Improving Performance of TCP over Wireless Networks [*]

Bikram S. Bakshi P. Krishna N. H. Vaidya D. K. Pradhan

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
E-mail: {bbakshi,pkrishna,vaidya,pradhan}@cs.tamu.edu
Phone: (409) 862-3411

Technical Report TR-96-014[†]

May 1, 1996

Abstract

Transmission Control Protocol (TCP) assumes a relatively reliable underlying network where most packet losses are due to congestion. In a wireless network, however, packet losses will occur more often due to unreliable wireless links than due to congestion. When using TCP over wireless links, each packet loss on the wireless link results in congestion control measures being invoked at the source. This causes severe performance degradation. In this paper, we study the effect of (a) burst errors on wireless links, (b) packet size variation on the wired network, (c) local error recovery by the base station, and (d) explicit feedback by the base station, on the performance of TCP over wireless networks.

It is shown that the performance of TCP is sensitive to the packet size, and that significant performance improvements are obtained if a 'good' packet size is used. While local recovery by the base station using link-level retransmissions is found to improve performance, timeouts can still occur at the source, causing redundant packet retransmissions from the source. We propose an "explicit feedback" mechanism to prevent these timeouts during local recovery. Results indicate significant performance improvements when explicit feedback from the base station is used. A major advantage of our approaches over existing proposals is that, no state maintenance is required at any intermediate host.

Experiments are performed using the Network Simulator (NS) from Lawrence Berkeley Labs. The simulator has been extended to incorporate wireless link characteristics.

[*]This research is supported, in part, by the Texas Advanced Technology Program under grants C-009741-052 and C-999903-029.

[†]This report supersedes TR-95-049

1 Introduction

A typical wireless network with mobile users is implemented using a wired network of fixed hosts, some of which are augmented with wireless interfaces. Such hosts are called *base stations.* The base stations provide a gateway for communication between the wireless and wired network.

Users of portable computers would like to execute popular applications like *ftp, telnet, www-access,* etc., over the wireless link, when they are mobile. Most of these popular applications employ TCP as their transport-layer protocol. One of the primary reasons for the widespread use of TCP on the internet is its inbuilt algorithms for congestion control and avoidance. Over the years, the internet community has incorporated new schemes into the TCP suite to make these protocols more robust to congestion. Details of schemes for congestion control and avoidance in TCP can be found in [2]. Here, we will give a brief overview of the general ideas behind these schemes, as this explanation proves useful in understanding the problems typical to a wireless network.

Two parameters of interest in this discussion are *congestion window* (*cwnd*), and *slow-start-threshold* (*ssthresh*) maintained by each TCP connection for use in flow-control. The value of *cwnd* fluctuates as new acknowledgments of previously sent data packets stream in. The maximum amount of unacknowledged data that TCP can have on the internet at any time, is the minimum of the receiver's advertised window and *cwnd.* The parameter *ssthresh* is used to control the rate of growth of *cwnd.*

Packets on the internet may get lost either due to congestion, or due to corruption by the underlying physical medium. Given the low bit error rates of wired links, almost all losses are related to congestion. TCP's reaction to losses is based on this very observation. Losses are detected either by timeouts at the source or by multiple duplicate acknowledgements (*dupacks*) from the receiver (referred to as the *fast-retransmit* policy [5]). Upon loss of a packet, TCP reacts by setting *ssthresh* to half the value of *cwnd*, subsequently decreasing *cwnd* to one, and entering the *slow-start* phase. This measure would appear severe, but works well, because cutting the window size and thus limiting the amount of unacknowledged data on the network, is the most effective way of dealing with congestion. In addition to the above measures, the timeout value is doubled upon each consecutive packet loss. Only upon receipt of an acknowledgement for a "non-retransmitted packet" is the timeout value recomputed [1].

While wired links offer a virtually error free transmission medium, errors on wireless links tend to be frequent and bursty, and are highly sensitive to direction of propagation, multipath fading, and general interference. As stated earlier, TCP assumes that each packet loss is **solely** due to congestion. However, in a wireless network, TCP will encounter packet losses that may be unrelated to congestion. Nonetheless, these losses trigger congestion control measures at the source and severely degrade performance. In addition, for wide-area wireless networks, the packet size over wireless links is typically much smaller than the packet size over the wired network. For example, the packet size over wireless links for CDPD Networks [12] is only 128 bytes. As a result, each packet on the wired network gets fragmented when transmitted over the wireless link. Loss of a fragment over the wireless link will initiate error recovery and congestion control mechanisms at the source, causing noticeable performance degradation.

In this study, we **do not** consider handoffs. In a separate study [17], we have proposed schemes to improve the performance of TCP in the presence of handoffs. In this study, we are only interested in the performance of TCP (for bulk data transfer) in the presence of losses in wireless networks. The performance metrics of interest in this study are:

• *Goodput*: This is the measure of how efficiently a connection utilizes the network. It is determined as the ratio of useful data received at the destination and the total amount of data transmitted by the source. If a connection requires a lot of extra packets to traverse the network due to retransmissions, its goodput is low. It is desirable that each connection have as high a goodput as possible. Clearly, this metric is of great significance for efficient operation of a network.

• *Throughput*: This is the measure of how soon an end user is able to receive data. It is determined as the ratio of the total data received by the end user and the connection time. A higher throughput will directly impact the user's perception of the quality of service.

In this paper we propose two approaches to improve the performance of TCP. They are:

- **Packet size variation:** As stated earlier, the packet size on wide-area wireless networks is typically much smaller than the packet size on the wired network. In this approach, we improve the performance of TCP by choosing an 'optimal' packet size on the wired network. It is observed that the optimal packet size depends on the error conditions on the wireless link. We show that choosing an optimal packet size over a non-optimal packet size can improve performance by upto 30% over basic TCP. It should be noted that this approach does not require any change in the transport or the link layer protocols at any host in the network.

