Buffer Control Strategies for the Transmission of TCP Flows over Geostationary Satellite Links Using Proxy–Based Architectures

Nicola Baldo, Andrea Odorizzi and Michele Rossi
University of Ferrara, Department of Engineering,
via Saragat 1, 44100, Italy.

***Abstract*— In this paper we consider the transmission of TCP flows over networks including geostationary satellite links. In this environment, due to the large delays and to the possibly high error rates introduced by the wireless channel, special protocols and architectures are required to obtain good performance. Our main goal in such a context is to efficiently manage the buffers that are usually placed between the cabled and the wireless part of the network [1] in order to harmonize the different bit rates that may be available over these different links. Many approaches based on proxies [2] [3], in fact, buffer TCP flows immediately before the wireless channel access and use dedicated transport protocols to optimize the transmission over the wireless links. However, while many contributions focus on the overall system performance and on the problem of maintaining the end-to-end semantic when TCP is terminated at proxy nodes, very few of them deeply investigate how buffers are to be managed in order to avoid congestion/losses while still be able to achieve good performance. This contribution is therefore aimed at filling this gap, by presenting buffer management techniques and evaluating them against previous solutions [4].**

I. Introduction

In this paper, we study the problem of providing good throughput and delay performance to TCP flows transmitted over geostationary satellite links. TCP assumes that the transmission over the channel is unreliable; various features such as timeout timers, packet reordering and retransmissions are used in TCP with the objective of providing a reliable channel to higher layer applications. However, these features have been designed to be effective over wired networks, where the main cause of packet error is due to network congestion. It is well known [5] that TCP is unable to properly react to wireless channel errors, where its throughput is strongly degraded. These scenarios include the case of heterogeneous networks including satellite links, where special countermeasures have to be taken to correct the inefficiencies of the TCP protocol. In this paper, as a possible remedy to the low performance of TCP over error-prone wireless links, we consider a TCP-split approach [3] [4]. The TCP-split architecture, that will also be considered here as a reference model to present the problem and state our solution, is presented in Fig. 1. The two communicating endpoints are fixed hosts exchanging packets over a path composed by wired and geostationary satellite links. A first host (leftmost side of the figure) transmits its data flow exploiting a TCP–like (LTL, see [4]) protocol to a first relay node, where such a flow is buffered and the TCP flow is terminated through a classical TCP-split approach. From this *relay buffer*, the flow is transmitted over the wireless satellite channel through a dedicated protocol (STL) which is aimed at increasing transmission and error recovery efficiency over the wireless link. Finally, the flow is gathered by a second relay node and transmitted to the final user (rightmost side of the figure) exploiting again a TCP–like flow control algorithm. The rationale behind this approach is to separate the different portions of the network, by optimizing the end-to-end transmission of the data flow thanks to the selection of the most suitable error control mechanism for each network segment. In particular, TCP is used as the transport protocol responsible for error recovery over wired links, whereas a dedicated SR–ARQ like protocol is used over the wireless satellite channel. The goal of this paper is to achieve the buffer control at the relay nodes [1] [4], i.e., to devise efficient techniques for the relay buffer occupancy management. In particular, we want to avoid the buffer overflow problem at the relay buffer while still providing good performance. In order to illustrate the problem, let consider the leftmost relay in Fig. 1. Further, assume that the bandwidth available over the wireless link (B_{sat}) is much lower than the one available over the wired connections (B_{wired}). In this case, it is clear that the buffer at relay 1 (Fig. 1) eventually saturates. This will lead to buffer overflow problems and subsequent packet losses. Moreover, it is noticeable that such packet losses are not being recovered as they are not in the TCP scope, which is instead acknowledged right before the insertion of the packets into the buffer [4]. These packet losses would therefore lead to considerable performance degradations since the only way in which they can be recovered is by invoking the intervention of the Upper Transport Layer (UTL) [4], which is a further protocol, placed on top of TCP and responsible for the correctness of the end-to-end transmission. The error recovery pursued by the UTL is however expensive in terms of time, due to the large delays characterizing the geostationary satellite links [3] [4] and to the inherent latencies associated to the UTL error control mechanism. These overflow problems are therefore to be avoided, by possibly providing ways to estimate the increased buffer occupancy and take actions such as stopping the incoming TCP flow in advance, i.e., before that a buffer overflow event will actually occur.

