WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links*

Karunaharan Ratnam
Dept. of Electrical and Computer Engineering
Northeastern University
Boston MA 02115
karu@ece.neu.edu

Ibrahim Matta
College of Computer Science
Northeastern University
Boston, MA 02115
matta@ccs.neu.edu

Abstract

The Transmission Control Protocol (TCP) used in the Internet has been mainly designed assuming a relatively reliable wireline network. TCP assumes that any *loss is due to congestion and consequently invokes congestion control measures. This has been shown to yield poor performance in the presence of wireless links as a large number of segment losses will occur more often because of wireless channel errors or host mobility. We present an efficient transmission control scheme (WTCP) that requires the base station to buffer data packets destined for mobile host and retransmit lost data. Through simulations, we show that our scheme yields better throughput than other existing proposals* while *maintaining TCP's end-to-end semantics. One salient feature of WTCP is that it effectively hides the time spent by the base station to locally recover so that the TCP's round trip time estimation at the source is not affected. This is critical since otherwise the ability of the source to effectively detect congestion in the fixed wireline network will be hindered.*

1. Introduction

Technology advances in communication, hardware and software, and the ever increasing need for untethered access to Internet and Intranet has created new opportuinities for wireless data comminication. However, the transmission quality of wireless channel is characterised by bursty errors with high bit error rate (BER) as oppose to randomly occuring errors with low BER in wired link. The transmission quality over wireless channel is further degraded by the hand-off related intermittent delay or loss of the connection. Hence, the design of a transmission protocol for a network with wireless links must take into account these additional losses in wireless link. The design of such transmission protocol must account for the fact that wireless hosts commonly communicate with hosts in the fixed part of the network, and the transmission procedures used over the wireless link must be compatible with their peer procedures employed in the fixed network. When used in a wireless environment, the Transmission Control Protocol (TCP) [5], developed for fixed networks with relatively reliable links, wrongly assumes segment loss due to wireless link errors to be a sign of network congestion and invokes congestion control mechanisms that curb the flow of segments on that connection. However, for better throughput, a segment loss due to wireless link errors must be detected and retransmitted as quickly as possible. We are proposing a new scheme to enhance the performance of the transport protocol in networks with wireless links. In our scheme the base station, through which the mobile user is connected to the rest of the network, plays a pivotal role by buffering the TCP segments and locally retransmitting them based on timeouts *as well as* duplicate acknowledgments.

Section 2 discusses related work. In Section 3, we present the proposed WTCP architecture. Section 4 describes the details of the simulations we have performed. In Section 5, we present the results of the simulation and compare the simulated protocols.

2 Related Work

There are many proposals to improve TCP performance for wireless users. Invariably, most of them, in one form or another, rely on the base station to improve performance.

*This work was supported in part by NSF CAREER grant NCR-9701988 and NEU grant RSDF-377090.

The first approach proposed was *Indirect-TCP* (I-TCP) [1]. It splits the transport link at the wireline–wireless border, and maintains two TCP connections, one over the fixed network, and another over the wireless link. This way, the poor quality of the wireless link is hidden from the fixed network. By splitting the transport link, I-TCP does not maintain end-to-end TCP semantics, i.e. I-TCP relies on the application layer to ensure reliability.

Another proposal *MTCP* [6] is similar to I-TCP, except that the last TCP byte of the data is acknowledged to the source only after it is received by the mobile host. However, as the authors have pointed out, MTCP semantics are not the same as TCP semantics.

In *Explicit Bad State Notification* (EBSN) [2], in addition to performing local retransmission upon detection of wireless losses, the base station sends a special message to source so as to avoid unnecessary source timeout. However, the main disadvantage of this approach is that it requires TCP code modification at the source.

The fast-retransmission approach [7] does not address the issue of wireless link reliability, but reduces the effect of mobile host hand-off related delay. In this approach, immediately after completing the hand-off, mobile host generates a certain number of duplicate acknowledgments which cause the source to retransmit the lost segment without waiting for the timeout period to expire. This requires modification to the TCP code at the mobile host.

In the Snoop [3] proposal, if the base station sees a duplicate acknowledgment from mobile host, it detects a segment loss and locally retransmits the lost segment and starts a timer. If this timer expires, the retransmitted segment is again retransmitted. Snoop does not completely shield the wireless link losses from TCP as duplicate acknowledgements them self can be lost in the presence of bursty error state of wireless link.

