

An Overview of Techniques for Improving TCP Performance Over Wireless Networks

Jayesh Vyas

Tejas Networks, Bangalore, India

Abstract—Reliable transport protocols such as TCP are tuned to perform well in traditional networks where packet losses occur mostly because of congestion. However, networks with wireless and other lossy links also suffer from significant packet losses due to bit errors and handoffs. TCP responds to a packet loss by invoking congestion control and avoidance algorithms, resulting in degraded end-to-end performance in wireless and lossy systems. There are various protocols proposed based on different schemes and the modes of operation. In this report, we discuss some of these methods, and do a qualitative study of their performance using throughput as a performance metric.

I. INTRODUCTION

The increasing popularity and demand of the wireless links indicate that wireless technology will be one of the leading technologies in future for communication. Reliable transport protocols such as TCP have been tuned to work satisfactorily well in the traditional networks comprising of wired links and fixed hosts. These protocols assume *congestion* to be the primary cause of packet loss, and perform well over such networks by adapting their rates of transmission in response to congestion. The TCP sender uses *cumulative acknowledgments* which it receives from the receiver and the *retransmission time out* to detect a packet loss in the network. For this purpose it maintains a running average of the estimated *round trip time* and the mean linear deviation from it. The sender identifies loss of a packet either by the arrival of several duplicate acknowledgments or by the absence of an acknowledgment for a packet within a *timeout* interval. TCP assumes the packet loss due to congestion in the network, and thus when it detects a packet loss it reduces its *transmission window size* before retransmitting the lost packet, initiating congestion control and avoidance mechanisms. This results in the reduction of load on intermediate links, thereby reducing congestion in the network.

TCP on Lossy Wireless Links

Connectivity over wireless links is often characterized by sporadic high bit error rates due to fading and the atmospheric conditions. During the handoff time, there is a temporary break in the connection, which also results in packet loss. Since TCP considers all packet losses in the network due to congestion, and invokes its congestion avoidance algorithm when a loss is detected, hence there is degradation in its performance and end-to-end throughput over such lines with high probability of errors.

Classification of Various Schemes

There have been several mechanisms proposed to improve the performance of TCP on networks which have wireless or similar lossy lines, such as local retransmissions, split-TCP connection, forward error correction, explicit notification and changing the sender and the receiver itself. All these methods can be classified into two *approaches*[7]. The first approach hides any non-congestive losses from the end hosts, and thus requires changes in the intermediate network elements. The rationale behind this approach is that since the problem is local, so it should be solved locally, and the transport layer need not be made aware of the characteristics of the underlying physical link. Protocols that use this approach attempt to make the lossy link appear as a high quality link with a reduced bandwidth. As a result most of the losses seen by the TCP sender are due to congestion only. Examples of protocols which use this approach are snoop TCP, and split connection approaches. The second class of techniques attempts to make the sender aware of the existence of the wireless hops in the networks and realize that some losses in the network are not due to congestion. Explicit notification and sender based algorithms use this approach. Finally there can be techniques in which there is a link aware transport layer protocol which co-exists with a TCP aware link layer protocol to improve the overall performance.

On the basis of the *fundamental philosophy* which the various schemes use, they can be divided into three main groups: end-to-end proposals, split connection proposals and link layer proposals[7]. The end-to-end protocols attempt to make the TCP sender aware of the non-congestive losses either through *Selective ACKnowledgments* (SACKs) or using *explicit notifications*. The split-connection protocols split the TCP protocol at the base station into two different connections. The second connection between the base station and wireless host then can use some other protocol suitable for that link for an improved performance, like negative or selective acknowledgments. The third link layer philosophy lies in between these two, as it tries to hide the wireless link losses from the end nodes by using local retransmissions or forward error correcting codes. The local retransmission techniques that are used are tuned to the characteristics of the wireless channel to provide enhanced performance. But the TCP sender may not be fully shielded from the wireless losses because by the time the link layer reliably retransmits the lost packets, the timer at the sender may timeout, resulting in the invocation of congestion avoidance methods by the sender. So some methods provide

complete shielding to the TCP sender based on the knowledge of TCP transmission control algorithm, like suppressing the duplicate acknowledgments.

In this report, we discuss the various techniques to make TCP more suitable to wireless networks. The techniques have been divided into the following classes and are discussed one by one[3]:

- Sender based discrimination
- Receiver based discrimination
- Explicit notification
- Link level mechanisms
- Split connection approach
- TCP aware link layer

But first, we study the TCP's congestion control technique in the next section.

II. THE TCP

TCP is a very widely implemented transport layer protocol which is used with IP over the internet since several years[2]. TCP ensures reliable and ordered delivery of data and prevents congestion in the network using its congestion control method. It maintains the end-to-end semantics, and requires the receiver to acknowledge the reception of data packets.

The TCP connection uses a window based control scheme. At any time t , the sender maintains the following variables for each connection[6]:

- $A(t)$ The lower window edge. All data numbered up to and including $A(t) - 1$ have been transmitted successfully and have been acknowledged by the receiver. The receipt of an ACK with sequence number n greater than $A(t)$ increases $A(t)$ to $A(t) + n$.
- $W(t)$ The congestion window ($cwnd$). The transmitter can send packets with sequence number n , such that $A(t) \leq n \leq [A(t) + W(t)]$, where $W(t) \leq W_{max}$. W_{max} is the maximum window size advertised by the receiver.
- W_{th} The slow start threshold ($ssthresh$).
- RTO Retransmission Time Out. If a packet is not ACKed before this timer is expired, then it is assumed to be lost. $RTO = mean(RTT) + 4\sigma$, where RTT is the round trip time, and σ is the mean linear deviation of RTT .

At the start of the transmission, $W(t)$ is set to 1, and $ssthresh$ to half of the receiver's initial advertised window[1][4]. The sender starts transmission in the *slow start*(exponential) phase, in which the $cwnd$ is increased by 1 for every received ACK. So the window size is doubled every Round Trip Time(RTT). After the window size increases up to the $ssthresh$, the source switches to *congestion avoidance*(linear) phase, in which for every ACK the window size is increased by $\frac{1}{[W(t)]}$. So in one RTT , the window size increases by one. Note that the window size is a floating point number, the integral part of which is the actual window size[4].

