TCP over Wireless Networks Using Multiple Acknowledgements *

(Preliminary Version)

Saad Biaz Miten Mehta Steve West Nitin H. Vaidya

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112, USA
E-mail: {saadb,mmehta,swest,vaidya}@cs.tamu.edu
Web: http://www.cs.tamu.edu/faculty/vaidya/mobile.html

Technical Report 97-001

January 30, 1997

Abstract

TCP protocol has been designed and tuned to perform well on wired network where the packet loss is due mainly to congestion. On heterogeneous networks where some links can be wireless this assumption is no more valid. In this paper, we propose a protocol which makes the TCP protocol layer aware of high bit error rates on wireless links while it tries to minimize the traffic load on the wired network. This is done by avoiding unnecessary retransmissions from TCP. The idea is to acknowledge *partially* a packet which reaches a base station, if it experiences difficulty (errors or congestion) on the wireless link. The base station is responsible for retransmissions on the wireless link, while it delays timeout at the sender by sending a *partial acknowledgement*.

*Research reported is supported in part by Texas Advanced Technology Program grant 009741-052-C and National Science Foundation grant CDA-9529442.

1 Introduction

This report describes briefly a protocol for end-to-end reliable communication on a channel consisting of several links, the last link being error-prone. We consider case (a) shown in Figure 1 where the path consists of one error-immune link and one error-prone link. Our protocol can be extended to other cases, for instance, case (b) shown in Figure 1, where the path consists of multiple error-prone and error-immune links. In the rest of this report, we assume that all

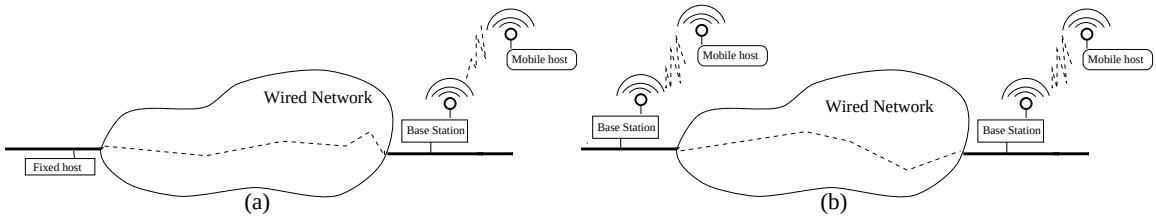

Figure 1: Case (a) : Fixed to mobile host. Cases (b) : mobile to mobile

wireline paths are error-immune, and wireless paths are *error-prone*. We may use the term *wired (wireless) link* interchangeably with *error-immune (error-prone) link*.

The proposed protocol is *end-to-end*. The term *end-to-end* implies that the sender is responsible for ensuring that the receiver receives transmitted data – if a sender believes that receiver has received the data, then that belief must be correct.

Rest of this report is organized as follows. Section 2 provides motivation for the proposed scheme, and provides brief discussion of previous related work. Section 3 discusses the distinction between different types of acknowledgements used in our protocol. The proposed protocol is outlined in Section 3. The implementation is described in Section 4. We describe briefly the testbed in Section 5. Section 6 is dedicated to the experiments planned. Several performance optimization to this protocol are possible, as described in Section 7.

2 Motivation and Related work

TCP is a protocol which has been designed, improved and tuned to work efficiently on wired network where the packet loss is very small ($\ll 1\%$) [7]. Whenever a packet is lost, it is reasonably assumed that congestion occurred on the connection path. Hence, TCP triggers slow-start and congestion avoidance algorithms when a timeout occurs for a packet. These algorithms proved their efficiency on a wired network. But, on a wireless link, the bit error rate is higher and the assumption that packet loss is mainly due to congestion is no more valid. The slow-start and congestion avoidance algorithms are triggered even when packets are lost due to corruption only on the wireless link. In this case, slow-start and congestion avoidance algorithms lower the connection throughput. So, there is a need to adapt TCP to heterogeneous networks which mix wired and wireless links. The proposed approach is based on previous work in [1, 3, 4, 5, 6], and combines ideas proposed in these papers to obtain a

unified scheme. The reader is assumed to be familiar with [1, 3, 4, 5, 6]. The similarities and differences between proposed approach and the previous work are as follows:

