodes between which a line exists can directly communicate. The same distances between neighboring nodes ensure that the nodes act equally in the simulation. In this paper, we assume that these connections carry large file transfers (i.e. infinite backlog which the TCP sender always has data to send out). Nodes are static.

![Fig. 1. String topology.](image-url)
4. Experiments of TCP Variants Without DA

We carry out the simulation results below to show:

(1) which TCP algorithm works best in wireless multi-hop networks; and
(2) the best TCP parameters in those TCP algorithms to fit the network.

In the experiments of this section, we turn the DA option off. The impact of DA option will be presented in Section 5. In this performance study, we set up a single TCP connection between a chosen pair of sender and receiver nodes and measured the successively received packets over the lifetime of the connection. The TCP good-put for each connection was measured after each simulation and averaged over 10 simulation runs.

4.1. TCP performance comparison

Figure 2 presents the measured TCP good-put as a function of the hop number. We repeat the simulation for different TCP sender versions, including Tahoe, Reno, New Reno, Sack, and Vegas. This figure shows that, for all these TCP versions, the good-put decreases rapidly when the number of hops is increased from 1, and then stabilizes once the number of hops becomes larger. This trend is similar to the throughput results reported in References [5] and [6]. We compare the performance among different TCP versions using this figure. They perform almost identical when the connections are less than three hops and they show some difference with larger numbers of hops. Except with Vegas, the differences are very small. The Vegas gains 15 to 20 per cent improvement over other TCP versions when the hop number is larger than 4. Although, with smaller hop numbers, it has almost the same, perhaps even slightly worse performance (around 0.1 per cent worse if the hop number equals 1).

In order to show the reason why other TCP versions perform worse, we present Figure 3 to compare the TCP throughput with the good-put. As we know, good-put only counts those packets effectively received once. The difference between these two metrics is rooted in unnecessary retransmissions—packets being received more than once. This occurs only when the TCP sender believes there is a packet loss, which is not always true. In Figure 3, to compare with Vegas, we use TCP Reno as an example, which is now the most popular TCP version. These two performance metrics are completely identical for Vegas. However, in the case of Reno, the throughput is bigger than good-put with connection paths longer than 4 hops. After carefully analyzing the trace of simulation, we found that one cause is time-out due to the loss of multiple TCP ACKs. Although the data packet has already been received by the destination, the ACK cannot travel back to the sender. Besides re-transmitting packets, the sender also supposes that the network is in congestion and sets its estimate of the capacity to half the current window. Usually, the sender has to start using SS (slow start) after timeout. Other versions like Tahoe, New Reno and Sack show similar results.

Figure 4 shows the related results. It includes six small figures. Figure 4(a), (b) and (c) shows the throughput results from one run using TCP Reno, Sack and Vegas, respectively. The experiment settings are exactly the same as those described above. In each figure, the measured throughput for the only
session using a specific TCP version (from node 1 to 5) is demonstrated. The plotted values are obtained by averaging over 1.0 s intervals. For example, let us take a look at Figure 4(a). The TCP sender (node 1) in this experiment ran TCP Reno. In the 120 s lifetime of this TCP connection, there are nine instances when the throughput went down to (or near) zero. That means the TCP performance degraded seriously. Every time after this, TCP restarted using ‘slow start’. Since only one connection exists in the experiment, this kind of pause is unexpected. The oscillation can only be explained by the TCP version not being well tuned for this wireless multi-hop network environment. We call it ‘instability’ of the TCP variant in this kind of network. From these four small figures in Figure 4, we find that only Vegas does not have this drawback. The other TCP variants suffer from this problem, including TCP Tahoe and New Reno (corresponding results not shown here).

