TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks

Ian F. Akyildiz, *Fellow, IEEE*, Giacomo Morabito, and Sergio Palazzo, *Member, IEEE*

***Abstract*—Current TCP protocols have lower throughput performance in satellite networks mainly due to the effects of long propagation delays and high link error rates. In this paper, a new congestion control scheme called TCP-Peach is introduced for satellite networks. TCP-Peach is composed of two new algorithms, namely Sudden Start and Rapid Recovery, as well as the two traditional TCP algorithms, Congestion Avoidance and Fast Retransmit. The new algorithms are based on the novel concept of using dummy segments to probe the availability of network resources without carrying any new information to the sender. Dummy segments are treated as low-priority segments and accordingly they do not effect the delivery of actual data traffic. Simulation experiments show that TCP-Peach outperforms other TCP schemes for satellite networks in terms of goodput. It also provides a fair share of network resources.**

***Index Terms*—Congestion control, high bit error rates, long propagation delays, satellite networks, TCP protocols.**

I. Introduction

BOTH experimental and analytical studies [30] confirm that the current TCP protocols have performance problems in networks with long propagation delays and relatively high link error rates such as satellite networks [37], [33], [6]. From the view of TCP, the throughput is reciprocal to the *round-trip time* (RTT) of a connection, and is approximately proportional to the *congestion window* ($cwnd$) which represents the amount of unacknowledged data the sender can have in transit to the receiver [39].

In satellite networks, TCP throughput decreases because [37], [33], [6]:

- the long propagation delays cause longer duration of the *Slow Start* phase during which the sender may not use the available bandwidth;
- the TCP protocol was initially designed to work in networks with low link error rates, i.e., all segment losses were mostly due to network congestions. As a result, the sender decreases its transmission rate each time a segment loss is detected. This causes unnecessary throughput degradation if segment losses occur due to link errors, as it is likely in satellite networks.

Manuscript received August 1, 2000; revised December 27, 2000; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Vaduvur. The work of I. F. Akyildiz and G. Morabito was supported by NASA-Ames under Contract NAG 2-1262.

I. F. Akyildiz and G. Morabito are with the Broadband and Wireless Networking Laboratory, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: ian@ee.gatech.edu; giacomo@ee.gatech.edu).

S. Palazzo is with Dip. di Informatica e Telecomunicazioni, University of Catania, 95125 Catania, Italy (e-mail: palazzo@iit.unict.it).

Publisher Item Identifier S 1063-6692(01)04730-6.

In [5], the TCP standard mechanisms are identified which provide the best performance in satellite networks. However, to our knowledge, these identified problems are still not solved to date [37].

In this paper, we introduce TCP-Peach, a new congestion control scheme for satellite networks which is an end-to-end solution to improve the throughput performance in satellite networks.

The paper is organized as follows. In Section II, we present the problems of TCP in satellite networks and the related work. We introduce TCP-Peach in Section III and describe its behavior in Section IV. In Section V, we evaluate the performance of TCP-Peach through simulation. Finally, in Section VI, we conclude the paper.

II. TCP Issues in Satellite Networks and Related Work

A. Slow Start Issues in Satellite Networks and Related Work

In the beginning of a new connection, the sender executes the Slow Start algorithm to probe the availability of bandwidth along the path [27]. The time required by the Slow Start to reach a bit rate B is [37]

$$t_{\text{Slow Start}} = \text{RTT} \cdot (1 + \log_2 B \cdot \text{RTT}/l) \qquad (1)$$

where RTT is the round-trip time and l is the average packet length expressed in bits. Equation (1) is satisfied if the *Delayed ACK Option* [13] is not implemented, i.e., the receiver sends one acknowledgment (ACK) for each received segment. In Table I, we give the duration of the Slow Start phase for different types of satellites, i.e., Low Earth Orbit (LEO), Medium Earth Orbit (MEO) and Geosynchronous Earth Orbit (GEO) satellites, and for different values of B, when $l = 1$ kB, which is a common value for segment size.

If the delayed ACK mechanism [13] is implemented, i.e., the receiver sends one ACK for each two received segments, then the time required by the Slow Start to reach the bit rate B becomes even higher than indicated in Table I. For the sake of simplicity, in the following we assume that the delayed ACK mechanism [13] is not implemented.

Many actual TCP applications, like HTTP, are based on the transfer of small files. Thus, it can happen that the entire transfer occurs within the Slow Start phase. In other words, it is possible that a TCP connection is not able to utilize all available resources in the network.

To cope with the performance problems of the Slow Start algorithm in long propagation delay networks such as satellite networks, there have been several proposed solutions in recent years.

1063–6692/01$10.00 © 2001 IEEE

TABLE I
DURATION OF THE SLOW START PHASE FOR LEO, MEO, AND GEO SATELLITES

Satellite Type	RTT	$t_{SlowStart}$ (B=1Mb/sec)	$t_{SlowStart}$ (B=10Mb/sec)	$t_{SlowStart}$ (B=155Mb/sec)
LEO	50 msec	0.18 sec	0.35 sec	0.55 sec
MEO	250 mesc	1.49 sec	2.32 sec	3.31 sec
GEO	550 msec	3.91 sec	5.73 sec	7.91 sec

- **Increasing Initial Window (IIW)** [4]. The congestion window $cwnd$[1] is initially set to a value larger than 1 but lower than 4, i.e., $1 \leq cwnd \leq 4$. With this option, $t_{\text{Slow Start}}$ values reported in Table I can be reduced by up to $(3 \cdot RTT)$ which can still be very high.
- **TCP Spoofing** [29], [10]. A router near the source sends back ACKs for TCP segments in order to give the source the illusion of a short delay path. TCP spoofing improves throughput performance but has some problems [37]:
  Problem 1: The router must do a considerable amount of work because it becomes responsible for the correct delivery of the TCP segments it acknowledges to the source.
  Problem 2: Spoofing requires ACKs to flow through the same path as data. On the contrary, in Internet it is very common that ACKs flow through a different path than data.
  Problem 3: If the path changes or the router crashes, data may get lost.
  Problem 4: If IP encryption is used, the scheme cannot be applied.
- **Cascading TCP** or **Split TCP** [7]. TCP connection is divided into multiple connections. This solution has the same problems as TCP spoofing with the exception of Problem 2 [37].
- **Fast Start** [36]. The Fast Start algorithm, alternative to the Slow Start algorithm, is introduced for Web transfers in [36]. The basic idea of the Fast Start is to reuse the values of the transmission rate from the recent past. However, the transmission rate used in the past might be too high for the current actual network condition, which may lead to congestion in the network. Thus, the TCP segments transmitted during this Fast Start period are carried by low-priority IP packets so that the throughput of actual data segments treated as high-priority segments will not be decreased. Note that one of the eight bits of the *Type of Service* (TOS) field—now renamed *Differentiated Service* (DS) field—in the IP header specifies the priority of the packet [38], whereas more recent IP implementations aimed to support the *Differentiated Service Model* (DiffServ) can define several priority levels. Experiments in [36] show the effectiveness of the Fast Start algorithm compared to Slow Start. However, the Fast Start has the following problems:
  Problem 1: The transmitted low-priority segments carry new information to the receiver, thus, they are still data segments, and if they are lost, then they must be recovered. Since these low-priority data segments may be lost easily, the sender needs to enhance its recovery algorithms [36].
  Problem 2: Fast Start can be used only if a recent value of the congestion window for the same path is available at the sender. This requires that within a short time the same server (sender) transfers several files to the same user (receiver), which may often not be the case.

[1]In the following, we consider the *segment* as the unit of data.

B. Error Rate Issues in Satellite Networks and Related Work

TCP was initially developed for wireline networks where the link error rate is low, such that the majority of the segment losses is due to network congestions. Thus, the sender assumes that all segment losses are caused by congestions and accordingly it decreases its transmission rate.

Although the application of *forward error correction* (FEC) algorithms can increase the reliability of satellite links, satellite networks have several orders of magnitude higher error rates than the wireline networks [2], [37]. As a result, we cannot ignore the errors in satellite links and assume that all segment losses occur due to congestions. This assumption may lead to drastic and unnecessary decrease in resource utilization [2], [8], [9], [16], [18], [20], [24], [37].

This problem could be solved if TCP could distinguish whether segment losses occur due to network congestion or due to link errors [20]. However, this is currently infeasible [37].

In [3], the authors suggest to decouple error and congestion control. TCP would then be responsible only for congestion control while the error control is handled by the link layer. However, this solution is impractical because the link layers of all subnetworks composing the Internet need to be redesigned.

An alternative solution is that the sender could contain an algorithm which can distinguish between congestion and errors. However, such an algorithm must be very reliable. In fact, if this algorithm does not respond correctly to an actual network congestion, the network utilization decreases drastically [37]. To our knowledge, such a reliable algorithm does not exist to date.

III. TCP-PEACH

TCP-Peach contains the following algorithms: *Sudden Start*, *Congestion Avoidance*, *Fast Retransmit*, and *Rapid Recovery*, as highlighted in Fig. 1. The Congestion Avoidance and Fast Retransmit algorithms may be those proposed either in TCP-Reno [28] or by TCP-Vegas [14], [15], [1]. Sudden Start and Rapid Recovery are the new algorithms and are presented in Sections III-B and III-C, respectively. The new algorithms

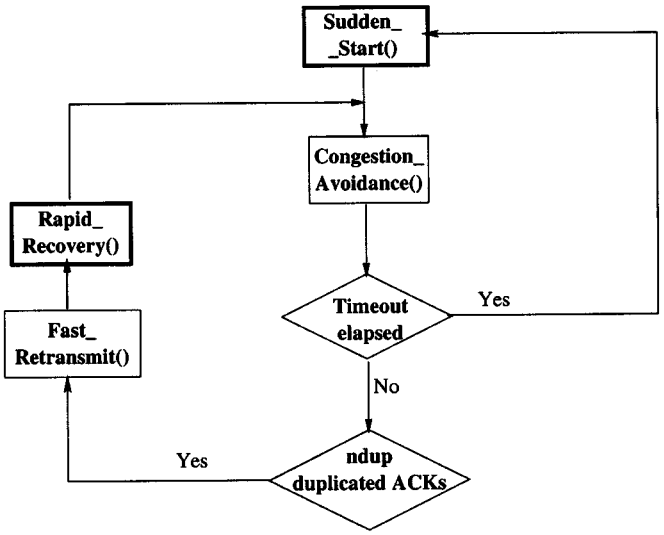

Fig. 1. TCP-Peach scheme.

are based on the use of *dummy segments*, which are explained in Section III-A.

