PERFORMANCE EVALUATION OF END-TO-END TCP ENHANCEMENTS FOR SATELLITE LINKS

Carlo Caini and Rosario Firrincieli

DEIS/ARCES, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy,
Email: ccaini@deis.unibo.it, rfirrincieli@arces.unibo.it, tel: +390512093062, fax: +390512095410.

ABSTRACT

Internet satellite communications are impaired by long RTTs and by segment losses on the satellite radio links. To cope with these problems, common to wireless internet communications, several TCP enhancement proposals have been presented in literature. The paper aim is to investigate the effectiveness of such modifications, when applied on TCP satellite connections. The focus is on two promising techniques, namely the TCP Westwood and the TCP Hybla (developed by the authors), which can be considered to some extent complementary, as they aim make TCP more robust against channel losses and long RTTs effects, respectively. Their performance is compared with NewReno plus SACK and Vegas, considering the presence of congestion and/or a non-null packet error rate on a reference network topology. Throughput, fairness and friendliness are the performance metrics, evaluated by means of ns-2 simulations. Results show that both techniques are able to introduce great advantages on TCP standard.

1. INTRODUCTION

Internet satellite communications have to face some peculiar challenges, due to the presence of a wireless link on the TCP connections. First, the Round Trip Time (RTT) is greatly increased by the long propagation time on the satellite radio channel, especially when MEO and GEO constellations are considered (for GEO, RTT ≅ 300 ms, considering the forward link via satellite and the return link via a fast terrestrial network, 600 ms if both the forward and the return links are via satellite). Second, the presence of random losses not originated by congestion cannot be considered negligible. Both these aspects pose severe threats to the transport layer performance and require the adoption of adequate countermeasures [1][2][3].

It is well recognized that the performance of standard TCP protocols (Tahoe, Reno and NewReno) largely depends on network delays, with significant throughput degradation for long RTT connections (e.g. satellite). This “unfairness” against longer RTT connections is due to the standard congestion control mechanism that increases the congestion window, cwnd, (and consequently the segment transmission rate) on an RTT time base, instead of on an absolute time one, as it would be preferable. Given that the congestion control is performed by the transport layer, this problem can only be addressed at this level.

By contrast, random losses due to radio channel can be reduced by enforcing the reliability of lower layers. However, more robust physical and link layers require an increase of complexity, a possible lower spectrum efficiency (if the FEC redundancy is increased) and also the introduction of additional delays (frame retransmissions at link layer, large interleaving and decoding time at physical layer), so that a trade off between them and the TCP Packet Error Rate (PER) must be pursued. It is worth noting that radio channel losses mainly affect performance in an indirect way, because of the inherent inability of TCP to distinguish them from congestion losses. As a result, the cwnd may be repeatedly halved in presence of channel losses, with a further penalization of longer RTT connections, unable to fast re-open the cwnd. The consequences may become dramatic on satellite connections, where a PER=1% may result in a throughput reduction of more than an order of magnitude, as it will be shown in the paper.

A number of interesting contributions has been presented in literature to solve the aforementioned problems. Among the potential solutions, we are specifically interested in the enhancement proposals that preserve the end-to-end semantics of TCP. In particular, the paper aims at evaluating the performance of TCP Westwood [4] and TCP Hybla [5][6], the latter recently presented by the authors. The rationale for this choice is that these proposals address two different but potentially complementary aspects of TCP. In particular, TCP Westwood specifically aims at limiting the negative effects of channel losses, through a bandwidth estimation that eliminate the standard cwnd halving, while TCP Hybla is mainly focused on solving the performance disparity which arises from long RTTs, by means of a set of procedures which includes the speed up of the cwnd increase rate and the use of packet spacing. As both the proposals have already been discussed in the literature, the focus of the paper is on the performance evaluation, assessed by means of the ns-2 simulator. Results refer to the presence of congestion and/or a non-null packet error rate on a reference network topology. Both techniques are evaluated alone and in comparison with several TCP variants (i.e.
NewReno plus SACK and Vegas), considering throughput, fairness and friendliness as performance metrics.

2. TCP VARIANTS

In the following, a brief description of the TCP variants considered in the numerical evaluations is presented, referring to the literature for a comprehensive description.