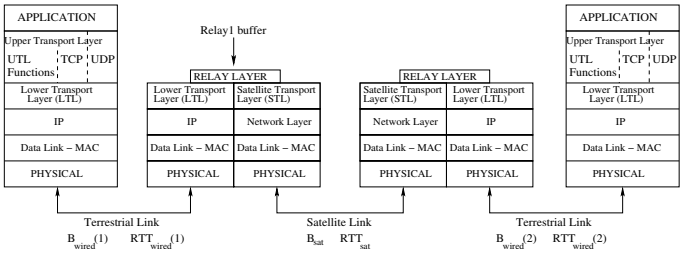

Fig. 1. End-to-end PETRA architecture.

The remainder of this paper is organized as follows. In Section II, we report the channel models that we considered to carry out the performance evaluation. In Section III, we review the buffer control algorithm proposed in [4], whereas two

novel mechanisms for buffer control are proposed in Sections IV and V. The performance evaluation is presented in Section VI, where the three buffer control algorithms are compared with each other. Finally, in Section VII we report the conclusions of our work.

II. Channel Models

For what concerns the terrestrial (wired) links, we consider an independent packet error probability. That is, TCP/IP packets are marked as erroneous with probability $P_e(IP)$. For the wireless channel, we considered the time slotted, where the slot duration corresponds to the transmission time of a single satellite link layer packet (LL PDU). To track the LL PDU error process, we used a discrete-time two-state Markov chain [6] with average error rate and burst length equal to ϵ and b, respectively. We calculated the PDU error probability ε from the bit error probability ε_b considering i.i.d. errors. ε_b and b were given as input to the simulator. As our goal is to avoid buffer overflow at the relay nodes, in the paper we will consider a very low IP packet error probability $P_e(IP) = 10^{-9}$. This is, in fact, the worst case for the relay buffer occupancy (refer to the leftmost buffer in Fig.1).

III. Review of the PETRA Satellite Buffer Control Scheme

A possible solution to this problem was proposed in [4]. In this approach the authors use a suitable buffer threshold B_{th} and stop the backward TCP acknowledgment (ACK) flow over the first (leftmost) wired channel as the buffer occupancy grows beyond B_{th}. As a reaction to that, the TCP running over the first wired link is forced to stop its outgoing flow as, due to the lack of acknowledgments, it window eventually results to be exhausted. The key idea is therefore to indirectly stop the TCP flow on the forward direction by stopping backward acknowledgments. The flow is then restored as the buffer occupancy decreases below B_{th}. In [4], the authors give an expression for B_{th} in order to minimize the probability of having TCP timeout events, while still achieving good performance

$$B_{th} \geq \left\lceil \frac{B_{sat} RTT_{wired}}{PKT_{len}} \right\rceil \quad (1)$$

where B_{sat} is the available bandwidth over the satellite channel, RTT_{wired} is the round trip time over the wired link and PKT_{len} is the number of bits needed to transmit an entire TCP segment over the wireless channel. Note that, with this approach we also have to carefully select the dimension of the buffer size B_{size} in order to avoid overflow events due to the inherent latency associated to the TCP reaction to the ACK stopping. In fact, at least $RTT_{wired}/2$ seconds are needed for the stopped ACK flow to have effect on the TCP transmission at the sender side. During these $RTT/2$ seconds, the relay node may receive additional packets that are *in flight* over the channel at the time the ACK flow is stopped. The buffer should be therefore overdimensioned to the worst case in order to accommodate such packets and therefore avoid overflow events.

