Efficient TCP over Networks with Wireless Links

Elan Amir, Hari Balakrishnan, Srinivasan Seshan, Randy H. Katz[1]
{elan,hari,ss,randy}@CS.Berkeley.EDU
Computer Science Division
University of California at Berkeley

Abstract

TCP is a reliable transport protocol tuned to perform well in traditional networks made up of wired links with stationary hosts. Networks with wireless links and mobile hosts violate many of the assumptions made by TCP, causing degraded performance. In this paper, we describe a simple protocol that improves TCP performance by modifying network-layer software only at a basestation without violating end-to-end TCP semantics. The main idea is to cache packets at the basestation and perform local retransmissions. Simulations of this protocol show that it is significantly more robust in the presence of multiple packet losses in a single transmission window as compared to TCP. This enables our protocol to tolerate at least 10 times as high an error rate without any performance degradation.

1 Introduction

The significant increase in recent activity in the area of wireless networks indicates that mobile hosts along with their wireless links will be an integral part of future networks. Communication over such networks shows quite different characteristics compared to traditional wired networks. Wireless links do not (at present) have bandwidths as high as on wired links and exhibit much higher error rates. Furthermore, since the predominant model for wireless communication is the cellular connection, frequent delays are caused due to *handoffs* as users move from cell to cell. These handoffs involve transfer of communication state (typically network-level state) from one *basestation* (a router between a wired and wireless network) to another, and typically last for anywhere between a few tens to a few hundreds of milliseconds [CI94, KMS+93, SLBR94].

Reliable transport protocols like TCP [CS94, RFC89] have been tuned for traditional networks made up of wired links and stationary hosts. TCP performs very well on such networks by adapting to end-to-end delays and packet losses. TCP provides reliability by maintaining a running average of estimated round-trip delay and mean deviation, and by retransmitting any packet whose acknowledgment isn't received within twice the deviation from the average. Due to the relatively low error rates over wired networks, packet losses are assumed to be as a result of congestion. Unfortunately, in the presence of the high error rates characteristic of wireless links, this assumption causes TCP to suffer a significant degradation in performance in the form of poor throughput and very high interactive delays when used in a wireless environment [CI94]. This behavior arises due to the fact that errors on the wireless link or delays due to handoffs are incorrectly interpreted by TCP at the sender as congestion. In response to this "congestion", TCP drops its transmission window size before retransmitting packets, initiates congestion control mechanisms (slow start [Jac88]) and resets the retransmission timer using an exponential backoff computation from the previous value (Karn's Algorithm [KP87]), thereby unnecessarily reducing its bandwidth utilization.

In this paper, we describe a simple protocol to alleviate this degradation and present the results of several simulations of this protocol. Our aim is to improve end-to-end performance on networks with wireless links without changing existing TCP implementations at the end hosts and without recompiling or relinking existing applications. We achieve this by a simple set of modifications to the network-layer software at the basestation. Our simulations indicate that we achieve equivalent or better performance over a channel with a given error rate as compared to unmodified TCP (based on the 4.3BSD-Reno version) over a channel with one-tenth that error rate. Furthermore, we achieve bandwidth speedups of up to 25 times that of the unmodified TCP connection.

The rest of this paper is organized as follows. In Section 2, we describe and evaluate some design alternatives and related work that deal with this problem. In Section 3, we describe the details of our protocol and the modifications to the router software at the basestation. We present

1. This work was supported by ARPA Contract J-FBI-93-153.

the results of our simulations in Section 4, and conclude with our proposals for future work in Section 5. We are in the process of implementing the protocol on a platform consisting of IBM ThinkPad laptops communicating over a 2 Mbit/s DEC WaveLAN.

2 Design Alternatives and Related Work

The design of a reliable transport-layer protocol for networks with wireless links has received a lot of attention in the recent past [BB94, CI94, Yav94]. Several alternatives have been proposed in these papers to alleviate the poor end-to-end performance shown by unmodified TCP and unmodified routers. We summarize these alternatives in this section and point out the disadvantages of each method. We then briefly outline our method, which we describe in more detail in Section 3.