TCP uses byte sequence numbers[8]. The receiver sends cumulative ACKs; if all data sequence up to $n - 1$ have been received then the receiver sends back the acknowledgment—*Next Expected = n*". If there is a packet loss in the network then the receiver keeps on sending the cumulative ACKs for the next expected packet(*duplicate ACKs*). The sender detects

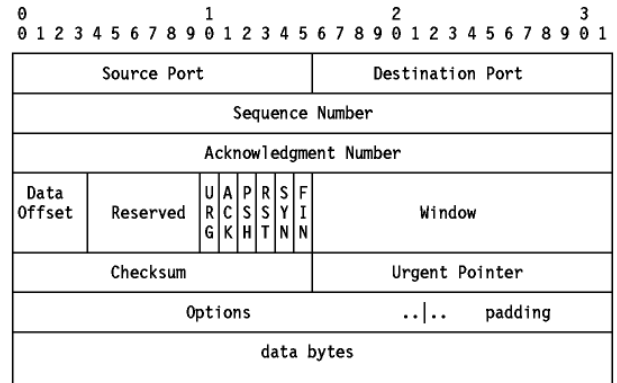


Fig. 1. The TCP Header [2]

a packet loss, if the number of dupacks for a packet exceeds a threshold(usually 3). If the network is very congested, and even the dupacks are also lost, then the sender detects a packet loss by the expiry of the coarse RTO . There are several variants of TCP which respond little differently to packet losses[1][4].

TCP Tahoe

The behavior of the source in the slow start and congestion avoidance phase is same is described above. TCP Tahoe detects a packet loss by the expiry of its coarse timer. It assumes that if the RTO expires, then there must be congestion in the network, and so it reduces the congestion window to 1 and the $ssthresh$ to half of the current window size. The RTO is also made twice of its present value, that is;

$$W_{th} = \frac{W(t)}{2}$$

$$W(t) = 1$$

$$RTO = 2 * RTO$$

Thus, the sender goes into the slow start phase.

TCP Reno

For the timeouts a TCP Reno source behaves the same way as the TCP Tahoe source, but it also responds for congestion in the network when it receives more than 3 dupacks. When the sender receives more than 3 dupacks, it reduces both its $cwnd$ and W_{th} to one half of the current window size, that is

```

if(time out)
  Wth = W(t)/2
  W(t) = 1 /*slow start is entered here */
  RTO = 2 * RTO
endif
if(no. of dupacks > 3)
  Wth = W(t)/2
  W(t) = Wth /*congestion avoidance is entered here */
endif

```

TCP Reno also uses the *fast retransmit* algorithm. When a packet loss is detected by the sender, then the sender after invoking the congestion control algorithm immediately retransmits the lost packet.

Apart from fast retransmit, TCP reno also uses *fast recovery* technique. After the fast retransmit, the $cwnd$ size is increased

by 1 for each received dupack. When a non-duplicate ACK is received, then the *cwnd* is set to *ssthresh*, that is:

```

cwnd = ssthresh + number_of_dupacks
if(a new nondup ack comes)
cwnd = ssthresh

```

III. SENDER BASED DISCRIMINATION

These are end-to-end methods, in which the sender tries to determine the cause of the packet loss using heuristics based on the round trip time (RTT), window sizes, and loss pattern. The figure below shows the typical behavior of a network as the load increases[3]:

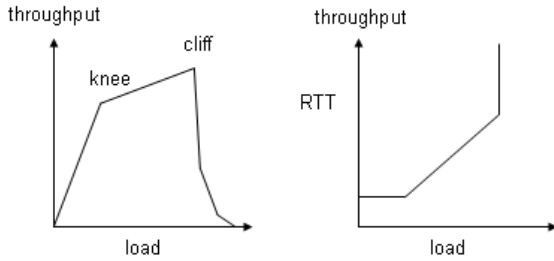


Fig. 2. Heuristics for Congestion Avoidance

There are various ways in which the source can estimate the condition of the network. For a TCP source we define a condition Ω as the function of the observed *RTT* and the congestion window sizes[3]. This condition is evaluated whenever a new ACK is received. The condition typically takes 2 or 3 different values, but for the present discussion we assume that Ω takes only 2 values *true* or *false*. If there is a packet loss, and the value of Ω is *true*, then it is an indication of congestion in the network. There are several proposals to calculate Ω [3]:

Normalized Delay Gradient

Here the argument for calculating Ω is that when there is congestion in the network, the round trip time(*RTT*) also increases when the window size is increasing, both in slow start and in congestion avoidance phase. We calculate the normalized gradients for the *RTT* and window size(*W*) as

$$r = \frac{RTT(i) - RTT(i-1)}{RTT(i) + RTT(i-1)}$$

$$w = \frac{W(i) - W(i-1)}{W(i) + W(i-1)}$$

where the values are calculated when the acknowledgment for the i_{th} packet is received.