Of all the proposals discussed in this Section, I-TCP [1], MTCP [6] and EBSN [2] are the only ones that completely shield the wireless link errors from the fixed network. However, these approaches either violate TCP end-to-end reliability or requires source code modification. Our scheme, referred to as WTCP, maintains end-to-end TCP reliability and requires *no* modification to the TCP code running in the fixed host or the mobile host. Also, WTCP effectively shields wireless link errors and attempts to hide the time spent by the base station to locally recover so that the TCP's round trip time estimation at the source is not affected. This is critical since otherwise the ability of the source to effectively detect congestion in the fixed wireline network will be hindered.

3 WTCP Architecture

We propose an efficient mechanism, where the base station is involved in the TCP connection. The conceptual view of the transport link is shown in Figure 1.

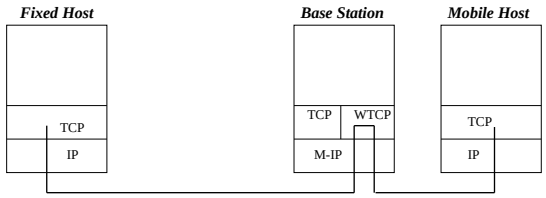

Figure 1. Transport connection.

WTCP receives data segment from source

```
wtcp_rcv_data()
1  if (pkt_seqno is for next pkt expected
   from src)
2     increment seqno of next pkt expected
      from src by pkt_size;
3     update receive array;
4     buffer pkt;
5     store arrival time;

6  if (pkt_seqno larger than seqno of next
   pkt expected from src)
7     update receive array;
8     buffer pkt;
9     store arrival time;

10 if (pkt_seqno less than seqno of next
   pkt expected from src)
11    ignore_packet;
```

The network layer protocol (viz. M-IP [8]) running in the base station detects any TCP segment that arrives for a mobile host and sends it to the WTCP input buffer. If this segment is the next segment expected from the fixed host, it is stored in the WTCP buffer along with its arrival time, and the receive array is updated. The receive array maintains the sequence numbers of the segments received by WTCP. The need for storing arrival time is explained in Section 3.1. The sequence number for the next segment expected from the fixed host is increased by the number of bytes received (`pkt_size`). If the newly arriving segment has a larger sequence number than what is expected, the segment is buffered, the arrival time is recorded and the receive array is updated, but the sequence number of the next packet expected is not changed. If the sequence number of the segment (`pkt_seqno`) is smaller than what is expected, the segment is ignored. Lines 1-11 show these steps in psedo-code.

WTCP sends data segments to mobile host

On the wireless link, WTCP tries to send the segments that are stored in its buffer. WTCP independently performs flow control for the wireless connection. It also maintains state information for the wireless connection, such as sequence number of last acknowledgment received from mobile host, and the sequence number of the last segment that was sent to the mobile host. From its buffer, WTCP transmits all segments that fall within the wireless link transmission window (`t_window`). Each time a segment is sent to the mobile host (including a retransmission), the timestamp of the segment is incremented by the amount of time that segment spent in the WTCP buffer (`residence_time`). When a segment is sent to the mobile host, the base station schedules a new timeout if there is no other timeout pending. Lines 12-18 show these steps.

```
wtcp_send_data()
12 while (seqno of next pkt to send to MH
   is within t_window)
13   retrieve pkt from buffer;
14   set timestamp to original timestamp
     plus residence_time;
15   if (no timeout pending)
        schedule new timeout;
16   if (no rtt estimation pending)
     // rtt: round trip time mark pkt
     // for new rtt estimate;
17   send(pkt);
18   increment seqno of next pkt to send
     to MH by pkt_size;
```

WTCP receives acknowledgement from mobile host

The base station acknowledges a segment to the fixed host *only after* the mobile host actually receives and acknowledges that segment. Hence, TCP end-to-end semantics is maintained throughout the lifetime of the connection. Also, the round trip time seen by the source is the actual round trip time taken by a segment to reach and return from the mobile host, *i.e.* it does not include residence time at the base station needed for local recovery. The corresponding pseudo code is given in lines 19-30.