A. Dummy Segments

Dummy segments are low-priority segments generated by the sender as a copy of the last transmitted data segment, i.e., they do not carry any new information to the receiver.

The sender uses the dummy segments to probe the availability of network resources. If a router on the connection path is congested, then it discards the IP packets carrying dummy segments first. Consequently, the transmission of dummy segments does not cause a throughput decrease of actual *data segments*, i.e., the traditional segments. If the routers are not congested, then the dummy segments can reach the receiver. The sender sets one or more of the six unused bits in the TCP header to distinguish dummy segments from data segments. Therefore, the receiver can recognize the dummy segments and acknowledge them to the sender. The ACKs for dummy segments are also marked using one or more of the six unused bits of the TCP header and are carried by low-priority IP segments. This requires simple modification of the receiver implementation. The sender interprets the ACKs for dummy segments as the evidence that there are unused resources in the network and accordingly, can increase its transmission rate.

The TCP-Peach sender can verify whether the receiver implements the required modification during the Sudden Start. If the required modification is not implemented, TCP-Peach sender stops transmitting dummy segments, i.e., TCP-Peach behaves like TCP-Reno [28].

In the following sections we show that the ACKs for the dummy segments transmitted during the Sudden Start and Rapid Recovery are received during the Congestion Avoidance phase. Consequently, the Congestion Avoidance is modified in TCP-Peach.

We introduce the variable $wdsn$. Upon receiving an ACK for a dummy segment, the sender checks the value of $wdsn$. Then:

- If $wdsn = 0$, then the congestion window $cwnd$ is increased by one, i.e., $cwnd := cwnd + 1$.

```
Sudden_Start( )
    cwnd=1;
    τ = RTT/rwnd;
    send(Data_Segment);
    for (i=1 to rwnd − 1),
        wait(τ);
        send(Dummy_Segment);
    end;
end.
```

Fig. 2. TCP-Peach: `Sudden_Start()`.

- If $wdsn \neq 0$, then the $wdsn$ value is decreased by one, i.e., $wdsn := wdsn - 1$, and the congestion window value $cwnd$ remains the same.

The variable $wdsn$ is used in order to match the behaviors of TCP-Peach and TCP-Reno [28] when the network is congested, i.e., this guarantees that TCP-Peach is TCP-friendly [23]. In the beginning of a new connection, $wdsn$ is set to zero.

TCP-Peach requires that all routers in the connection path support some priority discipline. In traditional IP networks, the IP TOS can be used for this purpose [38]. In fact, one of the eight bits of the TOS field in the IP header gives the priority level of the IP packet [38]. Instead, more recent IP versions, e.g., IPv6 [19], explicitly provide several priority levels.

Currently, some routers in the Internet do not apply any priority policy. However, in the near future, the Internet will support quality of service through the DiffServ [11], which requires all routers to support multiple service classes. As a matter of fact, all recent commercial routers, e.g., Cisco series 7000 and 12 000 [17], support at least the IP TOS.

Low-priority segments are used in [12] to measure the available bandwidth in the network to conduct admission control.

Low-priority segments are also used in [36] to improve the performance of TCP. However, the low-priority segments in [36] are different from the dummy segments because:

1) They are not used to probe the availability of network resources. In fact, their objective is to carry information to the receiver more rapidly without harming other flows.
2) Since they carry new information to the receiver, they are still data segments, and if they are lost, then they must be recovered.
3) They are used only in the beginning of a new connection.

B. Sudden Start Algorithm

The Sudden Start substitutes the Slow Start [27] in Fig. 1.

Let $rwnd$, which is specified by the receiver, be the maximum value for the congestion window $cwnd$. As shown in Fig. 2, the basic idea of the Sudden Start is that in the beginning of a connection, the sender sets the congestion window $cwnd$ to 1 and after the first data segment, it transmits $(rwnd - 1)$ dummy segments every

$$\tau = RTT/rwnd. \tag{2}$$

As a result, after one RTT, the congestion window size $cwnd$ increases very quickly. Note that the sender can estimate RTT during the connection setup phase.

Now we explain the Sudden Start algorithm in detail.

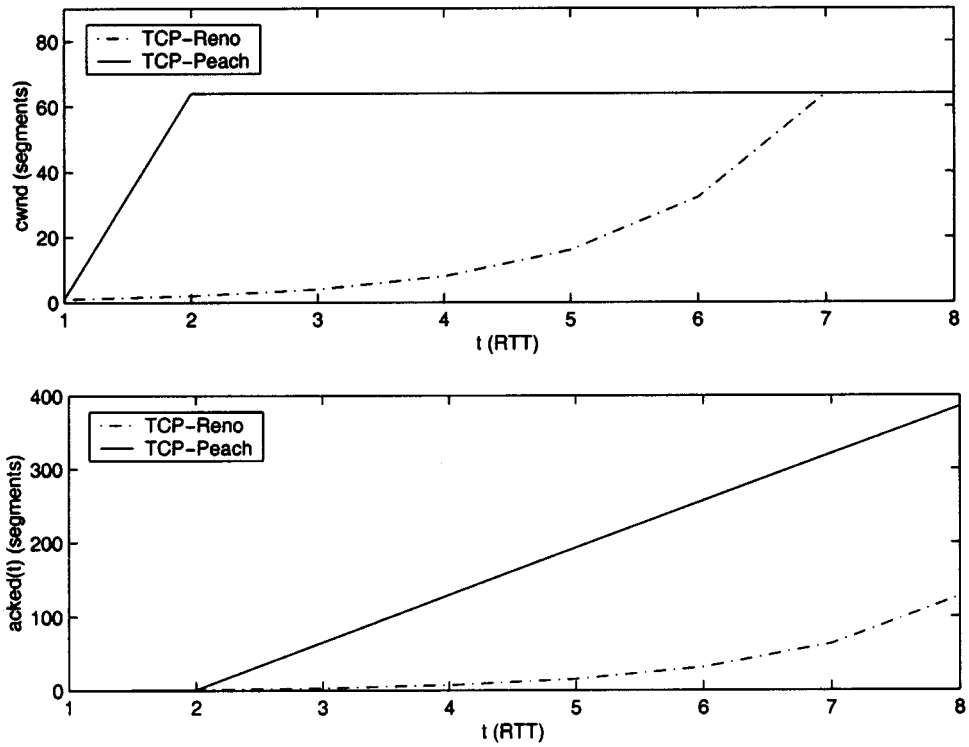

Fig. 3. Comparison between TCP-Peach and TCP-Reno in the beginning of a new connection.

Suppose that a new TCP connection begins at time $t = 0$.

- $0 \leq t < RTT$:
  The sender transmits a data segment first and $(rwnd - 1)$ dummy segments, in such a way that the data and dummy segment transmissions are uniformly distributed in an interval equal to RTT. Then, it executes the Congestion Avoidance.
- $RTT \leq t < 2 \cdot RTT$:
  The ACKs related to the data and dummy segments transmitted in the time interval $0 \leq t < RTT$ arrive at the sender. Since $wdsn = 0$, for any received ACK related to a dummy segment, the sender increases its $cwnd$ by 1 and transmits a new data segment. In this period, the data segments are transmitted at the same rate, v_{ACK}, of the ACK arrivals. Note that v_{ACK} is approximately

$$v_{\mathrm{ACK}} = \min\left\{\frac{\text{one segment}}{\tau}, \phi\right\} \tag{3}$$

  where ϕ is the maximum achievable rate with the currently available bandwidth and τ is given in (2). As a result, at time $t \approx RTT$, the transmission rate of the sender jumps suddenly from $\{\text{one segment}\}/RTT$ to v_{ACK}. Note that if the receiver does not implement the modifications required by the TCP-Peach scheme, then the ACKs for the dummy segments will not arrive in the expected format. If this is the case, the sender stops transmitting dummy segments and starts to use the TCP-Reno [28].
- $t \approx 2 \cdot RTT$
  At this time, the ACK related to the last transmitted dummy segment is received by the sender, i.e., the Congestion Avoidance continues as in [28].

In Fig. 3 we compare the TCP-Peach (solid lines) and the TCP-Reno (dashed lines) in the beginning of a new connection. In the upper plot, we show the behavior of the congestion window $cwnd$ dependent on time t. The time unit considered in Fig. 3 is set equal to RTT. In the bottom plot, we show $acked(t)$, which is the number of acknowledged data segments in the time interval $[0, t]$. These plots were obtained by considering the $rwnd$ equal to 64 segments. It can be seen that the Sudden Start reaches $rwnd$ (64 segments) much earlier than the Slow Start algorithm. Moreover, it can also be seen in the bottom plot that the Sudden Start algorithm delivers data segments much faster than the Slow Start.

C. Rapid Recovery Algorithm

The Rapid Recovery substitutes the classical Fast Recovery algorithm [28] with the objective of solving the throughput degradation problem due to link errors highlighted in Section II-B.

As shown in Fig. 1, when a segment loss is detected through $ndup$ duplicate ACKs, we use the original Fast Retransmit algorithm [28]. After completing the Fast Retransmit algorithm, we apply the Rapid Recovery algorithm, as in Fig. 4, which will terminate at the time, t_{END}, when the ACK for the lost data segment is received, as shown in Fig. 5. Consequently, the Rapid Recovery lasts for RTT. Then, the sender will enter the Congestion Avoidance phase as depicted in Fig. 1.

The Rapid Recovery first keeps the classical Fast Recovery conservative assumption that all segment losses are due to network congestion because the TCP layer does not know anything about the exact causes for the losses, i.e., due to network congestion or due to link errors [37]. Accordingly, the sender halves its congestion window $cwnd$, as in TCP-Reno [28]. Thus, if $cwnd$

```
Rapid_Recovery()
  cwnd=cwnd/2;
  adsn=2*cwnd;
  wdsn=cwnd;
  infl_seg=0;
  t_Retr=t;
  END=0;
  while (END=0)
    if (ACK_ARRIVAL)
      if (DATA_ACK_ARRIVAL)
        cwnd=cwnd+1;
        infl_seg=infl_seg+1;
      else if (DUMMY_ACK_ARRIVAL)
        if (wdsn=0)
          cwnd=cwnd+1;
          infl_seg=infl_seg+1;
        else
          wdsn=wdsn-1;
        end;
      end;
      if (cwnd>nackseg)
        while(cwnd>nackseg)
          send(Data_Segment);
          nackseg=nackseg+1;
        end;
      else if (adsn>0)
        send(Dummy_Segment);
        send(Dummy_Segment);
        adsn=adsn-2;
      end;
      if (LOST_SEGMENT_ACKED)
        END=1;
        cwnd=cwnd-infl_seg;
      end;
    end;
    if (t>t_Retr+RTO)
      Slow_Start();
    end;
  end;
end.
```

Fig. 4. TCP-Peach: **Rapid_Recovery()**.

was equal to $cwnd_0$, then it becomes $cwnd = cwnd_0/2$, which means that the sender will transmit approximately $cwnd_0/2$ data segments during the Rapid Recovery phase.