2.1 TCP NewReno

As TCP NewReno is the most widely adopted TCP version, it will be considered as a standard reference when describing the other TCP variants. In brief, its mechanism is the following. When a new TCP connection is established, the sender probes for bandwidth availability by gradually increasing the congestion window value \( W \). In the Slow Start (SS) phase, starting from an initial value \( W_0 \), typically equal to one or two maximum segment size (MSS), the congestion window is increased by MSS bytes per every non-duplicate ACK received. When the congestion window reaches the slow start threshold (ssthresh), the source switches to the Congestion Avoidance (CA) phase, during which the congestion window is increased by \( \text{MSS} / W \) bytes per every new ACK received [7]. In a real channel this rise continues until either the size of receiver buffer (advertised window) is reached, or a segment is lost. By expressing the value \( W \) of the congestion window in MSS units, the standard cwnd update rules previously described are given by

\[
W_{i+1} = \begin{cases} 
W_i + 1 & \text{SS} \\
W_i + 1/W_i & \text{CA}
\end{cases}
\]

where the index \( i \) denotes the reception of the \( i \)th ACK. Note that the actual transmission of TCP segments occurs in relation to a window size given by \( \text{min} (\text{cwnd}, \text{advertised window}) \).

When a loss occurs TCP NewReno triggers the Fast Retransmit and Fast Recovery procedures [8]. Let us start assuming that the segment \( n \) is dropped. As a consequence, a new duplicate acknowledgement (dupACK) for the segment \( n-1 \) is generated by the receiver every time a new segment arrives. The reception of the third dupACK triggers in the sender the Fast Retransmit and the Fast Recovery procedures, starting the recovery phase. First, the segment \( n \) is retransmitted, then the ssthresh is updated to max (FlightSize / 2, 2 MSS), where FlightSize represents the number of unacknowledged packets when the loss was detected. The cwnd value is stored (old_cwnd), and the new cwnd value is given by ssthresh plus 3 MSS. Then, the cwnd is incremented by a MSS every two (or one) additional dupACKs received (“cwnd inflating”), resulting in the transmission of a new segment if cwnd exceeds the old_cwnd.

The reception of the first non-duplicate ACK may determine two consequences. If the ACK is “partial”, i.e. if it confirms only a part of the segments in flight, the recovery phase is not terminated, the first unacknowledged segment is retransmitted, the cwnd is deflated by the amount of the data acknowledged, and a new segment is sent if allowed by the new cwnd value. Moreover, the first partial ACK arrival during the recovery phase resets the retransmission timer. If, instead, the ACK confirms all the packets in flight, the recovery phase ends, the cwnd is deflated to the ssthresh value, and the sender restarts transmission in congestion avoidance phase. Finally, it is worth noting that if multiple losses occur in the same window, NewReno remains in the recovery phase until either all the lost packets have been recovered or a Retransmission Time Out (RTO) expires. As NewReno is able to recover only one packet per RTT, the time spent in the recovery phase can become excessive [8] for long RTTs penalizing the wireless connections throughput.

2.2 TCP SACK

To overcome the TCP NewReno incapacity of recovering multiple losses in the same window, in [9] the selective acknowledgment (SACK) option was proposed. By means of the SACK procedure, the receiver can inform the sender about all the segments that have been successfully delivered, thus allowing the sender to recover more than one packet per RTT. In more detail, the receiver indicates in the SACK option field of each ACK segment a number of SACK blocks (usually three), where each SACK block reports a different data interval that has been received and buffered. The first block always reports the last interval of contiguous received segments, while the additional blocks report the last preceding intervals. In [8], the performance of TCP Reno with SACK option and TCP NewReno were compared by means of simulations, and the SACK ability to recover from multiple losses in a shorter time was confirmed.

