Network and Communication Technologies; Vol. 2, No. 1; 2013
ISSN 1927-064X E-ISSN 1927-0658
Published by Canadian Center of Science and Education

A New Survey on Improving TCP Performances over Geostationary Satellite Link

Alain Pirovano[1] & Fabien Garcia[1]

[1] Communication Networks Research Group, ENAC (French Civil Aviation University), France

Correspondence: Alain Pirovano, ENAC, 7 Avenue E. Belin Toulouse 31055, France. Tel: 33-562-174-145. E-mail: alain.pirovano@enac.fr

Received: November 16, 2012 Accepted: December 14, 2012 Online Published: January 6, 2013

doi:10.5539/nct.v2n1p1 URL: http://dx.doi.org/10.5539/nct.v2n1p1

Abstract

The idea of Internet everywhere makes the assumption that an Internet access should be available even in remote areas without network infrastructure. In this case satellite access represents an attractive solution. Nevertheless, experience shows that over satellite links, TCP is limited in terms of data speeds. Many enhancements and solutions, based for instance on tuning TCP parameters or TCP spoofing, have been proposed to avoid the underutilization of satellite link capacity. These topics have been often addressed, but considering recent high speed TCP variants, the evolution of end users habits, and recently proposed satellite link access scheme, a new study is necessary in order to reconsider some preconceptions and previous recommendations in such a context. This paper proposes an overview of TCP variants and a survey of commonly proposed solutions for TCP over satellite. Then a methodology for TCP performance assessment over satellite links is exposed. The approach is mainly based on a satellite link emulation platform and some tools developed at the ENAC. We assess the gain offered by a split TCP connection with PEPSal comparatively to end to end TCP connections based on NewReno and a recent widely deployed TCP version (TCP Cubic) on an emulated satellite link. Unlike existing studies, we compare PEP advantages with most recent TCP versions and propose as an extension, to assess PEP gain considering the number of simultaneous TCP connections. Finally, the results provided allow us to make some original recommendations toward TCP deployment over satellite links.

Keywords: TCP performances, satellite Internet access, Performance Enhancing Proxy, PEP

1. Introduction, Motivations, and Objectives

Seeing that a lot of geographic areas are still uncovered by terrestrial infrastructure networks and due to the fact that geostationary satellite links allow an easy and fast access network for mobile users, such communication systems represent a competitive and adequate solution for Internet access in several contexts. In the last decade, thanks to VSAT (Very Small Aperture Terminal) technologies and recent satellite link access scheme based on DVB-S/RCS (ETSI, 2009), satellite Internet access is getting easier to deploy, less expensive and therefore proposed by an increasing number of service providers.

Nevertheless, it has been often underlined that TCP performances may be affected by satellite links properties such as offered capacity, delay, and bit error rate. A link based on a geostationary satellite induces long propagation delay due to the satellite altitude (about 36,000 km). Furthermore, compared to terrestrial links, these links show high bit error rate, and hence high packet error rate.

Already in the 90's (Partridge & Shepard, 1997; Allman et al., 1997), and a RFC (Allman, 1999) explained how TCP should be tuned and enhanced in order to improve TCP performances in satellite environment. A well-known solution to those problems consists in splitting the end-to-end TCP connection on the satellite link at the ground station. As shown in figure 1, doing so allows the deployment of an enhanced TCP version dedicated specifically to satellite links and their specific properties. These solutions are generally known as TCP PEP (Performance Enhancement Proxy) or TCP accelerator. It has to be noted that PEPs are sometimes implemented at the application level such as proposed in (Davern et al., 2011), where the authors describe a novel HTTP PEP which improve HTTP performances in case of a satellite based Internet access. This particular implementation of PEP is not considered in the present article.

The majority of studies showing the benefit of PEP solutions in geostationary satellite contexts compare their

performances with those of relatively old versions of TCP such as NewReno. At the same time, with the increasing number of faster terrestrial access links like fibre, old standard TCP have shown their limits and particularly the difficulty to reach high data rates in those environments with typical packet loss rates. Many new transport protocols have been proposed to accommodate this, some of them are already implemented in the latest versions of the main operating systems, like Compound TCP (CTCP) introduced as part of the Windows Vista, or Cubic implemented on Linux. Even if the way the flow of data is managed by the congestion window in CTCP and Cubic TCP, is not the same, their main objective is to avoid the bandwidth underutilization in high speed long distance networks that has been observed with classical standard TCP. At the same time, these versions must ensure that they operate in a fair and friendly way when they share resources with others TCP connections. The very large scale of the deployment of these recent TCP variants and the specifics properties of recent satellite Internet accesses legitimize and justify a new study in order to assess the gain of TCP PEP comparatively to end-to-end TCP control implemented in more recent version. In other words, considering the recent evolution of satellite access solutions and transport protocols, we face again the old question: to split or not to split.

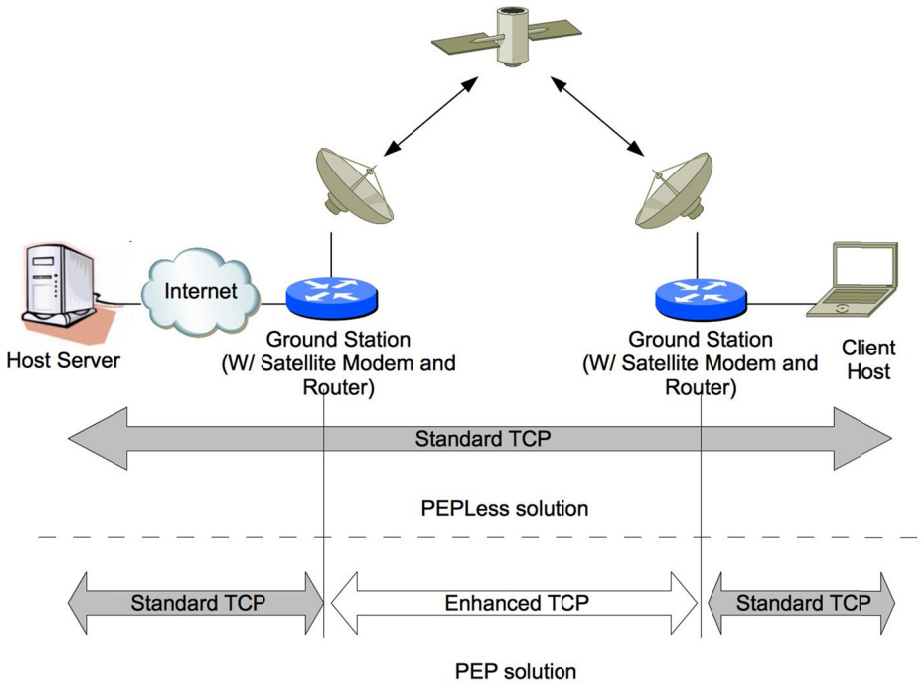

Figure 1. Typical satellite Internet access topology

2. An Overview of TCP Variants

Since its first version and a dedicated RFC 793 from the IETF (Internet Engineering Task Force) in 1981, a lot of improvements have been proposed and implemented in the famous transport protocol of the TCP/IP suite. These add-ons functionalities have been mainly motivated by the necessity to have TCP match with the properties of new networks links emerging in the time, particularly in term of capacity and delay. This led to the definition of several TCP versions.