Moreover, in order to probe the availability of network resources, the sender transmits a certain number n_{Dummy} of dummy segments. Note that the value for n_{Dummy} will be derived in the following. The ACKs for the dummy segments will be received after the ACK for the lost data segment, i.e., they will be received when the sender is in Congestion Avoidance phase, as shown in Fig. 5.

If the segment loss is due to congestion, then the congested router can serve $cwnd_0$ segments per round-trip time, approximately. As a result, the network will accommodate the $cwnd_0/2$ data segments, which have high priority, and $cwnd_0/2$ dummy segments out of the n_{Dummy} dummy segments transmitted during the Rapid Recovery phase.

Therefore, the sender must not increase its congestion window $cwnd$ when it receives the first $cwnd_0/2$ ACKs for dummy segments. In fact, these ACKs cannot be considered as the sign that the loss was due to link errors, i.e., not due to network congestion. With this objective, $wdsn$ is set to $cwnd_0/2$, i.e., $wdsn = cwnd$, in Fig. 4. This will prevent the sender to increase its congestion window when the first $cwnd_0/2$ ACKs for dummy segments are received during the Congestion Avoidance.

After receiving $cwnd_0/2$ ACKs for dummy segments, the sender increases its congestion window $cwnd$ by one segment each time it receives an ACK for a dummy segment.

We set n_{Dummy} equal to $cwnd_0$. As a result, if all dummy segments are ACKed to the sender, then the congestion window $cwnd$ reaches the value it had before the segment loss was detected, i.e., $cwnd = cwnd_0$.

Note that the retransmitted segment may get lost. Let t_{Retr} be the time when the lost segment is retransmitted. If at time $(t_{\text{Retr}} + RTO)$, no ACK has been received for the retransmitted segment, then this segment may be lost. Accordingly, the Rapid Recovery is terminated and the sender executes the Sudden Start because the loss may be due to heavy network congestion.

Now we explain the Rapid Recovery Algorithm, shown in Fig. 4, in detail. In the beginning, the sender sets some variables:

- $cwnd = cwnd/2$.
- $adsn = 2 * cwnd$.

  The variable, *allowed dummy segment number* ($adsn$), is the number of dummy segments that the sender is allowed to inject into the network. Initially, the sender sets *adsn* equal to n_{Dummy} which is given by

$$n_{\text{Dummy}} = cwnd_0 = 2 \cdot cwnd \tag{4}$$

- $wdsn = cwnd$.
- $infl\_seg = 0$.

  The sender can artificially inflate the congestion window $cwnd$ during the Rapid Recovery phase. The variable $infl\_seg$ gives the amount that $cwnd$ was inflated.
- $t_{\text{Retr}} = t$

  The variable t_{Retr} is set equal to the current time, t. Note that t_{Retr} is approximately the time when the lost data segment has been retransmitted.
- $END = 0$

  The variable END is a Boolean. When $END = 1$, the Rapid Recovery terminates. The sender initializes $END = 0$.

Then, until the end of the Rapid Recovery algorithm, upon receiving an ACK for a data segment (**DATA_ACK_ARRIVAL**), the sender artificially inflates $cwnd$ by 1 and then checks whether it can transmit a new data segment or not. If it cannot, i.e., if $cwnd$ is lower than or equal to the amount of unacknowledged data segments, $nackseg$, ($cwnd \leq nackseg$), then the sender checks the $adsn$ value. If $adsn > 0$, then the sender transmits two dummy segments and decreases the value of *adsn* by two. Finally, when the lost data segment is acknowledged (**LOST_SEGMENT_ACKED**), $cwnd$ is reduced by the amount it was artificially inflated before, $infl\_seg$, i.e., ($cwnd := cwnd - infl\_seg$), $adsn$ is set to 0, the Rapid Recovery phase is completed (**END** $= 1$) and the Congestion Avoidance is executed. The arrival of an ACK for a dummy segment (**DUMMY_ACK_ARRIVAL**) is treated such as the ACK of data segments, but the congestion window, $cwnd$, is inflated

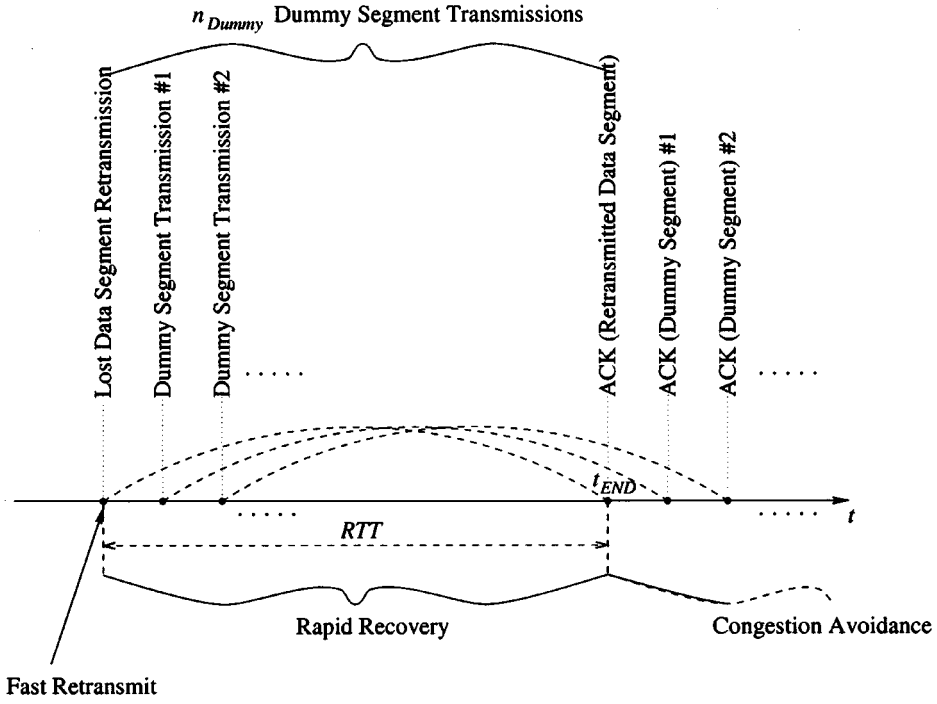

Fig. 5. Rapid Recovery phase.

only if $wdsn = 0$. Otherwise, $cwnd$ is not increased and the value $wdsn$ is decreased by one.

If at time $t \geq (t_{Retr} + RTO)$, the sender is in the Rapid Recovery phase, then none of the ACKs for the retransmitted data segment and the dummy segments have been received within RTO. This may be due to heavy network congestion. Accordingly, the sender terminates the Rapid Recovery and executes the Sudden Start.

Note that the Rapid Recovery algorithm relies on *self clocking*, i.e., data segment transmissions are triggered by receiving ACKs [32], in such a way that the amount of data segments in the network is kept at a constant level which guarantees the network stability.

IV. TCP-Peach Behavior in Case of Segment Losses

In this section, we present the behavior of TCP-Peach when a segment loss is detected. Thus, we show how the Rapid Recovery and the Congestion Avoidance work together. We consider two different cases:

Case 1: The segment loss is due to link errors (Section IV-A).

Case 2: The segment loss is due to network congestion (Section IV-B).

A. Segment Losses Due to Link Errors

Let t be time instance given in Fig. 6.

- $t = t_0^-$

  ($cwnd = cwnd_0$, $nackseg = cwnd_0$, $wdsn = 0$).

  Let t_0^- indicate the time instant immediately before t_0. Suppose that the congestion window of a sender is $cwnd_0$ at time t_0^-. Thus, there are approximately $cwnd_0$ outstanding unacknowledged data segments, i.e., $nackseg = cwnd_0$. Suppose that none but the first of these segments as well as their ACKs are lost. These ACKs reach the sender between time t_0 and t_1 where $t_1 \approx t_0 + RTT$.

- $t = t_0^+$

  ($cwnd = cwnd_0/2$, $nackseg = cwnd_0$, $wdsn = cwnd_0/2$, $adsn = cwnd_0$).

  Let t_0^+ indicate the time instant immediately after t_0. Suppose that at time t_0 the sender detects the data segment loss through receiving $ndup$ duplicated ACKs. According to the Fast Retransmit algorithm [28], the sender retransmits the lost data segment whose ACK is now expected to arrive at time t_1. As shown in Fig. 1, the Rapid Recovery then starts and the sender sets $cwnd = cwnd_0/2$, $adsn = cwnd_0$ and $wdsn = cwnd_0/2$. It follows that at time t_0^+

$$cwnd \leq nackseg = cwnd_0. \tag{5}$$

- $t_0 < t < t'$ (where $t' \approx t_0 + 0.5 \cdot RTT$)

  ($cwnd \leq nackseg$, $nackseg = cwnd_0$, $wdsn = cwnd_0/2$, $adsn \geq 0$).

  Upon receiving each ACK for data segments, the sender *inflates* its $cwnd$ by one, thus, (5) holds until the time t' when the sender receives $cwnd_0/2$ ACKs. Note that ($t' \approx t_0 + 0.5 \cdot RTT$). Since the congestion window $cwnd$ is smaller than the amount of unacknowledged data $nackseg$ in the time interval $]t_0, t']$, it follows that (5) holds; therefore the sender cannot transmit any new data segment. Upon receiving each ACK, the sender transmits two dummy segments and decreases $adsn$ by two segments. Thus, the dummy segments are transmitted at double the transmission rate of the data segments before t_0. If the network supports this rate, then the dummy

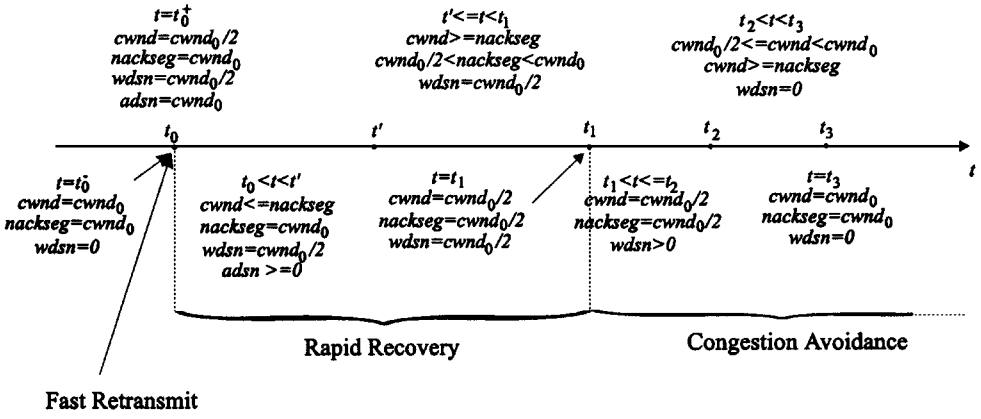

Fig. 6. Rapid Recovery behavior when a segment loss occurs due to link errors.

segments will be received and ACKed by the receiver. At time t', the sender parameters are $cwnd = cwnd_0$, $nackseg = cwnd_0$, $wdsn = cwnd_0$, and $adsn = 0$. Moreover, there will be the ACKs of $cwnd_0/2$ data segments expected by the sender, and $cwnd_0$ dummy segments in the path.

