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

Karunaharan Ratnam
Dept. of Electrical and Computer Eng.
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 the mobile host and retransmit lost packets. 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 wireline network will be hindered.*

1. Introduction

Technology advances in communication, hardware and software, and the ever increasing need for untethered access to the Internet and Intranet has created new opportunities for wireless data communication. However, the transmission quality of a wireless channel is characterized by bursty errors with high bit error rate (BER) as opposed to randomly occurring errors with low BER on a wired link. The transmission quality over wireless links 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 wireless losses. Such a design 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) [3], developed for fixed networks with relatively reliable links, wrongly assumes segment loss in the wireless portion of the network 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 wireless loss must be detected and retransmitted as quickly as possible. We are proposing a new scheme to enhance the performance of TCP in networks with wireless links. In our scheme the base station, through which the mobile user is connected to the fixed wired 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.

The first approach proposed was *Indirect-TCP* (I-TCP) [1]. It splits the transport connection at the wireline–wireless border, and maintains two TCP connections, one over the wired network, and another over the wireless link. This way, the poor quality of the wireless link is hidden from the fixed network. By splitting

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

the transport connection, I-TCP does not maintain end-to-end TCP semantics, i.e. an acknowledgment received by the source (fixed) host does not mean that the corresponding data has been successfully delivered to the receiving (mobile) host. Similar schemes to I-TCP are *MTCP* [11] and *M-TCP* [4], but they also violate TCP end-to-end semantics.

Snoop [2] is another proposal that works at the link layer. If the base station sees a duplicate acknowledgment from the 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 wireless link losses from TCP as duplicate acknowledgements themselves can be lost in the presence of bursty error conditions on the wireless link.

Our scheme, referred to as WTCP, maintains end-to-end TCP semantics 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 connection is shown in Figure 1.

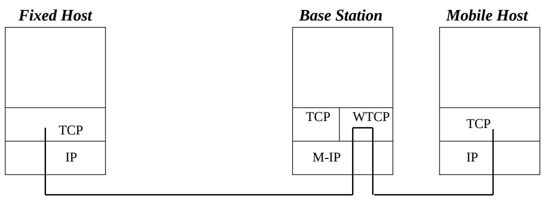

Figure 1. Transport connection.

WTCP receives data segment from source: The network layer protocol (viz. M-IP [5]) 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 times 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 dropped. Lines 1-11 of Figure 6 show these steps in pseudo-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 and error 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`)—see Section 3.1. 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 of Figure 6 show these steps.

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 of Figure 6 .

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.

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 opens the wireless transmission window in full 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 station, 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 [10]. 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 the 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 [3], 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 source 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). To bound the approximation error, WTCP at the base station replaces the timestamp only if the most recent segment received from the source has arrived after its last transmission to the mobile host. Otherwise, WTCP invalidates the timestamp field as shown in Figure 2, Case 2.

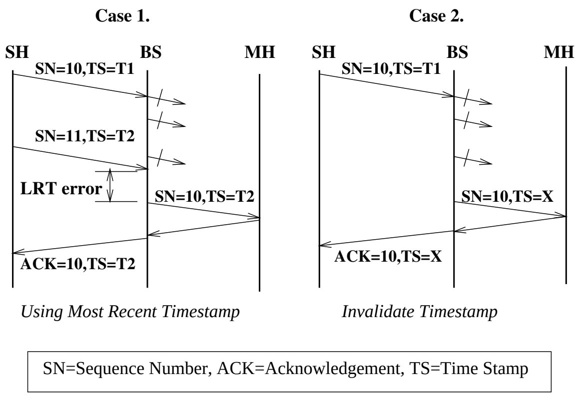

Figure 2. Hiding local recovery time

Parameter	Value
Maximum window size	10 Kbytes
Buffer size at base station	20 Kbytes
Segment size (including header)	1 Kbyte
BER in good state	10^{-6}
BER in bad state	10^{-2}
Mean good period	1 sec
Base station clock resolution	10ms
Source host clock resolution	100ms

Table 1. Simulation parameters.

4 Simulation Model and Performance Measures

In our evaluations, we used the network simulator **ns** v.1.2a3 [7], 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. See [8, 9] for more details. We model the wireless link using a two-state Markov process, where the channel is in either good state or bad state. We fixed the mean duration of the good state at 1 second. The mean bad state duration is varied from 10 ms to 100 ms.

5 Results and Discussion

We have simulated the network shown in Figure 3 with error-free wired link and erroneous wireless link. 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).