- **Explicit feedback:** Local recovery from the base station is found to improve performance of TCP. However, while the base station is performing local recovery, timeouts can still occur at the source. We propose an explicit feedback mechanism that eliminates timeouts at the source during local recovery. We performed experiments on wide-area wireless networks as well as local-area wireless networks using explicit feedback from the base station to the TCP source. It is observed that using explicit feedback improves performance of TCP by upto 100% over basic TCP in wide-area wireless networks, and upto 50% in local-area wireless networks. Our choice of error characteristics over the wireless link is conservative. We expect our schemes to yield even better performance if wireless links are more lossy.

The remainder of this paper is organized as follows. Section 2 presents a summary of the existing proposals for improving TCP performance over wireless networks. We present our simulation environment in Section 3. Section 4 presents the discussion of the proposed approaches, namely 'packet size variation', and 'explicit feedback'. Results and conclusions follow in Section 5 and Section 6 respectively.

2 Summary of Previous Approaches

Caceres and Iftode were among the first to investigate the impact of mobility on the performance of TCP [4]. The authors employ the fast retransmit procedure to recover quickly from packet losses during handoffs. This work, however, does not address the issue of packet losses due to lossy wireless links.

The *split-connection* approach [6, 7] suggests that a TCP connection between a mobile host and a fixed host should be split into two separate connections – one between the mobile host and the base station over the wireless medium, and another between the base station and the fixed host over the wired medium. Separation of flow control and congestion control of the wireless link from that of the fixed network, helps in improving TCP performance. However, the split-connection approach violates the semantics of end-to-end reliability. This is because, acknowledgments can arrive at the source even before the packet actually reaches the intended destination. Secondly, this approach requires a lot of state maintenance at the base station.

Balakrishnan et.al. incorporate a transport layer aware agent (snoop agent) at the base station in [11]. The snoop agent caches the TCP packets destined for the mobile host and performs local retransmissions after losses are detected by duplicate acknowledgments (*dupacks*) and timeouts. However, a timeout can occur at the source, and congestion control procedures invoked, while the snoop agent is trying to resend lost packets to the mobile host. Moreover, both snoop and the split-connection approaches do not perform well in the presence of bursty losses on the wireless links.

Several link level Channel State Dependent Packet (CSDP) scheduling policies are proposed in [9]. The performance of multiple TCP connections over a wireless LAN is investigated. It is observed that under FIFO packet scheduling at the base station, the head of line packet, if encountering burst losses, could block the transmission of other packets. In case of multiple TCP connections sharing the wireless link, scheduling protocols such as round-robin provide significant performance improvement over FIFO. The main limitation of this approach is that the performance improvement achievable depends mostly on the accuracy of the channel state predictor. The problem of source timeouts exists in this approach too.

3 Simulation Environment

We use the Network Simulator (NS) [13] from Lawrence Berkeley Labs with extensions incorporated to simulate wireless links, to evaluate the performance of our proposed schemes. NS is an extensible simulation engine built using C++ and Tcl/Tk that can simulate various flavors of TCP available today for wired networks. TCP-Tahoe is used for the purposes of our simulation.

3.1 Wireless Link Parameters

• **Error Model :** We consider a burst error model for errors on the wireless link. This error model is characterized by a 2 state markov model (as shown in Figure 1); the 2 states representing a good and a bad state. In each state, bit errors are poisson-distributed with a mean *Bit Error Rate* (BER) of λ_g for the good state and λ_b for the bad state. The transition from good-to-bad state, and from bad-to-good state are also poisson-distributed with a mean transition rate of λ_{gb}/sec and λ_{bg}/sec respectively. We fix the mean BER in the good state, $\lambda_g = 10^{-6}$, and the mean BER in the bad state, $\lambda_b = 10^{-2}$ (e.g. deep fades). The mean value of good period $\frac{1}{\lambda_{gb}} = 10sec$, and the mean value of bad period $\frac{1}{\lambda_{bg}}$, is varied from 1 sec to 4 sec.

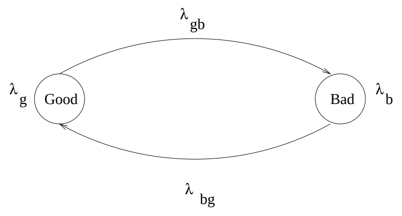


Figure 1: Two State Markov Model for Burst Error Characterization

• **Maximum Transmission Unit (MTU) :** This is the maximum link level packet size admissible on the wireless link. Any network layer data packet larger than the MTU gets fragmented while traversing the wireless link. Typically, the MTU for the wide-area wireless network is small. Unless otherwise mentioned, we use 128 bytes as the MTU for the wireless network.

• **Overhead :** A number of bytes is added to each network layer packet by the lower layers on the protocol stack before transmitting over the wireless link. These overheads are due to

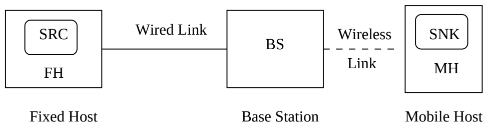


Figure 2: Simulation Setup

framing, error correction, segmentation, and synchronization. We assume that a packet over the wired network of length W bytes becomes 1.5W bytes after addition of these overheads. Since we assume a large overhead due to error correction, the BER during the good period (for burst error model) is kept low. As a result, losses over the wireless link occur primarily during a bad period. If a smaller overhead is chosen, then the BER during the good period should increase.

• **Bandwidth :** Symmetrical, 19.2 Kbps (raw). After overheads due to Forward Error Correction (FEC), etc. have been removed, the effective link bandwidth is equal to 12.8 Kbps.

• **Delay :** Transmission delay and propagation delay are the main delay components. We assume that there is only one connection being served by the base station. Therefore, MAC delay is assumed to be negligible.