- $t' \leq t < t_1$

  ($cwnd \geq nackseg$, $cwnd_0/2 < nackseg < cwnd_0$, $wdsn = cwnd_0/2$).

  In the time interval $[t', t_1[$, the sender inflates $cwnd$ by one segment upon receiving each ACK. After each inflation the congestion window becomes $cwnd = nackseg + 1$. Consequently, the sender transmits a new data segment and $nackseg$ is increased by one. If there is no congestion along the path in the time interval $]t', t_1]$, the sender receives the ACKs approximately at the same rate it was transmitting before t_0. Therefore, it also transmits new data segments approximately at the same rate it was transmitting before t_0.

- $t = t_1$

  ($cwnd = cwnd_0/2$, $nackseg = cwnd_0/2$, $wdsn = cwnd_0/2$).

  At time t_1, the sender receives the ACK of the lost and retransmitted data segment. Then $cwnd$ is decreased by the amount it was artificially inflated before and thus, $cwnd = cwnd_0/2$. In this way, the Rapid Recovery phase is over and the sender enters the Congestion Avoidance phase as shown in Fig. 1. Note that since the ACK for the retransmitted segments also acknowledges all data segments transmitted before time $t = t_0$, there are only $cwnd_0/2$ outstanding data segments, i.e., $nackseg = cwnd_0/2$, at time $t = t_1^+$.

- $t_1 < t \leq t_2$ (where $t_2 \approx t_1 + 0.25 \cdot RTT$)

  ($cwnd = cwnd_0/2$, $nackseg = cwnd_0/2$, $wdsn > 0$).

  The sender receives the ACKs for the first $cwnd_0/2$ dummy segments transmitted in the time interval $]t_0, t']$. Since $wdsn$ is higher than 0, i.e., $wdsn > 0$, the sender does not increase the congestion window size $cwnd$, and decreases the value of $wdsn$ by one. At time t_2, $wdsn$ reaches the value 0, i.e., $wdsn = 0$.

  Note that in the time interval $]t_1, t_2]$ the sender does not transmit any new data segment because the relation $cwnd = nackseg = cwnd_0/2$ always holds.

- $t_2 < t < t_3$ (where $t_3 \approx t_1 + 0.5 \cdot RTT$)

  ($cwnd_0/2 \leq cwnd < cwnd_0$, $cwnd \geq nackseg$, $wdsn = 0$).

  The sender receives the ACKs for the last $cwnd_0/2$ dummy segments transmitted in the time interval $]t_0, t']$. Since the $wdsn$ value is 0, the sender increases $cwnd$ by one segment each time it receives an ACK. As a result, $cwnd = nackseg + 1$, and thus, the sender transmits a new data segment and increases the $nackseg$ value by one segment.

- $t = t_3^+$

  ($cwnd = cwnd_0$, $nackseg = cwnd_0$).

  At time t_3^+, the congestion window is $cwnd = cwnd_0$ and the number of outstanding data segments is $nackseg = cwnd_0$. The TCP parameters ($cwnd$, $nackseg$ and $wdsn$) are the same as they were before the data segment loss was detected.

Note that the only effect of a segment loss due to link errors is that the sender stops transmitting new data segments in the time intervals $[t_0, t']$ and $]t_1, t_2]$. If there is enough bandwidth available, then in the time interval $t_2 < t \leq t_3$, the sender transmits new data segments at double the rate it was transmitting before the data segment loss was detected. If there are enough resources, the only effects of a data segment loss due to link errors on the throughput is that the sender transmits $cwnd_0/2$ less data segments.

In Fig. 7, we compare the Rapid Recovery (solid lines) and the Fast Recovery (dashed lines) algorithms. In both cases, we force the 100th segment to be lost at time $t_0 = 0$ when the congestion window $cwnd$ is equal to 30. In the upper plot, we show the behavior of the congestion window $cwnd$ dependent on time, and in the bottom plot, we show the number of acknowledged data segments $acked(t)$. We assumed $RTT = 0.55$ s. The segment loss is detected approximately at time $t = (t_0 + RTT) = 0.55$ s. Accordingly, in the bottom plot of Fig. 7, the congestion window for both TCP-Peach and TCP-Reno is halved, i.e., $cwnd = 15$ segments. Then the TCP-Peach sender executes the Rapid Recovery, whereas the TCP-Reno executes the Fast Recovery [28].

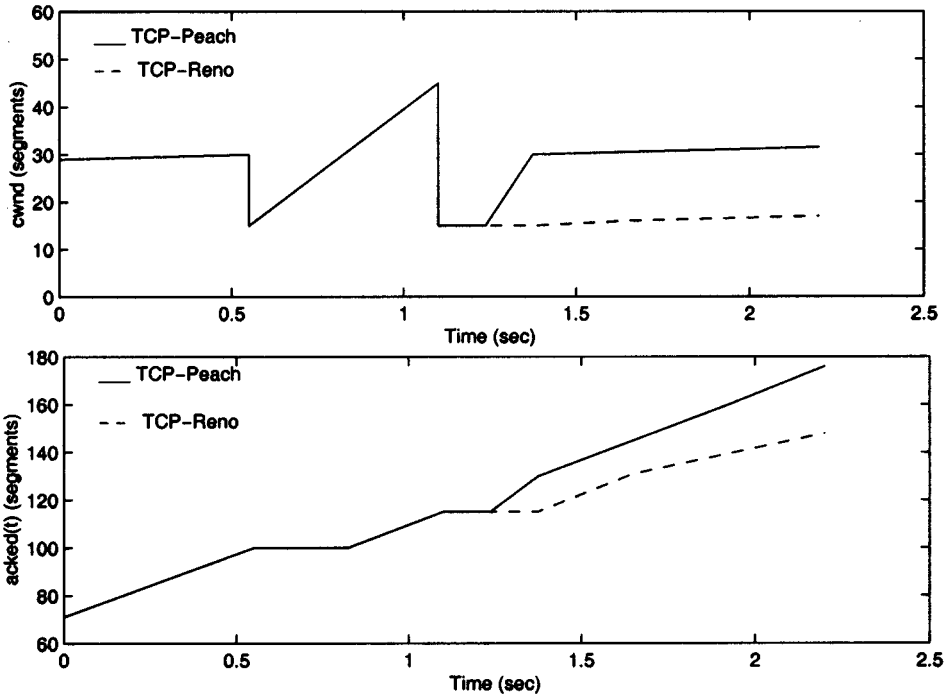

Fig. 7. TCP-Peach and TCP-Reno behavior when segment losses occur due to link errors.

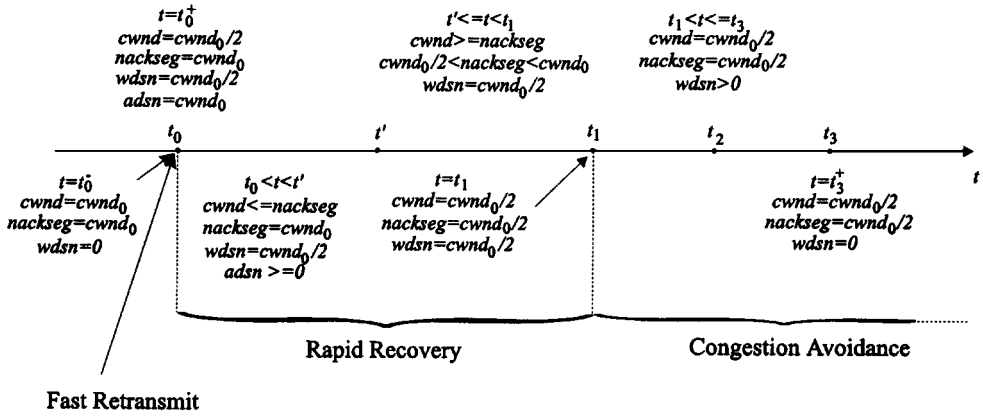

Fig. 8. Rapid Recovery behavior when segment losses occur due to network congestion.

In both cases, during this period the congestion window $cwnd$ is inflated by one segment for each ACK received. The sender receives the ACKs for 27 segments, thus the congestion window $cwnd$ increases up to 42 segments in the bottom plot of Fig. 7. At time $t \approx (t_0+2\cdot RTT) = 1.1$ s the sender receives the ACK for the lost segment. As a result, both TCP-Peach and TCP-Reno decrease the congestion window, i.e., $cwnd = 15$ segments, and enter the Congestion Avoidance phase. In this phase, the TCP-Peach receives the ACKs for the dummy segments transmitted during the Rapid Recovery phase. Accordingly, in the bottom plot of Fig. 7 from time $t \approx (t_0+2.5\cdot RTT) = 1.375$ s to time $t \approx (t_0+3\cdot RTT) = 1.65$ s the congestion window $cwnd$ for TCP-Peach increases from 15 to 30 segments. In this phase, the congestion window $cwnd$ for TCP-Reno [28] increases by only one segment per RTT [39]. This explains why in the upper plot of Fig. 7, $acked(t)$ increases more rapidly for TCP-Peach than the TCP-Reno.

B. Segment Losses Due to Network Congestion

In order to be a *good network citizen*, a flow should decrease its transmission rate by a factor two when the network is congested [23]. Here we show that TCP-Peach complies with this requirement.

Consider a single connection flowing through a link. Let t be time instance given in Fig. 8.

- $t = t_0^-$
  ($cwnd = cwnd_0$, $nackseg = cwnd_0$, $wdsn = 0$).
  Suppose that the congestion window of a sender is $cwnd_0$ at time t_0. Thus, there are approximately $cwnd_0$ outstanding unacknowledged data segments, i.e., $nackseg = cwnd_0$.
- $t = t_0^+$
  ($cwnd = cwnd_0/2$, $nackseg = cwnd_0$, $wdsn = cwnd_0/2$, $adsn = cwnd_0$).