2.1 A Brief Reminder on TCP Standard

TCP is an end-to-end connection-oriented protocol which accepts data from a data stream submitted by the application layer, segments it into chunks, and adds a TCP header creating a protocol data unit named TCP segment. It aims at enforcing end-to-end reliability using a sliding window flow control and loss detection algorithm based on the expiration of a timer used by the sender. In most of TCP versions, the sliding window size is fixed according to two processes. Firstly, the receiver specifies with each TCP segment it sends to acknowledge received data, the amount of additional data it is willing to buffer for the connection. This value is

known as the receive window *rwnd*. In order to take into account the state of the different links and intermediate systems between the sender and the receiver, TCP maintains another variable on the sender side named congestion window *cwnd*. Its value increases when a new acknowledgement indicates that new data is received, and it is decremented on loss detection, for instance when a timer expires. Finally, the amount of data that can be sent before a new acknowledgement, known generally as *flight size*, is the minimum between these two variables.

The first and most famous enhanced versions of TCP are Tahoe, NewReno and Vegas. These versions and the most recent ones mainly differs in the way they manage the congestion window size and the events or indicators that trigger its updates. The first one has been proposed by Jacobson (Jacobson & Karels, 1988) and introduced the 'fast retransmit' algorithm. Until then, TCP only used a time out based segment loss detection mechanism (i.e. the expiration of a pending timer managed by the sender), Tahoe version enhanced TCP with another way to detect losses based on duplicate acknowledgements. A TCP sender numbers the flow of bytes it receives from the application layer and the receiver periodically acknowledges the data it receives by sending back the number of the expected byte, i.e. the last byte number of the first continuous flow of data received incremented by one. Tahoe makes the assumption that in the event of duplicate acknowledgement, it has to retransmit the corresponding data even if the timer has not yet expired for these data. In cases of timer expiration or duplicate acknowledgements, Tahoe makes the assumption that the network is congested, and hence decreases its congestion window size to 1 maximum segment size (MSS, a sender network properties defining the maximum size of a TCP segment). With TCP Reno version, Van Jacobson proposed to differentiate these two events in the sense that if timer expiration is symptomatic of a serious problem which justify a drastic decrease of *cwnd* and hence of the *flight size*, duplicate acknowledgements revealed a loss followed by the reception of new segments, that is an isolated segment loss. That is why in the second case TCP Reno apply a 'fast recovery' algorithm which consists in dividing the congestion window size by two.

During a nominal behaviour without losses, these TCP versions increment the congestion window size by two mechanisms, Slow Start and Congestion Control. If the flight size is less than a threshold value ssthresh, they use Slow Start and increase cwnd by $cwnd = cwnd + 1MSS$ (maximum segment size) for each acknowledged segment. If the current *flight size* is greater than *ssthresh*, during a new phase called Congestion Avoidance the increment is given by $cwnd = cwnd + (1MSS/cwnd)$. The value of *ssthresh* is half the maximum flight size reached before the last loss detection or equals to *rwnd* for new connections. NewReno enhance Reno in the sense that after a fast retransmit, the sender stays in fast recovery state until the original window has been entirely acknowledged. Doing so, it improves performances in the case of multiple losses in a same window. In those, relatively old, TCP standards the end-to-end congestion control mechanisms is known as AIMD (additive increase multiplicative decrease) because the TCP sending rate is controlled by a congestion window which is halved for every loss, and increased by one packet per window of data acknowledged.

In the middle of the 1990s, L. Peterson and L. Brakmo researchers at the University of Arizona introduced TCP Vegas offering a new congestion avoidance algorithm (Brakmo et al., 1994). Rather than losses, this algorithm uses the observed packet delay in order to determine the rate at which to send its segments. Of course, this method depends heavily on an accurate calculation of the RTT (Round Trip Time) value which represents the time elapsed between the sending of a segment and the reception of the acknowledgement by the sender. TCP Vegas makes the assumption that under some conditions, an increase in Round-Trip Time reveals a network congestion.

2.2 Recent High Speed TCP Variants

More recently, since about 1998, some less known TCP versions have been proposed to accommodate the important growth of the Internet and a mutation of networks links technologies (fibre links, wireless links, ...) and their properties in term of delay and offered capacity. We can cite TCP Westwood, Illinois and BIC as examples of versions which seek to improve performances in case of high bandwidth-delay product networks known also as LFN (Long Fat Networks). Bandwidth-delay product reveals the amount of outstanding data a network can have, and with products higher than 64Kbytes, older TCP versions showed a waste of capacity. That is why, a lot of the newer versions aim at "keeping the pipe full", mainly enhancing packet loss detection and congestion control mechanisms.

For instance TCP Westwood (Casetti et al., 2002) or TCPW exploits end-to-end bandwidth estimation to adjust the values of *ssthresh* and *cwnd* after a congestion event. TCPW includes a recovery algorithm which avoids the halving of the flight size following a segment loss in order to ensure a high link utilization even in the presence of errors.

TCP Illinois (Liu et al., 2008) is a loss-delay based algorithm. It uses an AIMD algorithm but adjusts the increase and decrease parameters based on estimations of queueing delay and buffer size. The queuing delay is used as a congestion signal to adjust the pace of window size change.

TCP BIC (Binary Increase Control congestion) (Xu et al., 2004) is a protocol that tries to ensure a fair sharing of the link capacity among flows experiencing different RTTs. TCP BIC has been implemented and used by default in Linux kernels 2.6.18.

The most widely deployed TCP versions are Compound TCP (Tan et al., 2006), used as part of the Windows Vista and Window Server 2008, and Cubic TCP (Rhee & Xu, 2005). This latter is the default implementation provided since the 2.6.19 version of Linux kernels and considerer as the reference recent TCP candidate in our study in the rest of the paper.

TCP Cubic is seen as an enhanced version of TCP BIC using optimized congestion control algorithm. TCP Cubic is designed with an optimized congestion control algorithm for high speed networks which may induce high latency. It uses a cubic function of the elapsed time from the last congestion event and uses both concave and convex profiles of a cubic function for window increase (Rhee & Xu, 2005). Cubic is particular in the fact that it is ruled by the congestion epoch which is defined as the time between consecutive congestion event. It then makes the window growth independent from the round trip time (RTT) and hence ensures a better fairness than classical congestion control approaches.

Because it has been specifically developed for heterogeneous networks which may incorporate terrestrial or satellite radio link, another TCP version, named Hybla, is considered in our study. TCP Hybla is recommended as the alternative TCP version for the satellite hop in case of the use of a particular PEP solution named PEPsal that will be investigated in this study. In the presence of satellite links, TCP Hybla variant can be used either as an end-to-end protocol, with modification on the sender side, or as a satellite hop transport protocol, thus using the concept of TCP splitting. As explained in (Caini & Firrincieli, 2004), unlike in other approaches, the main idea in TCP Hybla is to open the congestion window at a rate independent of the RTT experienced by the connection. The congestion control algorithm is also modified in concordance with the measurements made during the connection. Further, TCP Hybla makes use of different techniques in order to reduce the impact of multiple losses, inappropriate timeouts and burstiness. It uses SACK (Mathis et al., 1996) (selective acknowledgement) and timestamp options. The first enhance classic TCP cumulative acknowledgement with selective ones and the second allows the add of a timestamp in each segment sent in order to accurately estimate the RTT. Also, in order to avoid possible bursts of segments, TCP Hybla uses a segment spacing and channel bandwidth estimation technique. TCP Hybla is designed to minimize the impact of long RTT, induced by satellite links, on TCP performance. Its congestion control policy increases the speed of the *cwnd* growth in order to provide the same transmission rate for connections with different RTT.