3.2 Simulation Model

A simple network topology is chosen to make it easier to understand performance dynamics. As shown in Figure 2, there are three nodes : a fixed host (FH), a base station (BS) and a mobile host (MH). There is a wired link (56 Kbps) between the fixed host and the base station and a wireless link (19.2 Kbps) between the base station and the mobile host. In this paper, we are only concerned with bulk data transfer from a fixed host to a mobile host. Therefore, a TCP source (SRC) is embedded in the fixed host, and a TCP sink (SNK) is embedded in the mobile host.

3.3 TCP Parameters

We use Tahoe TCP which incorporates slow start, congestion avoidance and fast-retransmit algorithms [2, 13]. Unless otherwise specified, the window size is set to 4 Kbytes. We run experiments for different packet sizes ranging from 128 bytes to 1536 bytes. The header size is set to 40 bytes. The granularity of the TCP clock is set to 100 msec, implying that the roundtrip times are measured to the nearest 100 msec.

3.4 Graphical Output

Packet traces for a connection may be obtained as graphical output from the simulator. For each graph (as shown in Figures 3-5), the horizontal-axis shows the time in seconds, while the vertical-axis denotes packet number *mod* 90. Each mark on the graph indicates a packet generated by the TCP source. Packet retransmissions are indicated by multiple adjacent marks having the same vertical-coordinate, but differing in transmission time. For example, in Figure 3, Packet 44 gets retransmitted twice; once at 25.9 sec, and then again at 28.3 sec.

4 Proposed Approaches

In this section, we provide detailed discussions on each of the following approaches:

- Effect of varying packet size on the wired network.
- Effect of local recovery and explicit feedback mechanism.

4.1 Effect of Packet Size Variation

In this section we will discuss the effect of variation of packet size on the wired network on the performance of TCP without local recovery from the base station. Here, our aim is to improve TCP's performance without making changes in the transport layer or the link layer at any host.

Typically, the MTU on a wide-area wireless network is kept small. This reduces the probability of packets getting corrupted during transmission over the wireless medium. For example, the MTU in CDPD networks is 128 bytes [12]. However, packets on the wired network larger than the wireless MTU get fragmented into multiple MTUs before transmission over the wireless link.

Fragmentation of packets on a heterogeneous network appears to have many advantages as pointed out in [15]. The end hosts are relieved from worrying about the size of their data-segments even though the intermediate links may have largely different MTU sizes. If the packets from the source are larger than the MTU of an intermediate link, the routers at the ends of this link are responsible for fragmenting and subsequently reassembling the packet. Clearly, this approach will give better throughput in cases where an intermediate high bandwidth link supports larger MTU sizes (those comparable to the size of the packet from the source).

While fragmentation may appear to be an attractive solution to the wide discrepancy in internetwork MTU sizes, the authors in [15] recommend it be avoided. It is pointed out that dropping or corruption of a single such fragment will result in the whole packet being dropped. The source would then have to retransmit the entire packet causing more fragments to litter the network and compound congestion problems. For these reasons, fragmentation of data packets should be avoided as far as possible.

The above argument has important ramifications for efficient operation of TCP over wide-area wireless networks. While throughput of TCP is sensitive to the error characteristics of the link, we show that the packet size on the wired network also affects results. Note that, if *Path MTU Discovery* (PMTU) [21] is used to decide the size of the data packet to be used for a TCP connection, the packet size chosen will be equal to the smallest MTU among all links along the route for the connection. In this case, it will be the MTU on the wireless link. If neither the MSS option, nor PMTU are used during TCP connection establishment, the source assumes the default IP datagram size of 576 bytes [22] as the packet size.

We performed experiments for different packet sizes under different error conditions over the wireless link. Our results indicate that for most error conditions, the optimal packet size differs from the MTU on the wireless link as well as the default IP datagram size. The numerical results of this study are presented in Section 5. We show that based on wireless link error characteristics, choosing a 'good' packet size will provide significant performance improvements without having to maintain any state information per connection at the the base station. This proposal may simply be implemented by maintaining a fixed table at each base station which maps a particular wireless link error characteristic to the 'good' packet size for that error characteristic.

Even though a judicial choice of packet size gives performance improvements, there exists a substantial difference between the performance obtained, and the theoretical maximum achievable. Using local recovery and explicit feedback mechanisms presented in the next section, we can obtain goodput and throughput values that nearly equal the theoretical maximum.

4.2 Explicit Feedback

We will first illustrate the effects of losses, local recovery and explicit feedback on TCP with the help of an example.

4.2.1 An Example

A simple experiment is run on our simulator for a network configuration shown in Figure 2, where bulk data is transmitted from a fixed host to a mobile host. The packets are subjected to burst losses on the wireless link. The losses are determined using the two-state markov model explained earlier. For this example we use a simpler model, where the bit-errors and state-transition values are assumed to be constant and do not follow a random distribution. This is done so that we can exactly duplicate the errors and state transitions for each of the three experiments; basic TCP, local recovery, and explicit feedback. Following are the parameters used in these experiments: mean BER in good state, $\lambda_g = 10^{-6}$, mean BER in bad state, $\lambda_b = 10^{-2}$, $\frac{1}{\lambda_{gb}} = 10sec$, and $\frac{1}{\lambda_{bg}} = 4sec$. The packet size on the wired network is equal to 576 bytes. The window size is equal to 4 Kbytes. As stated earlier, the MTU on the wireless link is equal to 128 bytes.

The simulation starts with the wireless link in a good state. It remains in the good state for 10 sec and then enters the bad state. It remains in the bad state for 4 sec and then reenters good state. This cycle continues for the length of the connection.