The condition Ω is then:

$$\Omega = \left(\frac{r}{w} > 0 \right)$$

Normalized Throughput Gradient

The heuristic used here is that if there is no congestion in the network, then the proportional increase in the throughput(*T*) with respect to the increase in the window size should not fall below a certain threshold as time of the connection increases. Usually this threshold is 0.5 times the base throughput gradient, which is the throughput gradient measured when the first packet was sent.

$$Throughput\ Gradient(TG(i)) = \frac{[T(i) - T(i-1)]}{[W(i) - W(i-1)]}$$

$$Normalized\ Throughput\ Gradient(NTG) = \frac{TG(i)}{TG(1)}$$

And the condition $\Omega = (NTG < 0.5)$

TCP Vegas

The main idea in TCP-Vegas is that the sender can compute the expected throughput of a connection as[4]:

$$Expected\ Throughput(ET) = \frac{W(i)}{Propagation_Delay}$$

The denominator is estimated by computing the smallest *RTT* seen so far, which is likely to be close to the actual propagation delay. The source can easily calculate the actual throughput by measuring how many packets has it sent in one round trip time.

$$Actual\ throughput(AT) = \frac{W(i)}{RTT(i)}$$

In practice the source sends a distinguished packet and records the transmission time of this packet. When the source receives the acknowledgment for the packet, it computes the estimated and actual transmission throughput. If the *ET* value is greater than the *AT*, then $(ET - AT)RTT$ packets which were transmitted in the previous *RTT* are still in the bottleneck buffer, so the source can guess that there is congestion in the network when $(ET - AT)$ is greater than some value, say β . Therefore the condition Ω is:

$$\Omega = ((ET - AT) > \beta)$$

To estimate the state of the network. the sender calculates and records the latest value of Ω by one of the methods above. If a packet loss is detected, and if the last recorded value of Ω is *true*, then the packet loss is assumed due to congestion in the network, else it is assumed that the packet is lost due to transmission errors.

But in many cases these sender based mechanisms do not work satisfactorily, because the statistics collected by the sender about the state of the network is often garbled by other traffic in the network. Also there is not much correlation observed between the short term statistics and the onset of congestion.

TCP Westwood - Congestion Window Control Using Bandwidth Estimation

TCP Westwood(TCPW)[5] is one more sender based scheme that controls the congestion window using end-to-end rate estimation in a way that is transparent both to the routers and to the destination. Thus, it is compatible with any network and TCP implementation. The key idea is to continuously estimate, at the sender, the available bandwidth to the TCP connection by monitoring the rate at which the acknowledgments are received. The estimated connection rate is then used to determine the congestion window size and the slow start threshold after a congestion period. Resetting the congestion window to match the available bandwidth makes TCPW more robust to sporadic wireless losses, to which the conventional TCP Reno and Tahoe often overreact by quickly reducing the congestion window to half or setting it to 1. TCPW uses the *bandwidth estimate* to directly drive the congestion window, instead of using it to compute the *backlog*, unlike done in packet pair and TCP Vegas. The reasoning behind this is that if a connection is currently achieving a given rate then it can safely use the window corresponding to that rate without causing congestion in the network.

Bandwidth Estimation: The TCP sender uses the following information to estimate the available bandwidth in the network for the connection (1) the ACK arrival times and, (2) the increment of data delivered to the destination. Let us assume that an ACK is received at the source at time t_k , notifying that d_k bytes have been received at the TCP receiver. We can measure the sample bandwidth used by that connection as

$$b_k = \frac{d_k}{(t_k - t_{k-1})}$$

where t_{k-1} is the time the previous ACK was received. Letting $t_k - t_{k-1} = \Delta t$, we get $b_k = \frac{d_k}{\Delta t}$.

The BandWidth Estimate (BWE) varies from flow to flow sharing the same bottleneck; it corresponds to the rate actually achieved by each individual flow. The duplicate acknowledgments are also considered in estimating the bandwidth, because they also indicate that a packet was successfully transmitted across the network. For the same reason delayed acknowledgments are also used in calculating the BWE . Another important parameter used in this procedure is the Round Trip Time (RTT). Ideally it should be measured when there is no queuing in the network and the bottleneck buffer is empty. So in practice, it is set to the overall minimum of the round trip delays seen so far on that connection, based on continuous monitoring of the ACKs RTT .

Setting of the cwnd and the ssthresh: The TCPW congestion window increments during the slow start phase and the congestion avoidance period are same as that in TCP Reno and TCP Tahoe, and it also detects a packet loss by either a coarse timer timeout or receipt of 3 duplicate acknowledgments. But the congestion avoidance algorithm is different. TCPW reacts to a packet loss in the following way.

```

if (3 DUPACKS received)
  ssthresh =  $\frac{BWE * RTT_{min}}{seg\_size}$ 
  if (cwnd > ssthresh) /*congestion avoidance*/
    cwnd = ssthresh
  endif
endif

```

where the BWE is in bits per second, and the seg_size is the length of the TCP segment in bits.

In case of a timeout:

```

if (the coarse timeout expires)
  cwnd = 1; /*slow start entered here*/
  ssthresh =  $\frac{BWE * RTT_{min}}{seg\_size}$ 
  if (ssthresh < 2)
    then ssthresh = 2
  endif
endif

```

Considering the above algorithm, one might feel that this aggressive behavior of TCPW might lead to unfairness for the other connections, and unfriendliness for the other TCP implementations. In the next section we show that it is indeed a fair-share and friendly scheme for all the connections.

TCPW Fairness and Friendliness: We will use an informal argument to show that TCPW ensures fairness for all the connection and that it is friendly with other TCP implementations as well. Consider a simple situation when there are only 2 senders, both with the same RTT . Let the bottleneck buffer capacity be X packets, and the bottleneck link capacity be μ packets/s. For simplicity let us assume that the bottleneck buffer size

is equal to the optimal window size that is $X = \mu * RTT$. One connection, say A, starts first. So as per the TCPW algorithm described in the previous section, the window size of A oscillates between X and $2X$, each cycle terminating when there is a buffer overflow when the window size is $2X$. Later the connection B starts, first in the slow start period, and then in the congestion avoidance phase. Now let at some time the bottleneck link is utilized to its full capacity, and R_A and R_B be the share of the bottleneck link for the connections A and B respectively, then

$$R_A + R_B = \mu$$

$$\Rightarrow \frac{R_A}{\mu} + \frac{R_B}{\mu} = 1$$

If X_A and X_B are A's and B's share of the buffer space at the bottleneck buffer, then

$$X_A = \frac{R_A}{\mu} * X$$

$$\text{and } X_B = \frac{R_B}{\mu} * X$$

because the buffer share of each connection will be proportional to its bandwidth share.