We outline that the PETRA scheme is not free of the timeout problem at the TCP sender. In fact, when the ACK flow is stopped for too a long period (due to, e.g., bad channel conditions on the satellite link), the TCP sender reacts by invoking the timeout and retransmitting. This, in turn, causes a substantial performance degradation. The buffer management schemes that we propose in the following aim at solving this problem.

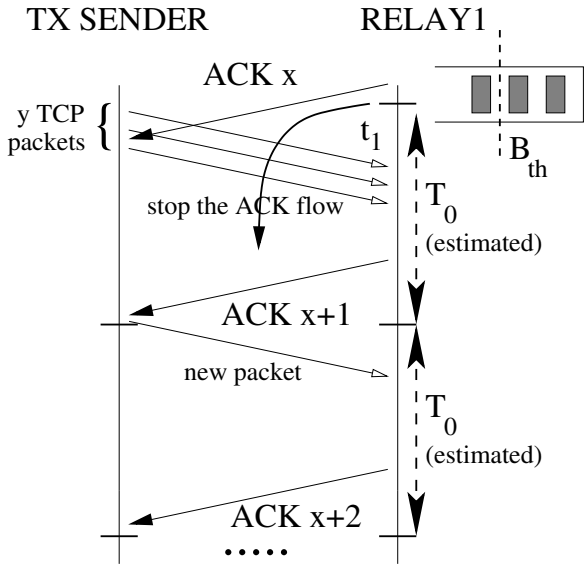

Fig. 2. Illustration of the mechanism implemented by the ACKRES scheme.

IV. ACKRES Satellite Buffer Control Scheme

In this paper, we go further with respect to previous work by presenting two different approaches to manage the TCP flow and buffer occupancy. The goals of the present contribution are to provide solutions where the knowledge of system parameters such as round trip time (RTT_{wired}) or satellite bandwidth (B_{sat}) do not have to be known. Moreover, these schemes should still be able to avoid buffer overflow problems and possibly improve the buffer utilization factor.

The first scheme that we present to accomplish to these goals is called ACK REServation (ACKRES). ACKRES still make use of the threshold B_{th} and it is aimed at further limiting timeout events of the TCP operating over the wired link. In order to explain how the scheme works consider Fig. 2. Assume that at time t_1 the relay1 buffer (Fig. 1) occupancy grows beyond the threshold value B_{th}. As a consequence, the buffer management scheme immediately stops the backward TCP ACK flow and starts tracking incoming TCP packets. These packets, as introduced above, are the *in flight* TCP segments at time t_1. In the example depicted in Fig. 2, we have that $y = 3$ TCP segments arrive after the stopping time t_1. Assume also that the last TCP ACK sent by ACKRES before time t_1 carries the acknowledgment number x. ACKRES keeps an estimate of the timeout expiration time at the TCP sender. As this expiration time approaches, ACKRES sends back a single new ACK (ACK $x + 1$), which is relative to the first (unacknowledged) packet received after the ACK stopping event and whose function is to refresh the timeout timer at the sending TCP, thereby avoiding its expiration. In particular, in order for this mechanism to be effective, the relay node has to send this acknowledgement at least $RTT_{wired}/2$ seconds before the timeout expiration time at the sender. The same mechanism applies for every further timeout event until the rely buffer occupancy decreases below B_{th} and the ACK flow can be completely restored, as for the scheme in Section III.

We stress the fact that obtaining an accurate estimate of the timeout at the sending entity may be difficult. In fact, the actual timeout at the sender depends on both the error events encoun-

tered by TCP and by the type of estimator adopted by the particular version of TCP in use. The relay buffer should therefore run a timeout estimator which is based on the incoming TCP traffic. Again, setting up this estimation may not be a trivial task as the timeout is not updated at regular intervals [7]. As will be discussed in the sequel, this algorithm presents some serious drawbacks. In this paper, we will present performance results for the case where the timeout estimate is perfect ($\Delta T_0 = 0$) and we will discuss the behavior that we found for the case where it is biased by a constant quantity $\Delta T_0 \neq 0$.