Figure 2 presents a timeline showing a chronological view of main TCP variants mentioned in this subsection.

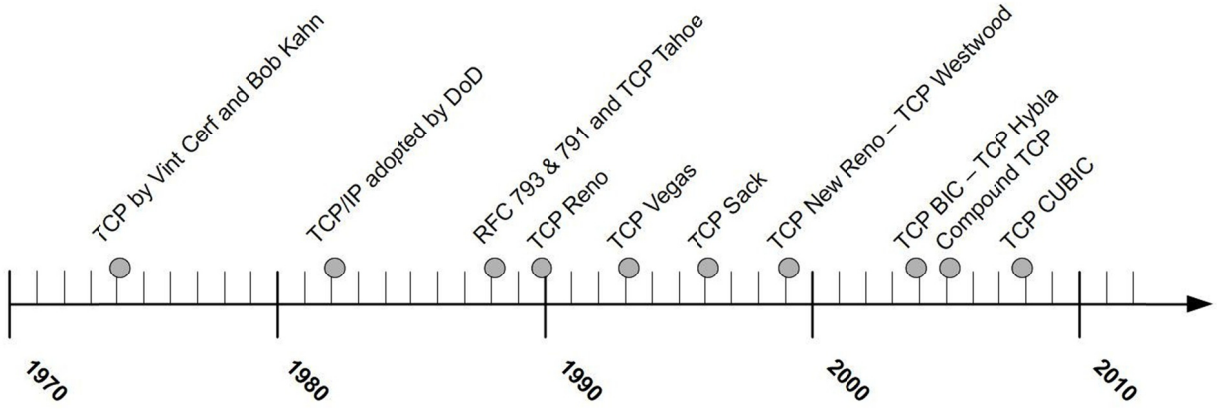

Figure 2. TCP timeline

3. Performance Enhancement Proxy

3.1 Main Reasons for Splitting End-to-end TCP

One problem of TCP is that it was not originally designed for use over satellite links. As TCP was not designed to be tied with one type or a class of link technology, the downside is that TCP is not optimal on any kind of link. Satellites links represent a kind of medium whose intrinsic specificities may heavily affect TCP performances, especially if one considers equipment and antennas for Internet Access by satellite that are affordable to a large public. Compared to terrestrial networks, those specificities are:

- an important propagation delay: about 250 ms considering the altitude of a geostationary satellite
- a high bit error rate which can lead to an important PER (Packet Error Rate) considering average packet sizes
- a low data rate, with a maximum upload rate of 10 Mbits/s

Considering TCP's mechanisms and algorithms, the main causes for TCP's inefficiencies over satellite links are:

- 3 way handshake: a TCP connection is established by a "3-Way Handshake" between the receiver and the sender. On satellite links, this may take at least a few seconds to be completed (even for very short data exchanges).
- Receive Window: An application using TCP to send data is only allowed to send an amount of bytes which is explicitly indicated in the last packet sent by the destination. This value known as the "Receive Window", is typically set to about 8 or 16 Kbytes, which severely limits the speeds that a TCP connection can achieve over a satellite.
- Default Slow Start algorithm: As previously explained, TCP starts sending at a very slow rate and increase it until the available capacity is reached. Since this process can take several RTT, the propagation delay inherent to the satellite hop becomes very penalizing. Hence, it will take a long time before TCP reaches its "cruising speed". In some cases, severe packet losses may also generate a drastic TCP behaviour with a new slow start stage.
- Packet loss: Basic TCP assumes that any packet loss is caused by congestion. In order to limit the perceived congestion whenever a packet is lost, TCP generally reduces its congestion window and hence its transmission rate. For instance, TCP NewReno reduced its transmission rate by 50 percent for each detected loss. Of course, because of the bit error rate on the satellite links, packet error rate is observed without congestion occurring. Moreover, because satellite links often acts as bottlenecks considering the difference between satellite and terrestrial links capacity, a lot of packet losses can occur in the considered topology leading to poor TCP performance.

Of course, the idea of adapting existing TCP standards or defining more adequate versions as explained sooner have been explored since the beginning of TCP. Very soon, in 1997, Partridge and Shepard have given accurate recommendation on TCP parameters tuning in order to avoid decrease of TCP performances in case of use in presence of satellite link. The conclusion mainly recommends to adopt new features like window scaling, PAWS (Protection Against Wrapped Sequence number), and Selective Acknowledgement. It must be noted that those enhancements do not represent an exhaustive list of the works done on this subject. Nevertheless, Even if end-to-end TCP performances are better with well tuned TCP parameters, satellite link characteristics still impact performances, in particular the observed delay for a starting TCP connection to fill up the offered capacity.

Furthermore, considering the end-to-end transport protocol approach based on recent high speed TCP variants, we can refer to (Marcondes et al., 2008). This study performs a cross-comparison of equal bandwidth delay product scenarios and very different network topologies based on high-speed networks and satellite networks. The authors experiment with two advanced TCP variants (TCP Westwood and Hybla) aimed at optimizing the throughput and then analyses issues related to improving RTT-fairness and coexistence with standard NewReno.

In another paper (Trivedi et al., 2010), the authors give a comparative performance evaluation of end-to-end TCP Hybla and Cubic on a satellite link and under low error conditions using forward error correction (FEC) techniques. The conclusions of this study are that Cubic performs better than Hybla. In fact, the aggressive *cwnd* policy adopted by Hybla results in an overall degraded performance in low PER conditions. The authors also explain that the oscillatory behaviour of the *cwnd* with TCP Hybla would make the performances less predictable, and hence unacceptable for many applications.

Even with the results of those studies, we can observe that satellite link characteristics still impact TCP performances with end-to-end transport protocol. That is why PEP solutions have been and are still proposed and

investigated.

3.2 Some Details on PEPs

It exists several types of PEPs. As satellite links represent the most challenging environment, they are the main target of PEPs but it is not the only one. Wireless networks more generally may improve their end-to-end performances by implementing PEPs. PEPs may be defined as network agents designed to improve the end-to-end performance. They operate by splitting the end-to-end connection into several successive connections and generally using different protocols for a same layer to transfer data across the different segments. Doing so allows a customization of the protocols without modifying the end systems. A typical implementation uses transport layer PEPs to improve TCP performance over a satellite link. The end systems use standard TCP without even being informed of the existence of the PEP along the link. The PEP is implemented on a router along the TCP connection, when a packet arrives at the router, it is forwarded and the PEP transmits the corresponding ACK to the source host in behalf of the destination host. It also stores a copy of the packet in case retransmission of the packet is required. TCP PEPs are generally placed on each end of a satellite link and are able to send acknowledgements back to the host which has sent segments before the destination host has received them.

Border et al. (2001) propose a survey of Performance Enhancing Proxies (PEPs) often employed to improve degraded TCP performance caused by characteristics of specific link environments, for example, in satellite, wireless WAN, and wireless LAN environments.