Based on duplicate acknowledgment or timeout, the base station locally retransmits lost segments. In case of timeout, the transmission window for the wireless connection is reduced to just *one* segment assuming a typical burst loss on the wireless link is going to follow. If only one segment is lost and the following segment(s) pass(es) through, the loss would likely be indicated by a duplicate acknowledgment (see below). In case of timeout, by quickly reducing the transmission window, potentially wasteful wireless transmission is avoided and the interference with other channels is reduced.

```
wtcp_rcv_ack(ack)
19 if (seqno of ack is larger than last_ack)
20    set t_window to receiver advertised
      window; // opening window in full
21    set last_ack to seqno of ack;
22    set dupack counter to zero;
      // initialize duplicate ack counter
23    if (seqno of ack covers a segment
      marked for rtt estimate)
24       compute new rtt;
25    forward ack to src;
26    delete acked segments from buffer;

27 if (seqno of ack same as last_ack)
   // duplicate ack
28    increment dupack counter;
29    if (first dupack)
30       send lost pkt to MH;
```

The opening of the transmission window in regular TCP is based on whether the connection is in the slow start or congestion avoidance phase. In the slow start phase, each time an acknowledgment is received, the congestion window (cwnd) is incremented by one segment size. In the congestion avoidance phase, each time an acknowledgment is received, the congestion window is incremented by a factor of 1/cwnd. WTCP however opens the wireless transmission window completely assuming that an acknowledgement indicates that the wireless link is in good state. That is, when an acknowledgment is received, the transmission window size is set to the window size advertised by the receiver.

TCP-reno assumes a duplicate acknowledgment is caused by a TCP segment being lost due to congestion and the connection is put in the congestion avoidance phase. In other words, after a duplicate acknowledgment, the transmission window is halved and then incremented only by one segment per round trip time. However, for duplicate acknowledgment, WTCP does *not* alter the transmission window assuming that the reception of the duplicate acknowledgement is an indication that the wireless link is in good state, and immediately retransmits the lost segment. The base station continues to transmit the remaining segments that are within the transmission window if they have not already been transmitted. However, until the mobile host receives the lost segment, the reception of each out-of-order segment will generate a duplicate acknowledgment. The number of these additional duplicate acknowledgments can be determined by the base sta-

tion, which ceases to retransmit during their reception. By avoiding more than one duplicate acknowledgment based retransmission for a segment, WTCP improves the utilization of the wireless channel.

Clock granularity

In most multi-hop networks, the round trip time is typically in the order of few hundred milliseconds. Because of this reason, many TCP implementations use a coarse clock granularity. For example, 4.4BSD-Lite uses a 500ms clock [12]. On the other hand, the base station–mobile host round trip time is typically in the order of tens of milliseconds. Hence, in order to accurately estimate round trip time and to trigger timeouts in the millisecond range, WTCP uses a 10 ms clock granularity.

3.1 Effect of Local Retransmission on RTT Estimation

In TCP specified in RFC 1323 [5], the sender writes the current timestamp in the optional timestamp field in the TCP header, and the receiver echoes back this timestamp without modifying it, so that the round trip time (RTT) can be estimated. However, when an intermediate host (base station) buffers the segments and locally tries to retransmit them, the RTT computation can be affected. For example, if a segment is successfully retransmitted after one or more timeouts at the base station, the RTT value computed at the source for this segment will overshoot by the residence time of this segment at the base station buffer. If the burst error duration of the wireless link is significantly long, the residence time of the segment in the WTCP buffer may span a few source clock ticks and affect the RTT computation, especially its variance. Even after successful transmission resumes on the wireless link and acknowledgments are sent back to source, the RTT computation at the source may take a long time to settle down to the actual value. This problem exists in all the proposals that use local retransmission by the base station except I-TCP [1]. To hide the wireless link errors from the source, ideally the residence time of the segment at the WTCP buffer could be added to the timestamp value in the TCP header before the acknowledgment is forwarded back to the source. This has the effect of subtracting the residence time at the base station from the RTT value computed by the source.

This approach requires the base station to know the clock granularity of the source since the timestamp field in the TCP header contains a clock tick value, not the real-time clock value. In particular, when a system is booted, the clock tick is set to 0 and is incremented by one every 500 ms in 4.4BSD-Lite. To get around the problem of not knowing the clock granularity, an *approximation* to subtracting the exact residence time at the base station is as follows: when a segment is sent to the mobile host the base station can replace the timestamp of that segment with the timestamp of the *most recent segment* received from the source (see Figure 2:Case 1). When the acknowledgment comes back from the mobile host, it is forwarded to the source without any modification. When there are no source timeouts, the base station receives TCP segments from source at least once every RTT interval. Hence the maximum error in the RTT computation using the most recent timestamp is bounded by one RTT (see Figure 2: Case 1) when there are no source timeouts.

