Presented at IEEE ICT 2001. Bucharest, June 2001.

Study of the TCP Unfairness in a Wireless Environment[1]

Llorenç Cerdà and Olga Casals

Polytechnic University of Catalonia
Computer Architecture Department
c/ Jordi Girona, 1-3, Mòdul C6-E105, 08034 Barcelona, Spain
e-mail: {llorenc,olga}@ac.upc.es
Phone: +34 93 401 6798. Fax: +34 93 401 7055.

ABSTRACT

In this paper we analyze the TCP performance degradation introduced by a wireless media having transmission errors and the capability of the following mechanisms to correct it (i) improvements over the TCP Reno release (SACK, New-Reno and FACK); (ii) improvements over the classical "tail dropping" mechanism used by routers (RED and ECN); and (iii) a TCP implementation based on the usage of ECN routers that would be able to distinguish among losses due to congestion and losses due to transmission errors. Furthermore, we have analyzed the unfairness that occurs when TCP sources having high transmission error rates compete with sources having low transmission error rates.

I. INTRODUCTION

Wireless communications are experiencing an enormous growth and Internet applications are foreseen to be widely deployed over them. Wireless networks are characterized by hand-offs and fading intervals that deteriorate the transmission links introducing transmission errors. Therefore, the applicability of the Transmission Control Protocol (TCP) [1] used in Internet in such scenarios has become of capital interest.

TCP is a variable window protocol where losses are considered as congestion signals. Basically, TCP increases the window as acknowledgments are received and reduces the window when losses are detected. Since TCP makes no distinction between losses due to congestion and losses due to errors in the transmission links, both produce window reduction. Consequently, losses due to transmission errors may unnecessarily produce a transmission rate reduction, and therefore, TCP to perform poorly.

Balakrishnan et al. [2] have studied different mechanisms that can be used to improve the TCP performance over lossy links. These mechanisms are classified into there categories: (i) end-to-end, (ii) link-layer and (iii) split-connection.

Among end-to-end improvements are Selective Acknowledgments (SACK) [10], and New-Reno [6]. These TCP enhancements try to avoid time-outs when multiple losses occur in the same window. Link-layer solutions attempt to hide transmission errors by using retransmissions at the link layer. Split-connection solutions consist of dividing the TCP connection in two separate ones. Typically, the separation point is located at the intermediate node where a link mismatch occurs. This could be, for example, a base station. In this case a standard TCP protocol would be used in the fixed part and another optimized protocol would be used over the wireless link. However, these solutions have two major drawbacks: the TCP end-to-end semantics is broken and end-to-end usage of IP layer security (and thus IPsec) is not possible.

Other mechanisms that may be advantageous for lossy links that do not fall into the previous classification are Random Early Detection (RED) [3], [7], and Explicit Congestion Notification (ECN) [5], [12]. These mechanisms where conceived to improve the performance of the classical "tail dropping" routers used in the Internet.

In this paper we analyze the impact of a lossy wireless media on the TCP performance and the benefits of using some of the former mechanisms. Namely, we have analyzed the benefits of using New-Reno, and two implementations of SACK: a basic SACK implementation proposed in [4] that we will refer to simply as SACK, and a more complex implementation called Forward Acknowledgements (FACK) [11]. Furthermore, we analyze the unfairness that occurs when different transmission error rates are found among contending sources. We have also investigated the use of RED and ECN that show to be beneficial for this situation. Finally, to solve the drawbacks that arise when

[1] This work was partly supported by the Ministry of Education of Spain under grant TIC-1998-1115-C02-01 and by the European project "MOEBIUS".

using TCP in a wireless environment we analyze a possible solution based on ECN routers.

The rest of the paper is organized as follows. We first give an overview of TCP, New-Reno, SACK RED and ECN in sections II, III, IV and V. In section VI we describe the model of the devices and scenarios we have simulated in this paper. We then give some numerical results in section VII. Section VIII is devoted to a TCP implementation based on ECN routers. Finally, conclusions are outlined in section IX.

II. TCP OVERVIEW

In this paper the reader is assumed to be familiar with TCP and its basic window adjustment algorithms: slow start, congestion avoidance, fast retransmit and fast recovery [8], [9]. For sake of notation these algorithms are briefly summarized in the following.

A. Slow Start and Congestion Avoidance

The Slow start algorithm is used to progressively increase the window used by TCP (`allowed_wnd`) seeking for a suitable size. This consists of the following rules [8], [13]:

- Add a congestion window (`snd_cwnd`),
- the `allowed_wnd` is the minimum of the advertised window and the `snd_cwnd`,
- when starting or restarting after a loss, `snd_cwnd` is set to one segment,
- on each ack for new data, increase `snd_cwnd` by one segment.