Suppose that at time t_0 the sender detects the data segment loss through receiving $ndup$ duplicated ACKs. Suppose that the above segment loss is due to network congestion, i.e., the connection path can accommodate at most a transmission rate given approximately by $cwnd_0/RTT$.

According to the Fast Retransmit algorithm [28], the sender retransmits the lost data segment and sets $cwnd = cwnd_0/2$. As shown in Fig. 1, the Rapid Recovery then starts and the sender sets $adsn = cwnd_0$ and $wdsn = cwnd_0/2$.

- $t_0 < t < t'$ (where $t' \approx t_0 + 0.5 \cdot RTT$)
  ($cwnd \leq nackseg$, $nackseg = cwnd_0$, $wdsn = cwnd_0/2$, $adsn \geq 0$).
  Upon receiving each ACK, the sender *inflates* its $cwnd$ by one segment, thus (5) holds until the time t' when the sender receives $cwnd/2$ ACKs. Note that $t' \approx t_0 + 0.5 \cdot RTT$. Since the congestion window $cwnd$ is smaller than the amount of unacknowledged data $nackseg$ in the time interval $]t_0, t']$, it follows that (5) holds; therefore the sender cannot transmit any new data segment.
  Upon receiving each ACK, the sender transmits two dummy segments and decreases $adsn$ by two segments. Accordingly, in the time interval $]t_0, t']$ the sender transmits $cwnd_0$ dummy segments at a rate which is approximately equal to $cwnd_0/(0.5 \cdot RTT)$. Since the network path can accommodate at most a transmission rate of $cwnd_0/RTT$, about the half of the transmitted dummy segments are lost, i.e., only $cwnd_0/2$ dummy segments reach the destination.
  At time t', the sender parameters are: $cwnd = cwnd_0$, $nackseg = cwnd_0$, $wdsn = cwnd_0/2$ and $adsn = 0$. Moreover, the sender is still waiting for the ACKs of $cwnd_0/2$ data segments transmitted before t_0 and there are $cwnd_0/2$ dummy segments in the path.
- $t' \leq t < t_1$
  ($cwnd \geq nackseg$, $nackseg \geq cwnd_0$, $wdsn = cwnd_0/2$).
  In the time interval $[t', t_1[$, upon receiving each ACK, the sender inflates $cwnd$ by one segment. After each inflation the congestion window is $cwnd = nackseg+1$. Consequently, the sender transmits a new data segment and $nackseg$ is increased by one.
- $t = t_1$
  ($cwnd = cwnd_0/2$, $nackseg = cwnd_0/2$, $wdsn = cwnd_0/2$).
  At time t_1, the sender receives the ACK of the lost and retransmitted data segment. Then, $cwnd$ is decreased by the amount it was artificially inflated before and thus, $cwnd = cwnd_0/2$. Moreover, there are only $cwnd_0/2$ outstanding data segments, i.e., $nackseg = cwnd_0/2$.
- $t_1 < t \leq t_3$ (where $t_3 \approx t_1 + 0.5 \cdot RTT$)
  ($cwnd = cwnd_0/2$, $nackseg = cwnd_0/2$, $wdsn > 0$).
  The sender receives the ACKs for the $cwnd_0/2$ dummy segments transmitted in the time interval $]t_0, t']$. Since $wdsn$ is higher than 0, i.e., $wdsn > 0$, the sender does not increase the congestion window size, $cwnd$, and decreases the value of $wdsn$ by one. At time t_3, $wdsn$ reaches the value 0, i.e., $wdsn = 0$.
  Note that in the time interval $]t_1, t_3]$ the sender does not transmit any new data segment because the relation $cwnd = nackseg = cwnd_0/2$ always holds.
- $t > t_3^+$
  ($cwnd = cwnd_0/2$, $nackseg = cwnd_0/2$).
  At time t_3^+, the congestion window is $cwnd = cwnd_0/2$ and the number of outstanding data segments is $nackseg = cwnd_0/2$ and there are no more dummy segments or their ACKs in the network. Note that the TCP parameters ($cwnd$ and $nackseg$) are the same as in the TCP-Reno case.

In Fig. 9, we show that the behaviors of TCP-Peach and TCP-Reno are the same when a segment loss occurs due to network congestion. We force the 100th segment to be lost for network congestion when the congestion window $cwnd$ is equal to 30. In the upper plot, we show the behavior of the congestion window $cwnd$ dependent on RTT, and in the bottom plot, we show the number of acknowledged data segments $acked(t)$. We assumed $RTT = 0.55$ s. At time $t = 0.55$ s, the sender (both in TCP-Peach and TCP-Reno case) detects a data segment loss and thus retransmits the lost segment and halves its congestion window $cwnd$, as shown in the upper plot of Fig. 9. For $0.55 < t < 0.55 + RTT$, the sender executes the Rapid Recovery in the TCP-Peach case and the Fast Recovery in the TCP-Reno case. In this time interval, the sender artificially inflates its congestion window $cwnd$ by 1 for each ACK received. Moreover, in the TCP-Peach case, the sender transmits dummy segments. At time $t = 0.55 + RTT$, the ACK for the retransmitted data segment is received. Accordingly, the sender decreases the congestion window $cwnd$ by the amount it was artificially inflated and it enters the Congestion Avoidance phase. Since the data segment loss was due to network congestion, in the TCP-Peach case, the majority of the dummy segments transmitted at $0.55 < t < 0.55+RTT$ are discarded by the network because they have low priority. As a result, for $t > 0.55 + RTT$, the sender does not increase its congestion window $cwnd$, due to the arrivals of the ACKs for the dummy segments. This explains why TCP-Peach and TCP-Reno results overlap in Fig. 9.

V. Simulation Experiments

We evaluate the performance of TCP-Peach in terms of goodput and fairness through simulations when several connections share the same link. More in detail, in Section V-A, we compare the goodput performance of TCP-Peach and TCP-Reno in satellite networks, while in Section V-B, we evaluate the fairness of TCP-Peach. Further simulation results along with an analytical model of TCP-Peach can be found in [34].

A. Goodput Performance

The TCP-Reno implementation considered here is suggested in [22] and is also known as New Reno because it removes

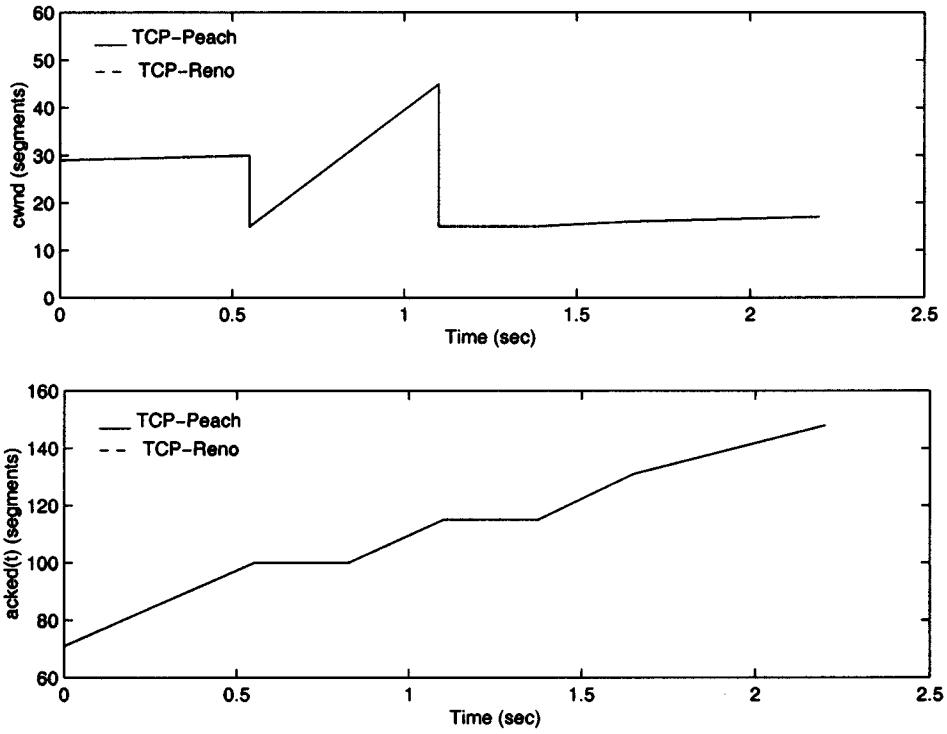

Fig. 9. The behavior of TCP-Peach and TCP-Reno when segment losses occur due to network congestion.

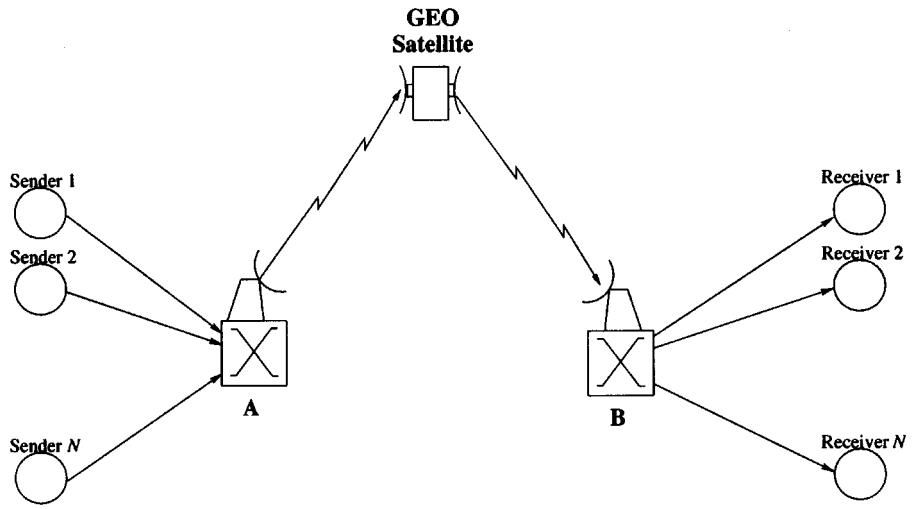

Fig. 10. Simulation scenario.

certain problems of the original Reno [21], [26]. Also, we assume that both TCP-Reno and TCP-Peach apply the *Selective Acknowledgment* (SACK) option [31]. Experimental results in [25] show that TCP-New Reno augmented with SACK option [31] gives the best performance in satellite networks.