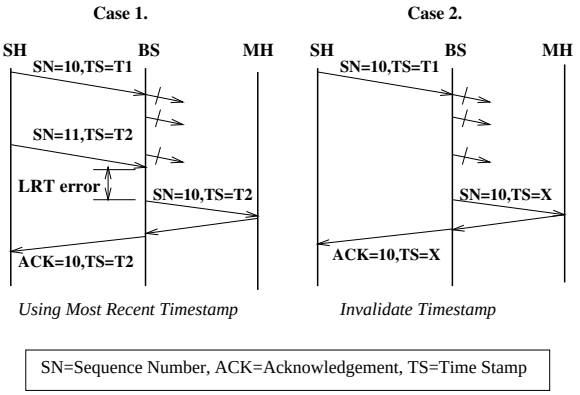

Figure 2. Hiding local recovery time

The above approach works fine as long as the base station frequently receives TCP segments from the source. However, if the source timeouts due to a prolong burst error state of the wireless link or due to a congestion in the wired portion of the network, the inter-arrival time between source TCP segments at the base station can be large. During this phase, the error in excluding residence time using the timestamp of the most recent segment received by the base station can be as large as the source back-off time (see Figure 2: Case 2). Any acknowledgment sent to the source using this timestamp will obviously affect the RTT estimate maintained at the source by a large factor. To avoid this problem, before sending a segment over the wireless link, the base station can invalidate timestamp option field if there has been any retransmission from the base station *after* receiving the most recent segment from the source. Timestamps are again used only after receiving a TCP segment from the source.

4 Simulation Model and Performance Measures

In our evaluations, we used the network simulator **ns** v.1.2a3 [10], a public domain software from Lawrence Berkeley National Lab. We extended ns to incorporate wireless link characteristics and to simulate WTCP, I-TCP, and Snoop. For all three protocols, the clock resolution of the base station is changed to 10 ms, and the resolution of the fixed host is maintained at 100 ms. No changes are made to the fixed network hosts or the mobile hosts. However, the base station node is modified to receive, buffer and transmit packets to the mobile host.

In the case of I-TCP, the incoming segment from the fixed host is accepted and buffered, and an acknowledgment is *immediately* sent back to the source. A separate TCP connection is created for the mobile host, and all segments in the base station buffer are transmitted over this wireless connection.

In order to avoid buffer overflow at the base station, the maximum window size of the TCP connection on both sides of the base station is chosen to be smaller than its maximum buffer size. However, in the case of I-TCP, buffer overflow can easily happen due to the split nature of the connection. TCP handles the receiver buffer overflow situation by using receiver window size advertisements. Thus for the I-TCP simulation, we modified the ns code at the base station to advertise available buffer size to the source as the window size in every acknowledgement. When there is no buffer space available, the base station advertises a window size of zero, and the source refrains from transmitting any more data until the window size is increased. The base station will increase the window size only if it can accept at least one full TCP segment [11]. For WTCP and Snoop, the base station buffer cannot become full as the transport connection is *end-to-end*. For an end-to-end connection, the number of unacknowledged segments sent out by the source is bounded by the transmission window. Since the maximum window size is chosen to be smaller than the buffer size at the base station, the base station buffer can never overflow.

In WTCP, when the base station receives a segment from the fixed host, the segment is saved in the buffer. Whenever the base station is ready to transmit that segment, a new header is created, and every header field from the buffered segment is copied to the new header except the timestamp value and the checksum. As discussed earlier in Section 3.1, we modify the timestamp in two different ways to hide the effect of wireless link errors on the RTT estimation at the source. The *first method* assumes an ideal scenario where the clock granularity of the source is known to the base station, and the base station increments the timestamp by the actual (exact) residence time of the segment at the base station. The *second method* makes no assumption about the source clock granularity, and uses the timestamp of the most recent segment received from the source as an approximation to excluding residence time at the base station from the RTT estimate. These two methods are simulated separately and the results are compared in Section 5.