- **The I-TCP Approach [BB94]:** This involves splitting the TCP connection into two separate connections at the basestation -- one between the fixed host and the basestation, and the other between the basestation and the mobile host. Since the second connection is over a one-hop wireless link, there is no need to use TCP on this link. Rather, a more optimized wireless link-specific protocol tuned for better performance can be used[Yav94]. The advantage of this approach is that it achieves a separation of flow and congestion control of the wireless link from that of the fixed network. However, there are many disadvantages of this approach, including:

  1. *Semantics:* I-TCP acknowledgments and semantics are not end-to-end. Since the TCP connection is explicitly split into two distinct ones, acknowledgments of TCP packets arrive at the sender even before the packet actually reaches the intended recipient. I-TCP derives its good performance from this splitting of connections, but as we shall show, there is no need to sacrifice precise semantics to achieve good performance.

  2. *Application relinking:* Applications running at the mobile host have to be relinked with the I-TCP library and need to use special I-TCP socket system calls.

  3. *Software overhead:* Every packet needs to go through the TCP protocol stack and incur the associated overhead *four* times -- once at the sender, twice at the basestation, and once at the receiver. This also involves copying data at the basestation to move the packet from the incoming TCP connection to the outgoing one.

- **The Fast-Retransmit Approach [CI94]:** This approach addresses the issue of TCP performance when communication resumes after a handoff. Unmodified TCP at the sender assumes the delay caused by a handoff process to be due to congestion (since TCP assumes that all delays are caused by congestion) and when a timeout occurs, reduces its window size and retransmits these packets. Often, handoffs complete relatively quickly, and long waits are required by the mobile before timeouts occur at the sender and packets start getting retransmitted. This behavior occurs due to coarse timeout granularities (on the order of 500 ms) in most TCP implementations. The fast retransmit approach alleviates this problem by having the mobile host send a certain threshold number of duplicate acknowledgments to the sender, a step that causes TCP at the sender to immediately reduce its window size and retransmit packets starting from the first missing one (for which the duplicate acknowledgment was sent). The main drawback of this approach is that it deals only with handoffs and not with the error characteristics of the wireless link. If used to deal with errors too, this would increase the amount of traffic and congestion in the fixed network, since *every* lost packet would have to be retransmitted from the source.

- **Link-level Retransmissions [PAL+95]:** In this approach, the wireless link implements its own retransmission protocol at the data-link level. In addition, TCP implements its own end-to-end retransmission protocol. Studies have shown that independent retransmission protocols such as these lead to degraded performance, especially as error rates become significant [DCY93]. A tight coupling of transport- and link-level retransmission timeouts and policies is necessary for good performance. In particular, information needs to be passed down to the data link layer about timeout values and policies reasonable for co-existence with the higher transport layer policy. Otherwise, retransmissions will be performed on packets even if the higher level protocols is unreliable.

3 Our Protocol Model

Since TCP is a ubiquitous protocol and most current network applications use it, it is desirable to achieve our goal of implementing TCP over our network without changing existing TCP implementations. We do not modify network-layer software anywhere in the fixed network except at the basestation, since that is the only node in the fixed network over which we can expect to have full control. At the basestation, we modify the router code to cache data meant for the mobile host, and introduce a few