At the time of congestion, that is when the buffer overflows

$$W_A - R_A * RTT = X_A = \frac{R_A}{\mu} * X$$

$$\Rightarrow W_A = R_A * \left[\frac{X}{\mu} + RTT \right]$$

Similarly

$$W_B = R_B * \left[\frac{X}{\mu} + RTT \right]$$

$$\Rightarrow W_A + W_B = X + \mu * RTT$$

This is a general property true for all TCP connections and particularly for TCPW. Note that the bottleneck link share for the connections (R_A and R_B) will be roughly equal to the BWE done by the sender. So after the congestion is detected by the sender due to the buffer overflow, the new TCPW window sizes are

$$W'_A = R_A * RTT_{min}$$

and

$$W'_B = R_B * RTT_{min}$$

Thus the ratio of the window sizes of the connections A and B are maintained after a congestion period. So the congestion period in the $W_A vs W_B$ graph is indicated by a straight line passing through the origin. Just after the congestion episode there is no backlog in the network routers and $W_A + W_B = \mu * RTT$, that is the total window size is the optimal window size. Thus the total window size for more than one connection also oscillates between X and $2X$. But the $\frac{W_A}{W_B}$ ratio keeps on decreasing (towards one) during the congestion avoidance (linear) period, because after each RTT , $\frac{W_A}{W_B}$ becomes $\frac{(W_A+1)}{(W_B+1)}$. So this follows the 45° line in the $W_A vs W_B$ graph. Equilibrium is reached when both the connection have equal share of the bandwidth and of the bottleneck buffer, as indicated in the graph below. The convergence point is the line $W_A = W_B$ as shown in the Fig. 3.

The above argument about the fairness of TCPW is valid even if the buffer size (X) is not equal to the optimal window size ($\mu * RTT$), and the RTT for the 2 connections are different. Similar argument can be used for the friendliness of TCPW with TCP Reno. Suppose the buffer size is again equal to $\mu * RTT$, then at equilibrium the window size of both TCPW and TCP Reno is X just before the congestion. After overflow the TCPW reduces the window size to $X/2$, while the TCP Reno simply reduces it to half of the current value, that is $X/2$. Thus friendliness is preserved. The design of the

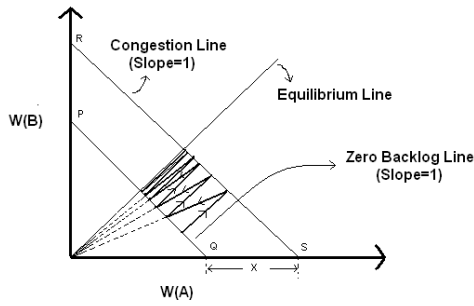


Fig. 3. Window Sizes for A and B

optimum buffer is important, because when the buffer size is smaller than the optimal, the TCPW gets a larger *cwnd* than TCP Reno, and captures the channel. And when the buffer size is smaller than the optimal, then TCP Reno tends to override TCPW.

In order to study the behavior of TCPW in presence of random errors, an analytic model was developed using the *Markov Chain Techniques* taking into the consideration the various factors like the propagation delay, bottleneck buffer size, bottleneck link capacity, and error rate. The results obtained from this model are confirmed by the simulation results as well as some of the test implementations of TCPW, which show a significant improvement in performance over TCP Reno over lossy wireless links[5].

IV. RECEIVER BASED DISCRIMINATION

When the receiver in the TCP connection is a mobile host, then it can use some heuristics to guess the reason for the packet loss, and then notify the sender about the loss through either explicit notification or by setting the Explicit Loss Notification (ELN) bit in the duplicate acknowledgments[3].

The inter-arrival time between consecutively received packets can be used to infer whether a packet is lost due to congestion in the network or due to wireless channel errors. Suppose the packet transmission time for each packet on the wireless link is T . So the receiver receives one packet after every T time. Now, if there is a packet loss due to channel or link error, then there is a *gap* between the successfully received packets. If the receiver received the last in-sequence packet at time $t = t(n)$, and there is a packet loss due to error in the channel, then the next correct packet received would be at time $t = t(n) + 2 * T$, instead of $t = t(n) + T$.

But if the packet loss is due to congestive errors in the wired part of the network, then the base station does not send the lost packet, and so there is no *gap* in the packets received by the mobile host. So even if there is a packet loss in the network, the receiver receives the next packet (after the last in-sequence packet) at time $t = t(n) + T$.

So when the receiver finds a packet missing, or out of order, then it observes the arrival time of the next received packet, and measures the inter-arrival time. If it is found to be close to

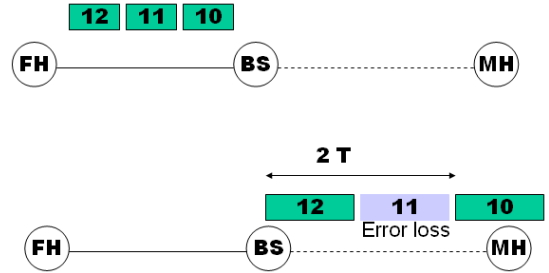


Fig. 4. Packet Loss Due to Channel Error

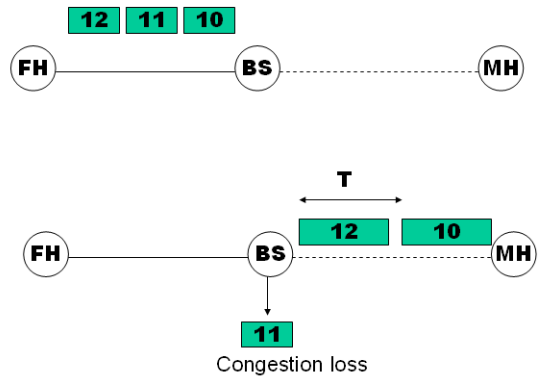


Fig. 5. Packet Loss Due to Congestion

the mean inter-arrival time then the receiver sends the duplicate acknowledgments, as in the normal TCP. But when the inter-arrival time is of the order of twice the mean inter-arrival time, then the receiver notifies the sender about the random error in the channel, either by explicit notification by sending a separate message, or by setting the ECN bit in the duplicate acknowledgments for the lost packet. The TCP sender on receiving the notification retransmits the lost packet, but does not reduce the congestion window.