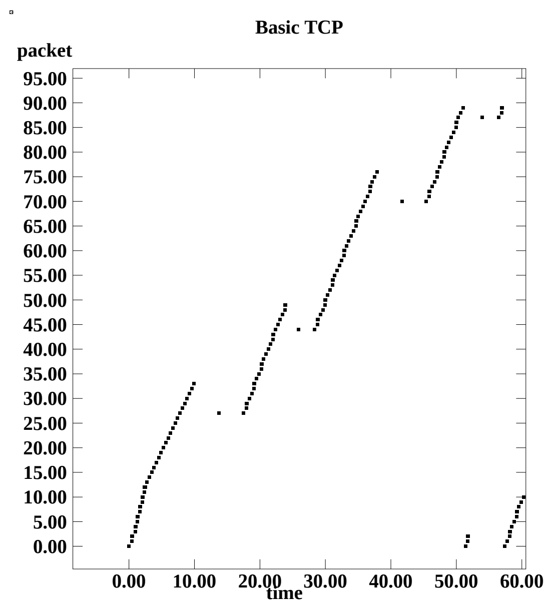


Figure 3: Basic TCP

Figure 3 shows the trace of packets for a connection using basic TCP. During the bad periods, all packets transmitted over the wireless link are lost. For example, consider the bad period between 24-28 sec, where packets 44-50 are lost on the wireless link. The source detects the loss of packet 44 only after a timeout at 25.9 sec as all acknowledgements from the mobile host also get lost during this bad period. The source now initiates congestion control measures and retransmits the lost packets, causing degradation of goodput as well as throughput.

Figure 4 shows the trace of packets for the same connection but using local recovery at the base station. (The basic setup of Figure 2 is modified to employ local link-level retransmissions from the base station.) The link-level protocol used is similar to to the protocol in [9]. This involves aggressive retransmission with packet discards. If the base station does not receive an acknowledgement following a packet transmission, it retransmits the lost packet after a random retransmission backoff. A maximum of RT_{max} successive retransmissions are allowed before a packet is discarded. We set RT_{max} to 13 [12]. It can be noticed that in most bad periods, local recovery at the base station prevents packet losses on the wireless link. For example, between 24-28 sec, no packets need to be retransmitted from the source. However, in some bad periods, the source may time out waiting for acknowledgments of packets that have already been sent. This is evident in the bad period of 10-14 sec, where a timeout for packet 27 occurs at the source at 13.75 sec.

The problem of redundant retransmissions (from the TCP source as well as the base station)

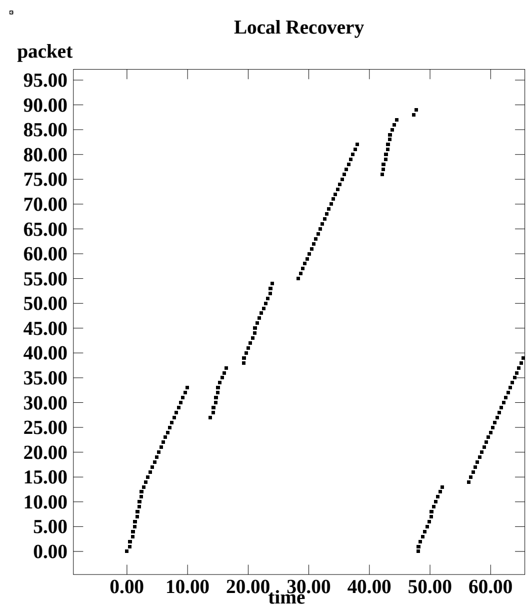


Figure 4: Local Recovery

was also pointed out in [3]. The authors in [3] suggested that error recovery employed at the link layer could potentially interfere with TCP's timeout mechanism. This leads to competing or redundant retransmissions. In our example, redundant retransmissions occur for packet 27. While the base station is trying to transmit the packet to the mobile host, the source times out and retransmits the same packet. This problem will not arise if TCP implementations use a very coarse timer. Current TCP implementations have a coarse timer granularity (of the order of 300-500 millisecond). Other approaches that employ local recovery [9, 11] assume a coarse timer, which is why they do not notice this problem of redundant retransmissions during local recovery. Recent proposals advocate the use of finer granularity timers, as this increases the sensitivity of the source TCP to congestion on the network [23]. In line with this trend, we use a timer granularity of 100 milliseconds for our experiments.

Explicit feedback from the base station can completely eliminate the possibility of timeouts occurring at the source, while the wireless link is in a bad state. The results of using explicit feedback are shown in Figure 5. As can be seen, there are no timeouts at the source, and therefore, the source does not invoke congestion control measures during any bad period. In the next section, we explore the possibility of using existing feedback mechanisms, like *Explicit Congestion Notification ECN* [23] for improving TCP performance over wireless links.

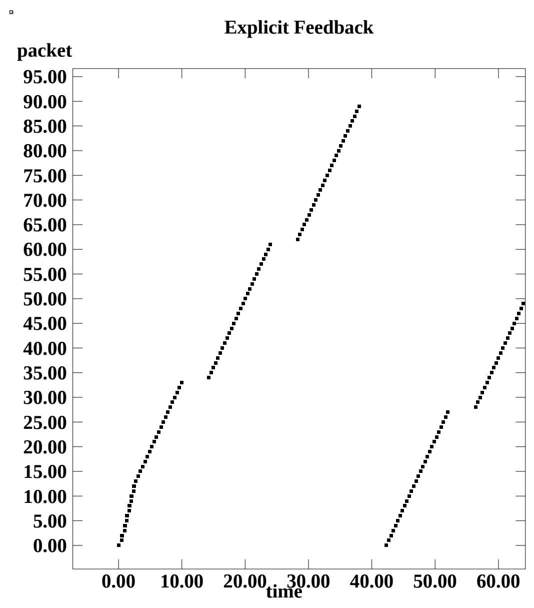


Figure 5: Explicit Feedback

4.2.2 Can ECN work for us ?

The use of explicit feedback for congestion control in the internet has already been shown to work well. ECN in the form of ICMP source quenches [20], is a host's means of informing the source of congestion in the network. This notification is sent to the source after either a packet has been dropped by the host due to an overflow in its buffers, or if it anticipates a dropping of packet(s) based on existing congestion conditions. The latter approach will be more effective in combating congestion as it *predicts* congestion and tries to take preventive measures *before* packets are actually dropped. Upon receipt of this explicit feedback, the source reduces its congestion window to allow a reduction of congestion in the network.