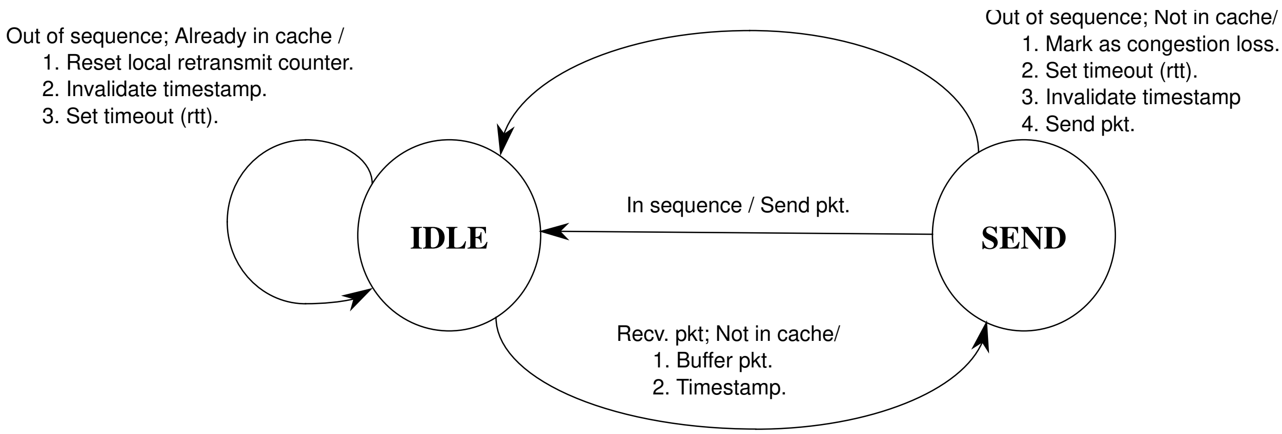


FIGURE 1. State Machine for `snoop_output()`

simple policies dealing with acknowledgments and retransmissions. In some sense, we maintain the advantage of the split connection of shielding the sender from the vagaries of the wireless link and the mobility of the receiver without sacrificing end-to-end semantics, modifying host TCP code, or relinking existing applications. We describe our protocol in the context of a single TCP connection consisting of a fixed host (FH) connected to a mobile host (MH) through a basestation (BS). The extension to multiple senders and receivers is not difficult.

The basestation routing code is modified by adding a layer, the *snoop* layer, which monitors every packet that passes through the connection in either direction. No transport layer code runs at the basestation. The snoop layer maintains a cache of TCP packets sent from the FH that haven't yet been acknowledged by the MH. This is easy to do since TCP has a cumulative acknowledgment policy for received packets. When a new packet arrives from the FH, the snoop layer adds it to its cache and passes the packet on to the routing code which performs the normal routing functions. The snoop layer also keeps track of all the acknowledgments sent from the mobile host. When a packet loss is detected (either by the arrival of a duplicate acknowledgment or by a local timeout), it retransmits the lost packet to the MH. Thus the basestation hides the packet loss from the FH by not propagating duplicate acknowledgments, thereby preventing unnecessary congestion control mechanism invocations.

3.1 Snoop Functions

In this section, we describe the details of the operations performed at the basestation by the snoop layer. We focus on TCP packets from a FH to a MH (it is unclear that a protocol involving changes only at the basestation can improve the performance of communication between a MH and a FH). The snoop layer has two linked state machines, `snoop_output()` and `snoop_input()`. `Snoop_output()` processes and caches packets intended for the MH while `snoop_input()` processes acknowledgments (ACKs) coming from the MH and performs local retransmissions. These local retransmissions are initiated using a timeout mechanism based on a TCP-style round-trip time (RTT) estimate of the last (wireless) link. We limit the maximum number of local retransmissions (the optimal value of this limit is still undetermined).

The state machines for `snoop_output()` and `snoop_input()` are shown in Figures 1 & 2 and are described below.

Snoop_output()

One of three types of packets can arrive from the FH:

1. *A new packet in the normal TCP sequence*. In this case add the packet to the snoop cache and also route it to the MH.
2. *An out-of-sequence packet that has been cached earlier*. In this case this packet has been transmitted either as a result of a sender timeout or as part of the data stream following a TCP fast-retransmission [RFC89]. In general, we allow the local retransmission scheme to deal with the delivery of cached packets. To allow local retransmissions to progress, we reset the count of such retransmissions to 0.
3. *An out-of-sequence packet that has not been cached earlier*. In this case this packet has either been delivered out of order by the network or was lost earlier due to congestion on the wired network. Therefore, this packet is marked as having experienced congestion loss. `Snoop_input()` uses this information to deal with TCP acknowledgments.