Note that with the former algorithm the `allowed_wnd` is opened one segment each time a segment is acknowledged, thus growing exponentially towards to the advertised window. If the transmission path is not able to store the full advertised window losses will occur. The congestion avoidance algorithm tries to stabilize the window to its optimum value. The time-outs are used as loss signals. Since time-outs are also used to restart the slow start algorithm, slow start and congestion avoidance algorithms are combined in the following way:

- A slow start threshold `ssthresh` is kept to switch between slow start and congestion avoidance algorithms,
- on a time-out, half of the current `allowed_wnd` is stored in `ssthresh` (this is the multiplicative decrease) and the slow start algorithm is initiated,
- slow start algorithm is applied until the `snd_cwnd` reaches `ssthresh`. Then the `snd_cwnd` is additively increased as `snd_cwnd = snd_cwnd + 1 / snd_cwnd` on reception of each ack.

In this way the slow start opens the window quickly to what is assumed to be a safe operating point (half the window where losses were detected). Then, congestion avoidance is applied and the window is slowly increased probing for higher bandwidth to be available.

B. Fast Retransmit and Fast Recovery

The fast retransmit algorithm exploits the fact that receiving consecutive duplicate acks is a likely indication that a segment has been lost. Therefore, the segment is transmitted without waiting for a time-out. After a fast retransmit, the following actions are applied (these are referred to as *fast recovery*):

- A congestion avoidance instead of a slow start algorithm is applied. This is because the full `allowed_wnd` may be in the transmission path when the fast retransmit is performed. Thus, it is not necessary to restart with the slow start algorithm.
- A new segment is allowed to be sent upon the reception of each duplicate ack. This is because the destination TCP module only generates an ack on the reception of a segment. Therefore, each ack indicates that a segment has left the transmission path and a new segment can be sent to take its place.

The fast retransmit and fast recovery algorithms are implemented together as follows [13], [14]:

- When the third consecutive duplicate ack is received: (i) the missing segment is retransmitted (this is the *fast retransmit*), (ii) `ssthresh` is set to max(`allowed_wnd` / 2, 2), (iii) `snd_cwnd` is set to `ssthresh` plus 3 segments. This inflates `snd_cwnd` by the segments that have left the network.
- Each time a duplicate ack arrives, increment `snd_cwnd` by one segment (for the corresponding segment leaving the network).
- When the next ack that acknowledges new data is received, the `snd_cwnd` is set to `ssthresh`. This ack should be the acknowledgment of the fast retransmit. Therefore, this step performs the congestion avoidance since the rate is reduced down to one-half the value it had when the packet got lost.

III. NEW-RENO OVERVIEW

The fast retransmit and fast recovery implementation described in the previous section allows TCP to recover from a single segment loss in a window. When multiple losses occur, this implementation ends up waiting for a retransmission time-out. This can be observed in Figure 1. The figure shows: (i) the sequence number of the segments at the instant where the TCP module passes the segments to the TCP driver (as would be recorded by the `tcpdump` utility [13] in a real scenario) indicated as Tx in the figure, (ii) the acknowledgments received by the source at the reception instants (ack), and (iii) the losses at

the instant when they occur. An ack is assumed to be sent for each received segment. The interval between the transmission of the segment and the reception of the ack includes the transmission time of the segment and the propagation and queuing delays in both directions. Note that each ack shown in the figure corresponds to the sequence number of the first segment missing at the receiver. The same interpretations apply to analogous traces depicted in the rest of the paper. All these traces have been obtained with the network topology and scenarios described in section VI.

Figure 1 illustrates that the first segment lost is retransmitted after the third duplicated ack is received, but the other two segments losses lead to the timer expiration.

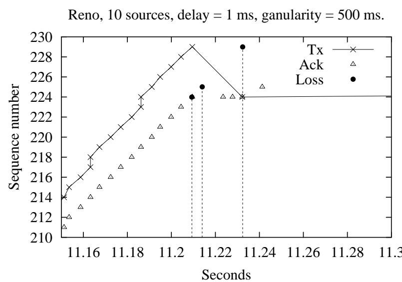

Figure 1: Example of multiple losses in a single window that are not recovered by fast retransmit and fast recovery algorithms.

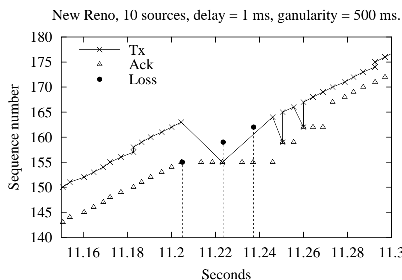

Figure 2: Example of multiple losses in a single window recovered by New-Reno.

The New-Reno improvement is based on the observation that partial acks received during the fast recovery phase are an indication of segment losses. Partial acks are defined as those confirming part but not all of the segments that were outstanding at the start of the fast recovery period. In our simulator the New-Reno modifications as defined in [6] have been implemented. These modifications consist of adding the following changes to the fast retransmit and fast recovery algorithms described in section II.B:

- When a third duplicate ack is received (and thus a fast retransmit is done), a variable called `recover` is used to record the highest sequence number of the transmitted segments.
- If an ack confirming all data ≥ `recover` is received during the fast recovery phase, the fast recovery phase is exited and the congestion window is set to: `snd_cwnd` = min(`ssthresh`, `FlightSize` + 1), where `FlightSize` is the number of outstanding data (tipically the value of `snd_cwnd` when the ack is received). Note also that in this formula we count in segments and not in bytes.
- If an ack confirming new data but less than `recover` is received during the fast recovery phase, this is a partial ack. In this case the first unacknowledged data is transmitted and `snd_cwnd` is reduced by the amount of data acknowledged. Furthermore, the retransmit timer is reset when the first partial ack is received.