- Balakrishnan et al. [1] evaluate several TCP-based protocols. Our approach uses the *snoop protocol* scheme in that some intermediate hosts on the path (e.g., *base station*) may buffer packets being sent to the receiver. The difference, however, is that in our protocol, the node that buffers the packet sends a *partial acknowledgement* Ack_p to the receiver. This cumulative acknowledgement will only prevent the sender from triggering the slow start and congestion avoidance mechanisms. In presence of bad transmissions conditions, *snoop* cannot prevent this. End-to-end semantics are still retained because the sender will consider that a packet has been received only if it receives a *complete acknowledgement* Ack_c from the receiver.

  Similar to the *explicit loss notification* (ELN) [1], our protocol also (implicitly) sends loss notification (using the partial acknowledgement Ack_p). The difference is that the loss notification (actually an ack) is not sent by the receiver, but by an intermediate node (in practice, a base station). In [1], the author recognizes the difficulty to distinguish between congestion and error losses. Our protocol uses a realistic way to achieve the ELN with a very good accuracy.

  Similar to the link-layer retransmission scheme in [1] our scheme also uses retransmissions, however, these retransmissions are unlikely to conflict with retransmissions from the sender. It is expected that our protocol will behave like snoop protocol for low bit error rates. If the channel experiences transient or permanent bad conditions, our protocol is expected to perform better than snoop. *Snoop* has a limited time to act to avoid sender retransmissions. With the partial acknowledgement, we prevent the sender from timing out. But we have to modify (slightly) TCP on the sender while snoop does not require any modification. However, our protocol will behave like snoop for a sender whose TCP is not modified. This sender will ignore the options field which we use to implement partial acknowledgements.

- Badrinath et al. [5] : The I-TCP protocol [5] does not provide end-to-end semantics, while the proposed approach does. Proposed scheme is similar to I-TCP in that different links are treated somewhat differently, as elaborated later. I-TCP has the advantage to tune (choose the packet size) every connection, (wired connection or wireless connection), to the errors characteristics. In our scheme, this may be done while respecting the TCP end-to-end semantics.

  Yavatkar et al. [6] proposes a similar scheme based on splitting the connection.

- Bakshi et al. [3, 4] present a protocol that sends, to the sender, an *explicit bad state notification* (EBSN) when a wireless link experiences errors. Our scheme also (implicitly) provides such a notification, using acknowledgement messages. The difference is that our scheme informs the sender of *which packets* are indeed experiencing bad conditions channel. The problem with EBSN is that, with a bad network latency, presence or absence of EBSN at the sender will not reflect the actual channel state. The channel bad

state can also be of a very short duration such that whenever EBSN reaches the sender, the channel is no more in bad state.

While proposed scheme, when applied to case (a) is similar to [3], proposed scheme also applies to (b) and other scenarios. The proposal here can be considered to be an improvement over [3] which does not use the *snoop* protocol.

The motivation behind the proposed approach is to combine the explicit notification schemes (ELN or EBSN), end-to-end semantics, and link-level retransmission (snoop) into a TCP-like protocol that handles a larger spectrum of bit error rates.

To obtain performance improvements using the proposed scheme, the sender TCP code must be modified. However, a constraint imposed on the scheme is that it should be able to work correctly even if the sender code is not modified.

3 The protocol

The idea is to distinguish the losses on the wired portion from the losses on the wireless link. Instead of splitting the connection in two connections, one for the wired link and the other for the wireless link, we use two types of acknowledgments :

Ack_p : This *partial* acknowledgment with sequence number N_a informs the sender S that the packet(s) with sequence numbers up to $N_a - 1$ had been received by the base station.