A base station can be configured to function as a gateway supporting ICMP messages. When a wireless link enters a bad state, little data is able to get across to the mobile host (Figure 6, Case 2). This results in queuing of packets from the source host at the base station. It would appear that under these circumstances sending a source quench message to the source would decrease the flow of new packets to the base station, and thus prevent unnecessary timeouts. This should improve the goodput as well as the throughput for the connection. Our results show otherwise.

After carefully tracing the flow of data and acknowledgement packets, we came to the following conclusion. When a wireless link enters a bad state, each datagram (acknowledgement) requires multiple retransmissions before it can get across to the mobile host (base station). This increased delay will result in the source TCP timer timing out (Figure 6, Case 3(a)). A source quench message from the base station at this stage will not be able to prevent timeouts of packets that are *already* on the network. It will of course, stem the flow of any further packets, and reduce the probability of their timeouts. This observation does, however, give us an important clue. When a wireless link is in a bad state, what is needed to prevent timeouts of packets queued at the base station, is a mechanism to update the TCP timer at the source. This mechanism will essentially thwart the source's attempts to invoke congestion control in response to delays on the wireless link. If the base station is able to provide such a mechanism, then we can hope for a measurable improvement in throughput.

4.2.3 Explicit Bad State Notification (EBSN)

The preceding observation led us to explore how the TCP timer could be updated at the source. Clearly, in the absence of acknowledgments coming from the mobile host when the link is in a bad state, we do not have an estimate for the round-trip time. We cannot also generate acknowledgments from the base station for packets that have not yet been received by the mobile host. Neither can we send acknowledgments of previously acknowledged packets as they will be interpreted as *dupacks*. What seems to be a solution is to send an *Explicit Bad State Notification* (EBSN) that would cause the previous timeout to be canceled and a new timeout put in place, based on the *existing estimate* of round trip time and variance[1]. Thus, the new timeout value is identical to the previous one. See Appendix for implementation details. Figure 6, Case 3(b) summarizes the working of EBSN and its role in preventing timeouts at the source. The EBSN approach does not interfere with actual round trip time or variance estimates, and at the same time prevents unnecessary (and detrimental) timeouts from occurring. When the wireless link is in a bad state, we do not want the source to decrease its window if there is no congestion. At the same time we should prevent timeouts for packets that had already been put on the network before the wireless link encountered the bad state. Clearly, EBSN is successful in accomplishing both these tasks.

In our simulations, EBSNs are sent to the source after every unsuccessful attempt by the base station to transmit packets over the wireless link. The correct timeout value at the source is readjusted upon receipt of the first new acknowledgement from the MH following any EBSN messages received. This acknowledgement may initially cause a large variance in the rtt calculation because of the large delay encountered on the wireless link. However, a new acknowledgement is

[1]If the new timeout value is chosen to be very large, deadlock might occur, where the source might never timeout. On the other hand, if we choose a very small value for the new timeout, a timeout might occur before the next EBSN arrives. The existing timeout estimate worked well for our simulations.

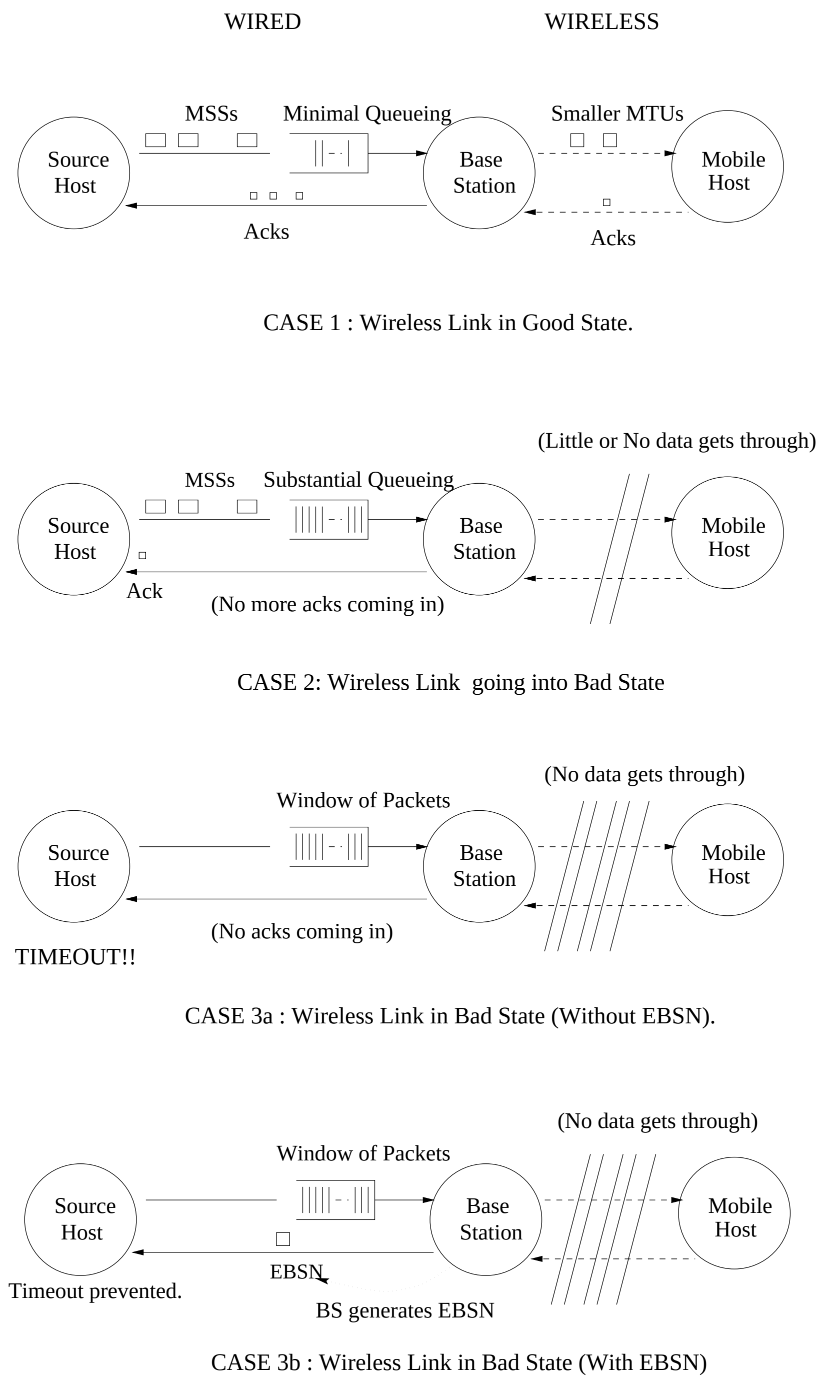
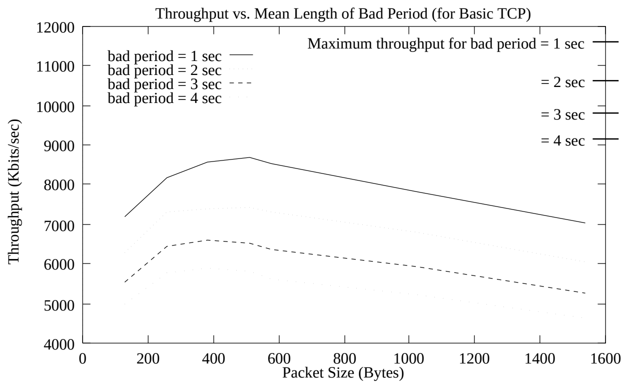