For all the TCP sender algorithms studied in this paper, more experiments show that the oscillation never happens when the connection only has one or two hops. Another observation is that the oscillation becomes more serious when the packet size in the TCP connection is larger. Figure 4(d) contains one set of results. In this figure, packet size is 1460 Bytes. The sender ran TCP Reno. In the 120 s lifetime of this connection, the throughput went near zero 20 times. The TCP parameter maximum window size \(w\) also has an effect on this problem. In fact, the problem can be lessened or eliminated with a smaller maximum window size. We will discuss
this in more detail in next sub-section. Figure 4(e) demonstrates the serious oscillation which occurs when using TCP Reno, with window set at 8 and the packet size of 1460 Bytes. It is better than the window = 32 case, but the oscillation is still very serious. In the 120 s lifetime of this TCP connection, the throughput went near zero 16 times! Note this is the TCP setting used in Reference [6]. We believe this kind of performance degradation is no better than that resulting from link breakage due to the movement in mobile ad hoc networks [6].

In the following part of this sub-section, we will reveal why this problem happens by analyzing the multi-layer traces. We find this instability problem is always due to one node failing to reach its next hop. This sounds very strange, since the distance between each pair of neighboring nodes in our experiments is 200 m, and each node is static which has a transmission radius of 250 m. (We will show that this derives from some MAC layer features.) That node which fails to reach its next hop triggers a route failure. If it is an intermediated node, the node drops all queued packets (usually ACKs) to the next hop node and reports the route failure to the source. Here source means data packet source—either the TCP sender or receiver. After the source receives this message, it starts a route discovery. Until a route is found, no ACK packet arrives at the TCP sender until a route from node 5 to 1 becomes available again after 6.1 s. In this period, the TCP packet with sequence number 111 is retransmitted two times. Although these two packets arrive at the TCP receiver (node 5) safely, the corresponding ACK packet cannot be sent out, since no route is available. Note that in NS2, the TCP packet size is fixed and the sequence number here is counted in packets (or segments) instead of bytes. Figure 6 illustrates another part of the trace. It shows the packet events from 19.5 s to 23.0 s. Figure 4(e) evinces another throughput degradation in this period. It is the TCP packet 516 that cannot find a route this time. Note that there is no data packet drop due to route failure. As we will show below, the route failure is because of node 1 cannot reach node 2. Since node 1 is the TCP sender, the TCP packet in the IFQ of this node will not be dropped. It will trigger a route discovery immediately after the route failure is reported. Figure 7 is a zoom of Figure 6 around 20.0 s.

Now we will look at the cause of route failure, focusing on the case shown in Figure 6 and 7. By analyzing the simulation trace, we find that this is rooted in the MAC layer. Node 1 cannot reach node 2. After node 1 tries to contact node 2 and fails seven times, the MAC layer reports a link breakage. Note that seven retry is defined in IEEE 802.11. A part of the MAC layer packet trace is shown in Figure 8. In this figure, Data means TCP packet or TCP ACK packet. In reference to the MAC layer, they are all data

![Fig. 5. Part of the packet events of TCP session shown in Figure 4(e), Reno, window = 8, packet size = 1460 bytes.](image1)

![Fig. 6. Another part of the packet events of TCP session in Figure 4(e), Reno, window = 8, packet size = 1460 bytes.](image2)
from upper layer. RTS means ‘request to send’; CTS means ‘clear to send’. M-Ack means the MAC layer acknowledgment. (Refer to References [9] and [10] for more details about the implementation of IEEE 802.11 MAC layer in NS2 software.) From this figure, we can find that the major cause of node 1’s failure to reach node 2 is that node 2 cannot successfully receive the RTS of node 1. Node 1 sends out seven RTS packets between 19.981 s to 20.012 s. However, node 2 only successively receives three of these RTS packets. For some reason we will describe later, node 2 does not send back any CTS to node 1 after receiving these three RTS packets. Due to collision, the other four RTS packets dropped at node 2. Note that there are, in total, five MAC packets dropped in this figure. Four of them occur at node 2. The other one occurs at node 1. It is because node 1 quits the delivery of the MAC layer data packet. The MAC layer data packet is dropped by node 1. At the same time, a link breakage event is reported to the upper layer. Of course node 1 does not receive any CTS from node 2. Thus, after each failure to get a reply, node 1 defers a random back-off period before transmitting again. The back-off interval is chosen using the binary exponential back-off scheme.

