PEPsal Performance Analysis on Disruptive Radio Channels

Carlo Caini[1], Rosario Firrincieli[1], and Daniele Lacamera[2]

[1] DEIS dep., University of Bologna, Bologna, Italy
[2] SADEL S.p.A., Castel Maggiore, Bologna, Italy
{ccaini,rfirrincieli}@arces.unibo.it
root@danielinux.net

Abstract. Fixed GEO satellite communications are impaired by long RTTs (especially GEO) and the possible presence of packet losses on the satellite radio channels. Moreover, when the satellite receiver is mobile, short and long disruptions due to line of sight obstructions associated with the presence of shadowing can cause further performance deterioration. In this paper, we evaluate the impact of a disruptive channel on PEPsal, a TCP-splitting PEP previously developed by the authors. Results, obtained by emulating the satellite link interruptions caused by tunnels of a real railway line, highlight the advantages of the TCP-splitting architecture. By enabling the adoption of optimized version of TCP on the satellite connection and a satellite-specific tuning of TCP parameters, PEPsal can offer a significant resilience against all kind of satellite impairments.

Keywords: PEP, TCP, disruptive channels, satellite communications, trains, testbed.

1 Introduction

Fixed GEO satellite communications are impaired by long RTTs (especially GEO) and the possible presence of packet losses on the satellite radio channels [1]. The channel is generally available, except for limited amounts of time when the system may go in outage due to particularly severe additional attenuations. However, when the satellite receiver is mobile, short and long disruptions due to line of sight obstructions associated with the presence of shadowing can cause further significant performance deteriorations.

The aim of this paper is to investigate the impact of disruptions on end-to-end TCP connections and PEP architectures [2], focusing in particular on PEPsal, a free software implementation of the TCP-splitting concept, developed by the authors [3], [4]. To this end, the algorithm of TCP RTO (Retransmission Time Out) timer [5], [6], whose role is essential in presence of disruptions, is briefly reviewed, by pointing out both RFC indications and Linux implementation. RTO is usually doubled by TCP at each unsuccessful retransmission, following the so called "exponential backoff algorithm". This conservative policy is justified if losses are caused by congestion, in order to preserve the stability of the network. However, in presence of long

K. Sithamparanathan (Ed.): Psats 2009, LNICST 15, pp. 68–76, 2009.

disruptions, this technique may introduce a long unjustified delay between the end of a channel disconnection and the consequent retransmission restart. RFC 1122 [6] considers as an optional feature the possibility of restricting the exponential growth of RTO, by setting a maximum RTO value (not lower, however, than 60 s, for the same stability concerns as before). Linux exploits this feature by setting the maximum RTO to 120 s by default in the kernel code. As shown in the paper, this feature plays an important role in limiting the restart delay. For this reason, the default Linux value is changed, to study the impact on performance. In particular, also values lower than 60 s, although not strictly RFC compliant, are considered in the PEP node only, by taking advantage of the isolation of the satellite link from the rest of the network granted by the TCP-splitting PEP.

As a practical case study, we consider disruptions caused by railway tunnels, focusing in particular on one of the most important Italian railways lines, the Bologna-Florence "Direttissima". Performance is evaluated by means of the Linux based TATPA (Testbed on Advanced Transport Protocols and Architectures) testbed [7], [8]. To the usual performance evaluation tools present in TATPA, a new module has been added to emulate disruptions on the satellite channel.

A series of test has been carried out to compare PEPsal and standard end-to-end TCP performance in presence of disruptions, to evaluate the impact of the train speed and that of the maximum RTO. Results are widely discussed in the last section of the paper.

2 TCP Timers

From the TCP perspective, a disruptive channel poses serious constraints to the end-to-end performance. The TCP connection is ruled by four main timers [5], [6] and [9]: the *retransmission timer*, the *keep alive* timer, the *2MSL* and the *persist* timers. Concerning disruptions, the first one has the greatest impact; therefore we will focus our attention on it in the rest of the paper.

2.1 The Retransmission Timer Algorithm

TCP is a reliable transmission protocol that requires positive acknowledgments of transmitted data. It uses the retransmission timer to react to a possible feedback absence from the remote data receiver. The duration of this timer is referred to as RTO (Retransmission Time Out). During the connection lifetime the current RTO, is computed from two estimated variables, SRTT (Smoothed RTT) and RTTVAR (RTT Variation). Details on how RTO is actually computed starting from these variables can be found in [6]. After the first RTO expiring, the last unacknowledged segment is retransmitted and the initial RTO value is doubled, by following the so called *exponential backoff* mechanism, and so on in case of successive RTO expirations, a case that may be easily triggered by disruptions. Before examining in details the RTO algorithm, let us recall the limits posed to the possible RTO values by RFC2988 [6]. It first states that RTO should be conservatively rounded up at 1 s to avoid spurious timeouts. It is worth noting that Linux OS implementation, however, fixes this minimum value to a lower threshold (TCP_RTO_MIN = 200 ms), perhaps because it can relies on a fine clock grain (1 ms) and on advanced spurious timeouts recognition

techniques, such as F-RTO [10]. The same RFC states that also a maximum RTO value may be optionally set. In this case, its value must be at least 60 s.