Figure 2 shows a trace of a New-Reno TCP source having multiple losses in a window. The figure shows that the retransmission of the partial acks allows the TCP sender to recover from the losses, and thus, avoiding the retransmission time-out.

IV. SACK OVERVIEW

With the cumulative acknowledgements used by TCP, the sender has little information to take retransmission decisions in case of multiple losses in a single window. To solve this drawback a Selective Acknowledgment (SACK) option for TCP was proposed [10]. Using SACK the TCP receiver informs about out of order segments that have been correctly received. Actually, the New-Reno improvement described in section III was conceived afterwards for those implementations that are unable to use SACK. This was motivated by the fact that New-Reno is easier to implement than SACK, and moreover, with New-Reno only the TCP sender needs to be modified, while SACK needs modification of both the TCP sender and receiver.

The RFC [10] describes the mechanisms to be used by the TCP sender and receiver to agree upon the usage of SACK option and the way by which the TCP receiver communicates the reception of correct segments received out of order. Using SACK the cumulative meaning of the ack number does not change. Additionally, the reception of out of order segments correctly received is communicated by specifying the edges of blocks of data correctly received that are contiguous and isolated. The space available in the TCP header to accommodate the options allows three of such blocks to be specified. Upon reception of SACK blocks the TCP sender may turn on a bit of the segments outstanding in the retransmission queue indicating that they have been correctly received. Therefore, when a retransmission is to be done, an unmarked segment is taken.

Although a detailed explanation of the SACK indications is given in [10], no description of the TCP sender behavior is proposed. In our simulator we have used two implementation of SACK. The first one is the simplest and was proposed in [4].

We shall refer to this as SACK. The second one [11] further exploits the information conveyed by the SACK option and we shall use the name coined by the authors of Forward Acknowledgements (FACK). In the following we give a brief description of these proposals.

The SACK implementation proposed in [4] consists of the following modifications to the fast retransmit and fast recovery algorithms described in section II:

- The TCP sender uses a variable called `pipe` to estimate the number of segments outstanding in the path during the fast recovery phase. When the fast retransmit is performed, `pipe` is initialized to `snd_cwnd-3` and then `snd_cwnd` is reduced by one half. Furthermore, a variable called `recover` is used to record the highest sequence number of the transmitted segments.
- The variable `pipe` is reduced each time an ack is received and is increased each time a segment is transmitted. Furthermore, if a partial ack is received (this is an ack confirming new data, but less than `recover`) `pipe` is decreased by 2.
- When the TCP sender receives an ack it is allowed to transmit segments while the condition `pipe < snd_cwnd` is satisfied.
- Packets outstanding in the retransmit list that are confirmed by the SACK options or retransmitted are marked.
- When the TCP sender is allowed to send a segment and there are blocks of non-consecutive marked segments in the retransmit list, then the first unmarked segment is retransmitted. Otherwise a new segment is transmitted.
- The fast recovery phase is exited when an ack confirming all data ≥ `recover` is received.

Figure 3 shows a trace where multiple losses in a window are recovered using SACK. Note that during the fast recovery phase the TCP sender is able to quickly retransmit the lost segments due to the SACK information conveyed by the acks.

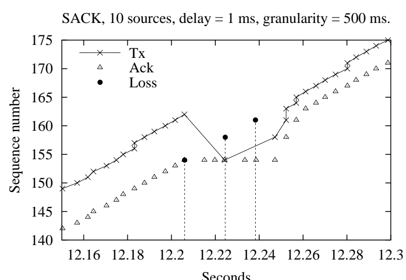

Figure 3: Example of multiple losses in a single window recovered by SACK.

The FACK implementation proposed in [11] adds several refinements to the basic SACK implementation previously described. For example, FACK allows the recovery of a lost retransmission. This implementation consists of introducing the following modifications to the fast retransmit and fast recovery algorithms described in section II:

- The TCP sender uses a variable called `snd_fack` that stores the highest sequence number acknowledged by the SACK option conveyed by the TCP header.
- The fast retransmit is triggered when three duplicate acks are received or a reassembly queue is estimated to be longer than three segments. The later condition is given by `snd_fack - snd_una > 3`.
- Another variable called `retran_data` is used to store the number of retransmitted segments during the fast recovery phase. This variable is increased each time a segment is retransmitted and decreased whenever a retransmitted segment is determined to have left the network.
- Furthermore, the current value of `snd_nxt` is stored for each retransmitted segment. These values allow recognizing retransmitted segments that are lost when the `snd_fack` is updated beyond the value stored for any retransmission.
- During the fast recovery phase, retransmitted segments that are determined to be lost or unmarked segments in the sack list are retransmitted as explained in the previous SACK implementation.
- When the fast retransmit is performed `snd_cwnd` is reduced by one half and is maintained constant during fast recovery. The amount of data outstanding in the network during fast recovery is estimated by:
  `awnd = snd_next – snd_fack + retran_data`
  Therefore, segments are allowed to be (re)transmitted during fast recovery while the condition `awnd < snd_cwnd` is fullfilled.