WTCP also measures the base station–mobile host RTT. If there is no RTT estimate pending, WTCP marks the next segment sent to the mobile host for RTT estimate and stores the sending time of that segment. When the first acknowledgement that covers the sequence number of this marked segment arrives at the base station, the base station computes the RTT by subtracting its sending time from its arrival time. Since only one hop is involved in the base station–mobile host link, the round trip time variation is small. Hence, the smooth round trip time estimation used in regular TCP is not employed in WTCP. When there is no base station timeout, timeout value is set to 1.5 times the wireless link RTT. However, after the first timeout, WTCP backs-off the next timeout by thrice the wireless link RTT. But, additional timeouts do *not* increase the timeout value any further. After receiving an acknowledgment, the timeout value is reset to 1.5 times RTT. The value of 1.5 is chosen so that any minor variation in the RTT do not trigger unnecessary timeouts. The selection of the back-off value depends on the nature of the wireless link. The first retransmission occurs at 1.5 $\times$ RTT. If the exact duration of the burst error (bad) state of the wireless link is known, the second retransmission can be made to occur immediately after the link returns to a good state. However, since the exact burst error duration is not known, the second retransmission can be made to occur after the mean burst duration by selecting a suitable back-off period. For example, if the mean burst duration is around 4 $\times$ RTT, then the back-off duration for the second retransmission can be set to 3 $\times$ RTT so that the total time for the second retransmission to take place is (1.5 $\times$ RTT + 3 $\times$ RTT =) 4.5 $\times$ RTT, which covers the mean bad state duration.

We also simulated the Snoop protocol based on [3] for comparison purposes. Snoop uses negative acknowledgment at the mobile host to improve performance. We did not implement negative acknowledgments because our intention is not to modify TCP at the mobile host.

Wireless channel error model

Unlike the wired link, wireless links are characterized by high bit error rates. The errors are generally bursty in nature with varying burst duration. We model the wireless link using a two-state Markov channel [4], where the channel is in either good state or bad state. In the good state the bit error rate (BER) of the channel is low, and in the bad state BER is high. We simulated the wireless link with bit error rates of 10^{-6} and 10^{-2} in good and bad states, respectively. The time spent in each state is modeled by an exponential distribution with mean good state duration fixed at 1 second. The mean bad state duration is varied from 10 ms to 100 ms.

4.1 Performance Measures

Our intention is to improve the throughput performance of a transport connection. The *throughput* is measured as the average number of bits successfully transmitted to the mobile host in a second.

Improving throughput may result in poor utilization of the wireless channel. For example, if the wireless channel is in bad state, aggressive retransmission by the base station will likely result in loss of segments. Though aggressive retransmission improves throughput, it also increases the interference with other wireless links. To capture this tradeoff, we measure *goodput* [2] defined as the ratio of the number of new (header + data) bytes transmitted over the wireless link and the total number of bytes transmitted (new and retransmissions).

5 Results and Discussion

We have simulated the network shown in Figure 3 with error free wired link and erroneous wireless link. The wired link bandwidth is 10 Mbps with 100 ms latency. The wireless side has an 800 Kbps link with 10 ms latency. Table 1 shows the parameters used in the simulation. The simulation is run for 200 seconds and the results are averaged over five independent runs (i.e. each run starts with a different random number seed).

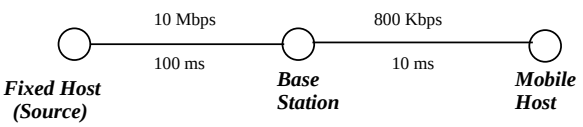

Figure 3. Simulated network.

Parameter	Value
Maximum window size	10 Kb
Buffer size at base station	20 Kb
Segment size (including header)	1 Kb
BER in good state	10^{-2}
BER in bad state	10^{-6}
Mean good period	1 sec

Table 1. Simulation parameters.

Figure 4 shows the trace of data transfer on the fixed link (left graph) and wireless link (right graph) using our WTCP protocol. The x-axis shows the time when the segment is put on the output queue of the fixed or wireless link, and the y-axis shows the segment's sequence number modulo 100. The plus sign shows the dropped segments. It can be seen from the fixed link trace that source retransmissions occur only at around 7 seconds. However, from the wireless link trace, it can be seen that the corresponding wireless burst error (bad) duration is around 1 second, which is around four times the RTT seen by the source.[1] If the burst duration is in the range of few hundred milliseconds as in typical RF wireless environments [4] there are no source timeouts.