The TCP retransmission algorithm is quite complex to describe as it consists of many mandatory and optional features, which leave space to multiple implementations.

We can distinguish between two algorithm phases: before and after the RTO expiration. In the former case, the TCP sender applies the following actions [6]:

1. every time a packet containing data is sent (including a retransmission), if the timer is not running, it is started so that it will expire after RTO seconds (for the current value of RTO);
2. when all outstanding data has been acknowledged, the retransmission timer is turned off;
3. when an ACK is received that acknowledges new data, the retransmission timer is restarted so that it will expire after RTO seconds (for the current value of RTO).

Then, after the RTO expiration, we have the following steps:

1. the earliest segment that has not been acknowledged by the TCP receiver is retransmitted;
2. the TCP sender backs off the RTO by doubling it (RTO = 2 RTO). A maximum value may be applied to provide a ceiling to this doubling operation; as aforementioned, this optional feature is adopted in Linux (TCP_RTO_MAX = 120 s);
3. the retransmission timer is started, such that it expires after RTO seconds (for the value of RTO after the doubling operation);
4. if the retransmission timer expires again for the same packet, the RTO is further doubled, the timer is restarted and the segment is retransmitted;
5. In [5] it is stated that the previous point must be repeated either for a given time or for a given number of retries; in Linux the choice is to set a threshold on the number of retries (TCP_RETR2 variable). After this threshold is reached, and the transmission of the segment still fails, the connection is closed. There is also a lower threshold (TCP_RETR1 variable) which allows the TCP to pass negative advice to the IP layer, which in turn triggers dead-gateway diagnosis. Linux default values for these variables are respectively: TCP_RETR1 = 3 and TCP_RETR2 = 15.

2.2 Remarks on the Maximum Tolerable Disruption Length, and TCP Agility on Restarting Transmission

The Linux choice in favor of retries threshold, implies that the maximum tolerable disruption length (i.e. that does not cause the connection closing) is not directly set, but depends on the RTO value at disruption start, the number of allowed retries (TCP_RETR2) and the max RTO (TCP_RTO_MAX). It is worth stressing the role played by this last parameter in both determining the maximum tolerable disruption length and the "agility" of TCP to restart transmission after relatively long disruptions. Actually, in the presence of a long disruption the channel availability is probed at doubling time intervals until the maximum RTO value is reached. Then, it

is probed at constant regular intervals until the maximum number of retransmission is reached. As the maximum RTO value represents the maximum time interval between a channel probe and the following, it also represents the worst case for the delay between a disruption end and the subsequent transmission restart. Moreover, if we assume disruption lengths uniformly distributed between two consecutive probes, the average restart delay is upper bounded by half the maximum RTO (first retransmissions are actually faster). The longer this delay, the higher the penalization in performance, as the channel is left unexploited fro more time.

In conclusions, the maximum RTO value represents the worst case delay before retransmission restart and its half the average delay for last retransmissions, when RTO has already reached its maximum value. The higher the maximum RTO threshold, the higher the restart delays. On the other hand, for a given maximum number of retries, the higher the maximum RTO, the longer the maximum tolerable disruption. In the following we exploit the possibility of tuning TCP_RTO_MAX and TCP_RETR2 variables to improve TCP agility while maintaining the maximum tolerable disruption length. To this regard we modified the Linux kernel in order to make the TCP_RTO_MAX variable accessible by the sysctl tool.

3 PEPsal

The category of Performance Enhancing Proxies (PEPs) embraces a wide variety of different techniques operating at different levels, as clearly shown in [2]. In brief, PEPsal [3] can be described as a TCP accelerator based on the TCP-splitting technique, implemented in Linux and made freely available [4] under the GPL license. Following the RFC classification PEPsal is a “multi-layer” proxy, because in order to implement the TCP splitting technique, it must operate at Network, Transport and Application layers. PEPsal can be classified as “integrated”, since it runs only on a single box on the forward link satellite gateway. It is worth noting that this characteristic differentiates it by most of commercial PEPs based on the same TCP-splitting techniques, which, by contrast, are usually “distributed”, running on two boxes at both ends of the satellite link. In the usual configuration PEPsal is “asymmetric”, i.e. it is active only in the forward direction (from the satellite gateway to the end user), but it can be made easily “symmetric” (i.e. active in both forward and return directions) just by introducing a few modifications on the receiver side. Finally, PEPsal is “transparent” to the users in the customary asymmetric configuration, as TCP users are unaware of the connection splitting performed at the intermediate satellite gateway.