Bisio et al. (2009) gives an overview of PEP mechanisms. As explained, one or more of these mechanisms may be present in a PEP implementation. Considering our objectives, in the case of splitting the end-to-end path in presence of a satellite link in order to implement an alternative adequate transport protocol, we may put the emphasis on three of these mechanisms:

- TCP Ack Handling: TCP PEP implementations are generally based on TCP ACK manipulation. For instance, this mechanism allows the PEP to act as a *virtual* end system hiding from the real end systems that the connection is split. In this case, data segments received by the PEP are locally acknowledged by the PEP.
- TCP ACK Spacing: because of the different capacity on the successive links between end to end systems, burst behaviour may occur on the satellite link. TCP Ack spacing helps with smoothing the flow of Acks and hence of segments.
- Local TCP Retransmissions: As a PEP acts as an end system, it may also locally retransmit data segments lost. Hence, it allows for faster recovery from lost data.

It has to be noted that other important functionalities, such as compression or tunnelling, may play important role in the PEP action.

An important characteristic of PEP concerns its distribution. A PEP implementation can be integrated or distributed. In the first case PEP software is installed on a single point and generally dedicated to satellite Internet access for a single user. In the second case a PEP installation is required on both side of the satellite link in order to share the satellite Internet access between several LAN nodes.

Some examples of PEP solutions are given in (ETSI, 2009). HPEP, which is mostly an HTTP PEP developed by Hughes, uses a distributed asymmetric architecture with a PEP client and a PEP server. HPEP can be considered as an HTTP Proxy. HTTP requests generated by the web client are sent to the Downstream Proxy Server which sends the requests to the Upstream Proxy Server across a single, persistent TCP connection. Doing so, the number of TCP acknowledgements is reduced, thereby reducing the overall traffic over the satellite link and avoiding a waste of resources. Therefore, faster and more efficient Web browsing is expected. But as HTTP is the only protocol supported, other traffics, such as FTP, bypass the proxy and do not get any benefit from HPEP.

I-PEP (Interoperable-Performance Enhancing Proxy) is a splitting architecture proposed by ESA Satlabs. It defines a protocol stack for both side of a satellite link making use of DVB-RCS as access architecture. It aims at improving TCP based applications using SCPS-TP (Space Communications Protocols Standards-Transport Protocol) in order to provide a reliable connection to upper layers on the satellite link. Nevertheless, Roseti et al. (2010) explains that considering the Demand Assignment Multiple Access (DAMA) schemes used to share radio resources in the return channel when DVB-RCS is implemented, protocols at the upper layers may experience an overall delay well above the propagation delay due to an important jitter. Applications based on Transmission Control Protocol (TCP) are particularly affected in such a situation, especially when transferring short objects or files as in the case of Web browsing. In order to avoid this latency and therefore improve performances, a new

burst-based TCP protocol for I-PEP named TCP Noordwijk (TCPN) is introduced and assessed.

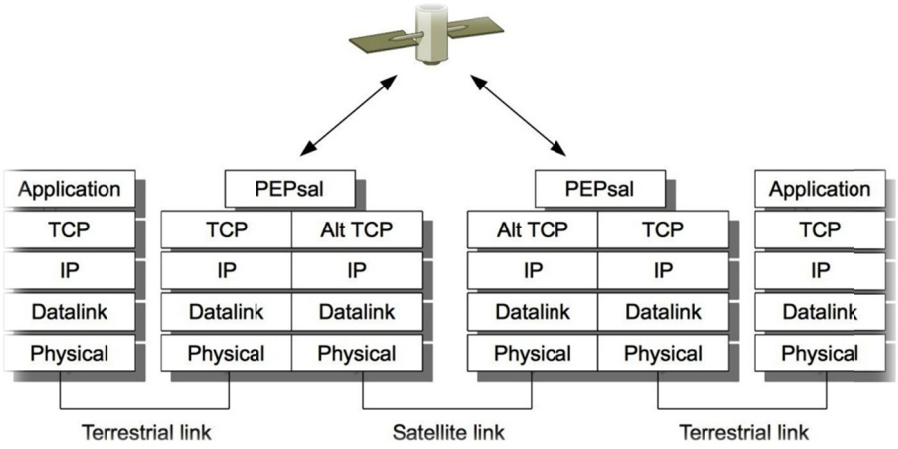

Figure 3. Satellite access with PEPsal protocol architecture

Caini et al. (2006) proposes a Performance Enhancing Proxy solution called PEPSal which is certainly the most investigated and implemented PEP solution. PEPSal is an open source TCP solution for the GNU/Linux operating systems (Sourceforge.net, 2012). It runs only on a single box, and hence can be classified as an integrated PEP. PEPSal splits the end to end TCP connection and makes use of TCP Hybla as a dedicated transport protocol on the satellite link. It has to be noted that since modifications are not required in both the connection endpoints, PEPSal is transparent for the end hosts.

PEPsal is now adopted by several satellite Internet provider and has even been recently recommended by ETSI as the PEP solution in the context of Broadband Satellite Multimedia (BSM) based on the last version of DVB-RCS2 (ETSI, 2011b; ETSI, 2011c; ETSI, 2011a).

Figure 3 shows the typical protocol stack architecture of distributed PEP based satellite Internet access for a distant user. As explained before, the use of PEP allows to choose an alternative TCP on the satellite link.

3.3 Expected Advantages and Limits of PEPs

Several studies or projects and articles have underlined the advantages of PEP solutions. Generally conclusions shows the benefit of splitting and the effective gain offered by PEP relatively to standard TCP. This gain is mainly possible because of the use of a dedicated TCP version on the satellite link which allows to avoid the capacity under-utilization generally observed with the end-to-end approach.

Results presented in (Caini et al., 2006) indicate that the PEPsal architecture is able to avoid the penalization encountered by satellite connections in heterogeneous environments. As previously explained, PEPsal is based on the use of a specific and dedicated TCP version to satellite link known as TCP Hybla. TCP Hybla has first been presented in (Caini & Firrincieli, 2004) as a promising solution to the problem of performance disparity in heterogeneous networks due to different RTTs. Unlike in classical TCP versions, in TCP Hybla the speed of the increase of the congestion window is not based on the RTT. Consequently, TCP Hybla is not expected to suffer from the important RTT induced by satellite links. Such results are confirmed in (Caini et al., 2007), and in particular, the goodput observed on a satellite link for different RTTs is given considering different approaches. PEPsaI performance is compared with end-to-end NewReno, SACK, and Hybla. For instance, in case of random loss on the satellite link and considering a typical RTT for a geostationary satellite, the PEPSAL solution shows a goodput about 7 times greater than end-to-end NewReno.

Kapoor et al. (2005) shown the benefits of the new transport protocol named XCP (eXplicit Congestion control Protocol) and uses it as the alternative transport protocol on a PEP satellite link. In the studied case XCP provides access to full link capacity for a single source, which represents more than 70 time the bandwidth usage of a source using only Reno TCP.

More recently, Cruickshank et al. (2008) presents the ETSI broadband satellite multimedia (BSM) PEP architecture which includes the satellite terminal protocol stack, PEP usage scenarios and security configurations for successful PEP implementations. Here again the factors and reasons which encourage PEP solutions are

explained. Then an overview of ETSI BSM is given and different PEP scenarios are described. The conclusions explain that PEPs have the potential to improve TCP and web performances over BSM network.