V. RED AND ECN OVERVIEW

Traditionally, routers in the Internet have implemented a simple FIFO queue with a "tail dropping" management mechanism. Tail dropping consists of discarding incoming segments when no more room is available at the queue. This simple mechanism may lead to the following drawbacks:

- Unfairness: some unbalanced situations among contending TCP sources may lead to some sources monopolizing the queue space. For example, sources with shorter round trip delays are more favored than sources having higher delays.
- High end-to-end delays: tail dropping tends to maintain the queues full and thus with high end-to-end delays. Furthermore, synchronization effects may grow leading to periods of high and low transmission rate where the queue is respectively built up and drained. In this case end-to-end delays may have strong oscillations.

To solve these problems an "active queue management" has been recommended [3]. This consists of dropping packets at the routers before buffer overflows according to some algorithm which tries to maintain the queue around a low threshold. One of these algorithm is the Random Early Discard (RED) [7].

The flow chart of Figure 4 summarizes the RED algorithm. First an exponential weighted moving average of the queue length is computed (the variable `avg`). Then the average is compared with two thresholds (`HighTh` and `LowTh`). When the average is between both thresholds, the packet is "marked" with a probability `Prob`. If the average is higher than `HighTh` the packet is always "marked" and if lower than `LowTh` the packet is always accepted.

At each paket arrival

```
if(q > 0) {
  avg = avg + wp * (q –avg) ;
} else {
  avg = (1 – wq)^(t – tq) / s * avg ;
}
if(LowTh <= avg && avg < HighTh) {
  count++ ;
  with probability Prob {
    Mark the packet ;
    Count = 0 ;
  }
} else if(HighTh <= avg) {
  Mark the packet ;
  Count = 0 ;
} else { // lower than LowTh
  count = -1 ;
}
```

Where:

`avg`: Queue average. `q`: Queue length.
`wq`: weight. `t`: current instant.
`LowTh`: Low threshold. `HighTh`: High threshold.
`tq`: instant when the queue became empty.

$$\mathrm{Prob} = \frac{\mathrm{pb}}{1-\mathrm{count}*\mathrm{pb}}, \mathrm{pb} = \frac{\mathrm{maxp}*(\mathrm{avg}-\mathrm{LowTh})}{\mathrm{HighTh}-\mathrm{LowTh}}.$$

Figure 4: RED algorithm.

The packet "marking" previously mentioned may consist of (i) discarding the packet, or (ii) setting the bit in the packet header. Obviously, the bit-setting mechanism requires that the congestion control algorithm reacts to this bit indication in a similar way TCP reacts to packet drops. Setting a bit instead of dropping the packet has advantages like avoiding the retransmission delay of the dropped packets and increasing the efficiency.

However, the current TCP standard does not have support for such bit indication. Thus, the Explicit Congestion Notification (ECN) [12] is a proposal to add this capability to TCP. In the following this proposal is briefly described.

- *TCP initialization*. ECN requires that both TCP sender and receiver collaborate to support the mechanism. Therefore, a TCP option is used during the TCP initialization to inform whether each side is ECN capable.
- *The IP packets*. Two bits are required to be used in the IP header: (i) an ECN-Capable Transport (ECT) bit, to indicate that the IP packet can be marked instead of discarded; and (ii) a Congestion Experienced (CE) bit which is the bit used by the router for packet marking.
- The TCP segments. Two bits are required to be used in the TCP header: (i) an ECN-Echo flag used to indicate that congestion was notified by means of the CE bit at the IP level; and (ii) a Congestion Window Reduced (CWR) bit used to inform that reaction to a congestion notification has already taken place.
- *TCP receiver*. Upon reception of a TCP segment conveyed by an IP packet with the CE bit set, the TCP acks are transmitted with the ECN-Echo flag set. This flag is set until a segment with the CWR bit set is received.
- *TCP sender*. Upon reception of a segment with the ECN-Echo flag set, the TCP sender must react as a packet loss was detected. Precautions are to be taken not to react to congestion more that once per window. When the TCP reduces its congestion window (either because of a timeout, a fast retransmit or upon reception of an ECN-Echo flag set) the next TCP segment is transmitted with the CWR bit set.

VI. SIMULATION MODEL DESCRIPTION

The simulator used to obtain the results shown in this paper is an event driven simulator written in C++. In the following the assumptions made for the simulation of the TCP module and the network topology are described.

A. The TCP Module

We assume a greedy TCP module (which has always segments ready to send). Our TCP module passes the maximum number of segments allowed by the TCP window to the network driver. The segments are immediately given to the driver after the TCP module initialization, when an ack allows the transmission of more segments, or when a time-out expires.