V. RWND-Based Satellite Buffer Control Scheme

In this section we present a third algorithm to control the incoming TCP flow and avoid buffer overflows. We control the buffer occupancy by measuring, each time a new TCP ACK is sent, the remaining space in the buffer B_{free} and delivering such an information to the TCP sender through TCP ACKs. To this end, we utilize the receiver advertised window ($rwnd$) field in the TCP ACK headers. Hence, we synchronize the TCP ACK flow $rwnd$ field with the remaining space in the buffer. In the limiting case, $B_{free} = 0$ and a TCP ACK including a $rwnd = 0$ is sent back to the transmitter. The sender, after receiving this ACK, freezes its status, by therefore stopping the downcounting of the timeout timer. By this way, we prevent the TCP sender to timeout by *forcing* it to decrease its transmission rate according to the still available space in the relay buffer and, in the limiting case, by putting it into the freeze mode.

Moreover, in order to prevent the TCP sender to remain into the *freeze* situation indefinitely, we implemented an *anti–freeze* mechanism working as follows. As soon as the relay node sends an ACK with a zero $rwnd$ field, it enters the so called *freeze state*, where it is forced to periodically check the buffer state and, in the case where $B_{free} > 0$, to send a (*forced duplicate*) ACK with $rwnd = B_{free}$. In the case where $B_{free} = 0$, nothing is transmitted and the buffer check is rescheduled. The relay node remains into the *freeze state* until a new TCP data packet is eventually received. In this case, the state is switched back to *normal*, the forced periodic buffer check is stopped and B_{free} is mapped again into the ACKs *rwnd* field. *We stress that this unfreeze algorithm is necessary and at the same time very effective to restore the TCP flow.*

In the next we prove that this strategy always avoids overflow events to occur at the relay node. TCP packets are transmitted in rounds. Consider the generic round $n \in \mathbb{N}^+$ and, in particular, the last incoming ACK relative to the TCP packets transmitted by the sender in round $n-1$. Assume that this ACK arrives at the transmitter at time t_n, and refer to the acknowledgement number contained in this ACK as $ACK(t_n)$. Moreover, let $rwnd(t_n)$ and $LBSN(t_n)$ be the receiver window advertized by the ACK at time t_n and the next octet to be transmitted over the satellite channel by the relay node at time t_n, respectively. Further, let us indicate with $HTSN(t_n) = H(t_n) + 1$, where $H(t_n)$ is the sequence number associated with the highest octet that can be transmitted by the sender at time t_n. Now, if we assume that a buffer overflow occurs at time $t > t_n$, i.e., due to the packets transmitted in the n-th round, we must have that:

$$HTSN(t_n) - LBSN(t) > B_{size} \tag{2}$$

where B_{size} is the relay1 buffer size. Furthermore, as $LBSN(\cdot)$ is by definition non-decreasing as a function of the time t, we have that $\forall\ t > t_n$, $LBSN(t) \geq LBSN(t_n)$ and therefore Eq. (2) implies:

$$HTSN(t_n) - LBSN(t_n) > B_{size} \tag{3}$$

But according to the way in which $rwnd$ is computed at the relay node we have that:

$$B_{size} = rwnd(t_n) + \{ACK(t_n) - LBSN(t_n)\} \tag{4}$$

Hence, from Eqs. (3) and (4) above we can write:

$$HTSN(t_n) > ACK(t_n) + rwnd(t_n) \tag{5}$$

But we reach an absurd as by definition $HTSN(t_n) = ACK(t_n) + \min(cwnd(t_n), rwnd(t_n))$, where $cwnd(t_n)$ is the congestion window at the TCP sender at time t_n.