Again in the context of ETSI BSM reference model, the aim of Giambene and Hadzic (2008) is to consider the problems of TCP performance in broadband GEO satellite networks and to propose a cross-layer approach for a transport layer PEP that makes spoofing actions on acknowledgements to modify them in case of satellite network congestion. The proposed cross-layer signalling allows information exchanges between DAMA (Demand Assigned Multiple Access) implemented in the link layer and TCP in transport layer. Thus this PEP can prevent congestion in the satellite network. This PEP proposal is innovative in the sense that it is non-transparent and requires the application of new cross-layer signalling at both the satellite terminal (ST) and the PEP/gateway. Nevertheless further investigations are needed to fully define the arguments for or against this particular approach.

As described in Caini et al. (2008) an alternative approach arises from the DTN (Delay/Disruption Tolerant Networking) architecture. This architecture is based on the introduction of the new bundle layer in the protocol stack which is inserted between the application and the transport layer. In this new architecture, end-to-end transport protocol features are confined to homogeneous network segments, while real end-to-end data reliability across a heterogeneous network is provided by the bundle layer. The aim of the paper is to discuss the advantages and disadvantages of this approach compared to more conventional solutions. DTN performance is compared with end-to-end NewReno, Hybla and PEPsal. The performances are evaluated by using a testbed. The results show that DTN coupled with TCP Hybla as transport protocol on the satellite segment outperforms TCP NewReno and shows performances close to that achieved by both end-to-end Hybla and PEPsal solutions. Furthermore, by introducing a sliding window, DTN performance is boosted and eventually the best. The DTN solution have been more recently discussed in Caini et al. (2011) which confirms previously observed trends. Furthermore this paper studies DTN security. The advantages over satellite architecture are examined with the threats faced in satellite scenarios, and the open issues. Also, the relation between DTN and quality of service (QoS) is investigated.

Finally, a lot of studies explaining the benefits of PEP solution in satellite context have been proposed. But, as underlined in the introduction and detailed in this section, most of these studies compare PEP solutions with end-to-end ones based on old standard TCP (e.g. NewReno, Vegas). Furthermore, a lot of the PEP studies compare performances of split and end-to-end TCP in a single user topology and even in presence of a single TCP connection. In the same time new TCP flavours, like TCP Cubic, have been proposed and widely deployed in order to match with new links properties, in particular with long high speed links.

We have also to keep in mind that since splitting TCP connection violate end-to-end semantics of TCP, it may present some limits and even drawbacks. For instance, IPSEC which is getting widespread in IP networks in order to provide VPN (Virtual Private Networks) functionality for instance, implies the encryption of TCP headers which is incompatible with PEP deployment since PEP has to access to TCP header without being the authenticated receiver. It has often been concluded, like in Chitre et al. (2004), that the solution to handle TCP protocol inefficiencies over high speed satellite links using the PEP is not applicable in the IPSEC environment. This problem has been investigated, and (ETSI, 2009) already explained that at least through the use of security policies, an end user can select the use of IPsec for some traffic and not for other traffic. In this case, PEP processing can be applied to the traffic sent without IPsec. Another alternative is to implement IPsec over the satellite link between the two satellite gateways where PEPs have been deployed. Even if an end-to-end use of IPsec is not implemented, the traffic between the two PEPs will be protected. Note that in both cases, we do not have a real TCP connection based on end-to-end IPsec encryption and PEPs.

4. Methodology and Tools

The study presented in this paper is mainly based on the use of a satellite link emulation platform developed at ENAC. This platform allows the emulation of the link capacity, the propagation delay and the packet error rate associated with a satellite link, as well as the jitter induced by access links. In this section, we present the architecture of the emulation platform, as well as the software tools used in our experiments.

4.1 Platform Presentation

4.1.1 Hardware Architecture

In its first configuration, our emulation platform consists in three linux servers and two 1000 Base TX Ethernet switches. One linux computer hosts the emulation application, the other two running the native applications. For the experiments presented in this paper, though, we added another linux computer as an end host (see Figure 4) and used one of the previous host as a router, with PEPSal software installed. This platform is IPv6 capable, though in the scope of this study, only IPv4 has been used as PEPSal technology is not available with IPv6.

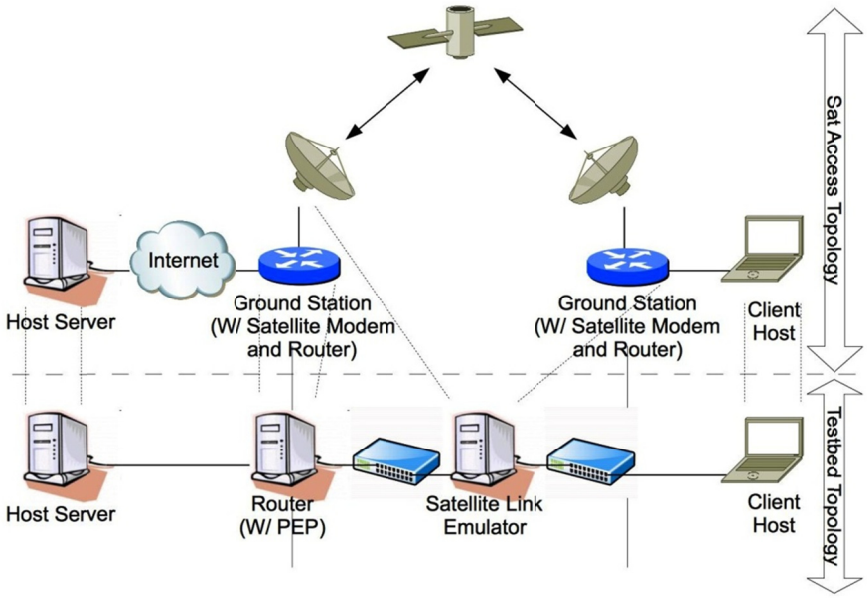

Figure 4. Satellite Internet access vs testbed topology

In all our experiments, we used a satellite link with 10Mbits/s link capacity, the link between the PEP router and the Satellite link emulator has hence be configured at this speed so as to better fit with the real topology.

4.1.2 Software Architecture

The philosophy of the software conception has been to give priority to the user experience and to make the emulator usable for an average user, unaware of the subtleties of the Linux kernel parameters for traffic control. In consequence, the GUI (see Figure 5) of the software allows the user to specify networks features and to keep ignoring the internal Linux machinery.

Three parameters can be specified in the emulation configuration GUI:

- The propagation delay in millisecond. It has associated jitter and probability law, as well as a correlation percentage.
- The loss percentage and correlation percentage between losses.
- The link capacity in bits per second.

To emulate a new link, the user has to associate two sets of parameters to two of the available network interfaces, one for each traffic flow direction. The parameters can be different for each selected interface which allows the user to emulate asymmetric networks (for example one which uses a satellite link on one way and a terrestrial one on the other). Once selected, an emulated link can be launched, stopped, restarted and deleted.