Let us focus on what happens to the first RTS packet drop at 19.9826 s. After deferring a while, node 1 sends out the second RTS. That is the one sending out at 19.9826 s. At this time, node 4 is sending a data packet to node 5. Since node 1 cannot sense the transmission occurring at node 4, it sends out the RTS packet. Unfortunately, this packet experiences a collision at node 2. So, the cause of this collision must be the interference from node 4. There are, in total, four MAC packets dropped at node 2 in Figure 8. All of them are caused by collision with the TCP data packet from node 4. It must be stated that, in a carrier sense wireless network, the interfering range (and sensing range) is typically larger than the range at which receivers are willing to accept a packet from the same transmitter [11]. WaveLAN wireless systems are engineered in such a way. According to the IEEE 802.11 protocol implementation in the NS2 simulation software, which is modeled after the WaveLAN wireless radio, the interfering range and the sensing range are more than two times the size of the communication range. This is the reason why a collision occurs at node 2 when node 1 and node 4 are sending at the same time, even though node 4 cannot directly communicate with node 2. Node 2 is within the interfering range of node 4. This is a typical ‘hidden node problem’ in wireless packet networks. Node 4 is the hidden node in this case. It is within the interfering range of the intended destination (node 2) but out of the sensing range of the sender (node 1). Since the nominal communication range is 250 m, which is smaller than the interfering range, node 1 cannot hear the CTS packet from
node 4. Thus the virtual carrier sense mechanism can-
not work either in this case. Now, we can explain why
node 2 still cannot send back a CTS when node 4 is
sending, even if node 2 successfully receives the RTS
from node 1. Note that node 2 can sense node 4. This
is a typical 'exposed node problem' in wireless packet
networks.

Since node 4 is sending several back-to-back TCP
data packets, the channel is keeping busy. After
failing seven times to receive CTS from node 2,
node 1 quits and reports a link breakage to its upper
layer. Then a route failure event occurs. The TCP
session has to pause until the route becomes available
again. Since the time of recovering from a route
failure is always more than one second, as shown
in Figures 5 and 6, the TCP throughput in this period
is zero. Since this time period is larger than the TCP
timeout threshold, the TCP session has to restart from
a window of one after the route is discovered in the
TCP source node. It also need do retransmissions for
those un-acked packets. This will further hurt the TCP
good-put.

Now, it is clear that the exposed station problem
and collisions disable the intermediated node to reach
its next hop. The binary exponential backoff scheme
used in the MAC layer makes this worse since it
always favors the latest successful node. Since larger
data packet sizes, and back-to-back packets sending,
both increase the chance of the intermediated node
failing to obtain the channel, the node has to backoff
a random time and tries again. If it continually fails
to obtain the response from its intended receiver after
several tries, it will declare a link breakage. The result
is the report of a route failure. This explains why
the Vegas TCP version does not have what we call
the 'instability problem'. The most important reason
is Vegas's less aggressive window increasing policy.
Vegas can increment, decrement, or not adjust the
window by one segment every RTT, as opposed to
other TCP versions that function in a state of con-
stant linear growth. This is called 'the new Conges-
tion avoidance mechanism' in Vegas. In other words,
TCP Vegas does not continually increase the conges-
tion window during congestion avoidance. Instead,
it tries to detect incipient congestion by comparing
the measured throughput to its notion of expected
throughput. A similar congestion detection mecha-
nism is applied during slow-start to decide when to
change to the congestion avoidance phase. This can
avoid the window size growing too big. Then this
reduces the chance of one pair of nodes capturing the
channel for a long time.

Since the distance between each pair of neighbor-
ing nodes in our experiments is 200 m, and each node
has a nominal transmission radius of 250 m, we never
thought node 4 could interfere node 2 before we con-
duct this work. This means that we did not expect the
hidden node and exposed node problems exist in this
simple multi-hop 802.11-based network. In fact, no
existing work ever talks about these problems, even
those latest papers about 802.11 unfairness [12, 13].
While, as shown above, these two problems are inher-
ent when we use 802.11-based radio in multi-hop
networks. They will cause many un-reported prob-
lems and affect the performance of these networks.
Since this is beyond the range of this paper, we do
not discuss it further. In fact, we have another paper
addressing this that will be available soon.