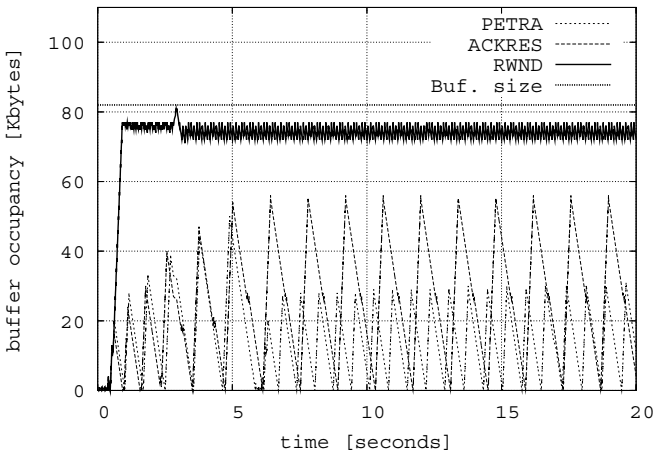

Fig. 3. Buffer occupancy as a function of time, $\varepsilon_b = 10^{-5}$, i.i.d. channel.

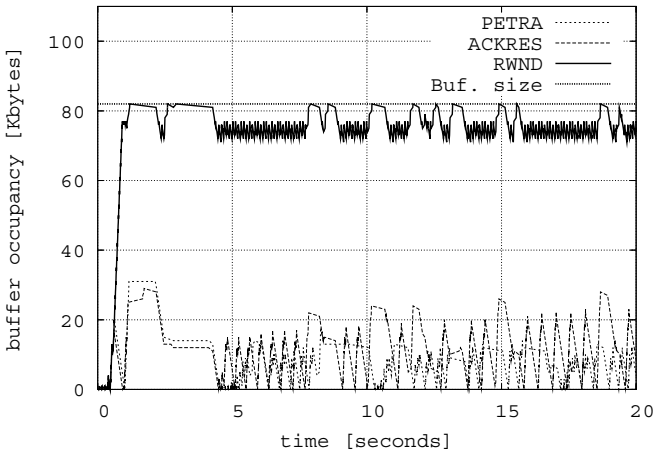

Fig. 4. Buffer occupancy as a function of time for $\varepsilon_b = 10^{-3}$, i.i.d. channel.

VI. Performance Evaluation

In this section, we present some performance measures to assess the effectiveness of the proposed solutions for buffer and flow control. In the first two figures 3 and 4, we report the

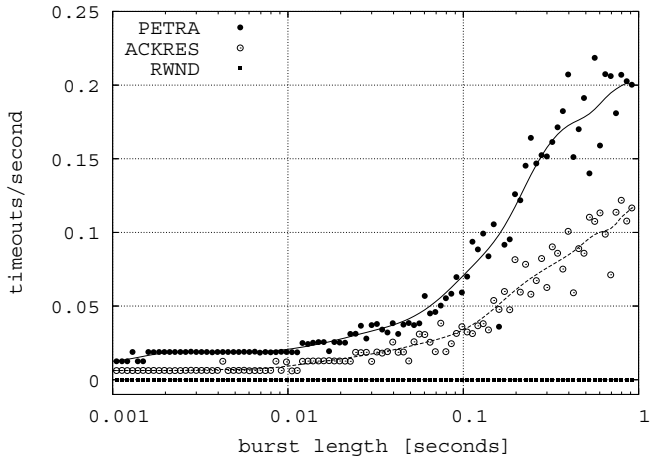

Fig. 5. Timeout frequency as a function of the error burst length b over the wireless link for $\varepsilon_b = 10^{-3}$.

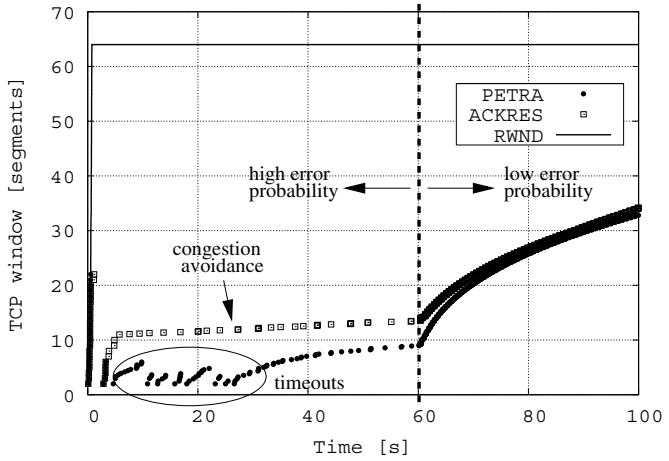

Fig. 6. TCP window trace.