2.3 TCP Vegas

TCP Vegas was proposed as a variant of the TCP NewReno aiming at preventing congestion losses by means of a dynamic estimate of the available bandwidth [10]. This protocol tries to detect changes in the sending rate by comparing the measured throughput rate with the expected one. First, it estimates BaseRTT as the RTT of a segment when the connection is not congested, in practice, the minimum of all measured round trip times. As a result, the
expected throughput is given by \( \text{cwnd} / \text{BaseRTT} \). Second, TCP Vegas calculates the current Actual sending rate by recording how many bytes are transmitted between the time that a segment is sent and its acknowledgment is received, computing the RTT and dividing the number of bytes transmitted by the sampled RTT. This calculation is done once per round-trip time. Finally, TCP Vegas compares Actual to Expected, and adjusts the window accordingly. Let assume \( \text{Diff} = \text{Expected} - \text{Actual} \) and \( \alpha \) and \( \beta \) two thresholds so that \( \alpha < \beta \). When \( \text{Diff} < \alpha \), TCP Vegas increases the congestion window linearly during the next RTT; when \( \text{Diff} > \beta \), it decreases the congestion window linearly during the next RTT; the congestion window is unchanged when \( \alpha < \text{Diff} < \beta \).

TCP Vegas presents also a number of minor modifications with respect to TCP NewReno, for example the use of timestamps to avoid spurious Retransmission Time Outs (RTOs), and a slight modification to the Slow Start cwnd update rule.

### 2.4 TCP Westwood

TCP Westwood [4] has been introduced for the purpose of limiting the consequences of the losses introduced by a wireless channel, which are always erroneously ascribed to congestion by the TCP protocol. To this end, TCP Westwood introduces a modification of the Fast Recovery algorithm called Faster Recovery. In contrast with the TCP NewReno, which halves the congestion window after three duplicate ACKs and “blindly” fixes the slow start threshold to this value, TCP Westwood attempts to select a slow start threshold more consistent with the actual available bandwidth. In this way, channel losses do not cause the dramatic slow down of the transmission rate typical of the standard TCP versions. The bandwidth is estimated by measuring and averaging the rate of returning ACKs. TCP Westwood makes use of a discrete-time filter to perform the estimate of the available bandwidth; the value of the filtered available bandwidth \( \hat{b}_t \) at time \( t = t_0 \) is given by:

\[
\hat{b}_t = \alpha_k \hat{b}_{t-1} + (1 - \alpha_k) \left( \frac{b_t + b_{t-1}}{2} \right)
\]

where \( b_t \) is the sampled value of the available bandwidth, \( \alpha_k = (2\tau - \Delta k)/(2\tau + \Delta k) \) and \( 1/\tau \) is the cutoff frequency of the filter. The ssthresh and cwnd (in MSS) are then calculated as

\[
sstresh = \hat{b} \ast \text{RTT}_{\text{min}} \]

\[
cwnd = \begin{cases} 
\text{ssthresh} & \text{if cwnd} > \text{ssthresh} \\
1 & \text{after a time out}
\end{cases}
\]

### 2.5 TCP Hybla

TCP Hybla [5] [6] has been conceived by the authors with the aim of eliminating the performance disparity that arises from long RTTs. It consists of a set of procedures which includes the modification of the cwnd increase rate in both the SS and the CA phases, the channel bandwidth estimation, the mandatory adoption of the SACK policy and timestamps, the recommended use of packet spacing.

The basic idea of the TCP Hybla is to obtain for long RTT connections (e.g. satellite) the same instantaneous segment transmission rate, i.e. the amount of segments transmitted per second, of a comparatively fast reference TCP connection (e.g. wired). The adoption of a continuous model [11] to describe the congestion window evolution in time \( W(t) \) made it possible to rigorously define the segment transmission rate as,

\[
B(t) = \frac{W(t)}{\text{RTT}}
\]

Eq. (4) shows that in order to achieve the goal of a transmission rate independent of the actual RTT, a faster \( W(t) \) increase rate for long RTT connections is required, while the contrary happens in standard TCP. By denoting with \( \text{RTT}_0 \) the round trip time of the reference connection to which we aim to equalize the performance, it is useful to introduce a normalized round trip time, \( \rho \), as

\[
\rho = \frac{\text{RTT}}{\text{RTT}_0}
\]

Then, in [5] and [6] it is shown that the same \( B(t) \) of the reference connection can be achieved even by longer RTT connections provided that the standard congestion control laws (1) are replaced by the following,

\[
W_{i+1} = \begin{cases} 
W_i + 2^\rho - 1 & \text{SS} \\
W_i + \rho^2/W_i & \text{CA}
\end{cases}
\]
Note that the CA update rule is similar to the Constant-Rate algorithm [14], which, on the other hand, didn’t consider the slow start phase. Another important difference is that in the actual implementation of (6), TCP Hybla sets to 1 the minimum value for $\rho$, in order not to slow down the connections that are already “fast” (i.e. connections with $RTT<RTT_0$). In addition to (6), in case of long RTT it is also useful (although not mandatory) to appropriately increase the advertised window at the receiver side. Bandwidth estimation is exploited in order to appropriately set the initial $ssthresh$.