Ack_c : This *complete* acknowledgment has the same semantics as the normal TCP acknowledgment, i.e, the receiver R received the packet.

Let us describe the data transfer from a fixed host S. We will consider the different receive and send events on the sender S (fixed host), the base station B and the receiver R (mobile host). There are potentially many variations possible on the protocol summarized below. Such variations will be investigated in our future work.

3.1 The sender

3.1.1 Data Sent

The sender will strictly follow the regular TCP when sending any packet as defined by RFC 793 [8]with the addition of algorithms introduced by [7] : *slow start, congestion avoidance*, *fast retransmit* and *fast recovery* [9].

3.1.2 Acknowledgement Ack_p received

Before describing the actions taken at the reception of Ack_P, we define some variables. We must distinguish the round trip time RTT between the sender and the receiver (end-to-end) and the round trip time RTT_B between the base station and the mobile host. These values will have respectively a direct impact on the time out values RTO and RTO_B. RTO is the maximum time the sender waits for an *end-to-end* acknowledgement. If RTO expires, the sender must retransmit the packet. RTO_B is the maximum time which can elapse between the reception of a packet at the base station and its acknowledgement by the mobile host. This acknowledgement Ack_p with sequence number N_a informs the sender that the base station received all the packets with a sequence number up to $N_a - 1$ and that it is experiencing problems to forward the packets through the wireless link. The sender can identify these packets. These packets have sequence numbers between the sequence number of the last Ack_c and $N_a - 1$. More precisely, these packets spent more than $RTOB$ on the base station without being successfully transmitted to the mobile host. Let RTT_B be the round trip time between the base station and the mobile host. The timeout value RTO_B will depend on RTT_B. Hence, the sender must update :

- the time out RTO.

  RTO must be updated to give more time to the base station to accomplish the retransmissions. This will avoid end-to-end retransmissions, thus avoiding to trigger the slow start and congestion avoidance mechanisms. There are different ways to update RTO.

  1: Reset

  We just set the timer at its initial value when the packet was sent

  2: Increase

  Increase the current timer value by some amount (e.g. a constant increment or a multiplicative factor)

  We will evaluate the different possibilities

- the Round Trip Time RTT

  RTT will be updated when either Ack_p or Ack_c is received for a packet. If we update RTT when Ack_p is received, RTT will not reflect the actual round trip time potentially leading to a premature time out. The advantage is that a transient bad state of the channel will not affect RTT. If RTT is updated with Ack_c then it will reflect the actual round trip time. We will investigate both possibilities.

  Another possibility is to ignore the round-trip time sample corresponding to any packet for which an Ack_p is received.[1] This is analogous to Karn's algorithm which ignores RTT samples corresponding to retransmitted packets.

[1] Thanks to P. Krishna for suggesting inclusion of this alternative.

The reception of Ack_p will not affect any other parameter of the connection.

There is another possible approach for handling Ack_p. When Ack_p is received, corresponding packets are just *marked*. Theses marks will be used when there is time out as follows : the sender considers packets for which it timed out. If the packet is not marked, the sender reacts exactly as in regular TCP. Otherwise, it will remove the mark and react as in regular TCP except that it will not retransmit, will not trigger slow start and congestion avoidance. In fact, the sender will only backoff the timer.

3.1.3 Acknowledgement Ack_c received

Receiving Ack_c means that the receiver got the packet. Hence, the sender takes the same steps as in normal TCP except for RTT estimate. If RTT has been updated using Ack_p then no update with Ack_c. Otherwise, RTT update must be done.

3.2 The base station

3.2.1 Data received

The base stations follow the *Snoop* protocol [1] with some modifications. When data is received from the sender, the base station will take different steps depending on whether the packet is out-of-sequence or not. If the base station receives :

1: A new packet in the normal TCP sequence

We buffer the packet, put a time stamp on it, set the timer RTO_B and try to forward it to the mobile host. We must observe that RTO_B is set when the packet is cached and not when the base station forwards it. RTO_B is set such that the packet will not spend at the base station more than RTO_B time without being acknowledged. If RTO_B expires then the base station sends back to the sender a cumulative acknowledgement Ack_p with the sequence number of this packet plus 1.