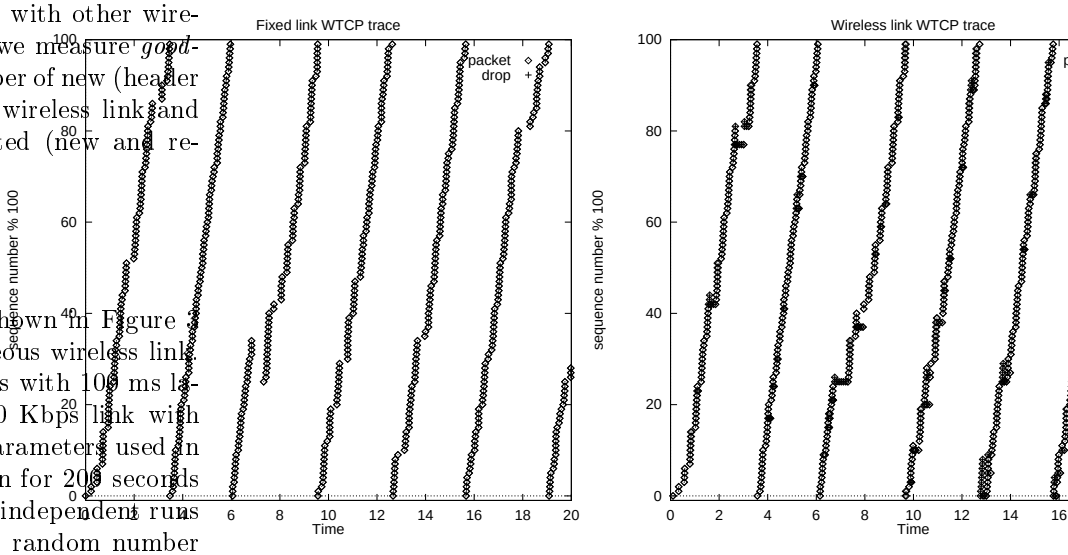

Figure 4. WTCP trace.

The throughput results of I-TCP, Snoop, WTCP and TCP-tahoe[2] are shown in Figure 5. For a good

[1] The round trip time from source to mobile host with 1 Kbyte data segment and 40 bytes acknowledgment for the network topology shown in Figure 3 is approximately 230 ms.

[2] We do not show results for TCP-reno as it was found to be less robust than TCP-tahoe in wireless applications [9].

quality wireless link, the throughput of I-TCP is better than other protocols. However, when the wireless link quality degrades WTCP yields better throughput mainly because of its aggressive retransmission policy over the wireless link. WTCP achieves throughput values 4-8 times higher than TCP-tahoe.

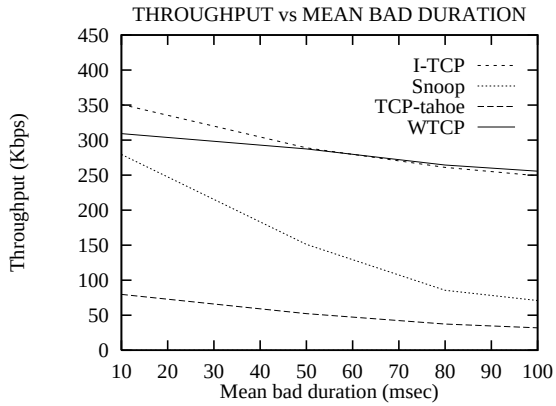

Figure 5. Throughput of different protocols.

Though Snoop achieves throughput values comparable to I-TCP and WTCP at low loss situations, the throughput of Snoop is poor when the wireless link is very bursty (in bad error state) for long durations. The reason is that Snoop triggers retransmission only after the base station receives a duplicate acknowledgment. However, when the wireless link is in bad error state, it is likely that acknowledgments be lost. In this situation, the base station fails to trigger duplicate acknowledgment based retransmission, and the throughput degrades.

WTCP's aggressive retransmission policy slightly reduces the goodput of the wireless link as shown in Figure 6. However, the throughput gain (4-8 times higher than TCP-tahoe) outweighs the small loss in goodput (around 2%). There is a clear tradeoff between throughput and goodput. One can improve goodput by using a less aggressive retransmission policy over the wireless link at the expense of decreased throughput. However, the gain in goodput will be minimal compared to the loss in throughput.