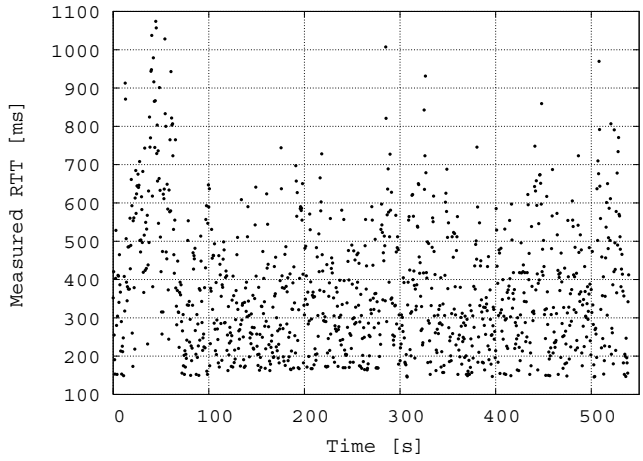

Fig. 7. Measured RTT_{wired} trace used for the second performance test.

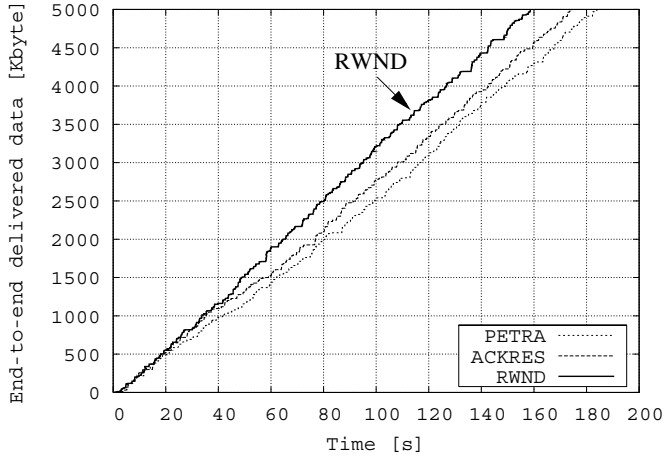

Fig. 8. Trace of the end-to-end delivered data, $\varepsilon_b = 10^{-3}$, i.i.d. channel.

buffer occupancy as a function of time considering an independent error satellite channel for two different values of the bit error probability $\varepsilon_b \in \{10^{-3}, 10^{-5}\}$ over the wireless link. From these figures the advantages offered by the RWND scheme are clear: the timeout event is always avoided, while the buffer size is used very efficiently, by almost covering its maximum capability. B_{size} in these simulations has been set to 82 TCP segments, the TCP packet size is 1 Kbyte, the bandwidth over the satellite (B_{sat}) and the terrestrial links (B_{wired}) are equal to 512 Kbps and 2 Mbps, respectively. The round trip delays over wired and wireless links are $RTT_{wired} = 100$ ms and $RTT_{sat} = 0.5$ s, respectively. These settings will be used for all the measurements reported in the paper.

From these figures it can also be observed that the oscillatory behavior on the buffer occupancy, which is present for both the PETRA and the ACKRES schemes, is largely reduced for RWND. This algorithm is very robust also at high error rates ($\varepsilon_b = 10^{-3}$). In Fig. 5, we plot the frequency of timeout events as a function of the wireless channel burst length by considering $\varepsilon_b = 10^{-3}$ and using the two–state Markov model described in Section II to simulate the wireless link error behavior. As can be observed from the figure, using RWND timeout events are successfully avoided for any considered duration of the channel error burstiness (b).