Figure 6: EBSN: How it works



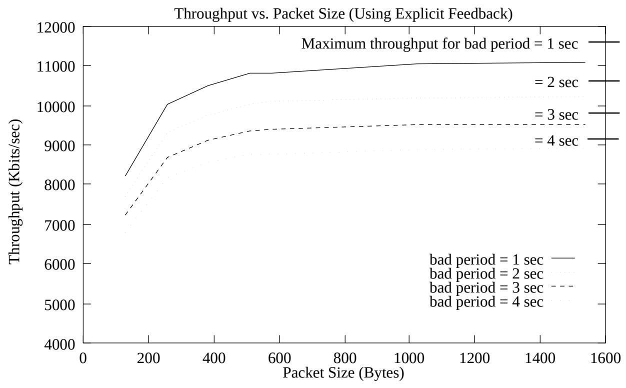


Figure 7: Basic TCP(Wide-Area): Mean Good Period=10 sec

Figure 8: EBSN(Wide-Area): Mean Good Period=10 sec

a good indicator that the wireless link has quit the bad state and has entered a good state. This will ensure a steady inflow of acknowledgments and ultimately reduce variance.

4.2.4 EBSN in Local-Area Wireless Networks

A TCP source is more susceptible to timeouts during local recovery when round-trip times are very small. The round-trip times are typically of the order of milliseconds in a LAN environment. Thus, a LAN environment is an ideal candidate for use of EBSN. We study the impact of EBSN in a typical local-area wireless network. In this case we assume no fragmentation over the wireless link. The setup in Figure 2 is modified for LAN experiments. The wired link bandwidth is set to 10 Mbps, and the wireless link bandwidth is set to 2 Mbps. The TCP window size is fixed at 64 Kbytes. Results of this study are presented in the next section.

5 Results and Discussion

In this section we first present the results of experiments performed on wide-area and local-area wireless networks. We then discuss various issues related to the explicit feedback mechanism using EBSN.

We take into account 40 bytes of header overhead while measuring connection throughput. The standard deviation for all results presented is less than 4%.

5.1 Wide-Area Wireless Networks

Figure 7 and Figure 8 illustrate the variation of throughput with packet size for basic TCP, and EBSN respectively. As stated earlier in the Section 3, the effective maximum throughput ($tput_{max}$) (the maximum throughput achievable without any errors) is kept as 12.8 Kbps. The theoretical maximum throughput ($tput_{th}$) in the presence of errors is determined as ($\frac{\lambda_{bg}}{(\lambda_{bg}+\lambda_{gb})} \times tput_{max}$) (which is $\leq tput_{max}$).

The value of $tput_{th}$ for each λ_{bg} is marked on the top right hand corner of Figures 7 and 8. We varied the packet size on the wired network from 128 bytes to 1536 bytes. Each run involved a 100 Kbyte file transfer from the fixed host to the mobile host. The mean length of the good period is set to 10 sec. We ran experiments for bad period lengths with means ranging from 1 to 4 sec.

Figure 7 shows that for a given packet size, throughput increases as the length of bad period decreases. It can also be seen that for each bad period length, there is an optimal packet size which delivers the maximum throughput. For example, in Figure 7, for bad period = 1 sec, packets of size 512 bytes give the best throughput, however, for a bad period = 3 sec, packets of size 384 bytes give the best throughput. Clearly, a good choice of packet size could provide significant performance improvements over basic TCP (about 30% improvement in throughput is obtained if 512 bytes is chosen as the packet size instead of 1536 bytes, for a bad period = 1 sec). We would like to draw the reader's attention to the large difference between $tput_{th}$ and the throughput obtained even for the optimal packet size for a particular error condition. For example, $tput_{th}$ for bad period = 1 sec is 11.8 Kbps. However, the throughput achieved using 512 bytes (the optimal packet size for bad period = 1 sec) is only 8.7 Kbps.