This method maintains the end-to-end semantics of the TCP connection, and does not require any support from the base station. It can be used even when the data and the acknowledgments traverse different paths, or when the TCP header is encrypted. But this approach has limited applicability. The mobile host has to be the TCP receiver only, and wireless hop from the base station to the mobile host should be the slowest link on the path, thus ensuring that some queuing will occur at the base station, so there will always be some packets to be sent by the BS and thus the packet inter-arrival time at the receiver would be constant. Also when there are more than one connection for the wireless hop, then the queuing delays at the base station should be more or less uniform for each TCP connection.

V. EXPLICIT NOTIFICATION

In these schemes the TCP sender is somehow informed or it deduces that the packet losses are not due to congestion, but are due to some other losses, so that it only retransmits the packets and does not reduce the window size[3].

Explicit Loss Notification (ELN)

One of the simple schemes is described below[3]:

1) *When wireless host is the TCP sender:* In this method the base station keeps track of the missing packets in the TCP flow. When a dupack is received from the receiver, the base station matches it with the missing sequence numbers it has cached. If there is a match, then the packet was lost in the wireless part, and not due to congestion in the wired part. So the BS sets the ELN bit on the ack. When the sender receives the dupack with the ELN bit set, then it only retransmits the packet and does not reduce the window size.

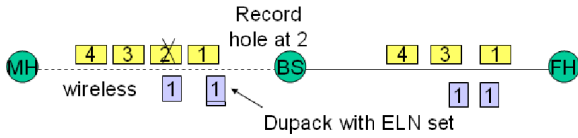


Fig. 6. MH is the TCP sender

2) *When wireless host is the TCP receiver:* This time the base station caches the TCP sequence numbers of the packets it has received successfully from the wired section of the TCP connection. When the base station sees a dupack, it matches its sequence number with the cache of sequence numbers. If there is a match then the BS sets the ELN bit in the dupack. So for the packet loss in the wireless part the BS informs the sender by setting the ELN bit, and the sender takes appropriate action for it.

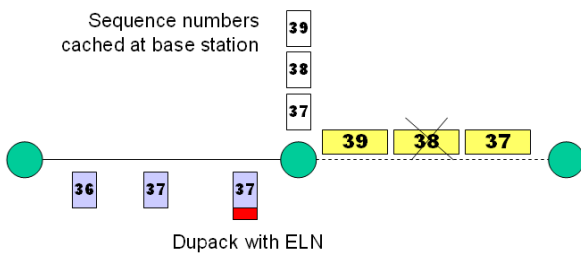


Fig. 7. MH is the TCP receiver

Explicit Bad State Notification (EBSN)

When the MH is the TCP receiver, the BS attempts to deliver packet using link layer retransmission scheme. If the packet cannot be delivered using small number of LL retransmissions, the BS sends an EBSN message to the TCP sender[3]. When the TCP sender receives an EBSN, it resets its timer. So the timeout is delayed, when wireless channel is in bad state.

Partial ACK protocols

In these protocols the sender is sent two types of acks. A *partial ack* from an intermediate host (usually the BS), informs the sender that it has received the packet. The normal TCP ack is also needed by the sender for reliability of the packet delivery[3]. When the sender receives a dupack, along with a partial

ack for the same packet, then it does not reduce the window size. The loss is then assumed to be due to wireless channel losses. So in this scheme, the base station need not buffer the TCP packets.

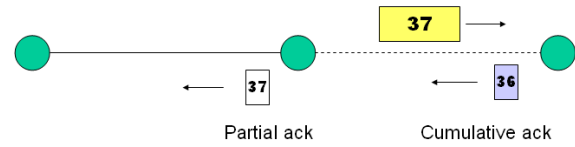


Fig. 8. Partial ACK protocol

Explicit Congestion Notification (ECN)

The sender is made aware of the non-congestive losses in the network by explicit feedback from the network and the sender in the form of Explicit Congestion Notification (ECN)[6]. The router in the network plays an important role in this.

Random Early Detection (RED): It is an active queue management mechanism which detects congestion before the queue overflows and informs the end nodes about the incipient congestion. The RED router continuously calculates the running average of the queue size, to avoid any bias to bursty traffic. Two levels of threshold minimum (min_{th}) and maximum (max_{th}) are maintained by the router. If the average queue size lies between min_{th} and max_{th} , then the arriving packets are dropped *probabilistically*, with the condition that the probability of dropping a packet belonging to a particular flow is approximately proportional to its share of the bandwidth. So the user who uses more bandwidth is penalized more by having dropped more packets. If the average queue size is more than the maximum threshold, then the arriving packets are always dropped. Thus the problem of *global synchronization* is also avoided.

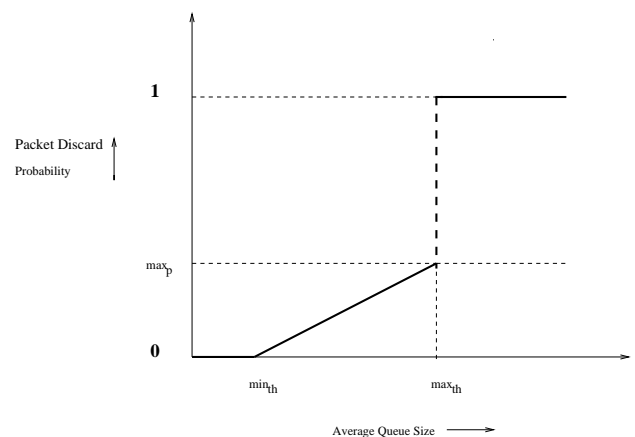


Fig. 9. Queue Management at the RED Router

Explicit Congestion Notification (ECN): This is an extension proposed to the RED, in which the packets are marked instead of being dropped when the average queue size lies between

min_{th} and max_{th} . Since ECN marks packets before congestion actually occurs, it is useful for protocols like TCP which are sensitive to even single packet losses. When the receiver receives the marked packets, it informs the sender in the subsequent acks about the incipient congestion, which triggers the congestion avoidance algorithm by the sender.