One may think that this, what we call 'TCP insta-
ibility problem in multi-hop wireless network', is not
really a TCP problem. That is correct at some extent.
This problem is rooted in the MAC layer. The hid-
node problem (collision) and the exposed node
problem, along with the backoff scheme of 802.11
MAC protocol are the causes of the instability prob-
lem shown in TCP. However, this problem indeed
appears in TCP performance when TCP runs in this
kind of a network. It is a problem of TCP variants
except Vegas. It indicates that these TCP variants are
not well tuned for this wireless multi-hop network
environment. The bursty transmission feature in TCP
gives occasion to this problem. Moreover, in the next
sub-section, we will show a way to eliminate this
problem by adjusting one TCP parameter.

4.2. The TCP maximum window size

From the above analysis, it is clear that reducing
the back-to-back packets sending might be a way to
relieve the instability problem of TCP in multi-hop
wireless networks. On the other hand, in previous
works [5, 13], it was claimed that increasing the
TCP window to more than one packet size had no
beneficial impact on the network performance. If
the claim were true, then we should not need TCP
congestion control any more.

Having these questions in mind, in this sub-section,
we investigate the effect of TCP maximum window
size (window\textsubscript{\textit{\textendash}}). As we explained in Section 2, maxi-
mum window size is the limit of the real transmission
window size in a TCP connection. With this param-
eter being set at 4, the TCP source can only send out a
maximum of four packets before it receives an ACK
from the receiver. With window\textsubscript{\textit{\textendash}} 1, the next packet
can be sent out only after this packet’s ACK arrives at the sender. This means it is a ‘stop-and-go’ case. There is no contention in the channel, since the packet and the ACK take turns in using the channel.

Figure 9 shows the good-put performance between TCP Reno connections with different window_. Figure 10 is its zoom. They show the same trend with different window_. Furthermore, the performance is best when window_ is set at 4 and degrades with a larger maximum window size. Especially, at window_ 16 or 32, the good-put performance is even worse than the ‘stop-and-go’ case with window_ 1. That is because they suffer from the instability problem. In the ‘stop-and-go’ case, the performance does not compare badly with window_ = 4. That means using a very simple reliable transport protocol, which just waits for the ACK before sending out another packet, can achieve good-put performance that is comparable with the best parameter.

Figures 11 and 12 show the good-put performance of connections using TCP Sack with different a window_. They are almost completely identical to Figures 9 and 10. The other two TCP versions, Tahoe and New Reno, have similar performance results. We show some of these results in Tables 1 and 2.

Figures 13 and 14 are results from TCP Vegas. We found that the performance does not degrade if the maximum window size window_ is greater than 4.

Table 1 lists out the numerical results for the five TCP versions investigated in this work with window_ 1. Table 2 shows the results with window_ 4. We find that in these two cases, these TCP versions have almost the same performance. Since Table 2 shows the performance with the best value of this TCP parameter window_, we would assert that there is no major difference between these TCP versions with the parameter set at 4. The corresponding results with window_ 32 are illustrated in Table 3.

Comparing Tables 1 and 2, we find several interesting phenomena, which are not consistent with those reported in References [5] and [13]. Their results show that increasing the TCP window to more than...
Fig. 13. TCP Vegas good-put vs. number of hops; packet size = 512 bytes.

Fig. 14. Zoom of Figure 13.

Table 1. Good-put result with different TCP versions, window = 1, packet size = 512 bytes.

<table>
<thead>
<tr>
<th>Hops</th>
<th>Good-put(Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tahoe</td>
</tr>
<tr>
<td>1</td>
<td>894.737</td>
</tr>
<tr>
<td>2</td>
<td>443.569</td>
</tr>
<tr>
<td>3</td>
<td>276.987</td>
</tr>
<tr>
<td>4</td>
<td>178.918</td>
</tr>
<tr>
<td>5</td>
<td>145.359</td>
</tr>
<tr>
<td>6</td>
<td>134.286</td>
</tr>
<tr>
<td>7</td>
<td>130.198</td>
</tr>
</tbody>
</table>

Table 2. Good-put result with different TCP versions, window = 4, packet size = 512 bytes.