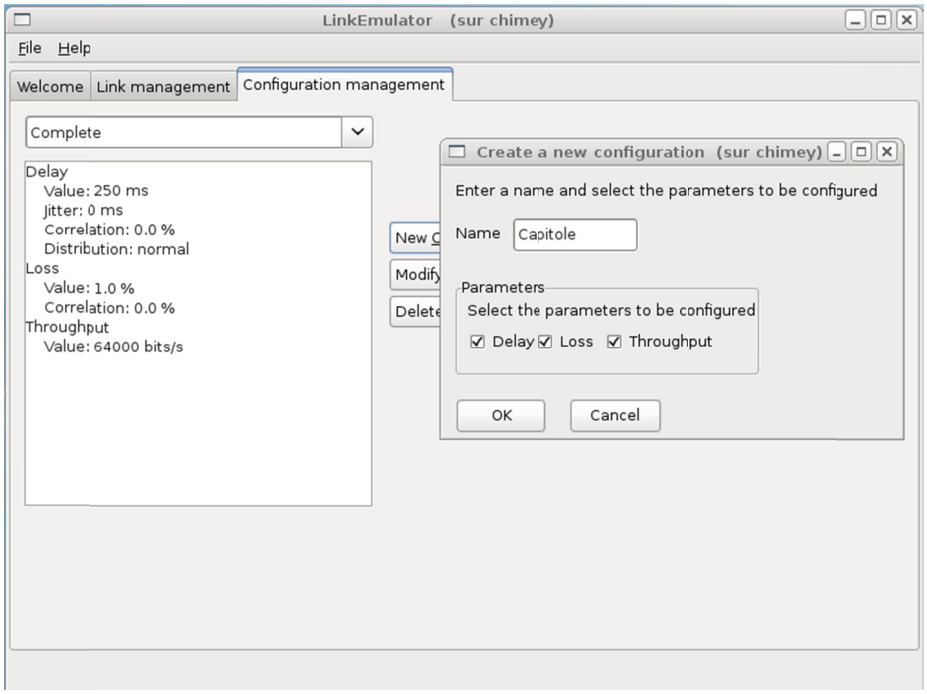

Figure 5. Link emulator software GUI

In order to gain independence from any external tool, we chose not to build the network emulator on top of the command line tool *tc* which is the standard tool for Linux traffic control (Hubert et al., 2003). We chose instead to directly address kernel modules via the C open source library libnl. Besides libnl, the various components of the software were written in JAVA and interfaced with the C component by using the Java Native Interface framework. The graphical user interface is based on the SWT/JFACE libraries, chosen for performance and aesthetic reasons. Attention has been paid to make a clean separation between the various components such that it should be possible to substitute another implementation to one of them without changing the others.

We used two modules in kernel space, netem and tbf. Each emulates some parameters of the link, Netem (Hemminger, 2005) is used to emulate delays and losses, whereas tbf (Hubert et al., 2003) is used to emulate the delay of the link. The Token Bucket Filter (TBF) is in charge of slowing down packets so they respect a given maximum rate. In the terminology introduced in (Blake et al., 1998), TBF is a traffic shaper.

4.2 Software Tools

In order to measure TCP performance, we decided to use two approaches, fine grain TCP behaviour measurement and Application level goodput measurements. Those measures were made both in the single user case with one TCP flow and in the multi-user case with several concurrent flows.

To this end, we used both COTS tools, namely *TCPDump*, *tcptrace*, *wget* and *wput*, and one homemade tool, *TCPMeter*. Those tools are described in the following sections.

4.2.1 COTS Tools

(a) TCP behaviour measurement

In order to assess what TCP parameters were limiting its performance, we used *TCPDump* (Note 1) and *tcptrace* (Note 2). *TCPDump* is a well known and widely used packet analyser, and *tcptrace* is a tool that analyses *TCPDump* capture files to extract specific TCP connection details.

TCPDump was configured to capture packets on the sender side and to dump its output in a capture file in pcap format. This file was then processed through tcptrace to extract a number of graphs and measures for each TCP connection.

The main graph we extracted shows the number of unacknowledged TCP segments on the sender side along with the advertised receiver window (called Outstanding data graph in *tcptrace* manual). The number of unacknowledged outstanding TCP segments is a measure of TCP's *flight size* as long as the TCP sender queue contains data (i.e. as long as TCP as data to send it will send up to its allowed *flight size*). This graph thus show

clearly which of the TCP windows, *rwnd* or *cwnd*, limits its throughput.

(b) TCP traffic generation

To generate TCP traffic, both with HTTP and FTP style flows, we first used two programs named *wget* (Note 3) and *wput* (Note 4).

Wget is a tools for retrieving files from HTTP, HTTPS or FTP servers. We chose this tool because it is easy to integrate in shell scripts and thus allows for an easy automation of tests.

Wput is a tool similar to *wget* but allows for sending to FTP servers and thus was easier to use along with *TCPDump* on the sender side.

Those tools proved useful, but when we increased the number of TCP connections, we noticed that they reached a maximum throughput that was not limited by TCP performance (this occurred with at about 7 connections in parallel on our testbed). This comes from the fact that both tools use disk access to send and receive data and thus are not only limited by general network performance, but also by other system performances. This lead us to develop a new tool we called *TCPMeter* for multi-flow TCP traffic generation.

(c) TCPMeter

In order to free ourselves from the possible disk speed limitations, we wrote a small tool that generates several concurrent TCP flows without using data from the disk. This tool uses boost C++ threads (Note 5), each thread managing a TCP connection. A main thread monitors the sending and receiving threads and collect statistics on each connection once they terminate. The collection of statistics is done on the receiving side and is then transmitted to the sender side for easier collection of tests results (remember the *TCPDump* capture are done on the sender side).

TCPMeter allows the user to specify the number of bytes to send on each connection and the number of connection to launch in parallel. The first parameters allows for varying length tests, and the second allows for testing TCP with multiple flow that compete for network resources.

TCPMeter outputs its results in an easy to read text file containing one line for each connection with its id, the port numbers used, the number of bytes sent, the times at which the connection started and finished and the measured average goodput at the Application level.

Here is an example command line used to create three TCP connections sending 100000 bytes on

each connection to a host called *orval*:

```
TCPMeter_V1.0 -n 3 -b 100000 -v -s orval
```

On *orval* the only command necessary is:

```
TCPMeter_V1.0 -r -v
```

The -v option is used to generate detailed debug information on *TCPMeter* behaviour.

5. TCP Performances Assessment over Geostationary Satellite Link

5.1 Considered TCP Flow Types and Performances Parameters

We considered in our study two kinds of flow types. First, in order to validate known results and expected advantages of PEPs, we considered basic flows obtained during the reception of a simple webpage. We then investigated the TCP behaviour and performances considering a single file transfer. Hence we make several measurement campaigns using our testbed and each time we observe the results and performances considering at least three different scenarios concerning TCP solutions:

• end to end TCP NewReno connection (*e2e_Nreno*),

• end to end TCP Cubic connection (*e2e_Cubic*),

• split TCP connection using PEPsal with TCP Hybla on the satellite linka nd TCP Cubic on the end systems (*Cubic & PEPSal*).

Doing so, our objective is to consider *e2e_NReno* as the reference case for our measurements, the second as the typical case considering current TCP Cubic deployment, and finally the third one as an improved approach in satellite link case.

Furthermore, we extend our study considering two conditions: the case in which a single user uses exclusively its satellite Internet access for web browsing or file transfer and the case in which he shares this access with others TCP connections.