Details of the protocol: The ECN capable transport(ECT)bit is set by the sender when both the end system are ECN capable. Packets which are not ECN capable, are still dropped probabilistically by the router when the average queue size lies between min_{th} and max_{th} .

During the call set up phase, the source and the destination TCP negotiate whether each one of them is capable and/or willing to use ECN. When there is an agreement to use ECN, the source sets the ECT bit on the *IP packet headers*. When the queue size at the router is between the thresholds, and when the router finds the ECT bit set on the packets, it sets the Congestion Experienced (CE) bit on the IP headers with probability proportional to the average queue size. When the receiver receives a CE data packet, it sets the ECN-Echo (ECE) flag and the CE bit on the subsequent acknowledgments, until it receives a Congestion Window Reduced (CWR) message from the sender. When the TCP is using the delayed ACKs scheme, as used in many current implementations, where the TCP receiver sends one acknowledgment for 2 successfully received packets, the ECE flag and the CE bit in the ACK packet will be set to the OR of the CE bits of all the data packets that are being acknowledged by that ACK. This increases the robustness of the mechanism. When the sender gets an ack with ECE bit set, it invokes the congestion avoidance mechanism and reduces the window size. After the window size is reduced, the TCP sets the CWR bit on the next sent data packet.

Extension of ECN to Wireless Networks: The ECN can be extended to wireless networks which use TCP to determine the cause of packet loss. This is done by observing the Congestion Experienced(CE) bit in the dupacks received if the TCP is ECN capable.

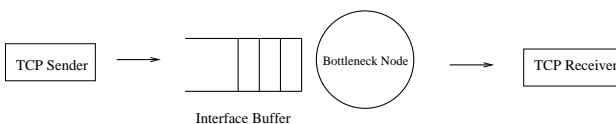


Fig. 10.

Consider the simple network as shown above. If the queue size at the router is between the maximum and minimum threshold value of the average queue length, and if the packets are ECN capable, then the router sets the CE bit on the IP header, which is forwarded to the sender in the subsequent ACKs by the receiver. If the sender receives third duplicate acknowledgment with its CE bit set, then it can infer that the packet loss has occurred due to congestion in the network and it can start its congestion avoidance methods to take care of it. This is because the intermediate router already warned the end nodes about the incipient overflow of the buffers by setting the CE bit. When the buffer actually overflows and a packet is dropped, the receiver sends duplicate acknowledgment and the sender can conclude

about the congestion when it notices the CE bit also set. But, if the average queue size at the buffer is less than the minimum threshold, then the CE bit in the TCP packets will not be set by the bottleneck node, and as a result the dupack will also not have the ECE and CE bits set. Now if there is a random packet loss in the network which is informed to the sender by sending three duplicate ACKs by the receiver in which none of the ACKs have their CE bit set, then the sender can conclude that packet loss is due to random wireless loss in the network. So once the loss has been recognized as a wireless loss, the TCP sender need not use the congestion avoidance algorithm. But the window size is still reduced, because the wireless hop of the TCP connection is still is a bad state. Hence if the packet loss is detected due to random wireless loss, then the approach is to reduce the window size but recover fast when the connection becomes good again.

So there are some changes at the sender's side. Upon receipt of an ACK with the CE bit set, and if the ACK is third duplicate in a row, then the sender infers it to be a sign of congestion in the network, and congestion avoidance algorithm is triggered reducing the window size and the slow start threshold is reduced to half of the current window size. But if the sender receives 3 dupack's with none of them having their CE bits set, then the window size is reduced, but the recovery is also fast. Once the source has taken appropriate steps, it sets the Congestion Window Reduced (CWR) bit in the next packet, to inform the receiver that it has responded to its congestion notification.

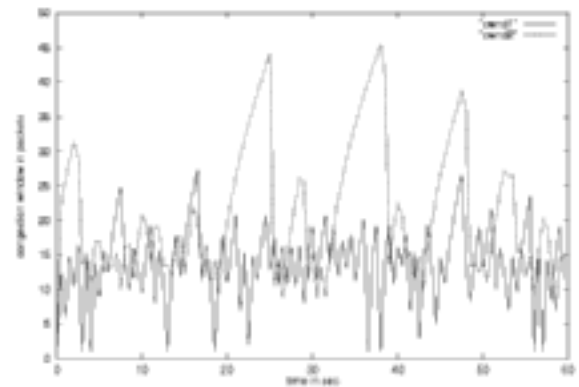


Fig. 11. TCP Window Variation

Simulation results with TCP Tahoe shows improvement in performance with this modified TCP scheme [6]. With the following simulation parameters in the LBNL network simulator NS, the modified TCP achieved a throughput of 30 more than the unmodified TCP during a period of 60s.

The buffer capacity at the bottleneck buffer was 120kbits. Data rate of the link is 2Mbps with round trip time 4μs. The technology used is 914Mhz Lucent WaveLan radio interface, with transmission power .2818W, and received power threshold $3.652 * 10^{-10}W$. For the RED router $min_{th} = 96kbits$, $max_{th} = 120kbits$, mean packet size= 8kbits and $max_p = 0.02$. It was also noticed that the congestion window

size for the modified TCP was most of the time higher than that of the unmodified one.