It shall be observed that ACKRES can also effectively reduce the timeout event probability (Fig. 5). However, we verified that the correct estimate of the timeout expiration time is pivotal for the algorithm to be effective. In particular, when this estimate is affected by errors its performance scheme *strongly degrades* and, in such cases, this scheme performs even worse than PETRA. On the one hand, as the timeout bias $\Delta T_0 > 0$, ACKRES tends to perform as PETRA. In the limiting case ($\Delta T_0 = +\infty$) the timeout refreshing algorithm is never invoked and ACKRES and PETRA coincides. On the other hand, when $\Delta T_0 < 0$, TCP ACKs are sent in advance and if ΔT_0 decreases below a given threshold, the whole TCP/buffer system becomes unstable. That is, our ACKRES algorithm behaves as a self-clocking mechanism which ends up in continuously triggering new ACKs. In this case, the relay buffer is continuously filled by leading to numerous timeout events. These arguments explain why it is not recommended to implement this type of algorithm. In fact, not only the TCP timeout may be hard to be estimated, but estimation errors could also lead to unstable behaviors.

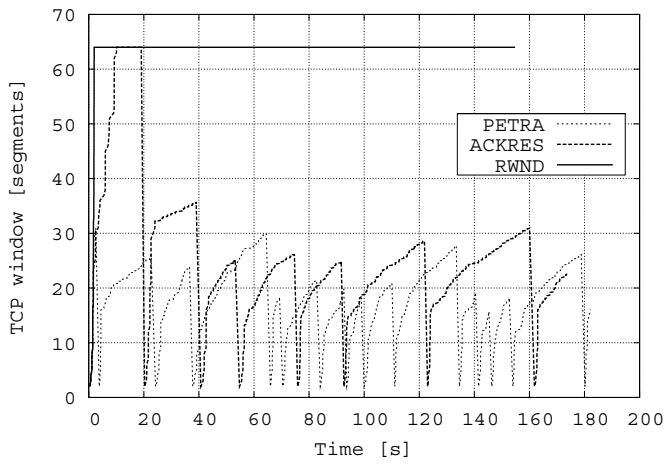

Fig. 9. TCP window trace, $\varepsilon_b = 10^{-3}$, i.i.d. channel.

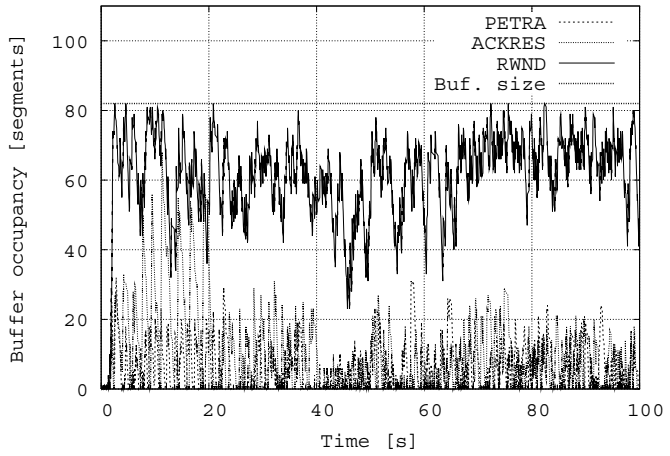

Fig. 10. Relay buffer1 trace, $\varepsilon_b = 10^{-3}$, i.i.d. channel.

In the sequel, we proceed with our performance evaluation by discussing the following two tests:

1) In the first performance test, we consider the case where the wireless channel sharply transits between two very different error rates. In particular, we consider a simulation of 100 seconds, where from time 0 to time 60 s $\varepsilon_b^1 = 10^{-3}$, whereas after time $t^* = 60$ s, the bit error probability suddenly drops to $\varepsilon_b^2 = 10^{-12}$. This may describe the situation of a mobile satellite connection, where the Line of Sight (LoS) component of the signal is subject to sharp variations due to obstacles.
2) In the second test, we consider a constant bit error rate $\varepsilon_b = 10^{-3}$, whereas we introduce some variability on the round trip time RTT_{wired} concerning the leftmost terrestrial connection (Fig. 1). In Fig. 7 we report the RTT trace used to carry out this second test. This trace has been obtained by ping measurement performed from an ADSL home Internet connection to the site `http://www.google.it`.