We simulate the system in Fig. 10 where N senders transmit data to N receivers through a satellite channel. Note that although we consider only a GEO satellite system, we have obtained similar results for LEO and MEO satellite systems as well. The N streams are multiplexed in the Earth Station A, whose buffer can accommodate K segments. Both data and dummy segments may get lost due to link errors with a probability P_{Loss}. As in [25], we assume that $N = 20$, $K = 50$ segments, $rwnd = 64$ segments. We also assume that the link capacity is $c = 1300$ segments/s which is approximately 10 Mb/s for TCP segments of 1000 bytes. The RTT value considered is $RTT = 550$ ms.

All the results shown in this section have been obtained by considering the system behavior for $t_{\text{Simulated}} = 550$ s, which is 1000 times the round-trip time value. Preliminary experiments on the physical testbed provided by "ACTS Experiment 154—Investigation of TCP Performance Relative to Distinguish between Errors and Congestion" confirm the simulation results shown in this section [35].

In Fig. 11 we compare the goodput values of TCP-Reno and TCP-Peach for different values of loss probabilities due to link

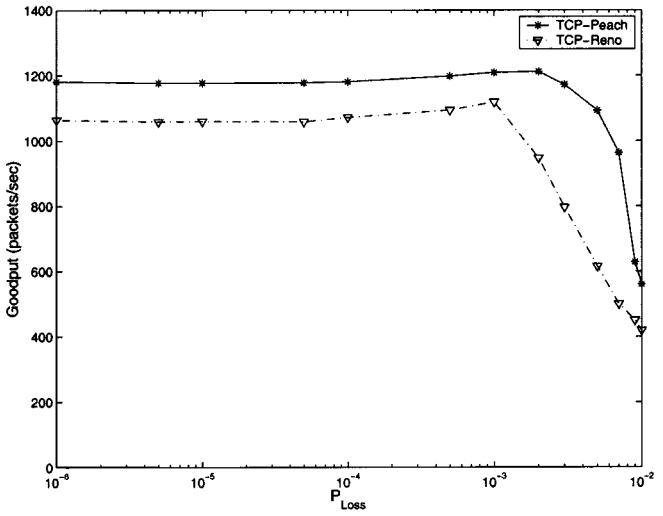

Fig. 11. Goodput performance comparison of TCP-Peach and TCP-Reno for different values of P_{Loss}.

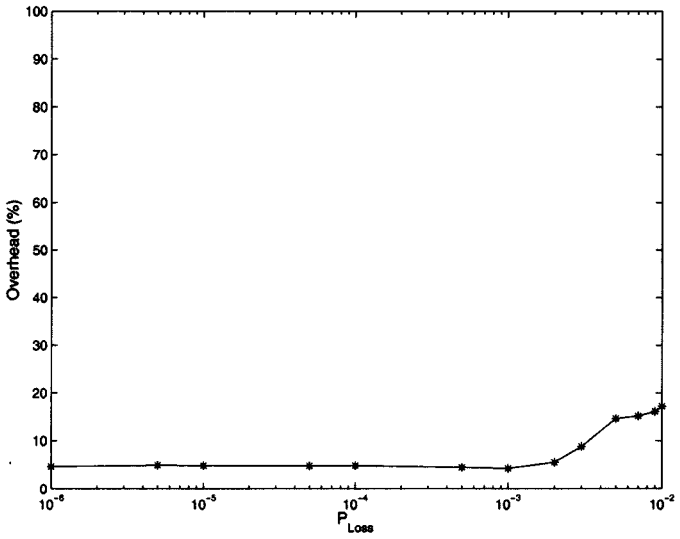

Fig. 12. Overhead introduced by dummy segments for different values of P_{Loss}.

errors, P_{Loss}.[2] In Fig. 11, the goodput obtained using TCP-Peach is always higher than in TCP-Reno case. When the loss probability for link errors is low $P_{\text{Loss}} \leq 10^{-3}$, the goodput increase obtained using TCP-Peach is mostly due to the Sudden Start Algorithm. In fact, packet losses due to link errors are very rare, i.e., almost all packet losses are due to network congestion, thus, the Rapid Recovery does not give any advantage in this case compared to Fast Recovery [28]. However, for P_{Loss} values higher than 10^{-3}, the goodput of TCP-Reno decreases because of the packet losses due to link errors. The goodput of TCP-Peach is higher because using the Rapid Recovery the congestion window $cwnd$ increases more rapidly when packet losses occur due to link errors.

TCP-Peach obtains higher goodput transmitting dummy segments. Since dummy segments do not carry any new information, they cause overhead in the network. In Fig. 12, we show the overhead dependent on P_{Loss}. As it can be seen in Fig. 12, when $P_{\text{Loss}} = 10^{-2}$, the overhead is equal to 17.21%, which is the maximum value. However, in the same case, TCP-Peach achieves 30.65% higher goodput than the TCP-Reno as shown in Fig. 11.

In Fig. 13, we show the goodput of TCP-Peach and TCP-Reno for different values of the link capacity c. We assumed $P_{\text{Loss}} = 5 \cdot 10^{-3}$. Note that for low values of the link capacity c, the goodput values obtained by TCP-Peach and TCP-Reno are almost equal, because the majority of segment losses is due to network congestion. For segment losses caused by link errors, the congestion window $cwnd$ of TCP-Reno is always small. As a result, in Fig. 13 the goodput for TCP-Reno does not exceed 620 packets/s even when the link capacity c is very high. The Rapid Recovery algorithm solves this problem, as shown in Fig. 13.

Currently, web applications are very popular in the Internet. Therefore, we simulated the case in Fig. 10 where N TCP-Peach senders transmit web pages, each with S segments. As soon as a web page transfer is completed, i.e., all ACKs for S segments of

[2]The bit-error rate (BER) in satellite networks can be as high as 10^{-4}. For TCP segments of 1000 bytes, the BER 10^{-4} gives a segment loss probability P_{Loss} higher than 10^{-2} even if a powerful decoding algorithm is applied.

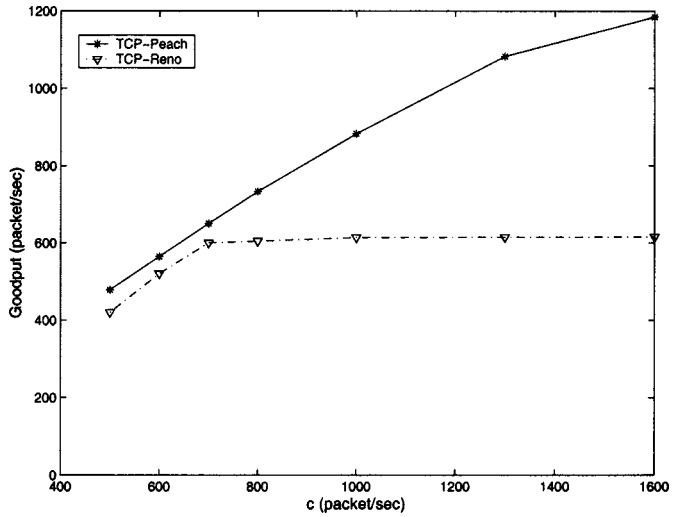

Fig. 13. Goodput performance comparison of TCP-Peach and TCP-Reno for different values of the link capacity c.

one web page are received, the sender then begins to transmit a new web page. In Fig. 14, we show the average time t_{Avg} needed by each TCP-Peach sender to transfer a web page with $S = 50$ segments. Note that t_{Avg} is an important performance parameter for web users. In Fig. 14, we also show the simulation results in cases of connections using TCP-Reno [28] and TCP-Reno with the Increased Initial Window option [4]. We assume the SACK option [31] is implemented in all cases. In Fig. 14, TCP-Peach outperforms TCP-Reno because the Sudden Start is faster than the Slow Start. In Fig. 15, we show the goodput values where TCP-Peach achieves the highest goodput performance.

Now let $\phi_{\text{Peach/Fast Start}}$ be the ratio between the goodput obtained by TCP-Peach and the goodput obtained by TCP-Reno with the Fast Start modifications [36] (TCP-Reno + FS). In Fig. 16 we show $\phi_{\text{Peach/Fast Start}}$ dependent on the segment loss probability due to link errors, P_{Loss}. In our simulations, TCP sources transmit files of S segments. The time interval between two consecutive file transfers from a sender is equal to

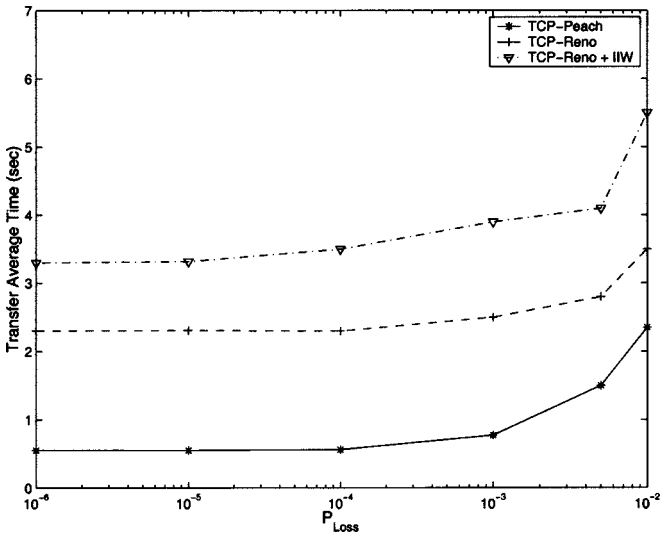

Fig. 14. Transfers of files of $S = 50$ segments: average transfer duration.

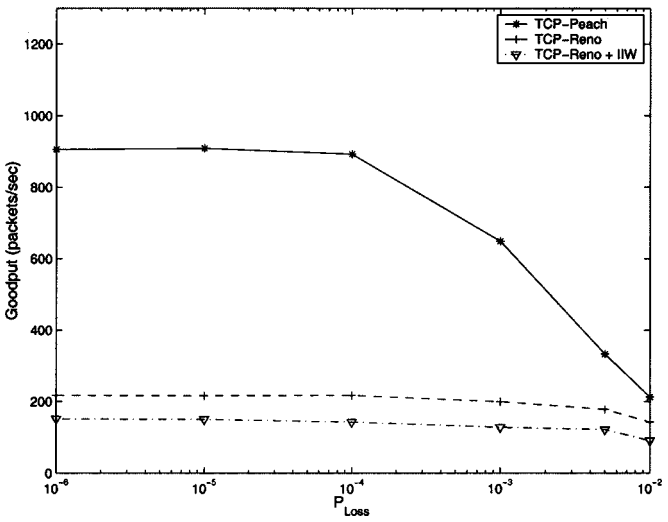

Fig. 15. Transfers of files of $S = 50$ segments: average goodput.