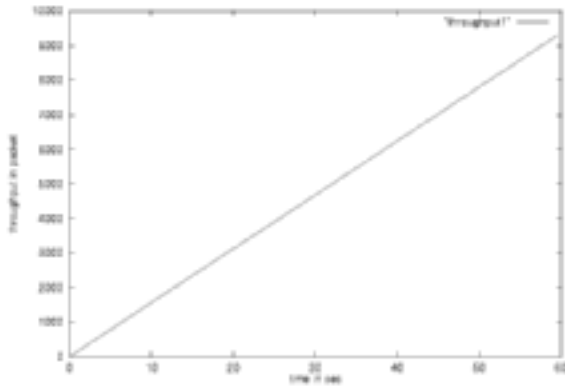


Fig. 12. Throughput of Unmodified ECT TCP

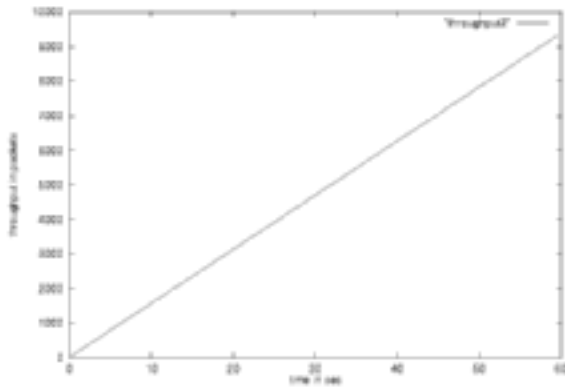


Fig. 13. Throughput of Modified ECT TCP

VI. LINK LAYER APPROACHES

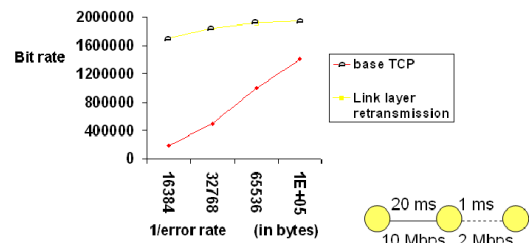
There have been many proposals for reliable link layer protocol. Two main classes of techniques which are employed by these protocols are: error correction, using techniques such as forward error correction(FEC), and retransmission of the lost packet in response to the Automatic Repeat Request(ARQ) messages[9]. So the TCP and higher layers are unaware of the local link layer transmissions. The link layer may be *semi-reliable* that is there is a bound on the maximum number of attempts that the link layer makes to retransmit a packet; or it may be totally *reliable*, that is it guarantees that all packets will be transmitted across the link[3]. The link-layer protocols for the digital cellular systems in the U.S. (both CDMA and TDMA) primarily use ARQ techniques. While the TDMA protocol guarantees reliable, in-order delivery of link-layer frames, the CDMA protocol only makes a limited attempt and leaves eventual error recovery to the (reliable) transport layer[7].

The link layer(LL) ARQ is typically a negative acknowledgment(NAK) based selective repeat scheme. The receiver does not acknowledge successfully received LL packet, but it only requests for retransmission of the packets which are not received correctly. FEC coding can be applied by adding parity bits to the data bits of the LL packet, or other error correction codes like CRC, BCH can be used[9].

The main advantage of the link layer approach is that it is independent of the higher layers, and works locally without any awareness of the higher layers, and thus follows the protocol layering guidelines.

But with TCP there are certain limitations with the LL transmissions. The LL retransmissions increase the *RTT* for the TCP sender, and thus the retransmission time out (*RTO*). This will delay the congestion response of TCP. But when the *RTT* of the wireless hop is small and the *granularity* of the *RTO* is large, then this problem is not significant. If the LL retransmission is not successful within the *RTO* of the TCP source than both the source and the sender retransmit the same data(interference), which will waste the wireless bandwidth. The LL should deliver the TCP packets in order, otherwise the dupack generated by the receiver due to out of order(OOO) delivery of the packets invokes the TCP congestion mechanism, although there is no congestion in the network.

So a reliable link layer is beneficial to TCP when it provides in-order delivery of TCP packets, and TCP's *RTO* is large enough to tolerate the additional delays due to link layer retransmissions. Also the retransmissions at the LL must persist long enough to outlast the bad state of the channel in order to achieve performance better than TCP without LL ARQ.

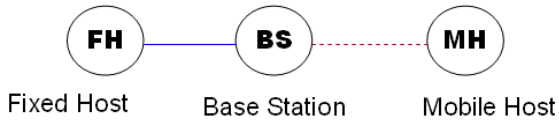


2 Mbps wireless duplex link with 1 ms delay
Exponential error model
No congestion losses

Fig. 14. Performance of Base TCP and TCP with LL Retransmissions[3]

VII. SPLIT CONNECTION APPROACH

Split connection protocols break the TCP connection between the sender and the receiver into two separate connections - one for the wired part and the other for the wireless part between the base station and the mobile host[3]. If wireless link is not last on route, then more than two TCP connections are needed. The connection between the sender and the receiver goes through the base station (BS).



$$FH-MH = FH-BS + BS-MH$$

So now there are two separate connections with different flow/error control protocols, RTT and timeouts.

One of the early proposals for split connection was *indirect* TCP(I-TCP), which uses two different standard TCP connections for the above 2 connections, which was not successful because of the basic TCP behavior[3]. In the selective repeat protocol(SRP) the FS-BS connection is the normal TCP connection, but the BS-MH connection has SRP on the top of UDP[3][7]. Although it's performance is better than I-TCP, but still studies have concluded that it obtains no significant performance improvement over I-TCP when handoff issues are taken into account[7]. A better solution would be to terminate the TCP connection at the BS and use another protocol optimized for the wireless channel for the BS-MH connection. Then the base station must guarantee the ordered delivery of packets to the mobile host.

The main advantage of split connection approach is that the wireless part of the connection can be optimized according to the wireless channel. This also ensures that there is local recovery of local errors. But the end-to-end semantics of TCP are violated. The sender of the TCP gets the ACK from the BS, even if the MH has not actually received it. The base station needs to maintain *hard state* for each TCP connection. When there is hand off, then all the hard states for the TCP connection and the buffered data must be transferred to the new BS for the MH. So the hand off becomes very complicated and slow. All the data has to be processed twice at the base station, as the packets need to be converted from one protocol semantics to another. This makes the base station very complicated. This scheme may not work if the data and ACK traverse different path(they do not go through the same BS).