In Fig. 6 we report the TCP window ($cwnd$) trace as a function of time by comparing the three buffer control algorithms and focusing on the first test. Noticeably, before $t^* = 60$ seconds, the TCP sender is strongly impacted by the high error rate ε_b^1. In both PETRA and ACKRES, the backward ACK flow is often stopped thereby leading to frequent timeouts at the sender side. This reduces both the TCP congestion window and the window threshold, by therefore limiting the promptness of TCP in the subsequent phase ($t > t^*$, i.e., when the channel error rate drops to 10^{-12}). The RWND based scheme is instead very robust as, thanks to the freeze mechanism, TCP state variables are not degraded.

In Figs. 8,9 and 10 we focus on the second test. Again, the combination of high error rates ($\varepsilon_b = 10^{-3}$) with the high variability of RTT_{wired} causes frequent timeouts for both PETRA and ACKRES. However, even in such a harsh environment RWND performs well, by avoiding spurious timeouts (Fig. 9). The effect of the higher RWND performance on the end-to-end data transfer can be seen from Fig. 8, where we report the number of end-to-end delivered TCP/IP segments as a function of time. In this scenario, RWND leads to time savings of about 20 seconds for a 5 Mbytes file transfer. By comparing Fig. 10 with Fig. 5, we also observe that a highly variable round trip delay also impacts the buffer occupancy behavior, which is affected by a larger variability. Nevertheless, RWND succeeds in staying close to the maximum buffer size, thereby efficiently exploiting the available resources. In fact, a larger buffer occupancy means an immediate availability of fresh data to be transmitted over the wireless channel when its error rate suddenly decreases. This also corresponds to a reduced probability to leave the wireless channel underutilized.

VII. Conclusions

In this work we focused on buffer control management strategies to enhance the performance of TCP operating over TCP-split architectures, for its transmission over geostationary satellite links. In the paper, we first reviewed recently proposed buffer control solutions and then we proposed two novel schemes, where the first one (ACKRES) tries to minimize spurious timeouts at the transmitter by pro-actively refreshing the timeout timer at the sender. The second algorithm, instead, realizes the buffer control by properly tuning the receiver advertized window fields in the backward TCP ACKs. ACKRES presents some serious drawbacks, in particular, when the timeout at the TCP sender can not be perfectly estimated it leads to unexpected behaviors and the system performance is strongly impacted. Instead, we found the second algorithm (RWND) to be very effective under any setting.

References

[1] C. Barakat, N. Chaher, W. Dabbous, and E. Altman, "Geostationary Satellite Links," in *Proceedings of IEEE Globecom*, Rio, Brazil, Dec. 1999.
[2] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "IETF RFC3135: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," June 2001.
[3] T. R. Henderson and R. H. Katz, "Transport Protocols for Internet-Compatible Satellite Networks," *IEEE J. Select. Areas Commun.*, vol. 17, no. 2, pp. 326–344, Feb. 1999.
[4] M. Marchese, M. Rossi, and G. Morabito, "PETRA: Performance Enhancing Transport Architecture for Satellite Communications," *IEEE JSAC*, vol. 22, no. 2, pp. 320–332, Feb. 2004.
[5] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, and N. Vaidya, "IETF RFC 3155: End-to-end Performance Implications of Links with Errors," Aug. 2001.
[6] M. Zorzi, R. Rao, and L. Milstein, "Error Statistics in Data Transmission over Fading Channels," *IEEE Trans. Commun.*, vol. 46, no. 11.
[7] W. R. Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*, N. Y. Addison Wesley, Ed., 1994.