A main metric in our study is the time needed to transfer an amount of data, such as a set of contents from a web page or a single file. Generally, the transfer speed is described as the throughput which includes the original data, the protocol overheads (i.e. such as packet headers and signalisation) and packets that are retransmitted. Unlike throughtput *t*, goodput *g* only measures the transfer speed of the original data. Goodput can be calculated by dividing the size of a transmitted file by the time it takes to transfer it. It has to be noted that goodput can be considered as the application level throughput or the throughput as seen by the end user. Because our study aims at comparing performances offered to the end user by different TCP solutions, the other main metric considered as a result is the goodput in KBytes.

As our contribution not only focusses on the single user case, we also took into account other metrics. In this case of multiple TCP connection sharing a single satellite link we used the sum of the N different individual goodputs. This metric is named aggregate goodput and the TMRG (Transport Modeling Research Group) at IETF proposed in (IRTF, 2011) a formula for this metric:

$$G(t)=\sum_{i=1}^{N} g_i(t) \tag{1}$$

It must also be underlined that because packet losses are randomized and because a single experiment of a case may generate a particular set of conditions, it is necessary to play each case several time in order to ensure an accurate assessment. So, each experiment corresponding to a single scenario, for instance e2e NReno, has been played twenty times. Hence, we used those others metrics:

• *mean aggregate goodput*, considering the set of results obtained after several experiments on one scenario, this metric represents the mean of the aggregate goodput on each experiment.

• *max aggregate goodput*, considering the set of results obtained after of several experiments of one scenario, this metric represents the maximum aggregate goodput that have been observed.

• *mean total aggregate goodput gain* (vs *e2e_NReno*), because *e2e_Cubic* and *Cubic & PEPSal* are expected to improve performances relatively to *e2e_NReno*, we measured the gain offered by each of them. Once again, considering the set of results obtained after a number experiments on one scenario, this metric represents the mean observed gain.

• *mean fairness*, fairness is a very important feature for any TCP variant. It refers to the capacity of a specific TCP version to ensure a fair band subdivision among competing connections that use the same version of the protocol. Jain et al. (1984) proposed a formula for this metric sometimes called the Jain's index *J*, that have been often used in TCP fairness assessments:

$$G(t)=(\sum_{i=1}^{N} g_i)^2 / N(\sum_{i=1}^{N} (g_i)^2) \tag{2}$$

Once again, considering the set of results obtained after several experiments on one scenario, this metric represents the mean fairness. It has to be underlined that in order to indicate the reliability of this result, we also give the 98% confidence interval.

5.2 Single User Topology

5.2.1 Single User Web Browsing

In order to have a first feeling and to check if we observed the expected advantages of PEP, we have done a trivial measurement campaign on our testbed. This first step allows us to measure for web page load, the throughput and the total time elapsed to receive the considered webpage. The chosen webpage is quite simple with 7 objects: a main html document and six jpeg pictures (with an average picture size of 847 Kbytes). Finally, the overall webpage content shows a size of 4955 Kbytes. Using the linux *wget* command with adequate options in order to avoid the use of local proxy and to force the load of all files, we observe the time the web page takes to load and we deduce the obtained goodput. The *linkemulator* is configured in order to emulate an errored satellite link inducing a packet error rate (PER) of 1% and the typical propagation delay of 125 ms.

Of course, this experience is not strictly equivalent with the results that could be obtained using a web browser, mainly because doing so we serialize the transfer of each objects contained in the considered web page. Nevertheless, this approach has the interest to ensure that no hidden parameters or functionalities will induce on the results. And, of course, as we use the same method for each tested TCP solutions the results can be directly compared. The results are presented in Table 1.

Table 1. Web page load with different testbed configuration

Testbed Configuration	Web Page Load Time (s)	Goodput (KBytes/s)
e2e_NewReno	158	31.4
e2e_Cubic	72	68.8
Cubic & PEPsal	37	133.9

As expected, we observe that the use of PEP improves TCP performances observed at the end point with an offered goodput about four times greater than the configuration based on end to end TCP NewReno. Nevertheless, it has also to be underlined that end to end TCP Cubic solution already allows to double the goodput comparatively to end to end TCP NewReno. As TCP Cubic is with CTCP currently widely deployed, it has from now on to be considered as a reference point to assess the gain offered by PEP approaches. Furthermore, we have to keep in mind that the entire offered capacity is in this studied case dedicated to the considered TCP connection, e.g. no exogenous TCP traffic.

5.2.2 Single File Transfer

As first experiment of this new part dedicated to file transfer and once again in order to verify if we obtain the expected trends, we assess end user performances obtained in case of file transfer in our main scenarios. And yet here, the *linkemulator* is configured on our testbed in order to emulate an errored satellite link (PER of 1%) and a propagation delay of 125ms observed in case of geostationary satellite link. The considered file have a size of 5 MBytes and its download is emulated using our home made tool TCPMeter presented in 4.2. The results are shown in Table 2.

Table 2. Single file transfer with different testbed configuration

Testbed Configuration	File transfer Time (s)	Goodput (KBytes/s)	Avg/Max owin (KBytes)
e2e_NewReno	77.5	26.9	17.3/56.5
e2e_Cubic	34.8	61.9	34.6/97.1
Cubic & PEPsal	18.8	121.3	9.5/23.2

In the present experiment, the entire offered capacity is dedicated to a single TCP connection, e.g. no exogenous TCP traffic. Like in the previous section, we observe the expected trend: *e2e_Cubic* case improve greatly performances comparatively with *e2e_NReno* but the best performances are provided by the third solution which is based on PEPSal. It is important to understand that such results are due to the relative aggressiveness of the new TCP variants, particularly TCP Hybla which is used by PEPSal on the satellite hop. On a packet loss event this allows to retrieve more quickly the congestion window size reached before the loss and hence to better fill the offered capacity on the link. As we have here a single TCP connection in each experiment, the aggressiveness of the best solutions does not present drawbacks. Thanks to *tcptrace*, the third column of Table 2 shows the average and maximum amount of outstanding data. In our cases, e.g. in presence of high packet loss, the amount of unacknowledged outstanding data, or in flight data, is bounded by *cwnd*. The values obtained in *e2e_Cubic* case explain finally the gain observed on the goodput and file transfer time. But when PEPSal is used in order to split the end to end TCP connection, we observed nevertheless smaller values for the goodput. This is coherent with the fact that the TCP connection has been split and *tcpdump* command is used at the end point. We observe there the average and maximum amount of outstanding data concerning the TCP connection between the host server and the first ground station.

The question we address in the next section is, what is the expected gain if the satellite link is shared by several TCP connections based on the same approach?

5.3 Multiple Simultaneous File Transfers

This last part is dedicated to the study of TCP performances with multiple simultaneous TCP connections sharing a single satellite link. The number of simultaneous TCP connections is taken from 1 to 20. As we generate these several TCP flows of data from a single client node to a single server node, we have first verified on our testbed that such a configuration does not imply limits induced by the end points. To do so we have done a first measurement campaign without any link emulation, e.g. based on the real testbed properties and characteristics. We have verified that even with 20 simultaneous TCP connections the amount of transferred data is only bounded by the link capacity between the two nodes which will emulate the ground stations. Hence, the following experiments may be considered equivalent to the case of several TCP connections from different client nodes to different server nodes, which is equivalent to a LAN from which several TCP connections are opened with several servers all sharing a geostationary satellite link. We consider our three configurations: *e2e_NReno*, *e2e_Cubic*, and *Cubic & PEPSal*. Yet again our testbed is configured in order to behave as a geostationary satellite link offering Internet access. As explained in subsection 5.1, each experiment corresponding to a testbed configuration, for instance *e2e_NReno*, have been played twenty times and results averaged in order to provide reliable results.