Effect of residence time on source RTT estimation

By storing TCP segments at the base station and locally retransmitting them, WTCP and other proposals shield most of the wireless link errors from the source and improve throughput for reliable bulk data transfer. But, the amount of time a segment spends in the base station buffer *is* an indication of the wireless link state, and should be hidden from the source as much as possible. Otherwise, the ability of the source to effectively detect congestion in the fixed wireline network will be affected. Depending on the nature of the wireless link, the amount of time a segment spends in the base station buffer (residence time) can span a few source clock ticks and offset the RTT estimate maintained at the source by a large value.

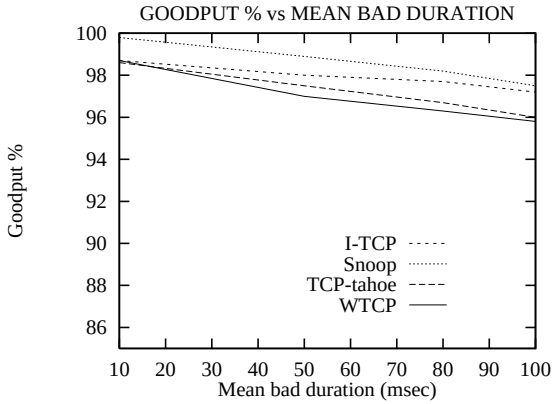

Figure 6. Goodput of different protocols.

In Figure 7 we plot the source timeout values, sampled every second, as a function of time. The timeout value is based on the smooth round trip time estimate[3]. The granularity of the source clock is 100 ms, and the mean bad period of the wireless link is 100 ms. Figure 7 shows the effect of (a) not hiding the residence time; (b) excluding actual residence time assuming the granularity of the source clock is known to the base station; and (c) using the timestamp of the most recent segment received from the source with timestamp removal option (discussed earlier in Section 3.1). When the residence time is not excluded, the timeout value reaches as high as 2.4 seconds, a five fold jump from the timeout value when segments are not buffered at the base station.

When the timestamp of each segment is replaced by the timestamp of the most recent segment received by the base station, or when the exact residence time is added to the timestamp, the timeout value varies between 500–700 ms. However, when the exact value of the residence time is excluded, the timeout value varies more slowly compared to when most recent timestamp is used. Variations in the timeout value when the exact value of residence time is used are due to queuing delays. In particular, if there is a retransmission at the base station, WTCP closes the transmission window

[3] This does not include the back-off value. The amount of time scheduled for a timeout, *i.e.* retransmission timeout (RTO), is defined as the product of the timeout value and the back-off value.

to just one segment. However, when an acknowledgment is received, the transmission window is opened fully, and the packets are queued on the wireless link for transmission. For an 800 Kbps link, the maximum queuing delay is 90 ms if ten 1 Kbyte packets are queued.

If, as in most TCP implementations, a coarser clock granularity is used at the source, the queuing effect is indiscernible. In Figure 8 we show the plot of timeout values when the source clock granularity is 300 ms. When the timestamp of the most recent segment is used or the exact residence time is excluded, the timeout value is always 900 ms (i.e. 3 clock ticks). However, if the timestamp is not modified (i.e. residence time is not excluded) the timeout value reaches as high as 2.4 seconds.

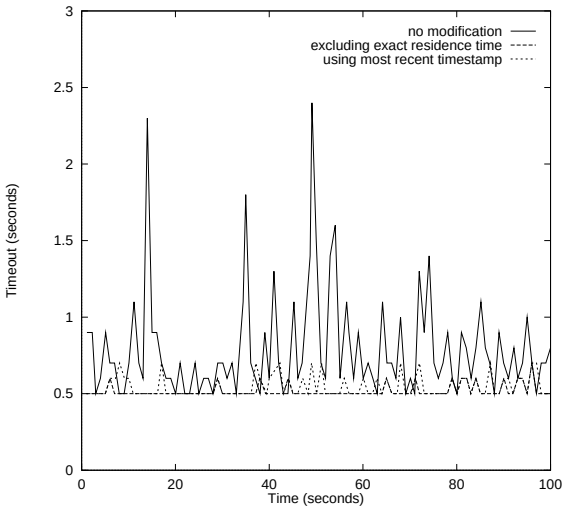

Figure 7. Plot of sample timeout values with 100 ms clock.