2: An out of sequence packet that has been buffered earlier

This can happen if, for example, the acknowledgements Ack_p or Ack_c have been lost. The actions taken will depend on the sequence number S of the packet. Let S_a be the sequence number for the last acknowledged packet. If $S > S_a$ and the packet is still in the cache then we send back to the sender a partial Ack_p (the packet will continue to be handled by the base station). Otherwise, i.e $S > S_a$ and the packet is no more in the cache then we process the packet as if it is a new packet. If $S \leq S_a$ then we generate a fake Ack_c holding the identity of the mobile host and send it back to the sender.

3: An out of sequence packet that has not been buffered earlier

This means that packets have been lost on the wired network or that there is an out-of-order delivery. The problem here is to distinguish between out of order delivery and

retransmitted packets. In [1], Balakrishnan identifies an out-of-order delivered packet as a packet whose sequence number is not more than one or two packets away from the last packet seen so far. The retransmitted packet must be distinguished for the processing of duplicate acknowledgements received from the mobile host.

An Ack_p will be generated for this packet if it spends more RTO_B on the base station, provided that meanwhile all packets before it have been received. Anyway, this packet will be forwarded to the mobile host.

3.2.2 Retransmissions

When RTO_B expires, this means that either the packet has not been sent yet or that it has been lost. The packet must then be retransmitted and an Ack_p is sent back to the sender if all packets before it have been received by the base station. We have the choice to send Ack_p only once when first time RTO_B expires, or to send it at each RTO_B expiration (or something in between). We will investigate each approach.

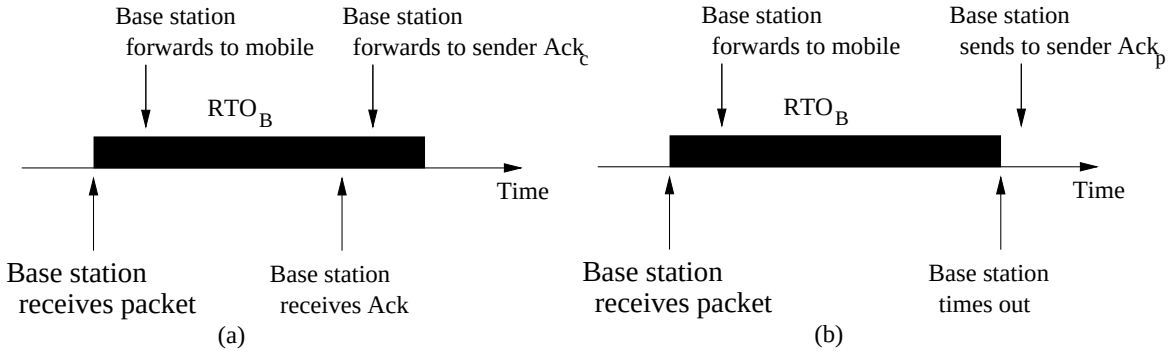

Figure 2: (a): No time out on the base station. (b) : Time out.

3.2.3 New acknowledgement received

Since the protocol must hold even if the TCP-sender does not recognize Ack_p, acknowledgements from the receiver must be processed (filtered) at the base station. These acknowledgements will be processed exactly as in Snoop[1]. We recall here how Snoop processes acknowledgements. The idea is to forward what is needed by the TCP protocol and to discard what can trigger unnecessary retransmissions from the sender. Snoop will act differently depending on three types of acknowledgements it will receive :

1: A new ack

Snoop will purge its buffer, forward the acknowledgement Ack_c to the sender and update RTT_B. RTT_B is the time which elapses from the time the base station receives a packet to the reception of its acknowledgement from the receiver. In fact, [1] updates RTT_B

only once per window and respects Karp's condition which states that RTT_B must not be updated with any acknowledgement for a retransmitted packet. RTT_B must reflect an estimate of the time spent by the packet on the base station plus the round trip delay to the mobile host.