It can be noticed in Figure 7, that for packet sizes beyond the optimal packet size, throughput decreases. One of the main reasons for this degradation, is increased fragmentation of packets on the wireless link. Loss of a single fragment causes the retransmission of the whole packet from the source. The impact of larger packet size over the wired link is better illustrated in Figure 9 which shows the extra data transmitted by the source due to retransmissions. As is evident, the amount of retransmitted data increases with both the packet size and the length of the bad period; larger the amount of retransmitted data, lower is the goodput of the connection.

The performance of TCP using EBSN is illustrated in Figure 8. There is significantly more improvement in performance of TCP using EBSN than performance of basic TCP. An interesting observation is that unlike basic TCP, the throughput now increases with increase in packet sizes. This is because, irrespective of packet size, timeouts are being completely eliminated when EBSN is used. In the absence of timeouts, there are no redundant retransmissions from the source which is shown in Figure 9. The performance is no longer sensitive to fragmentation over the wireless link, and larger packets perform better. For example, for 1536 bytes, and for a bad period = 4 sec, we notice a 100% improvement in throughput using EBSN; the throughput for basic TCP is 4.5 Kbps, and for TCP using EBSN is 9.0 Kbps. Indeed, we expect improvements to be much higher for larger bad period lengths. The effectiveness of EBSN is evident from Figure 8, where throughput obtained using EBSN is quite close to the theoretical maximum $tput_{th}$ for larger packet sizes.

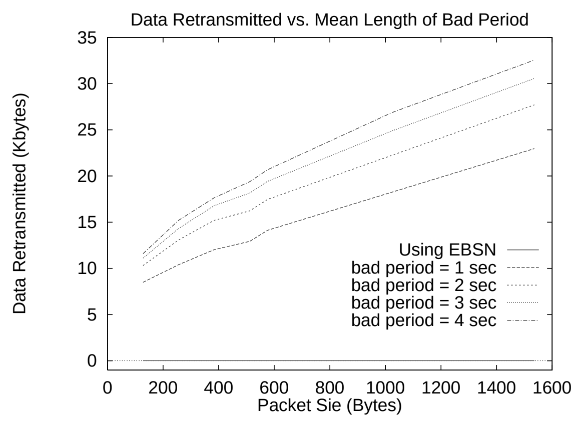


Figure 9: Basic TCP vs EBSN(Wide-Area): 100 Kbyte File, Mean Good Period=10 sec

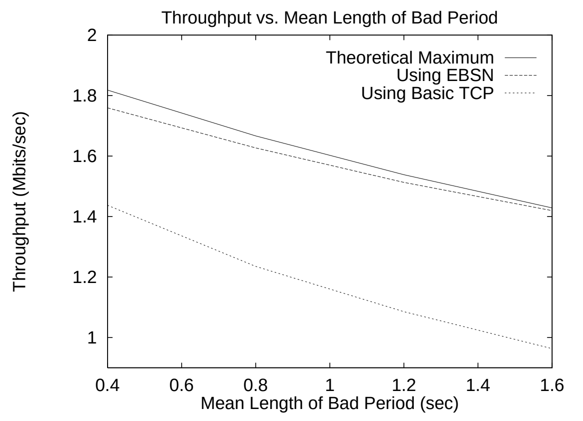


Figure 10: Basic TCP vs EBSN(Local-Area): Mean Good Period=4 sec

5.2 Local-Area Wireless Networks

The packet size for this study is fixed at 1536 bytes. We also assume that there is no fragmentation on the wireless link. The maximum throughput on the wireless link in the absence of errors ($tput_{max}$) is chosen to be 2 Mbps. As before, the maximum theoretical throughput ($tput_{th}$) in the presence of errors is given by ($\frac{\lambda_{bg}}{\lambda_{bg}+\lambda_{gb}} \times tput_{max}$).

Each run involved a 4 Mbyte file transfer from the fixed host to the mobile host. The mean good period length is set to 4 sec. We ran experiments for different bad period lengths with means ranging from 400 milliseconds to 1.6 sec. Figure 10 illustrates the variation of throughput with bad period length for basic TCP and EBSN. The value of $tput_{th}$ for the corresponding bad period lengths is also plotted. TCP using EBSN clearly outperforms the basic TCP scheme. For some bad period lengths, there is about 50% improvement in throughput using EBSN. As before, the primary reason for performance improvement using EBSN is zero timeouts at the TCP source (Figure 11). Once again, the goodput achieved using EBSN is 100%. This is in contrast to the basic TCP scheme, where source timeouts result in a large number of retransmissions.

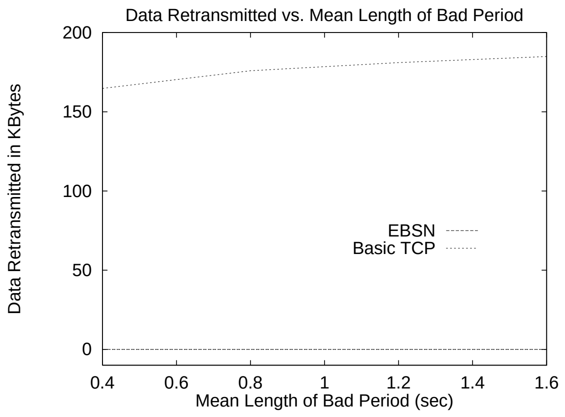


Figure 11: Basic TCP vs EBSN(Local-Area): 4 Mbyte File, Mean Good Period=4 sec

6 Conclusions

This paper presents a study of the effect of burst errors, packet size variation, local recovery, and explicit feedback, on the performance of TCP over wireless networks. First, we propose variation in packet size over the wired network to improve TCP performance. Since, the MTU on the wide-area wireless links is small, a packet on the wired network gets fragmented before transmission over the wireless link. Loss of a fragment causes significant degradation in performance of TCP. Results show that an optimal packet size can lead to about 30% performance improvement over a non-optimal packet size. We have shown that the optimal packet size varies with the error conditions on the link.