<table>
<thead>
<tr>
<th>Hops</th>
<th>Good-put(Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tahoe</td>
</tr>
<tr>
<td>1</td>
<td>918.746</td>
</tr>
<tr>
<td>2</td>
<td>453.53</td>
</tr>
<tr>
<td>3</td>
<td>282.699</td>
</tr>
<tr>
<td>4</td>
<td>211.9</td>
</tr>
<tr>
<td>5</td>
<td>186.382</td>
</tr>
<tr>
<td>6</td>
<td>172.428</td>
</tr>
<tr>
<td>7</td>
<td>162.912</td>
</tr>
</tbody>
</table>

Table 3. Good-put result with different TCP versions, window = 32, packet size = 512 bytes.

<table>
<thead>
<tr>
<th>Hops</th>
<th>Good-put(Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tahoe</td>
</tr>
<tr>
<td>1</td>
<td>922.776</td>
</tr>
<tr>
<td>2</td>
<td>454.584</td>
</tr>
<tr>
<td>3</td>
<td>276.987</td>
</tr>
<tr>
<td>4</td>
<td>178.918</td>
</tr>
<tr>
<td>5</td>
<td>145.359</td>
</tr>
<tr>
<td>6</td>
<td>134.286</td>
</tr>
<tr>
<td>7</td>
<td>130.198</td>
</tr>
</tbody>
</table>

one packet size has no beneficial impact on the network. However, for all TCP versions, and for most of the cases in our experiments, the good-put performance is best with window around 4. Our explanation is that there are buffers in the wireless nodes, even if they are only one hop away, and the receiver can gain better performance with bigger window. The only anomaly is in the case of TCP connections having 3 or 4 hops. In such cases, the performance is a little bit worse with the bigger window than with the parameter set as one. The results are the same for all the TCP versions investigated. We believe this is because the interfering range (and sensing range) is larger than the communication range. When the sender is 3 or 4 hops away from the receiver, the interference between the ACK and TCP packet becomes so serious that even the ‘stop-and-go’ case with window = 1 works better. We believe that the different conclusions in References [5] and [13] arose from a different MAC layer. The completely implemented IEEE 802.11 MAC layer (as in NS [9,10]) was not used in these works.

From this set of experiments, we can draw the following conclusions:

(a) Maximum window size affects TCP performance, especially for TCP versions other than Vegas.
(b) For all TCP versions, the good-put performance is best when the maximum window size window is set at approximately 4. Since this is the minimum window size required to enable the fast retransmission scheme, we believe it should be the choice for TCP connections running over IEEE 802.11 in ad-hoc networks.

Copyright © 2001 John Wiley & Sons, Ltd.
(c) With this ideal parameter, there is no significant difference between the performance of Vegas and other TCP variants. This is not consistent with those results from previous studies in the wired networks [14]. They claimed that TCP Vegas yielded 40 to 70 per cent better throughput than TCP Reno.

We emphasize that the results reported here are based on the TCP packet size of 512. With other TCP packet sizes, the above conclusions are still correct. However, due to space limitation, we have not shown them here. We believe that the limited capacity of the network is the major reason that a 4-segment window is the optimal setting for the experiments. Note that the available bandwidth of the multi-hop radio link is much less than that of the one-hop link, and the round-trip delay is larger than that of the one-hop link. Considering other issues, like TCP fast retransmission scheme, we would recommend using this ideal parameter in multi-hop TCP connections.

Note that with window_4, none of the TCP versions has the instability problem anymore. That is why they can achieve the same performance as Vegas. Figure 4(f) shows one run of the TCP four hops connection with window_4 and packet size being 1460. Comparing Figures 4(d)(e)(f), we see that the TCP instability problem in wireless multi-hop networks can be lessened or eliminated with a smaller maximum window size.