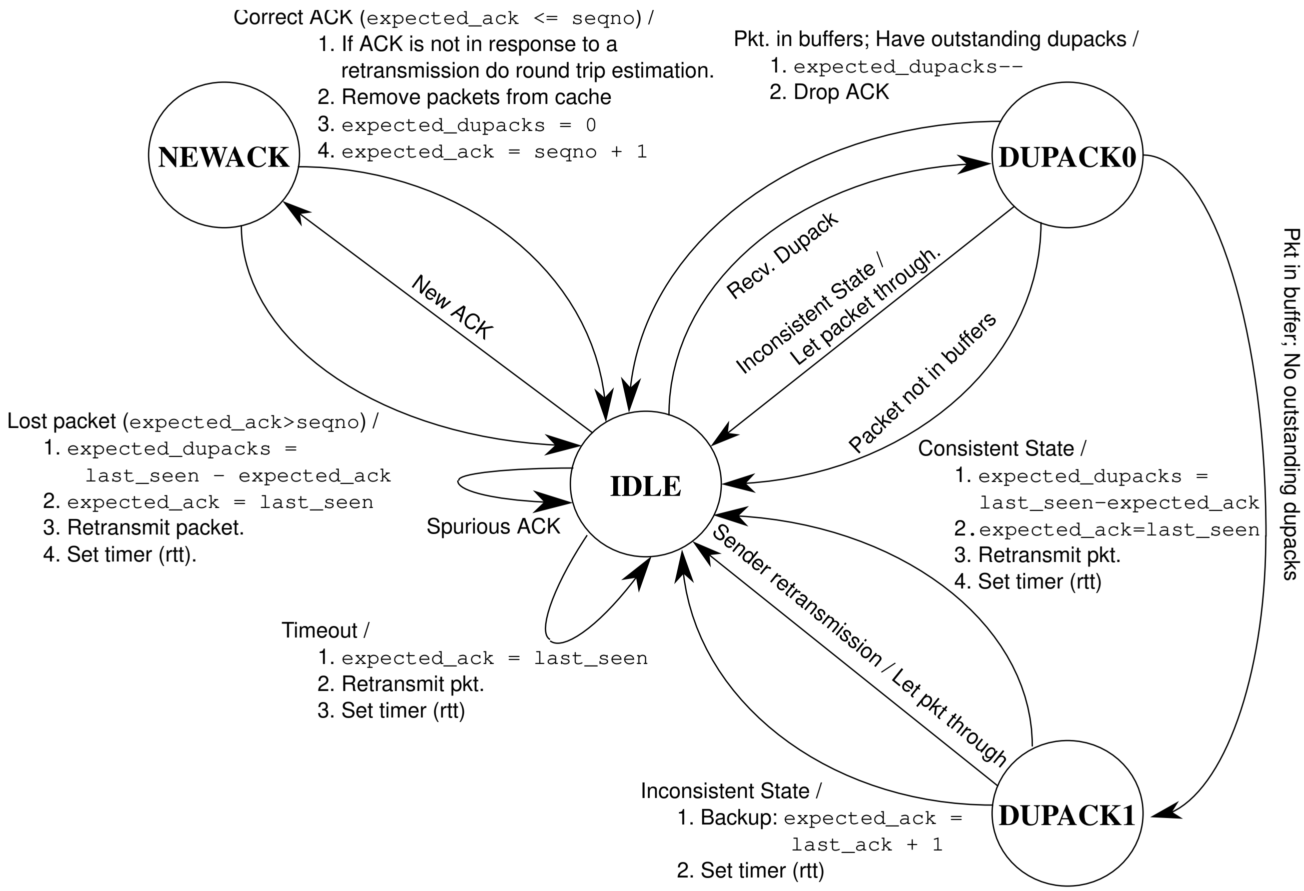


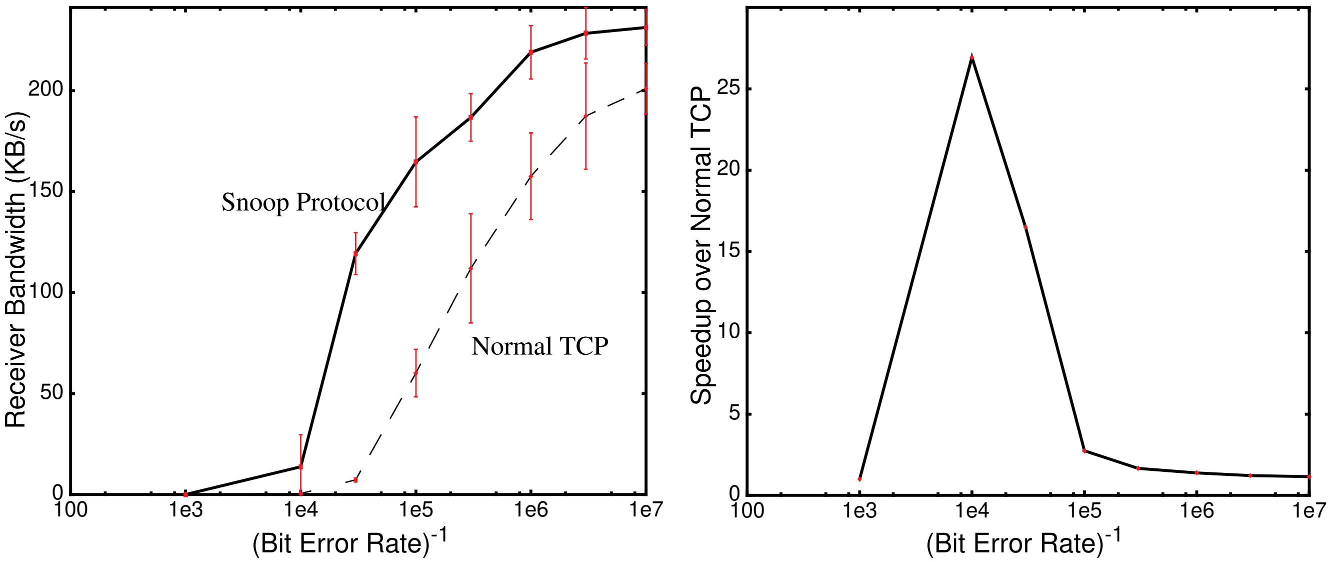
FIGURE 2. State Machine for `snoop_input()`

Snoop_input()

`Snoop_input()` monitors the acknowledgments (ACKs) sent back by the MH. These ACKs fall into one of three categories:

1. A *spurious* ACK, out of the normal non-decreasing order of TCP ACKs. This ACK is discarded since TCP's acknowledgments are cumulative.
2. A *duplicate* ACK (DUPACK), identical to a previously received one. In this case the next packet in sequence from the DUPACK has not been received by the MH. However, some subsequent packets in the sequence have been received. We take one of several actions:
    - If the desired packet is not in the snoop cache or is marked as a congestion loss, we route the DUPACK to the FH. This is because we need the FH to resend the packet after invoking its congestion control mechanisms.
    - Otherwise, if we are not expecting a DUPACK for this packet, we retransmit the packet to the MH. Based on state we maintain (sequence numbers of the last transmitted packet and the expected ACK), we estimate the number of DUPACKs that the lost packet will create, update the state and set the wireless retransmission timer.
    - Finally, if we are expecting a DUPACK for this packet (based on the above estimate), we discard the DUPACK.
3. A *new* ACK. In this case the ACK signifies an increase in the packet sequence received at the MH.
    - Based on the state we maintain (the expected ACK), we determine if a packet has been lost on the wireless link, and if so, retransmit it. We also set the retransmission timer and update the state.