For the network driver we have assumed an infinite queue (i.e. with no losses) which stores the segments received by the TCP module until they are sent into the transmission link. However, we have considered the buffer occupancy at the TCP receiver due to segment reordering. Therefore, in the simulations the TCP receiver always advertises a window equal to the free buffer space. We have also assumed that the TCP receiver sends an ack for each received segment. A realistic simulation is done of the slow timer used by TCP for the segments retransmission control (see [15]). In fact, the slow timer function is called each time interval

equal to the granularity used by the TCP module as in a real implementation.

For sake of simplicity, we have not considered queuing delays but only the propagation delay in the return path of the acks. However, in order to avoid phase effects we have added a small random component in the return path delay of the acks.

The sources run the different TCP implementations with the following parameters:

- Segments of 576 bytes (this is the default value used by TCP if no MSS is indicated at the connection setup).
- Buffer space at the TCP receiver equal to 50 segments.
- The timestamp option is used for the round trip measurements (see [13]).
- We use the retransmission time-out mechanism described in [8].

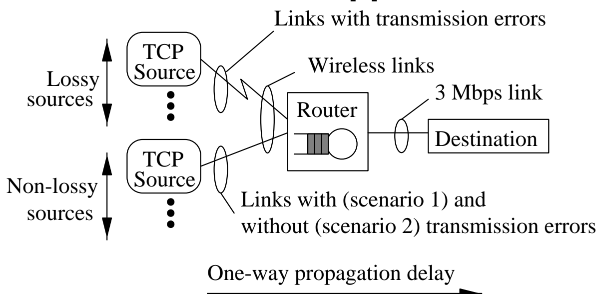

Figure 5: Simulation scenarios.

B. Simulation Topology and Scenarios

Figure 5 shows the network topology we have simulated in this paper. It consists of 20 TCP sources that compete for a congested link at the router output. The transmission links of the sources are 1 Mbps (we shall refer to these links as the wireless links), and the congested link at the router output is 3 Mbps.

We have considered two scenarios:

- ***Lossy* scenario**: all the wireless links have the same transmission error rate,
- ***half-lossy* scenario**: only half of the wireless links have transmission errors. We shall call *lossy* sources the sources having a transmission link with transmission errors and *non-lossy* otherwise.

In the simulation the errors have been introduced only in the forward direction, i.e. no acks are lost due to transmission errors. We have used a Bernoulli distribution in order to generate these transmission errors (i.e. each segment is lost due to transmission errors with probability *loss_p* and properly transmitted with probability 1 – *loss_p*). Several simulations have been carried out for each of these scenarios changing the behavior of the TCP sources and the router as follows:

- **TCP sources**: (i) standard TCP-Reno described in section II, (ii) TCP with the New-Reno improvement described in section III, (iii) TCP with the SACK implementation described in section IV and (iv) TCP with the FACK implementation described in section IV.
- **Router**: (i) "tail dropping", (ii) with the RED discarding policy and (iii) with the ECN-RED marking mechanism described in section III.

Finally, simulations have been done varying the following parameters:

- **Number of sources**: this has been set to (i) 10 sources (5 *lossy* and 5 *non-lossy* in scenario 2) and (ii) 40 sources (20 *lossy* and 20 *non-lossy* in scenario 2),
- **end-to-end propagation delay**: this has been set to (i) 1 ms and (ii) 20 ms,
- **TCP granularity**: this has been set to (i) 50 ms and (ii) 500 ms.

The buffer size of the router has been fixed to 150 segments in all cases. RED and ECN-RED have been implemented with the following parameters: `LowTh` = 20 segments, `HighTh` = 60 segments, `wq` = 0.5, `maxp` = 0.02.

VII. NUMERICAL RESULTS

Each of the graphs of Figure 6 and Figure 7 shows the results obtained for the lossy and half-lossy scenarios described in section VI.B with a different set of parameters. What we call *gain* in the graphs is given by the average goodput obtained for each group of sources having the same ratio of transmission errors to the fair goodput (the link rate divided by the number of sources) multiplied by 100. We compute the goodput of one source as the sequence number increment achieved during the simulation multiplied by the segment size in bits divided the simulation time. Remember from section VI.B that in the lossy scenario all the sources have the same segment loss probability due to transmission errors and in the half-lossy scenario only half of the sources have transmission errors. This loss probability is given in the abscissa and we shall refer to it as *loss_p*. Therefore, only one point is depicted in the graph for each simulation with the lossy scenario, and two points corresponding to the group of sources having and not having transmission errors are depicted in the half-lossy scenario. Note that these points are easy to identify: (i) those in the middle of the graph correspond to the lossy scenario and (ii) those in the top and the bottom of the graphs respectively correspond to the lossy and non-lossy sources of the half-lossy scenario.