The PEPsal architecture is shown in Figure 1. The end-to-end TCP connection is split into a pair of separated connections, with the aim of isolating the satellite channel. On the first connection, from the TCP server to the satellite gateway, it is adopted a standard TCP variant, like NewReno. By contrast, on the second, from the satellite gateway to the TCP client, it is definitively better to adopt an enhanced TCP variant, to better cope with long RTTs and the possible presence of segment losses due to residual bit errors on the satellite link. The suggested choice is TCP Hybla [11], a TCP variant specifically designed to cope with the long RTTs typical of satellite channels. However, PEPsal allows all the other TCP variants implemented in

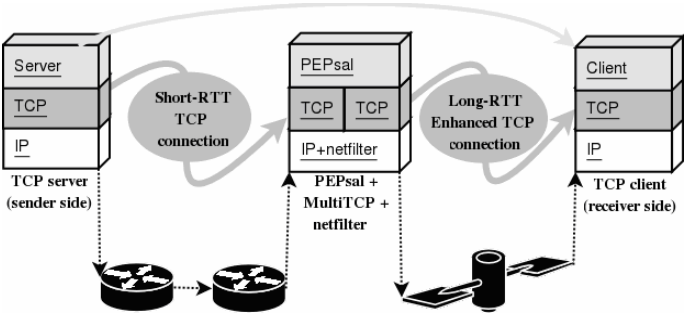

Fig. 1. PEPsal architecture (based on integrated TCP-splitting approach)

Linux (more than ten at present) to be adopted as well. As all of them are compatible with standard TCP clients, both the TCP server and the TCP client can continue to use standard TCP, being actually unaware of the PEPsal presence in the middle.

The splitting of the end-to-end connection in two segments, allows a greater freedom of choice in the setting of TCP timers related to timeout and retransmissions. In fact, being the satellite connection isolated from the network core, it is possible to consider more frequent channel probes of what usually allowed on end-to-end channels, if necessary. This is a possibility that is explored in the paper, by tuning the TCP_RTO_MAX value. In particular, this parameter is reduced from 120 s (the Linux default) to 10 s in order to limit the restart delay after disruption ends. Although not RFC compliant [6], its adoption is safe also from the network point of view, because the more frequent channel probes made through segment retransmissions have a null impact on the open network, being strictly confined on the satellite connection.

4 The TATPA Testbed

The design of TATPA (Testbed on Advanced Transport Protocol and Architectures) [7] aims to reproduce the essential characteristics of heterogeneous networks that include satellite links. Its logical layout (much simpler than the corresponding physical layout based on a cluster of Linux PCs) is depicted in Figure 2. The choice of the Linux platform was dictated by a number of reasons that make it very appealing to the research community: GNU/Linux system is free, fully customizable by the user, and offers the network researcher a wide number of TCP variants already available in the standard OS package. In addition to widely adopted network tools, such as NistNet [12] and Iperf [13], TATPA also exploits some software packages specifically developed by the authors to extend its features (these software tools are also made available to the scientific community under the GNU license). Among them, we cite the Multi TCP package [14], which allows us to include the full version of TCP Hybla [11] and to collect logs of internal variables (like RTO), the PEPsal package, discussed in the previous section, and the reference implementation of Bundle Layer DTN (Delay Tolerant Networks) [15]. Last but not least, in order to emulate satellite channel disruptions, we realized a time-discrete Gilbert-Elliot [16] channel emulator. Such a channel discards all arriving packets when in its bad state,

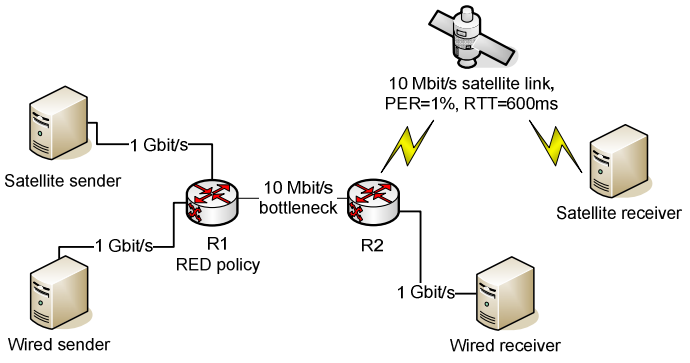

Fig. 2. The TATPA logical layout

while leaves untouched flowing packets when in its good state. In order to determine the good-bad states transitions, both statistical and deterministic trace-based methods have been implemented. In this paper, we exploited the second possibility, by making use of real railways tunnel traces.