Local recovery using link-layer retransmissions from the base station is shown to improve performance. However, while the base station is performing local recovery, the source could still timeout. To prevent these timeouts, we propose an explicit feedback mechanism. The central idea of our approach is to send an Explicit Bad State Notification (EBSN) Message to the source from the base station during local recovery. Upon receipt of an EBSN, the source resets its timeout value. This way, timeouts at the source during local recovery are totally eliminated. It is observed that TCP using EBSN provides upto 100% performance improvement over basic TCP in wide-area wireless networks, and upto 50% performance improvement in local-area wireless networks. These results are for a very conservative error model for the wireless link. We expect greater performance improvements for wireless links having higher BERs. We also show that the throughput and goodput values obtained for TCP using EBSN are very close to the theoretical maximum.

We now summarize the main advantages and disadvantages of the explicit feedback mechanism using EBSN.

Advantages:

• Source timeouts are prevented during local recovery. This improves goodput and throughput of the connection. Note, that the improvements will be more pronounced in high BER wireless links. • TCP using EBSN does not require state maintenance at any intermediate host.

• Effectiveness of some of the proposed approaches [9, 11] depend on the granularity of the TCP timer. A TCP timer with finer granularity will result in a larger number of timeouts during local recovery. This will cause significant degradation in throughput and goodput. With our explicit feedback mechanism, the timeout value at the source is reset upon receipt of every EBSN. This reduces the number of timeouts, and in a large number of cases prevents timeouts from occurring at all. The effect of clock granularity on performance is now greatly reduced, thus improving the robustness of the network [23].

Disadvantage:

• The main disadvantage of EBSN is that it requires modification to TCP code at the source. Note, however, the changes involved are minimal as shown in the Appendix.

We view this work as a preliminary investigation into the effectiveness of explicit feedback mechanisms, to improve performance of TCP in wireless networks. In this paper, we have assumed that the wired network is not congested. We are separately studying the impact of congestion in the wired network on the effectiveness of EBSN [18]. This involves looking into issues related to the interaction between ECN and EBSN. We are also investigating schemes to make a source timer more robust to larger delays on the wireless link without using explicit feedback mechanisms. If this is possible, we will be able to achieve performance improvements comparable to those using EBSN without changing TCP code at the end hosts.

References

[1] P. Karn and C. Partridge, "Estimating round-trip times in reliable transport protocols," *Proc. SIGCOMM*, Aug., 1987.

[2] V. Jacobson, "Congestion Avoidance and Control," *Proc. SIGCOMM,* pp. 314-329, Aug., 1988.

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

[4] 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.

[5] W. Stevens, *TCP/IP Illustrated*, Volume 1, Addison-Wesley, 1994.

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

[7] A.Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," *ICDCS,* Oct., 1994.

[8] Tim Alanko et al, "Measured Performance of Data Transmission Over Cellular Telephone Networks." Technical report TR C-1994-53, University of Helsinki.

[9] P. Bhagwat et. al., "Enhancing Throughput over Wireless LANs Using Channel State Dependent Packet Scheduling," *INFOCOM*, 1995.

[10] A.Bakre and B.R. Badrinath, "Handoff and System Support for Indirect TCP/IP," *Proc. Second Usenix Symp. on Mobile and Location-Independent Computing,* April, 1995.

[11] H. Balakrishnan, S. Seshan, E. Amir, R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," *Proc. 1st ACM Conf. on Mobile Computing and Networking,* November 1995.

[12] Cellular Digital Packet Data System Specification: Release 1.0, CDPD Forum Inc., 1993.

[13] Sally Floyd, Steve McCanne, "Network Simulator." LBNL public domain software. Available via ftp from ftp.ee.lbl.gov.

[14] R Braden, "Requirements for Internet Hosts – Communication Layers", RFC 1122, October 1989.

[15] C. A. Kent, J. C. Mogul, "Fragmentation considered harmful," *SIGCOMM 1988.*

[16] Authors of this paper, "Performance of TCP over Wireless Networks," Tech Report # TR-95-049, December, 1995.

[17] Authors of this paper, "Seamless Communication over Mobile Wireless Networks," Tech Report # TR-96-25, April, 1996.

[18] Authors of this paper, "Explicit Feedback in Wireless Networks," Tech Report in preparation.

[19] A. Conta and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)," RFC 1885, December, 1995.

[20] J. Postel, "Internet Control Message Protocol," RFC 792, September 1981.

[21] J. Mogul and S. Deering, "Path MTU Discovery," RFC 1191, November, 1990.

[22] J. Postel, "The TCP Maximum Segment Size and Related Topics," RFC 879, November, 1983.

[23] Sally Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, V.24, No. 5, October 1994.

Appendix: Implementation of EBSN in our Simulator

In the normal operation of TCP, each time a data packet is sent out by a TCP source, any previously set timer is first disabled, and a new one put in place based on the latest estimate of round trip time. We reproduce the basic algorithm below. Details may be found in [stev94].

Basic TCP

```
tcp_recv()
{
  if ((fast_retransmit) || (timeout)) {
     update ssthresh;
     update cwnd;
     update timer_backoff;
     set_rtx_timer(); /* explained below */
     retransmit lost pkt;
     return;
  }
  /* Other packet processing */
}
```

The set_rtx_timer() procedure cancels any previous timer, and puts a new one in place based on the latest estimate of round trip time.

```
set_rtx_timer()
{
   if previous timer(TCP_TIMER)
         cancel(TCP_TIMER);
/* Now calculate new timeout based on latest
   estimate of round trip time and variance */
   new_timeout = calculate_new_timeout();
   set_tcp_timer(new_timeout);
}
```

Response to an EBSN message requires minimal changes to the tcp code, which are localized to the tcp_recv() routine only. On receipt of an EBSN message, the source replaces any previous timer with a new timer retaining the current timeout value. Note, that EBSN can be implemented as a new type of ICMP message.

TCP's response to EBSN

```
tcp_recv()
{

  if EBSN received {
    set_rtx_timer();
    return;
  }

  /* Other packet processing */
}
```