5. Experiments for TCP Variants with DA

In this section, we present the experimental results in which TCP variants run with the DA option. The TCP receiver refrains from transmitting an ACK for every incoming segment. Since the TCP data packet stream has to compete for limited bandwidth with its ACK stream in multi-hop wireless networks, we believe this DA option can improve the TCP performance significantly.

As in Section 4, we set up a single TCP connection between a chosen pair of sender and receiver nodes and measured the successively received packets over the lifetime of the connection. All results below are averaged over 10 runs unless otherwise indicated. First, we examine the impact of the maximum window size window_. We will see whether the TCP performance is still best when we set this parameter at 4.

5.1. Impact of the TCP maximum window size

As we did in Section 4.2, we investigate the effect of TCP maximum window size (window_). Figure 15 shows the good-put performance between TCP Reno connections with different window_. Figure 15 corresponds to Figure 9, except when the DA option is used. The major variation between Figures 8 and 9 are the results of simulations in which window_ is set at 1. When the DA option is activated, the ‘stop-and-go’ model cannot work, since the DA scheme defers the ACK packet for which the sender is waiting. Since only one packet can be sent out in a round trip, the receiver cannot get the second packet to complete the delayed acknowledgment. Instead, it has to wait for a time-out, which occurs 100 ms after it receives the TCP packet [9]. So, the good-put performance is very bad. Figure 16 is a zoom of Figure 15. From it we can see that, when the DA is on, the performance is best when window_ = 4. With other values of this

![Fig. 15. TCP Reno good-put vs. number of hops; packet size = 512 bytes, receiver running DA.](image)

![Fig. 16. Zoom of Figure 15.](image)
parameter, TCP Reno performs either worse or almost similar to that performance from window = 4.

The corresponding results for other TCP variants, such as Tahoe, New Reno, and Sack, are similar. For TCP Vegas running with DA option, the results have little difference between different values of \(D\). When this parameter is set larger than 4. Since this value is the minimum value required to enable the fast retransmission scheme, it will be a good choice for TCP connections running over multi-hop wireless ad-hoc networks, either with or without DA option. Since DA option does not delay any out-of-order packet, so a window size equal to 4 still guarantees three duplicate ACKs if the first packet in a window is lost. We emphasize that the results reported here are based on the TCP packet size of 512. With other TCP packet sizes, the above conclusions are still correct. However, due to space limitation, we have not shown them here.

5.2. Performance improvement from the DA option

Table 4 lists the numerical results for the two TCP versions investigated in this work with window = 4. This table shows the performance with the best value of window. We also list the percentage of the improvement which results from the DA option, in addition to those good-put results. We find that running DA can result in 13 to 18 per cent good-put improvement when window = 4.

Table 5 lists the good-put results w/o DA option from three TCP algorithms, Reno, Sack and Vegas, with window set at 32. The percentage of the improvement which results from the DA option is also listed, along with those good-put results. It is clear that running DA can result in 15 to 32 per cent good-put improvement. That means with a larger value of window, DA option benefit the TCP good-put more. It is easy to explain this improvement is significant than that with window = 4. Before we do this, let us look at the reasons why the DA option increases the TCP good-put in wireless multi-hop networks.

There are two reasons. First, it cuts the amount of ACK packets traveling through the network roughly in half. This allows the data packet from the sender to meet less collision in the wireless link. Since the wireless channel is a shared resource for all neighbor nodes, a decrease in ACKs saves more bandwidth for the data packets. Another reason is that DA also reduces the instability in TCP algorithms other than Vegas. That is why the improvement in Vegas is less dramatic than other TCP algorithms. It is also the reason why TCP Reno can benefit more from DA option when its window set at 32 than that with a window of 4. Figure 17 illustrates this. By comparing this figure to Figure 4(a), we can see that DA option significantly reduce the instability in TCP algorithms.