As the congestion control algorithm of TCP Hybla is much more efficient of the standard in assuring a satisfactory transmission rate to long RTT connections, it follows that a larger average cwnd has to be expected. As a result, multiple losses in the same window will be more frequent, which suggests considering the SACK policy mandatory. Large cwnds also frequently cause severe inefficiencies of the “exponential back-off” RTO policy. These can be avoided by exploiting timestamps [12], at present largely implemented in the most widespread operating systems.

Finally, to counteract the burstiness due to large congestion windows, with the consequent possible large losses at intermediate router queues, the adoption of packet spacing [13] has been envisaged and recommended.

3. **TOPOLOGY AND SIMULATION SETUP**

To implement the TCP Hybla in the ns-2 simulator, it was necessary to write a new software module, while for the other variants of TCP we used the existing packages. The network topology considered in most simulations is shown in Fig.1. The satellite TCP connections are composed of both wired legs and a satellite link, while the background traffic is represented by entirely wired paths. All the connections share the R1-R2 bottleneck link, whose bandwidth has been deliberately limited to study the congestion effects. The routers R1 and R2 follow a RED (Random Early Detection) policy [14] because of its superiority in ensuring fairness among competing TCP connections and in reducing the burstiness impact, even with moderate queues lengths (qlen=50 seg., maxth=15 seg., and minth=5 seg. in the simulation), while all the hosts follow a simple DT (Drop Tail). The two-way propagation delay of the satellite links varies in such a way that the RTT of the satellite connections ranges from 25 ms (considered only for comparison purposes with the wired connections) to 600 ms (corresponding to the case of a forward and return GEO satellite link). The wired links are supposed to be error free, while a uniformly distributed error model has been considered with a variable PER (0-5%) for the forward link. Note that with respect to [6] both the MSS and the bit rate of the client wired links have been increased, to 1460 bytes and to 100Mbit/s, respectively, to be closer to real network characteristics. For every TCP connection, a persistent FTP file transfer process is considered. The performance is evaluated in terms of goodput, i.e. the amount of packets correctly received divided by the transfer process time (600 s). TCP Hybla requires an estimation of the wireless connection RTT to derive the suitable $\rho$ value ($RTT_0$ is chosen in the simulations equal to the RTT of the wired connections, for the sake of comparison); this estimation is provided by all the TCP versions and it is also implemented in the ns-2 simulator.

Fig.1: ns-2 testbed

4. **NUMERICAL RESULTS**

In this section the outcomes of the simulation campaign will be presented and discussed distinguishing between the scenarios where congestion is present and single flows where only a non-null PER is the limiting performance factor.
4.1 Performance in Presence of Congestion

In order to analyze the effects of RTT performance disparity in presence of congestion, a single foreground satellite connection with variable RTT and 5 background wired connections are considered simultaneously active. In Fig.2, Fig.3, and Fig.4 the goodput of the satellite connection is shown versus the RTT for several PER values, namely, 0%, 1% and 5%. Note that the goodput of the background connections is not reported for the sake of clarity (in brief, they equally share most the bandwidth not used by satellite connections). The results were obtained adopting TCP SACK in the wired connections (SACK option is the default in most TCP NewReno implementation), and four TCP variants for the satellite connection (SACK, Vegas, Westwood, and Hybla). By inspecting the figures, it can be noted that when the first two protocols are applied, the satellite connection performance fast degrades with RTT. By contrast, a substantially independence of the RTT is achieved by TCP Hybla, in accordance with its primary aim. In particular, when an error free satellite channel is considered (Fig.2), the goodput of our proposal is very close to the “maximum fair share” (the capacity of the bottleneck link divided by the number of sharing connections), confirming the outcomes presented in [6]. For all the cases considered, intermediate performance is achieved by TCP Westwood, whose best results are obtained for RTT less than 200 ms. This is coherent with the absence of specific countermeasures against RTT disparity. Finally, by comparing Fig.2 with Fig.3 and Fig.4, we can observe that the penalization that stems from the introduction of a non-null PER leaves basically unaltered the relative performance of the TCP variants considered.