VIII. TCP AWARE LINK LAYER

A. The Snoop Protocol

The snoop protocol introduces a module, called the *snoop agent*, at the base station[7]. This agent monitors every packet that passes through the TCP connection in both directions and maintains a cache of TCP segments sent across the link that have not yet been acknowledged by the receiver. A packet loss is detected by the arrival of a small number of duplicate acknowledgments from the receiver or by a local timeout. The snoop agent retransmits the lost packet if it has it cached and *suppresses* the duplicate acknowledgments.

So the link layer at the BS uses the knowledge of the higher layer transport protocol(TCP) to suppress the duplicate acknowledgments and retransmit locally the lost packets, and thereby avoiding unnecessary fast retransmission and congestion control procedures by the sender[3]. Unlike the split connection techniques, here the base station need not maintain hard

state about each TCP connection. The per connection state is *soft*; it is not essential for error free working of TCP[7]. But if the TCP header is encrypted then this method cannot be used. It cannot be used if TCP data and TCP header traverse different paths, that is they do not go through the same base station.

If wireless link level *delay-bandwidth product* is less than 4 packets, a simple (TCP-unaware) link level retransmission scheme can suffice. Since delay-bandwidth product is small, the retransmission scheme can deliver the lost packet without resulting in 3 dupacks from the TCP receiver[3].

B. Wireless TCP

WTCP works similar to snoop protocol, but it uses the *timestamp* option to estimate the RTT for the TCP packets[3]. The base station adds the *base station residence time* to the timestamp when processing an ACK from the wireless host. The sender then takes care of this while calculating the *RTT*, so that it is not affected by retransmissions on the wireless link.

C. TCP-Unaware Approximation of TCP Aware Link Layer-The Delayed Dupacks Protocol

The Delayed DUPACKs Protocol is designed to give the link layer more time to retransmit the lost or corrupted packets. When a packet is lost, then the sender delays its third and subsequent duplicate acknowledgments for an interval D [3]. So when the link layer finds a packet loss by a duplicate acknowledgment, then it gets D more time to retransmit it successfully compared to the normal TCP. So the loss is tried to get recovered from local retransmission only, and not invoking the congestion avoidance from the sender up to duration D . The link layer may actually need not be TCP aware, and this method can be a sender based scheme only. This scheme works well for networks with the wireless hop having small *RTT* as compared to the total *RTT*. But this also delays the congestion loss recovery. So choosing the appropriate value of D is also difficult.

IX. CONCLUSIONS

Several of the methods described in this report have been tested by simulations and experiments, and some of them have been implemented as well.

The split connection protocols completely hide from the sender the presence of wireless link *on route*. But many times the sender stalls due to timeouts, which decreases the throughput. In such situations, explicit notification schemes such as Explicit Bad State Notification(EBSN) become handy. The complexity at the base station in split connection approach also increases because of the change of the protocol for the packets from the wired to wireless parts and *vice versa*. The SACK schemes and the SMART-based selective acknowledgment schemes have shown better results than the split connection methods, which demonstrate that breaking the end-to-end connection is not a requirement for the achieving better throughput[7]. Selective acknowledgment schemes are very useful in presence of lossy links, especially when the losses occur in bursts.

It has been found out and can be deduced intuitively also that a reliable link layer which uses the knowledge of TCP

(TCP-aware-link-layer) to shield the sender from the losses performs better than a link layer which operates independently of TCP and does not attempt in-order delivery of packets[7]. Also the TCP-aware-link-layer avoids interference in transmissions with the transport layer retransmissions, thereby increasing the goodput.

But schemes which require the intervention of intermediate network elements violate the end-to-end semantics of the TCP protocol. Providing the transport layer the characteristics of the underlying physical and link layers does not follow the protocol stack layering rules. The link layer when made TCP aware becomes more complicated to implement.

So we look for the end-to-end schemes, which are either sender based and/or receiver based. Although these methods are not as effective in handling the local wireless losses, but are promising since significant performance gains can be achieved without any extensive support from the intermediate nodes in the network. The explicit loss notification protocols have given throughput about twice as much as that of TCP Reno, with comparable goodput values[7]. TCP Westwood has also outperformed TCP Reno when there are sporadic losses in the network, albeit assuring *fair-share* and *friendliness*[5].

So keeping in mind the performance of various schemes, an improved and reliable transport layer protocol needs to be developed, which responds well both to congestion and transmission errors satisfactorily, thus integrating the core wireless network and mobile users efficiently.

Acknowledgments

I would like to express my sincere gratitude to **Prof. Abhay Karandikar** for his invaluable guidance and immense support. More importantly, I would like to thank him for making this report on TCP over wireless networks, possible for me.

REFERENCES

- [1] K. Fall and S. Floyd. *Simulation Based Comparisons of Tahoe, Reno and Sack TCP*. Computer Communications Review, 1996.
- [2] <http://www4.ulpgc.es/tutoriales/tcpip/pru/3376fm.htm>. *TCP/IP Tutorial and Technical Overview*.
- [3] Nitin H. Vaidya. TCP for Wireless and Mobile Hosts. *MobiCom'99 Tutorial*.
- [4] S. Keshav. An Engineering Approach to Computer Networking. *AT&T Labs-Research*. Pearson Education Asia.
- [5] Mario Gerla, M. Y. Sanadidi, Ren Wang, and Andrea Zanella. *TCP Westwood: Congestion Window Control Using Bandwidth Estimation*. ULCA Computer Science Department. <http://www.cs.ucla.edu/~mvalla/tcpw/papers/TCPWGIobecomBasicPaperFinalDraft.pdf>
- [6] Rohit Ramani and Abhay Karandikar. Explicit Congestion Notification(ECN) in TCP over Wireless Networks. *IEEE, ICPWC'2000*
- [7] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, December 1997.
- [8] H. Chaskar, T. V. Lakshman, U. Madhav. TCP Over Wireless With Link Level Error Control: Analysis and Design Methodology. *MILCOM, 1996*
- [9] A. Chockalingam and Michele Zorzi. Wireless TCP Performance with Link Layer FEC/ARQ. *Communications, 1999. ICC'99. 1999 IEEE International Conference on*, 1999 page(s): 1212-1216 vol.2