5.3. Small file transmission

In last two sub-sections, we have shown that DA option can improve the TCP good-put significantly. It is good for large file transmission. When we transmit
the receiver will trigger a DA time-out; an ACK is sent back. After receiving this ACK, the TCP sender increases the window size to two; it sends out two packets. After receiving these two TCP packets, the receiver sends out another ACK, then the TCP increases the window further. So, with DA option, TCP slow start phase becomes really slow. For large file transmission, this has only little impact. However, for small file transmission, the increased delay is significant.

Fortunately, when we set another TCP option, this drawback will not exist anymore. In the experimental IETF RFC 2414 [15], a TCP initial window of 2 (or even larger) is proposed. We set this option in the TCP sender and set the initial TCP window size at 2. Then we repeat the above experiment. The corresponding results are shown in Figure 19. It is clear DA option can reduce the required time even a file (such as a FTP), we care about how much time is needed to finish the file transmission. We expect that less time will be needed with the DA option. However, a problem will arise when we set this option for small file transmission.

Let us look at Figure 18, in which we show the required time for a TCP connect transferring a small file, which only has 32 TCP packets (around 16kB), with window set at 4. For both TCP Reno and Vegas, the required time for the transmission with DA option is even larger than that without this option. This demonstrates that DA option is not good for small file transmission if we do not make any change to the current TCP algorithms. Since the normal TCP window starts from 1 in the beginning, the receiver with DA option does not send back an ACK immediately after receiving this packet. Due to DA option, it will wait for the second packet arriving. However, at this stage, the TCP sender is also waiting for the ACK from the receiver. Until 100 ms later,
for small file transmission. Figure 20 illustrates the required time for a larger file transmission, which has 2000 TCP packets (around 1 MB). These results show that TCP connections work better with DA option on.

In this paper, we use an extremely simple topology in which only two direct neighbors are competing for the link. In a real ad hoc network, there would usually be more than two nodes. Thus, more nodes will benefit from the reduction of ACK packets traveling in the network. The overall throughput of the network will improve more dramatically than our simple string topology simulations.

6. Discussion and Related Works

It is worth emphasizing that our above works are based on the NS2 simulator, in which wireless radio is modeled after the commercially available 802.11-based WaveLan. Since 802.11-based wireless multi-hop networks are widely used in almost all test-beds and simulations in the research of MANET (mobile ad hoc network) area, it is hard to believe that no one has explicitly indicated the potential existence of the hidden node and exposed problems. Even in the latest works related to the fairness of 802.11 [12,13], the authors did not mention these problems. In this work, we have shown that these problems are inherent in 802.11-based multi-hop networks. They could affect the TCP performance seriously. Since this is not the main topic of this paper, we do not discuss it further. Another important thing we need mention here is that the conclusions drawn in the above sections are correct not only in our simple static string scenario, but also in all other experiments in which we use different topologies and mobility patterns. Since the related experimental results are almost the same as that in this paper, and also due to space limitation, more results are not shown in this paper.

Recently, some researchers have studied the performance of TCP on multi-hop networks [5–7]. Gerla et al. [5] and Tang and Gerla [13] investigated the impact of the MAC protocol on the performance of TCP on multi-hop networks. They found that the interaction between TCP and MAC layer backoff timers causes severe unfairness and capture conditions. Their results show that increasing the TCP window to more than one packet size has no beneficial impact on the network performance. We believe that this differing conclusion in References [5] and [13] came from a different MAC layer. They did not use the completely implemented IEEE 802.11 MAC layer (as in NS [9,10]).

Holland and Vaidya [6] investigated the effects that link breakage due to mobility has on TCP performance. Chandran et al. [7] proposed the TCP-F (TCP-F-F) protocol, which uses explicit feedback in the form of route failure and re-establishment control packets. Since a wireless mobile ad hoc network must be a multi-hop network, we believe our work is a basis upon which to investigate the mobility problem. Furthermore, since link breakages are not caused by movement in some cases (as we have shown in this paper), some adjustments for the assumptions made in those papers might be needed.