Note also that in Figure 6 the curves obtained for each of the three types of routers (tail dropping, RED and ECN) have been superimposed for each scenario. The TCP implementation used for these graphs is FACK (see section IV). Instead, in Figure 7 the curves obtained for each of the TCP implementations (Reno, New-Reno, SACK and

FACK) have been superimposed. In the following some general guidelines are derived from these graphs.

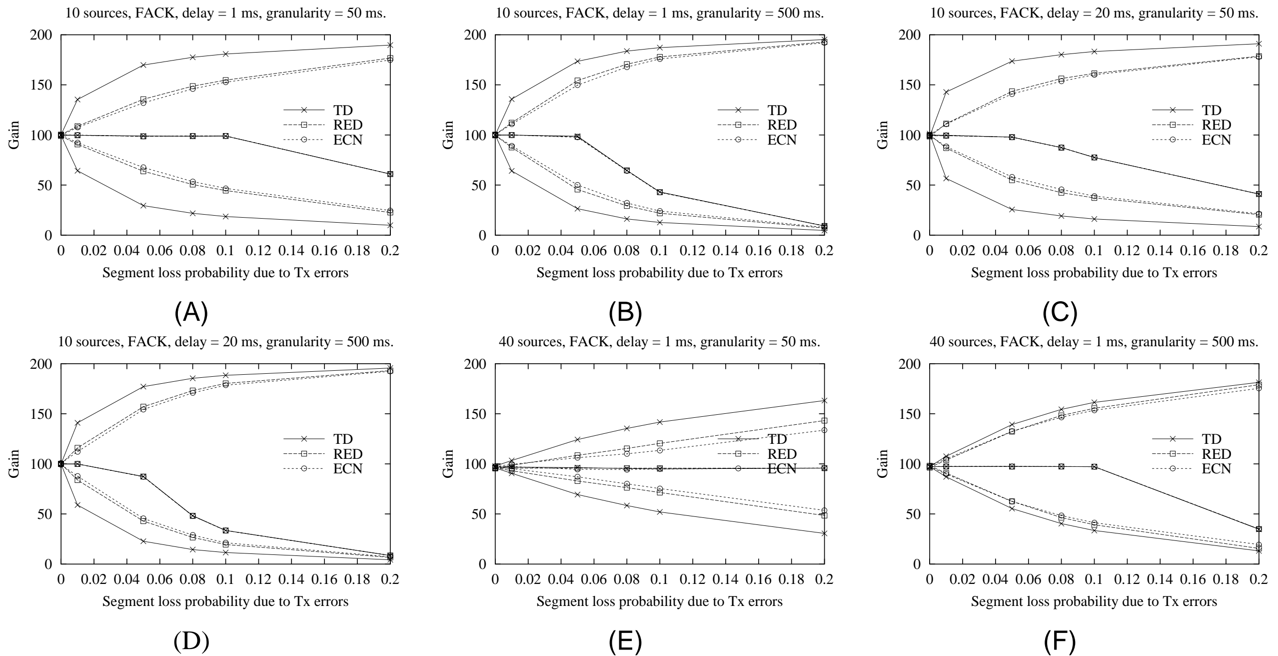

Figure 6: Gain obtained in the lossy and half-lossy scenarios varying the number of sources, delay and granularity using a tail dropping (TD), RED and ECN router.

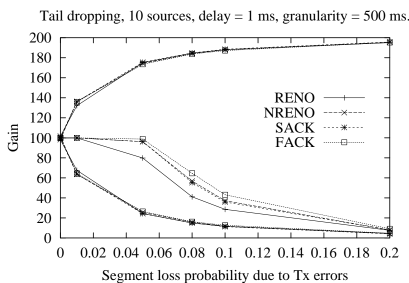

Figure 7: Gain obtained in the lossy and half-lossy scenarios using different TCP implementations.

- **Unfairness**: the first effect that becomes apparent from the graphs of Figure 6 arises when comparing the results obtained with the lossy and half-lossy scenarios. In all cases the degradation of the gain obtained by the sources having losses due to transmission errors is higher when the congested link is to be shared with sources having no transmission losses. For example, graph (A) shows that only for values of *loss_p* higher than 0.1 the gain starts to degrade in the lossy scenario. Instead, the gain of sources having transmission errors starts to degrade for small values of *loss_p* in the half-lossy scenario (e.g. using the “tail dropping” router this is only 20% when *loss_p* is equal to 0.1). This effect can be seen as an unfairness behavior of TCP when different transmission error rates occur among the sources, since the sources without transmission errors have the tendency to lock-out the sources with transmission errors.
- **RED and ECN advantage over tail dropping routers**: Figure 6 shows that the advantage of using RED and ECN arises in the half-lossy scenario. In this case, the unfairness effect described in the previous paragraph is reduced. For example, graph (A) shows that when *loss_p* is equal to 0.1 the gain of sources having transmission errors in the half-lossy scenario is around 45% when RED and ECN is used, while it is only 20% when using tail-dropping. These graphs show also that this fairness benefit of using RED and ECN depends on the delay and the granularity. In fact it can be observed that the higher the granularity, the lower the benefit of using RED and ECN in terms of fairness. Furthermore, the lower is the RTT, the higher is the influence of the granularity. Note that there are other differences between the router algorithms analyzed that cannot be derived from Figure 6. For example, using tail dropping the queue length has stronger oscillations than using RED. Therefore, the end-to-end transmission delays and their variance are reduced using RED and ECN-RED.
- **Granularity**: the influence of this parameter is twofold since it determines the accuracy in the RTT measurement and the coarse of time-outs. Comparing the graphs (A) and (B) of Figure 6 it can be seen that a higher value of the granularity not only reduces the fairness benefit of RED and ECN-RED as explained in the previous paragraph, but also increases the influence of *loss_p* on the gain.
- **Delay**: comparing the graphs (A) and (C) of Figure 6 it can be observed that the higher is the RTT, the higher is the influence of *loss_p* on the gain. However, graphs (E) shows that