5 A Case Study: Disruptions Caused by Railway Tunnels

As mentioned above, disruptive channels can be present when mobility is concerned. Tunnels on railway or highway are a practical significant example of disruptiveness induced by mobility. Here, we will focus on railway tunnels, considering real data referring to one of the most important and significant Italian railways lines.

5.1 The Characteristics of the "Direttissima" Bologna-Florence Railway Line

Bologna is one of the most important railway nodes in Italy. It is located roughly a hundred kilometers North of Florence, from which is separated by the Apennines mount chain. Bologna and Florence are at present connected by two lines, of which one is of local interest only, while the second, called "Direttissima" and considered here, is the most important railway link for both passengers and freight traffic between North and Central Italy. "Direttissima" was opened to the service in 1934. It is about 96 km long and has been chosen not only because of its practical relevance but also because it is characterized by an interesting mix of tunnels of different lengths interlaced with open sky segments. In the whole, there are 33 tunnels on the line, of which the longest is of about 18.5 km, while the average length is of 1.1 km. The whole length of tunnels is of 37 km, equal to 39% of the line. Of course, the longer the tunnel, the longer the corresponding satellite channel disruption, the actual values being dependent on the train speed. Assuming a constant speed of 120 km/h (typical of the average speed of passenger train on the line) we obtain the values given in Figure 3. Value for other speeds can be directly derived from these.

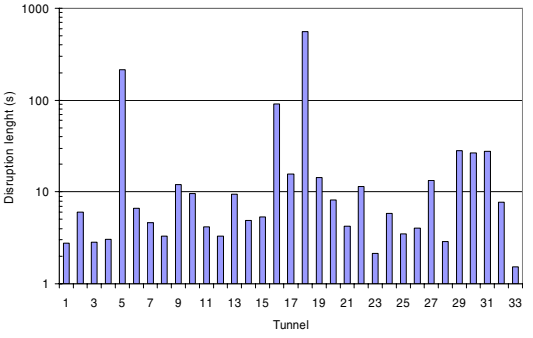

Fig. 3. Bologna-Florence railway line; length of channel disruptions caused by tunnels (train speed=120 km/h)

A new high speed railway line is going to be completed by the end of 2009. It has not been considered here because most of it will be in tunnels, with very short open sky segments.

5.2 Numerical Results

Numerical results, obtained through TATPA, are presented in Figure 4 and Figure 5. They refer to a single satellite connection that lasts for the entire train journey, from Bologna to Florence. Performance is given in terms of final goodput, calculated considering only the clear sky percentage of the connection lifetime (61% in the considered scenario). The objective is to highlight the TCP ability to exploit the channel when it is not disrupted by tunnels. Moreover, we have assumed the link affected by a 1% Packet Error Rate (PER), to take into account possible radio link impairments outside the tunnels. It is worth noting that the tested link is extremely challenging, encompassing at the same time a long RTT (600ms), a high PER (1%) and frequent disruptions (39% of the entire connection lifetime).

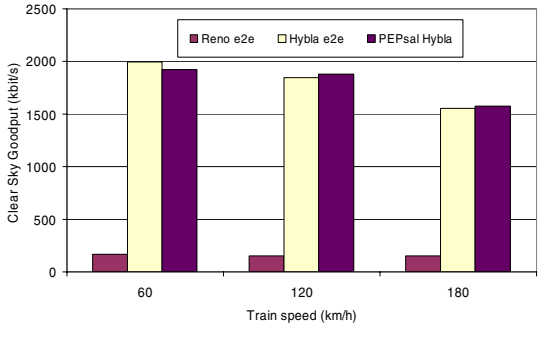

Fig. 4. Comparison between end-to-end TCP and PEPsal; clear sky goodput versus different train speeds

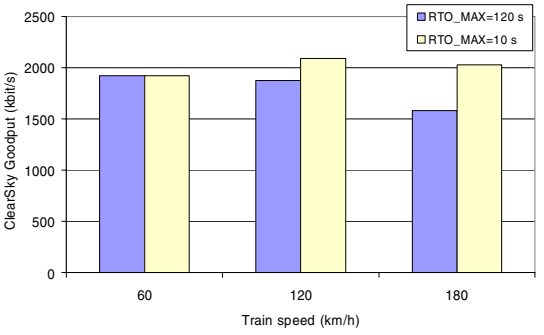

Fig. 5. Impact of TCP_MAX_RTO on PEPsal clear sky goodput for different train speeds