6 Conclusion and Future Work

We presented a new reliable transport layer protocol, called WTCP, for a network with wireless links. While maintaining end-to-end TCP semantics and requiring no modification to TCP running on the fixed or mobile host, WTCP runs on the base station and achieves very high throughput compared to TCP-tahoe. We have simulated other protocols (I-TCP and Snoop) for comparison purposes. WTCP performs better or as well as other protocols in terms of throughput with only a slight loss in goodput. WTCP effectively hides the time spent by the base station to locally recover so as not to hinder the source's ability to effectively detect congestion in the fixed wireline network.

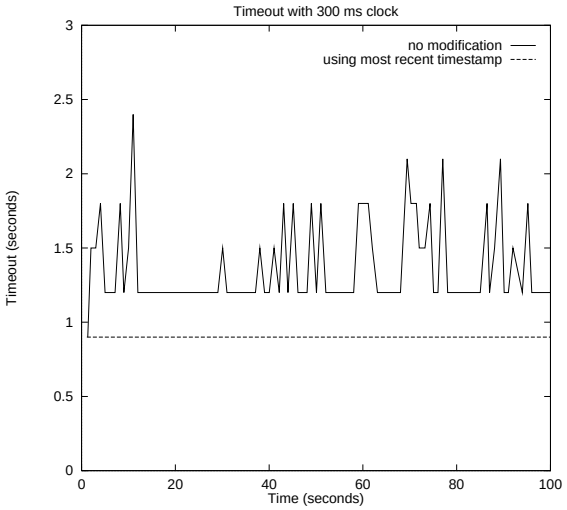

Figure 8. Plot of sample timeout values with 300 ms clock.

In this paper, mobility hand-off of the mobile host is not considered. The mobile host is assumed to be connected to a base station throughout the lifetime of the data transfer. We are currently developing a protocol to support mobility. This protocol is multicast-based, where duplicate segments are sent to where the user is *about* to move. In this scheme, we expect WTCP to perform effectively as the time required to complete the hand-off to the new base station is small (in the order of tens of milliseconds).

References

[1] Ajay Bakre and B. R. Badrinath. Handoff and systems support for indirect TCP/IP. In USENIX Association, editor, *Proceedings of the second USENIX Symposium on Mobile and Location-Independent Computing: April 10–11, 1995, Ann Arbor, Michigan, USA*, pages 11–24, Berkeley, CA, USA, April 1995. USENIX.

[2] Bikram S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan. Improving performance of TCP over wireless networks. Technical Report 96-014, Texas A & M University, 1996.

[3] Hari Balakrishnan, Venkata N. Padmanabhan Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance in wireless networks. In *ACM SIGCOMM Symposium on Communication, Architectures and Protocols*, August 1996.

[4] Pravin Bhagwat, Partha Bhattacharya, Arvind Krishna, and Satish K. Tripathi. Enchancing throughput over wireless LANs using channel state dependent packet scheduling. In *Proc. IEEE INFOCOM'96*, pages 1133–40, March 1996.

[5] D. Borman, R. Braden, and V. Jacobson. RFC 1323: TCP extensions for high performance, May 1992. Obsoletes RFC1185.

[6] Kevin Brown and Suresh Singh. A network architecture for mobile computing. In *Proc. IEEE INFOCOM'96*, pages 1388–1396, March 1996.

[7] Ramon Caceres and Liviu Iftode. Improving the performance of reliable transport protocol in mobile computing environment. *IEEE Journal of Selected Areas in Communications*, 13(5), June 1995.

[8] C. Perkins. RFC2002: IP mobility support, October 1996.

[9] T.V. Lakshman and Upamanyu Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3), June 1997.

[10] Steven McCanne, Sally Floyd, and Kevin Fall. Network simulator. Public domain software, http://www-nrg.ee.lbl.gov/ns/.

[11] W. Richard Stevens. *TCP/IP Illustrated* The Protocols. Addison-Wesley, Reading, MA, USA, 1994.

[12] Gary Wright and W. Richard Stevens. *TCP/IP Illustrated: The Implementation*. Addison-Wesley, Reading, MA, USA, 1995.