    - Otherwise, no packets have been lost and we use this ACK to update our RTT estimate for the last link and free the buffers in our cache.

In addition, `snoop_input()` must retransmit a packet when a timeout occurs on the wireless link.



(A) (B)

FIGURE 3. Performance of snoop protocol compared to normal TCP - (A) shows end-to-end bandwidth with standard deviations. (B) shows bandwidth speedup of the snoop protocol.

4 Simulation Results

We simulated our protocol by modifying REAL [Kes88], a network simulator for studying the dynamic behavior of flow and congestion control schemes in packet-switched networks. Network sources in REAL can be specified to be a TCP-like source, in which case they simulate TCP Reno (based on the 4.3BSD-Reno implementation). Our modifications involved creating two new node types to simulate a snooping basestation and a mobile host. Additionally, a bit error rate over the last link to the mobile host was introduced to model the wireless link.

We simulated a three-hop network with four hosts; one source (FH), one router, one basestation (BS), and one mobile host (MH). The simulations compared the performance of two connections: one which had an ordinary BS (non-snooping) connected to the MH over the wireless link and one in which that BS was replaced by a snooping BS. The bandwidths of the fixed links were 10 Mbps with a latency of 100 μs, while the peak bandwidth of the wireless link was 2 Mbps with a latency of 200 μs. Each TCP packet was 500 bytes in size, the maximum router buffer size at the nodes was 20 packets, and the scheduling policy at the router and basestation was FCFS.