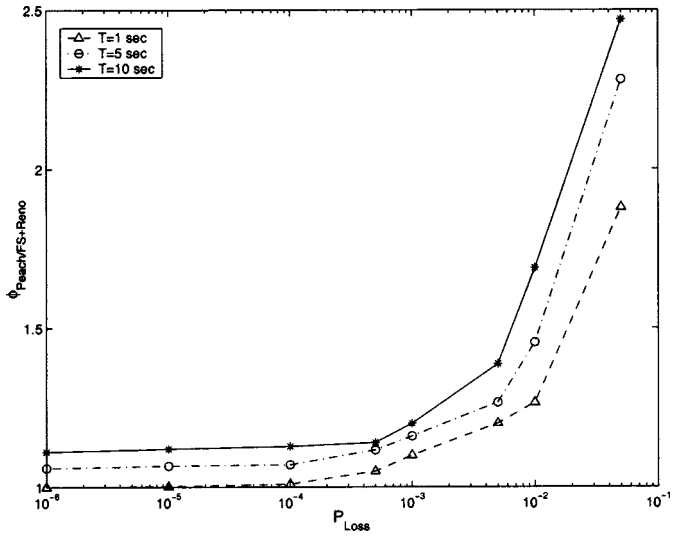

Fig. 16. Goodput performance comparison of TCP-Peach and TCP-Reno with the Fast Start modifications.

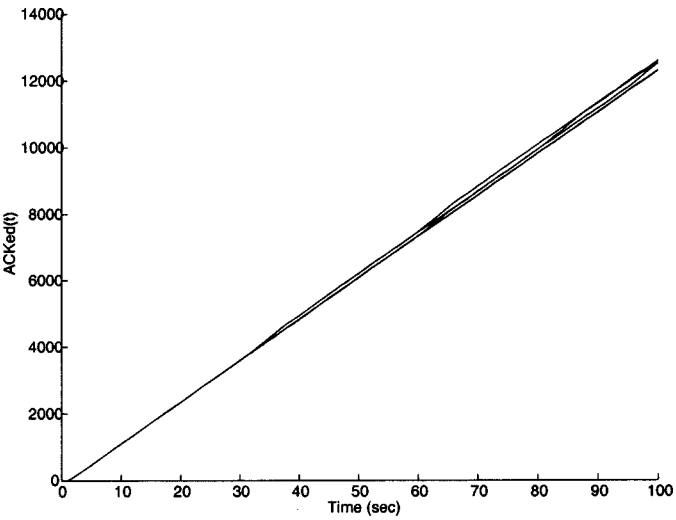

Fig. 17. Fairness evaluation in a homogeneous scenario.

a random variable which is exponentially distributed with average value DT. We considered $DT = 1$, 5, and 10 s. For each value of DT, the file size S was set in such a way that S/DT is constant. As a consequence, the offered traffic load is approximately constant. In Fig. 16, when P_{Loss} is low and $DT = 1$ s, the performance of TCP-Peach and TCP-Reno + FS [36] is approximately equal, i.e., $\phi_{\text{Peach/Fast Start}} \approx 1$. Otherwise, the goodput of TCP-Peach is higher than the goodput of TCP-Reno + FS [36]. In Fig. 16, the value of $\phi_{\text{Peach/Fast Start}}$ increases when P_{Loss} increases because the Rapid Recovery algorithm becomes more effective. Also, we observe that when DT increases, then the value for the initial congestion window utilized by the Fast Start becomes obsolete and therefore, may not be appropriate for the current traffic load condition. As a result, the performance of Fast Start decreases as shown in Fig. 16.

B. Fairness

1) Homogeneous Case: We assume that all connections pass through the same path and run TCP-Peach. Let $acked_i(t)$ represent the number of segments acknowledged in the time interval $[0, t]$ for connection i, for $i = 1, 2, \ldots, N$. In Fig. 17, we show $acked_i(t)$ dependent on time t for $i = 1, 2, \ldots, N$, which are obtained by simulating the system in Fig. 10 with parameters $N = 10$, $K = 50$ segments, $rwnd = 64$ segments, $c = 1300$ segments/s, $P_{\text{Loss}} = 0$, $RTT = 550$ ms, and all connections using TCP-Peach. In Fig. 17, at any time t, $acked_{i'}(t) \approx acked_{i''}(t)$, for any i' and i''. This means that each TCP-Peach connection is given a fair share of the system resources. We obtained similar results using other values for system input parameters.

2) Heterogeneous Case: We consider the system shown in Fig. 18. There are M connections of type X and N connections of type Y. All of them pass through the link connecting the routers A and B, which is assumed to be the bottleneck. We assume that the capacity of this congested link is $c = 1300$ segments/s. The round trip time and loss probability are assumed to be RTT_j and $P_{\text{Loss, j}}$, respectively, for connections of type $j = X, Y$.

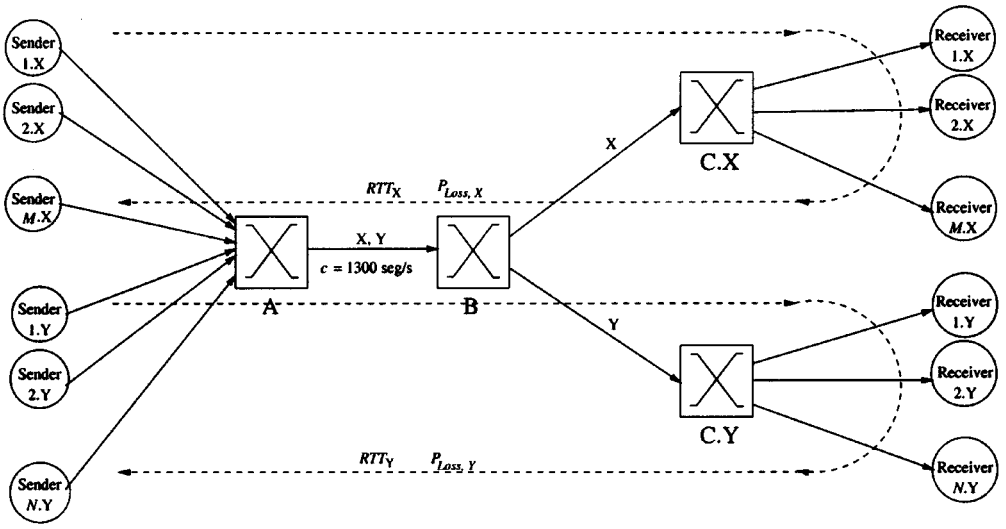

Fig. 18. Simulation scenario for the fairness evaluation in heterogeneous environment.

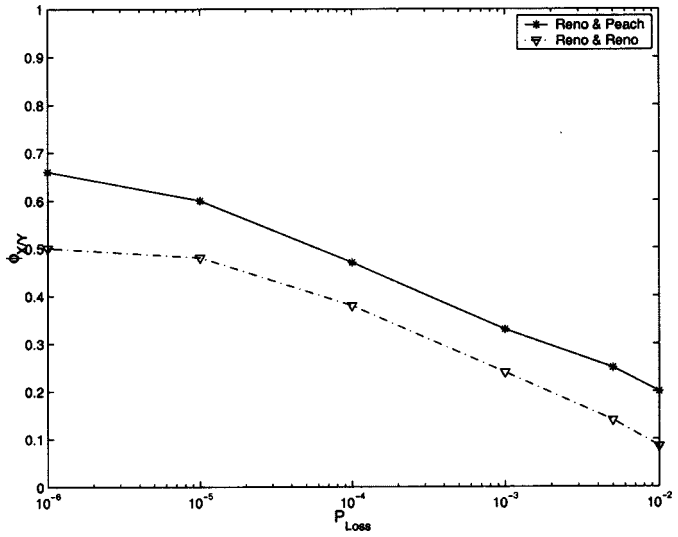

Fig. 19. Fairness evaluation in a heterogeneous scenario: satellite and terrestrial connections.

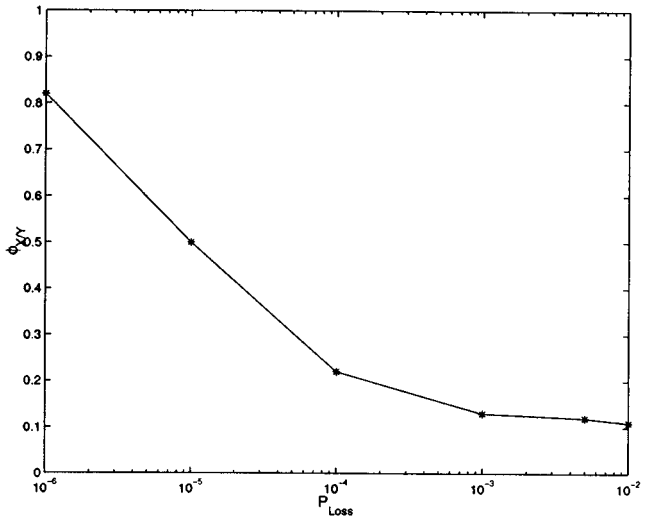

Fig. 20. Fairness evaluation in a heterogeneous scenario: TCP-Reno and TCP-Peach flowing through a GEO satellite channel.

In Figs. 19 and 20, (Peach) and (Reno) means that connections X use TCP-Peach and connections Y use TCP-Reno. We measure the fairness, ϕ, as the ratio between the goodput, r_X, of connections of type X, and the goodput, r_Y, of the connections of type Y, i.e.

$$\phi = r_X / r_Y. \tag{6}$$

It is obvious that the fairness becomes higher as ϕ approaches 1.

Using TCP-Peach for satellite connections, we achieve a more fair share of resources between terrestrial and satellite connections than the TCP-Reno. In Fig. 19, we assume that connections X pass through a GEO satellite link and thus their round-trip time is about $RTT_X = 550$ ms and their loss probability $P_{\mathrm{Loss,\,X}}$ is assumed to be $P_{\mathrm{Loss}} = 0$, 10^{-4}, 10^{-3}, and 10^{-2}. Connections Y are assumed to pass only through terrestrial links, thus no losses occur due to link errors, $P_{\mathrm{Loss,\,Y}} = 0$ and the RTT values are lower. We assumed $RTT = 200$ ms, such as the case of a connection between the mail server of the School of Engineering of the University of Catania, Italy, and the web server of the School of Electrical and Computer Engineering of the Georgia Institute of Technology, Atlanta. In Fig. 19, we show the fairness, $\phi_{\mathrm{Peach/Reno}}$, obtained in the (Peach) & (Reno) case, i.e., TCP-Peach is used for satellite connections and TCP-Reno is used for terrestrial connections. We also present the fairness, $\phi_{\mathrm{Reno/Reno}}$, obtained in the (Reno) & (Reno) case, i.e., TCP-Reno is used for both terrestrial and satellite connections. For the experiments in Fig. 19, we used $N = M = 5$, $K = 50$ segments and $rwnd = 64$ segments. In Fig. 19, we see that when TCP-Peach is used for satellite connections we obtain a higher fairness. This result was expected because TCP-Peach helps satellite connections to recover from their problems presented in Section II compared to terrestrial connections.