this effect is reduced when increasing the number of sources.

- **Number of sources**: comparing the graphs (A)-(B) respectively with the graphs (E)-(F) it can be observed that the higher is the number of sources, the lower is the reduction of the gain when *loss_p* increases. This is logical since the higher is the number of sources the lower is the average rate obtained for each of them, consequently, the higher is the ratio of discarded (or marked) to transmitted segments by the router in order to adjust the transmission rate governed by TCP. Therefore, the higher loss rate due to transmission errors is needed to interfere with segments discarded (or marked) by the router.
- **Reno, New-Reno, SACK and FACK**: Figure 7 depicts the gain obtained with each of these TCP implementations in the lossy and half-lossy scenarios using the same parameters than those of graph (B) of Figure 6. The traces obtained using the other sets of parameters of the graphs of Figure 6 are not shown because similar conclusions that those given in the following where obtained. The first conclusion that can be derived from Figure 7 is that in the half-lossy scenario, nearly the same result is obtained regardless of the TCP implementation. Therefore, these TCP implementations are not effective to solve the unfairness problem described in the previous paragraphs. Furthermore, from Figure 7 it can be observed that the influence of the granularity is even higher than the use of a refined TCP implementation. This can be explained since the most important benefit of better TCP implementations is reducing the number of time-outs, but the impact of time-outs is only predominant when the value of the granularity is much higher than the RTT. Finally, from Figure 7 the following conclusions can be derived: (i) Reno is clearly outperformed by the other TCP implementations, (ii) New-Reno achieves the same performance as the basic SACK implementation (that one we refer to as SACK), (iii) using the more complex TCP implementation (FACK) does not represent a significant improvement over the much simpler New-Reno implementation.

VIII. SOLUTION BASED ON ECN ROUTERS

In the previous section we have seen that the main problem that losses due to transmission errors produce on TCP are performance degradation and unfairness when the loss rate is not the same for all sources. These drawbacks are a consequence of the fact that TCP makes no distinction between losses due to congestion and losses due to transmission errors. In this section we investigate the behaviour of TCP in a hypothetical scenario where this distinction would be possible. In order to achieve this goal we have made the following assumptions: (i) all routers are ECN capable, (ii) routers mark the segments for congestion control and only discard them due to buffer overflow. Furthermore, we have modified the FACK TCP described in section IV as follows (we shall call this implementation ECN-FACK): Upon ack reception (regardless it is a duplicated ack or not) the congestion window is updated as shown in the flowchart of Figure 8.

```
if(the segment is ECN marked &&
  time > ecn_backoff && snd_una > ecn_seq)
{
    ecn_backoff = time + t_srtt;
    ecn_seq = snd_max;
    ssthresh = min(snd_cwnd,snd_wnd)/2;
    snd_cwnd = ssthresh ;
  } else {
    if(snd_cwnd >= ssthresh) /* C.A. */
      snd_cwnd = snd_cwnd + 1/snd_cwnd;
    else /* Slow Start */
      snd_cwnd = snd_cwnd + 1 ;
  }
```

Figure 8: ECN-FACK congestion window (`snd_cwnd`) processing upon ack reception

The `ecn_backoff` and `ecn_seq` variables shown in this figure are used to prevent from reacting to ECN indications more that once per window. The variable `t_srtt` is the current RTT measurement kept by TCP. Note from Figure 8 that `ssthresh` and `snd_cwnd` are set to the effective window divided by two as a reaction to an ECN indication, otherwise the classical slow start and congestion avoidance algorithms are applied. This implementation only reduces the congestion window upon reception of ECN indication (as shown in Figure 8) or when a retransmission time out occurs. When a retransmitted segment is found to be lost, or when there are "holes" in the sack list ECN-FACK retransmits a segment, otherwise a new segment is transmitted. Segments are transmitted while the condition `snd_max – snd_fack + retran_data < snd_cwnd` is fullfilled. Remember from section IV that `snd_fack` stores the highest sequence number acknowledged by the SACK option conveyed by the TCP header and `retran_data` stores the number of in flight retransmitted segments.