A three-hop network like the one we chose is sufficient to model congestion since output queues overflow most often on the slowest link which in most wireless networks is the wireless channel. We have not yet experimented comprehensively with multiple TCP connections contending for the same resources in the network although we feel that this configuration will actually improve the relative performance of our protocol over unmodified TCP because of our caching and retransmission mechanism.

Our protocol is designed to handle both bit-errors and mobility of hosts. We focus only on errors here due to space constraints. We ran simulations using Poisson-distributed errors with error probabilities ranging from 10^{-3} to 10^{-7}. The results are shown in Figures 3 (A) and (B). Figure (A) shows the bandwidth achieved at the MH in KB/s as a function of the bit-error rate (with standard deviations), for both our protocol and unmodified TCP. Figure (A) shows that our protocol handles error rates at least 10 times higher than normal TCP without any degradation, for error rates between 10^{-4} and 10^{-7}. Figure (B) shows the bandwidth speedup of our protocol over normal TCP. These figures indicate that TCP does not perform well in the presence multiple packet losses in a single transmission window. On the other hand, our protocol is significantly more robust in dealing with such losses and achieves speedups of up to 25 times. The performance gain is maximum around an error rate of 10^{-4} and tapers off for error rates larger than 10^{-6} when there is a very low probability of multiple errors in any TCP transmission window.

5 Conclusions and Future Work

We have presented a simple protocol to improve the performance of TCP on networks with wireless links and mobile hosts. This protocol works by modifying the network-layer software at the basestation, and involves no other changes elsewhere in the network. The main idea is the caching of packets intended for the mobile host at the basestation and performing local retransmissions. Simulations of this show that this protocol can handle 10 times as high a bit-error rate without any performance degradation as compared to unmodified TCP. Our simulation results are very promising, and we are currently in the process of implementing the snoop protocol on a platform consisting of IBM ThinkPad laptops and Pentium PCs running BSD/OS 2.0 from BSDI, communicating over a 2 Mbit/s DEC WaveLAN.

6 References

[BB94] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. Technical Report DCS-TR-314, Rutgers University, October 1994.

[CI94] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE JSAC Special Issue on Mobile Computing Networks*, 1994. To appear.

[CS94] D.E. Comer and D.L. Stevens. *Internetworking with TCP/IP, Volume II*. Prentice Hall, Englewood Cliffs, N.J, 1994.

[DCY93] A. DeSimone, M.C. Chuah, and O.C. Yue. Throughput Performance of Transport-Layer Protocols over Wireless LANs. In *Proc. Globecom '93*, December 1993.

[Jac88] V. Jacobson. Congestion avoidance and control. In *SIGCOMM 88*, August 1988.

[Kes88] S. Keshav. REAL: A Network Simulator. Technical Report 88/472, Computer Science Division, Univ. of California at Berkeley, 1988.

[KMS+93] K. Keeton, B.A. Mah, S. Seshan, R.H. Katz, and D. Ferrari. Providing Connection-Oriented Service to Mobile Hosts. In *Proc. 1993 USENIX Symp. on Mobile and Location-Independent Computing*, August 1993.

[KP87] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. In *SIGCOMM 87*, August 1987.

[PAL+95] S. Paul, E. Ayanoglu, T.F. LaPorta, K.H. Chen, K.K. Sabnani, and R.D. Gitlin. An Asymmetric Link-Layer Protocol for Digital Cellular Communications. In *Proc. InfoComm '95*, 1995.

[RFC89] Requirements for internet hosts – communication layers. Internet Engineering Task Force, R. Braden (ed.), October 1989. Request for Comments: 1122.

[SLBR94] S. Seshan, M.T. Le, Burghardt, and J. Rabaey. Software Architecture of the InfoPad System. In *Mobidata Workshop*, November 1994.

[Yav94] R. Yavatkar and N. Bhagwat. Improving End-to-End Performance of TCP over Mobile Internetworks. In *Mobile 94 Workshop on Mobile Computing Systems and Applications*, December 1994.