Now we evaluate the fairness when connections of type X using TCP-Peach and others of type Y using TCP-Reno flow

through the same path, i.e., a GEO satellite link. In Fig. 20, we show the fairness $\phi_{\text{Reno/Peach}}$. We observe that $\phi_{\text{Reno/Peach}}$ is much lower than 1, which is the approximate value for the fairness if all connections use TCP-Reno. However, note that when, for example, $P_{\text{Loss, X}} = P_{\text{Loss, Y}} = 10^{-2}$, in the (Reno) & (Peach) case the goodput for connections of type X and Y are $r_X = 12.1$ segments/s and $r_Y = 100$ segments/s, respectively, whereas in the (Reno) & (Reno) case we have $r_X \approx r_Y = 19.7$ segments/s. This means that in the (Reno) & (Peach) case we have a low decrease in r_X but a high increase in r_Y.

VI. Conclusion

In this paper, we introduced TCP-Peach, a new congestion control scheme improving the goodput performance and fairness in satellite networks. TCP-Peach is based on the use of dummy segments, which are low-priority segments that do not carry any new data to the receiver. TCP-Peach requires the routers along the connection to implement some priority mechanism at the IP layer. Priority can be supported at the IP layer by the Type of Service (TOS) option in the traditional IP, whereas IPv6 explicitly supports several priority levels. TCP-Peach contains two new algorithms: the Sudden Start and the Rapid Recovery, as well as the Congestion Avoidance and the Fast Retransmit as presented in [28] or [14], [15], [1].

The main features of TCP-Peach is that it only requires modifications in the end user behaviors and that it is compatible with traditional TCP implementations. If the receiver implements the SACK option [31], straightforward modification of TCP-Peach as presented here provides goodput performance improvement.

Acknowledgment

The authors would like to express their sincere thanks to D. Beering and G. Romaniak, as well as the Navy Research Laboratory for providing the physical testbed and helping with the experiments.

References

[1] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: Emulation and experiment," in *Proc. ACM SIGCOMM*, Aug. 1995, pp. 185–196.

[2] I. F. Akyildiz and S.-H. Jeong, "Satellite ATM networks: A survey," *IEEE Commun. Mag.*, vol. 35, pp. 30–43, July 1997.

[3] I. F. Akyildiz and I. Joe, "A new ATM adaptation layer for TCP/IP over wireless ATM networks," *ACM-Baltzer J. Wireless Networks*, vol. 6, no. 3, June 2000.

[4] M. Allman, S. Floyd, and C. Partridge, "Increasing TCPs initial window,", Internet RFC 2414, 1998.

[5] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over satellite channels using standard mechanism,", Internet RFC 2488, 1999.

[6] M. Allman *et al.*, "Ongoing TCP research related to satellites,", RFC 2760, Feb. 2000.

[7] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. 15th Int. Conf. Distributed Computing Systems (ICDCS)*, May 1995, pp. 136–143.

[8] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. ACM Mobicom*, Nov. 1995, pp. 2–15.

[9] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport protocol and handoff performance in cellular wireless networks," *ACM-Baltzer Wireless Networks J.*, vol. 1, no. 4, pp. 469–481, Dec. 1995.

[10] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, Dec. 1997.

[11] Y. Bernett *et al.*, "A framework for differentiated services,", Internet draft. draft-ietf-diffserv-framework-02.txt, Feb. 1999.

[12] G. Bianchi, A. Capone, and C. Petrioli, "Throughput analysis of end-to-end measurement-based admission control in IP," in *Proc. IEEE Infocom*, Mar. 2000.

[13] R. Braden, "Requirements for Internet hosts-communication layers,", STD 3, IETF 1122, Oct. 1989.

[14] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, Aug. 1994, pp. 24–35.

[15] L. Brakmo and L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1465–1480, Oct. 1995.

[16] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 850–857, June 1995.

[17] [Online]. Available: http://www.cisco.com.

[18] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, and N. Vaidya, "End-to-end performance implications of links with errors,", Internet draft, work in progress, Mar. 2000.

[19] S. Deering and R. Hinden, "Internet protocol version 6 (IPv6) specification,", IETF RFC 2460, Dec. 1998.

[20] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP extensions for space communications," in *Proc. ACM Mobicom*, Nov. 1996.

[21] S. Floyd. (1995, Feb.) TCP and successive fast retransmits. [Online]. Available: ftp://ftp.ee.lbl.gov/papers/fastretrans.ps

[22] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *ACM Computer Commun. Rev.*, vol. 26, no. 3, pp. 5–12, July 1996.

[23] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Trans. Networking*, vol. 7, pp. 458–472, Aug. 1999.

[24] R. Fox, "TCP big window and nac options,", Request for Comments 1106, IETF, June 1989.

[25] T. R. Henderson and R. H. Katz, "Transport protocols for Internet-compatible satellite networks," *IEEE J. Select. Areas Commun.*, vol. 17, pp. 326–344, Feb. 1999.

[26] J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," in *Proc. ACM SIGCOMM*, Aug. 1996, pp. 270–280.

[27] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, Aug. 1988, pp. 314–329.

[28] ——, "Congestion avoidance and control,", Tech. Rep., Apr. 1990.

[29] S. Keshav and S. Morgan, "SMART retransmission: performance with overload and random losses," in *Proc. IEEE Infocom*, Apr. 1997, pp. 1131–1138.

[30] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. Networking*, vol. 5, June 1997.

[31] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. (1996, Apr.) TCP selective acknowledgment options. [Online]. Available: ftp:ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-tcplw-sack-00.txt

[32] M. Mathis and J. Mahdavi, "Forward acknowledgment: Refining TCP congestion control," in *Proc. ACM SIGCOMM*, Aug. 1996, pp. 281–291.

[33] C. Metz, "TCP over satellite… The final frontier," *IEEE Internet Comput.*, pp. 76–80, Jan./Feb. 1999.

[34] G. Morabito, I. F. Akyildiz, and S. Palazzo, "TCP-Peach: Analytical model and performance evaluation," *Int. J. Satellite Commun.*, to be published.

[35] G. Morabito and I. F. Akyildiz, "TCP-Peach: Experimental results," Georgia Tech., Atlanta, GA, Tech. Rep., May 2000.

[36] V. N. Padmanabhan and R. Katz, "TCP Fast Start: A technique for speeding up web transfer," in *Proc. IEEE Globecom*, Nov. 1998.

[37] C. Partridge and T. J. Shepard, "TCP/IP performance over satellite links," *IEEE Network Mag.*, pp. 44–49, Sept./Oct. 1997.

[38] J. Postel, "DoD Standard Internet Protocol,", IETF RFC 760, Jan. 1980.

[39] W. Stevens, *TCP/IP Illustrated*. Reading, MA: Addison-Wesley, 1994, vol. 1.

Ian F. Akyildiz (M'86–SM'89–F'96) is a Distinguished Chair Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, and Director of the Broadband and Wireless Networking Laboratory. His current research interests are in wireless networks, satellite networks, and Internet.

He is the Editor-In-Chief of *Computer Networks Journal* (Elsevier), and an Editor for *ACM-Springer Journal for Multimedia Systems*, *ACM-Kluwer Journal of Wireless Networks (WINET)*, and *Journal of Cluster Computing*. He served as an Editor for IEEE/ACM TRANSACTIONS ON NETWORKING from 1996 to 2001, and for IEEE TRANSACTIONS ON COMPUTERS from 1992 to 1996. He was the program chair for the Ninth IEEE Computer Communications Workshop held in Florida in October 1994, the ACM/IEEE MobiCom'96 (Mobile Computing and Networking) conference, and for IEEE InfoCom'98. He will be the General Chair for the ACM/IEEE MobiCom'2002 and the Technical Program Chair for the IEEE ICC 2003.

Dr. Akyildiz is a Fellow of the Association for Computing Machinery (ACM). He received the Don Federico Santa Maria Medal for his services to the Universidad of Federico Santa Maria in Chile. He served as a National Lecturer for the ACM from 1989 until 1998 and received the ACM Outstanding Distinguished Lecturer Award for 1994. He received the 1997 IEEE Leonard G. Abraham Prize award for his paper entitled "Multimedia Group Synchronization Protocols for Integrated Services Architectures" published in the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS in January 1996.

Giacomo Morabito received the Laurea degree in electrical engineering and the Ph.D. degree in electrical, computer and telecommunications engineering from the Istituto di Informatica e Telecomunicazioni, University of Catania, Catania, Italy, in 1996 and 2000, respectively.

Since November 1999, he has been with the Broadband and Wireless Networking Laboratory of the Georgia Institute of Technology, Atlanta, as a Research Engineer. He serves on the Editorial Board of the Computer Networks Journal. His research interests include satellite and wireless networks, quality of service and broadband-network performance analysis.

Sergio Palazzo (M'92) received the degree in electrical engineering from the University of Catania, Catania, Italy, in 1977.

Until 1981, he was with ITALTEL, Milano, Italy, where he was involved in the design of operating systems for electronic exchanges. He then joined CREI, which is the center of the Politecnico di Milano for research on computer networks. Since 1987, he has been with the University of Catania, where he is now a Full Professor of Telecommunications Networks. In 1994, he spent the summer at the International Computer Science Institute (ICSI), Berkeley, CA, as a Senior Visitor. Since 1992, he has served on the Technical Program Committee of InfoCom, the IEEE Conference on Computer Communications. He was appointed as the General Vice Chair of the ACM MobiCom 2001 Conference. He will also be General Chair of the 2001 International Tyrrhenian Workshop on Digital Communications, focused on "Evolutionary Trends of the Internet." Since 1997, he has been a member of the Editorial Board of the *IEEE Personal Communications Magazine* and he was a Guest Editor of the Special Issue of the same journal on "Adapting to Network and Client Variability in Wireless Networks." Recently, he joined the Editorial Boards of the *Computer Networks Journal* and *Wireless Communications and Mobile Computing*. Also, he was a Guest Editor of the Special Issue of the IEEE JOURNAL OF SELECTED AREAS IN COMMUNICATIONS on "Intelligent Techniques in High-Speed Networks." His current research interests include mobile systems, broadband networks, intelligent techniques in network control, multimedia traffic modelling, and protocols for the next-generation Internet.