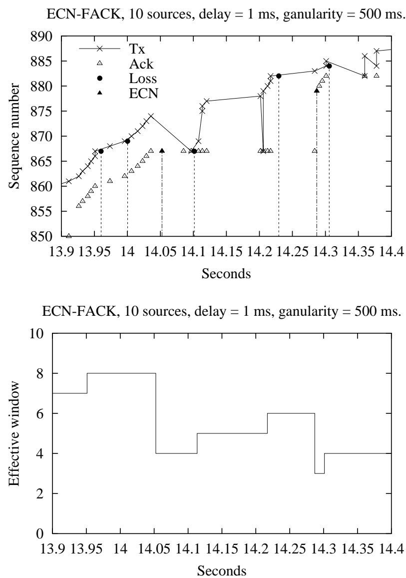

Figure 9: ECN-FACK behavior.

Figure 9 depicts a trace showing the ECN-FACK behaviour. The upper graph of this figure shows the evolution of the transmitted segments, acks, losses due to transmission errors and ECN adjustments. The lower graph shows the evolution of the effective window during the same time interval. These graphs show how ECN-FACK retransmits all lost segments reducing the transmission window only when ECN indications are received.

Figure 10 shows the benefits of ECN-FACK. The graph has been obtained with the same simulation parameters than the graph (B) of Figure 6. This figures show that the performance degradation of TCP due to transmission errors is avoided by ECN-FACK. Furthermore, the unfairness problem due to different transmission error rates among competing sources is minimized.

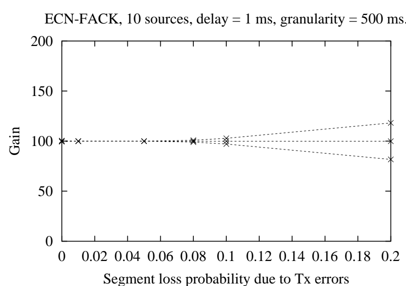

Figure 10: Gain obtained in the lossy and half-lossy scenarios using ECN-FACK.

IX. CONCLUSIONS

Several mechanisms that may be of interest to improve the TCP degradation in wireless environments are described and analysed by simulation. Three of these mechanisms (NewReno, SACK, FACK) are modifications to the TCP protocol that try to improve the TCP response when multiple segments are lost within a window. The other two mechanisms we analyse (RED and ECN) are modifications that have been proposed to improve the classical tail dropping management of router buffers.

The simulation results demonstrate the ability of New-Reno, SACK and FACK to mitigate the performance degradation due to transmission errors. However, nearly the same results are obtained with New-Reno and a basic SACK inplementation, although SACK is more complex and more detailed information about losses is conveyed to the TCP sender using SACK. Only FACK, a more sophisticated implementation of SACK, is able to outperform New-Reno.

The simulation results also show that in a scenario with sources having different transmission error rates, sources with lower errors have the tendency to lockout the sources with higher transmission errors. Using RED or ECN at the base station alleviates this drawback.

Finally, we have shown that the previous drawbacks can be solved in a scenario having all routers ECN capable. In this scenario the congestion control of routers is made by means of ECN marking, and thus, TCP can asume that losses are due to transmission errors.

REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens. "TCP Congestion Control". RFC 2581, April 1999.

[2] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz. "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links". In Proc. of the SIGCOMM'96, 1996.

[3] B. Braden et al. "Recommendations on Queue Management and Congestion Avoidance in the Internet". RFC 2309, April 1998.

[4] K. Fall and S. Floyd. "Simulation-based Comparisons of Tahoe, Reno and SACK TCP". ACM Computer Communication Review, July 1996. ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z

[5] S. Floyd. "TCP and Explicit Congestion Notification". ACM Computer Communication Review, 24(5), October 1994.

[6] S. Floyd and T. Henderson. "The NewReno Modification to TCP's Fast Recovery Algorithm". RFC 2582, April 1999.

[7] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance". IEEE Transactions on Networking, August 1993.

[8] V. Jacobson. "Congestion Avoidance and Control". ACM Computer Communication Review, 18(4), 314-329, August 1988. ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z

[9] V. Jacobson. "Modified TCP Congestion Avoidance Algorithm". end2end-interest

mailing list, April 1990. ftp://ftp.isi.edu/end-2end/end2end-interest-1990.mail

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. "TCP Selective Acknowledgment Options". RFC 2018, October 1996.

[11] M. Mathis and J. Mahdavi, S. Floyd. "Forward Acknowledgement: Refining TCP Congestion Control". ACM Computer Communication Review, 26(4), October 1996.

[12] K. Ramakrishnan and S. Floyd. "A proposal to add Explicit Congestion Notificacion (ECN) to IP". RFC 2481, January 1999.

[13] W.R. Stevens. "TCP/IP Illustrated, Volume 1: The Protocols". Addison-Wesley, 1994.

[14] W.R. Stevens. "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms". RFC 2001, January 1997.

[15] G.R. Wright and W.R. Stevens. *TCP/IP Illustrated, Volume 2, the implementation.* Ed: Addison-Wesley, 1995.