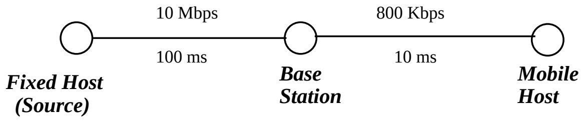

Figure 3. Simulated network.

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. The throughput results of I-TCP, Snoop, WTCP and TCP-tahoe[1] are shown in Figure 4. For a good 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.

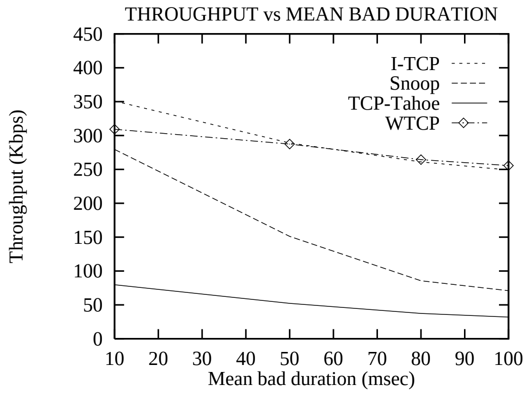

Figure 4. Throughput of different protocols.

As discussed in Section 3.1, the residence time a TCP segment spends in the base station buffer can affect the RTT value estimated at the TCP source. In order to see the effect of this residence time, we introduced congestion (random) loss on the wired link with probability 0.01. Figure 5 shows the throughput results of WTCP (including residence time, and excluding residence time using most recent timestamp), Snoop, and TCP-tahoe. By excluding residence time, the congestion losses are effectively detected, and WTCP gains around 15% more throughput.

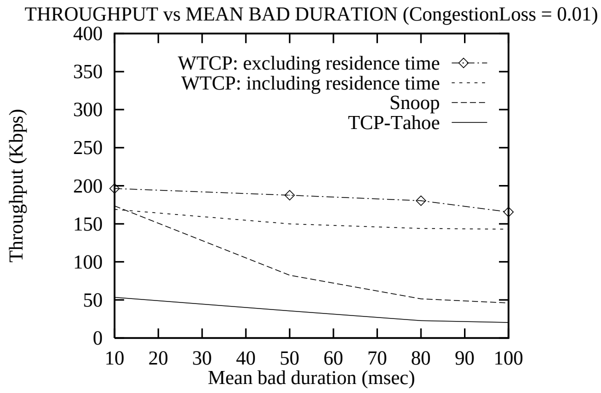

Figure 5. Throughput in the presence of congestion loss.

6 Conclusion and Future Work

We presented a new reliable transport-level scheme, called WTCP, for a network with wireless links. While maintaining end-to-end TCP semantics and requiring

[1] We do not show results for TCP-reno as it was found to be less robust than TCP-tahoe in wireless applications [6].

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 [8]. 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.

References

[1] A. Bakre and B.R. Badrinath. Handoff and systems support for indirect TCP/IP. In *Proceedings of the second USENIX Symposium on Mobile and Location-Independent Computing*, April 1995.

[2] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. A comparison of mechanisms for improving TCP performance in wireless networks. In *ACM SIGCOMM*, August 1996.

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

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

[5] IP Mobility Working Group. IP mobility support. In Charles E. Perkins, editor, *Internet Draft*, April 1996.

[6] T.V. Lakshman and U. 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.

[7] S. McCanne, S. Floyd, and K. Fall. Network simulator. Public domain software, http://www-nrg.ee.lbl.gov/ns/.

[8] K. Ratnam and I. Matta. WTCP: An efficient transmission control protocol for networks with wireless links. Technical Report NU-CCS-97-11, Northeastern University, 1997. http://www.ece.neu.edu/personal/karu/papers/WTCP-NU.ps.gz.

[9] K. Ratnam and I. Matta. Effect of local retransmission at wireless access points on the round trip time estimation of TCP. *IEEE 31st Annual Simulation Symposium '98*, April 1998. http://www.ece.neu.edu/personal/karu/papers/ss98.ps.

[10] G. Wright and W.R. Stevens. *TCP/IP Illustrated: The Implementation.* Addison-Wesley, Reading, MA, USA, 1995.

[11] R. Yavatkar and N. Bhagawat. Improving end-to-end performance of TCP over mobile internetworks. In *IEEE 1994 Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, 1994.


```
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     drop_packet;

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_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;
```

Figure 6. WTCP pseudo-code.