2: A spurious Ack

Let S_n the last sequence number acknowledged by the receiver. A spurious ack is an acknowledgement for the sequence number A with $A < S_n$. It is discarded.

3: A duplicate acknowledgement

The idea here is to forward the necessary acknowledgements to the TCP protocol and to discard those who trigger unnecessary retransmissions from the sender. We exactly follow the Snoop protocol for the duplicate acknowledgements

3.3 The receiver (mobile host)

The receiver is not modified : it follows TCP-Reno protocol.

4 Implementation

To implement Ack_p, we can use the options field in the header of TCP packet. An option is implemented with the following syntax : $b_k b_l option$. b_k is a byte giving the *type* of option. We must make sure that we choose b_k such that it does not interfere with other options like SACKs. b_l is a byte giving the length of the option of kind b_k including the kind byte and the length byte. If we do not send a partial acknowledgement Ack_p, we will just have a TCP header without any option field. If there is an Ack_p, we will have an option field of length 6 bytes : one byte for b_k, one byte for b_l and four bytes for the sequence number acknowledged.

Another possibility is to send Ack_p as an ICMP message.

5 Testbed

We will use Pentium based workstations running Free BSD 2.1.5 Release 0. These workstations are interconnected using a 10 Mbps Ethernet and 915 Mhz AT&T Wavelans, a shared wireless LAN with a raw bandwidth of 2Mbps.

6 Experiments

We will implement snoop and measure the throughput of the system when transmitting a large file under different conditions of bit error rates (bursty and not bursty). We will implement

our scheme and perform the same measurements to compare with *Snoop*. This report will be updated when the experimental results become available.

7 Performance expected

We believe that below some threshold of bit rate error BER, Snoop and our protocol will behave the same and will have the same performance. When the bit error is very low on the wireless link, Snoop will be able to recover without any time-out on the sender in most cases. But, if the bit error rate on the wireless link is high or if the channel is overloaded, *Snoop* would not be able to prevent time out and slow start on the sender will be triggered. With our scheme, the sender will time out only if the base station abandons retransmissions or if the Ack_p is lost. In presence of congestion on the wired network, the following situation will happen : packets will be dropped, so packets will reach the base station out of order and will not be acknowledged by Ack_p. Hence, packets already received by the base station may be retransmitted by the sender.

Acknowledgements

Thanks are due to P. Krishna for his comments on a version of this report.

References

[1] Balakrishnan,H., V. N. Padmanabhan, Seshan, S., Katz, R., *A Comparison of Mechanisms for Improving TCP Performance over Wireless Links*, ACM SIGCOMM'96, CA, August 1996.

[2] Balakrishnan,H., Seshan, S., Amir, E., Katz, R., *Improving TCP/IP Performance over Wireless Networks*, 1st ACM Int'l Conf. on Mobile Computing (Mobicom), November 1995.

[3] Bikram S.Bakshi, P. Krishna, N.H. Vaidya, D.K. Pradhan, *Improving Performance of TCP over Wireless Networks*, Texas A&M University, Technical Report TR-96-014, May 1996.

[4] Bikram S.Bakshi, P. Krishna, N.H. Vaidya, D.K. Pradhan, *Performance of TCP over Wireless Networks*, to appear in the 17^{th} International Conference on Distributed Computing Systems (ICDCS), Baltimore, May 1997.

[5] A. Bakre, B.R. Badrinath, *I-TCP : Indirect TCP for mobile hosts*, Technical Report DCS-TR-314, Rutgers University, October 1994.

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

[7] V. Jacobson, *Congestion Avoidance and Control*, ACM SIGCOMM'88, pp 314-329, Aug. 1988. An update version is available via ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[8] J. Postel, *Transmission Control Protocol*, RFC 793, 85 pages, Sept 1981.

[9] W.R. Stevens, *TCP/IP Illustrated, Volume 1, The Protocols*, Addison Wesley, 1994.