Allman [16] showed that delayed acknowledgment mechanism hurts TCP performance, especially during slow start. They offered some schemes to overcome this shortage. However, in their work the target network was not a wireless network, as in this paper. Balakrishnan et al. [4] demonstrated that using delayed ACKs could increase performance in asymmetric networks. Their network was not the same as ours. The compete resource between the data stream and the ACK stream in neighbor nodes could not be modeled as bandwidth asymmetry. We also show that using the experimental draft 2414 [15], small file transmission can also benefit from the DA option.

7. Conclusions

In this paper, we have investigated and evaluated the performance of several prevalent TCP algorithms in wireless multi-hop networks over the wireless LAN standard IEEE 802.11 MAC layer.

After extensive comparison of the existing TCP versions (including Tahoe, Reno, New Reno, Sack and Vegas) by simulation, we have shown that, in most cases, TCP Vegas works best. The Vegas shows 15 to 20 per cent improvement over other TCP versions when the hop number is larger than 4. Since TCP Vegas also works better than other TCP versions in wired networks, it is recommendable to use Vegas in the new application in wireless multi-hop networks.

In addition, we demonstrate the reason why other TCP versions perform worse than Vegas and reveal the TCP instability problem by multi-layer trace analysis. Vegas does not have this problem. We also show a method to avoid this by tuning a TCP parameter—maximum window size. For all TCP versions investigated in this work, we show that the TCP connection performs best with this parameter tuned to approximately 4. Since this is the minimum...
window size required to enable the fast retransmission scheme, we believe it should be the choice for TCP connections (other than Vegas) running in these networks.

When the maximum window size is set at approximately 4, other TCP variants differ little in performance from Vegas. This is not consistent with those results from previous studies of wired networks [14]. They claimed that TCP Vegas yielded 40 to 70 per cent better throughput than TCP Reno. This is because the advantage of it cannot show up with such small a maximum window size, TCP Vegas does not perform better than the other TCP versions.

Furthermore, we investigate the performance of these TCP algorithms using the delayed acknowledgment (DA) option defined in IETF RFC 1122, which allows the TCP receiver to transmit an ACK for every two incoming packets. We show that the TCP connection can gain 15 to 32 per cent good-put improvement by using the DA option. Along with another option defined in RFC 2414 [15], DA option can also benefit small file transmission significantly. Moreover, the reduction for the number of ACK packets traveling in the network, many more nodes will benefit from DA option. The overall throughput of the network will show greater improvement than in our simple string topology simulations. So, we strongly recommend using DA option in TCP applications in wireless multi-hop networks.

Acknowledgments

We would like to thank the Monarch group at CMU, VINT group at USC and Berkeley for making the NS2 simulator available.

References


Authors’ Biographies

Shugong Xu (xu001@ieee.org) received a B.Sc degree from Wuhan University, China, and his M.E. and Ph.D. from Huazhong University of Science and Technology (HUST), Wuhan, China (all in electrical engineering). Between 1997 and 1999 he was with Tsinghua University, China, and Michigan State University. In 1999 he joined the Information Networking and Telecommunications group at City College of New York (CCNY), where he is a research scientist. His current research interest lies in wireless LAN, wireless mobile ad hoc networks, internet QoS, and multimedia communication. He has published about thirty technical papers in these areas.

Tarek N. Saadawi received a B.Sc. and an M.Sc. from Cairo University, Egypt, and a Ph.D. from the University of Maryland, College Park (all in electrical engineering). Since 1980 he has been with the Electrical Engineering Department, City University of New York, City College, where he currently directs the Information Networking and Telecommunications group at CCNY. His current research interests are telecommunications networks,
high-speed networks, multimedia networks, ad hoc networks, and packet radio networks. He has published extensively in the area of telecommunications networks. He is a co-author of the book Fundamentals of Telecommunication Networks (Wiley, 1994). He is also the lead author of the Egypt Telecommunications Infrastructure Master Plan covering the fiber Networks, IP/ATM, DSL, and the wireless local loop. He is a former Chair of IEEE Computer Society of New York City (1986–1987). He has received the IEEE Region 1 Award, 1987, and the NTT of America award for research on broadband telecommunication networks.