In Figure 4 end-to-end Reno, end-to-end Hybla and PEPsal are compared considering different train speeds (60, 120 and 180 km/h). All of tested protocols and architectures are considered with the default TCP_RTO_MAX, while TCP_RETR2 has been properly increased to cope with the almost 20 minutes disruption caused by the longest tunnel in the 60 km/h case. The outcome confirms the well known Hybla superiority over Reno, when applied in satellite environments. With respect to previous results presented in the literature [3], here the channel is more challenging, being affected, as previously pointed out, not only by a high PER but also by frequent disruptions. PEPsal, using Hybla in its second connection, shows the same performance of end-to-end Hybla, with the great advantage of being transparent also to the sender, which, consequently, does not require to be modified as in the end-to-end approach.

As aforementioned, the splitting nature of PEPsal allows the modification of TCP_RTO_MAX on the second connection without affecting the rest of the network. This important parameter can be reduced to decrease the TCP restart delay after a disruption. However, a reduction of the RTO max value without a correspondent increasing of the maximum number of retransmissions (TCP_RETR2) results in a diminution of the maximum tolerable disruption length. In Figure 5 we focused on PEPsal performance to have a first assessment of the TCP_RTO_MAX impact. To this end we have compared previous results (120 s) with those achievable by reducing TCP_RTO_MAX to just 10 s. In this case we have also further increased the TCP_RETR2 to preserve the maximum tolerable disruption length. Results show that, especially at high train speeds, reducing the RTO max value effectively decreases the restart delay, with a consequent enhancement of the final goodput.

6 Conclusions

In the paper, the performance achievable by a TCP-splitting PEP architecture operating on a GEO satellite link have been analyzed, with particular emphasis on the impact of channel disruptions. Results, obtained by emulating the satellite link interruptions caused by tunnels of a real railway line, highlight the advantages of the TCP-splitting architecture. By isolating the satellite impairments from the rest of the

network, it enables the adoption of optimized version of TCP on the satellite connection and a satellite-specific tuning of TCP parameters, like the RTO maximum value investigated in the paper. In this way, a better resilience against all kind of satellite impairments, included channel disruptions, can be easily achieved.

References

1. Hu, Y., Li, V.O.H.: Satellite-based internet: a tutorial. IEEE Commun. Mag. 39(3), 164–171 (2001)
2. Border, J., Kojo, M., Griner, J., Montenegro, G., Shelby, Z.: Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations, IETF RFC 3135 (June 2001)
3. Caini, C., Firrincieli, R., Lacamera, D.: PEPsal: a Performance Enhancing Proxy for TCP satellite connections. IEEE Aerospace and Electronic Systems Magazine 22(8), B–9–B–16 (2007)
4. PEPsal: `http://sourceforge.net/projects/pepsal/`
5. Braden, R.: Requirements for Internet Hosts—Communication Layers, IETF RFC 1122 (October 1989)
6. Paxon, W., Allman, M.: Computing TCP's Retransmission Timer, IETF RFC 2988 (November 2000)
7. Caini, C., Firrincieli, R., Lacamera, D., Tamagnini, S., Tiraferri, D.: The TATPA Testbed, a Testbed for Advanced Transport Protocols and Architecture performance evaluation on wireless channels. In: Proc. IEEE TridentCom, Orlando, Florida, pp. 1–7 (2007)
8. TATPA website: `https://tatpa.deis.unibo.it`
9. Stevens, W.R.: TCP/IP Illustrated, vol. 1. Addison-Wesley, Reading (1994)
10. Sarolahti, P., Kojo, M.: Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP), IETF RFC 4138 (August 2005)
11. Caini, C., Firrincieli, R.: TCP Hybla: a TCP Enhancement for Heterogeneous Networks. Int. J. Satell. Commun. Network 22, 547–566 (2004)
12. NistNet web site: `http://snad.ncsl.nist.gov/itg/nistnet/`
13. Iperf web site, `http://dast.nlanr.net/Projects/Iperf/`
14. Caini, C., Firrincieli, R., Lacamera, D.: A Linux Based Multi TCP Implementation for Experimental Evaluation of TCP Enhancements. In: Proc. SPECTS 2005, Philadelphia (July 2005)
15. Cerf, V., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., Weiss, H.: Delay-Tolerant Networking Architecture, Request for Comment, IETF RFC 4838 (April 2007)
16. Gilbert, E.N.: Capacity of a burst-noise channel. The Bell System Technical Journal 39, 1253–1265 (1960)