Fig.2: Performance of different TCP versions in the presence of congestion only (PER=0%).

Fig.3: Performance of different TCP versions in the presence of congestion and 1% packet error rate.
4.2 Performance in Presence of Link Losses

As TCP does not distinguish the origin of packet losses, link errors cause spurious interference on the congestion control mechanism, causing many unnecessary cwnd halvings. The consequences of this erroneous interpretation may become severe, as highlighted in Fig.5 and Fig.6, where only the satellite connection is active and all the losses are due to the radio channel, with a PER of 1% and 5%, respectively. In both cases only a limited fraction of the 10 Mbit/s bottleneck bandwidth is actually exploited (with a scale factor due to the diverse PER values), with a fast degradation with RTT for all the TCP variants except TCP Hybla. TCP Vegas provides only a limited improvement with respect to TCP SACK, while TCP Westwood performance depends on the combined effects of PER and large RTT. For PER=1%, its countermeasures against link losses are really effective for RTTs less than 200 ms, while for longer RTTs the performance decay is quite abrupt. By contrast, for PER=5%, the improvement of TCP Westwood with respect to TCP SACK and TCP Vegas is more pronounced and less dependent on the RTT. As far as TCP Hybla is concerned, it must be remarked that, although this variant is not specifically designed against link losses, it generally provides the best performance, except for RTT very close to 25 ms (the reference RTT), where TCP Westwood prevails. Finally, it is worth noting that the performance deterioration due to the simultaneous presence of both congestion and link errors is lower than the sum of the two disjoint penalizations, as previously observed in [6] and here largely confirmed.

Fig.4: Performance of different TCP versions in the presence of congestion and 5% packet error rate.

Fig.5: Performance of various TCP techniques in presence of a residual packet error rate (PER=1%).
Fig. 6: Performance of various TCP techniques in presence of a residual packet error rate (PER=5%).

4.3 Fairness and Friendliness

Fairness and friendliness are two important features for any version of TCP protocol. Fairness refers to the capacity to assure a fair band subdivision among competing connections that use the same version of the protocol, while friendliness indicates the same ability with reference to different protocol variants. TCP standard versions are known to be fair as long as the competing connections have similar RTTs, while on the opposite case longer RTT connections (e.g. satellite) are highly penalized. This phenomenon is illustrated in Fig. 7, where the R1-R2 bandwidth share of three satellite connections (RTT=300 ms, PER=0%) and three wired connections (RTT=25 ms), all simultaneously active, is reported.

Fig. 7: TCP fairness and friendliness: bottleneck share in presence of congestion only (PER=0%).

SACK on the terrestrial connections (RTT=25 ms) and:

a) SACK, b) Vegas c) Westwood d) Hybla, on the satellite ones (RTT=300 ms).
TCP SACK, Vegas and Westwood show a good fairness among connections with the same RTT but also a large unfriendliness between wired and satellite connections. By contrast, TCP Hybla demonstrates noticeable properties of both friendliness and fairness, at the expense of a slight increase in the amount of unused bandwidth.

5. CONCLUSIONS

In the paper, the performance of known variants of TCP (NewReno with SACK, Vegas) is compared with proposals specifically designed to cope with the problems introduced by satellite links, namely non-null PER for TCP Westwood and large RTTs for TCP Hybla. Results, achieved in presence of congestion and/or non-null PER, show that a large performance improvement may be achieved. In particular, TCP Hybla shows its superiority in all the cases considered, except in absence of congestion, where for a PER=1% and RTTs less than 100 ms, TCP Westwood offers a limited advantage.

ACKNOWLEDGEMENTS

This work was supported in part by the IST-507052 SatNEx Network of Excellence.

REFERENCES