The Figure 6 presents the obtained results in term of *mean total aggregate goodput*. As previously underlined, with a single TCP connection, the use of PEPSal allows to improve greatly the performance comparatively to a solution based on *e2e_NReno*. Nevertheless, the use of *e2e_Cubic* already offers a gain of 100%. But the main interest of these figure is to reveal that the greater the number of simultaneous TCP connections sharing the satellite link, the fewer the gain offered by PEPSal. Indeed, starting at 3 or 4 simultaneous TCP connections, PEPSal approach does not offer any gain comparatively to end-to-end TCP Cubic configuration which allows to ensure a gain even for 20 simultaneous connections.

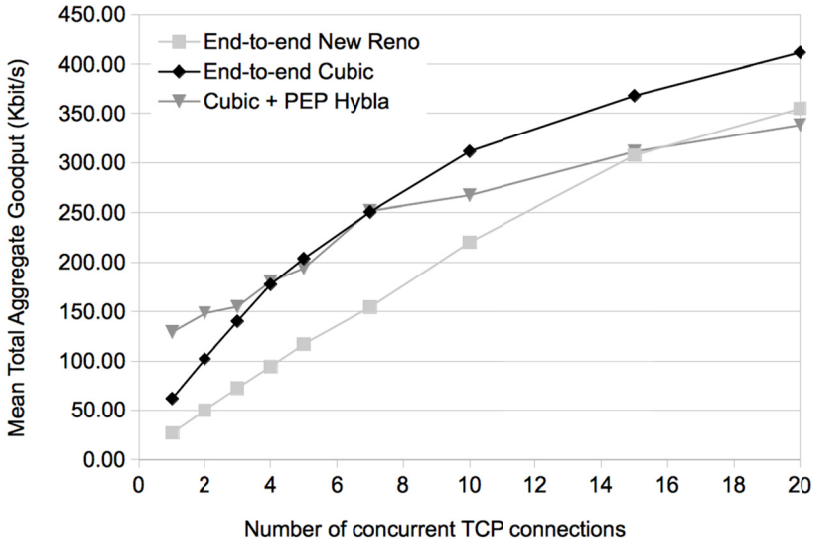

Figure 6. Mean total aggregate goodput

The Figure 7 presents the *maximum total aggregate goodput* versus the number of simultaneous TCP connections sharing the satellite link. This new result allows to confirm the trends of Figure 6. In this case, PEPSal solution shows the best gain up to 10 connections where it becomes almost equivalent to *e2e_Cubic* in term of offered performances. This may be interpreted as a first indication of the unfairness of TCP Hybla proposed with PEPSal. Actually, Figure 6 shows that *e2e_Cubic* results are equivalent or better than PEPSal's for 4 or more connections, instead of 10 in Figure 7. This indicates that the results obtained during the twenty experiments with PEPSal approach are more scattered around the mean value.

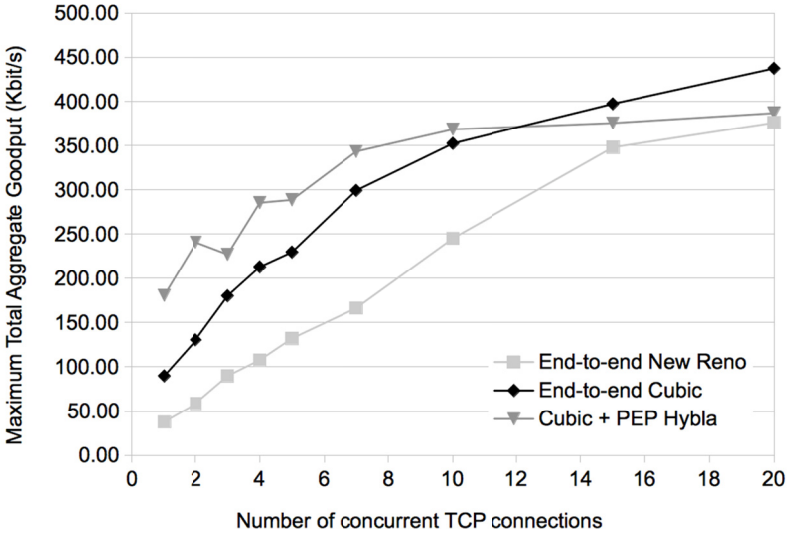

Figure 7. Maximum total aggregate goodput

The Figure 8 shows the gain observed on the mean total aggregate goodput by *e2e_Cubic* and PEPSal approaches comparatively to *e2e_NReno* versus the number of simultaneous TCP connections. The aim of this metric is here to confirm the previous results. It underlines that even if the gain provided by PEPSal is important when the number of simultaneous TCP connections is quite low, this gain decreases drastically as the connection number increases. With end-to-end TCP Cubic approach, the gain seems more stable, and considering PEPSal cost and limits exposed in 3.3, for instance its complexity or unsuitability with IPSec, *e2e_Cubic* should be preferred as soon as the number of simultaneous TCP connection is greater than three.

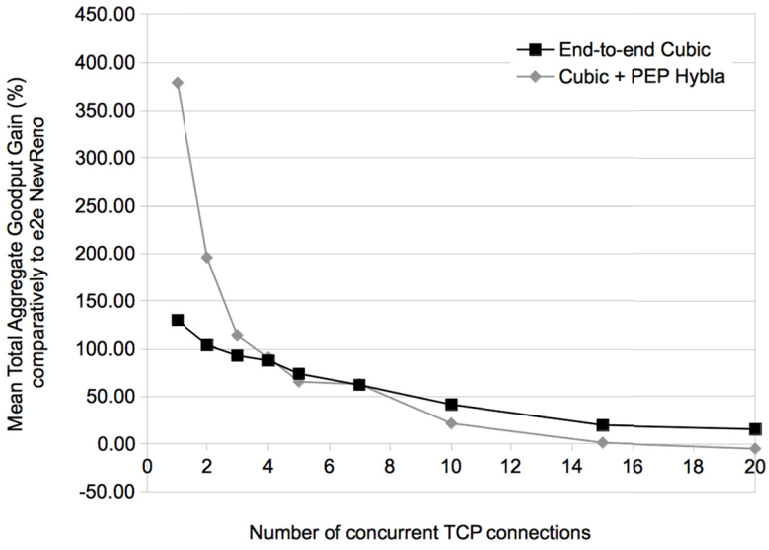

Figure 8. Mean total aggregate goodput gain

Finally, the Figure 9 presents a comparison of each 3 main scenarios in term of the fairness index as introduced in (Jain et al., 1984). This index always lies between 0 and 1. A fairness index of 1 indicates a complete fair behaviour of the considered TCP version. In order to provide reliable results, we enhance this metric with a 98% confidence interval. Like end-to-end TCP NewReno approach, end-to-end TCP Cubic shows very good results, at least 0.97 with quite small confidence interval and whatever the number of simultaneous TCP connections. And the results obtained with split TCP approach implemented with PEPSal reveal a less fair behaviour, always

under 0.94 as soon as there is more than a single TCP connection. Furthermore, the confidence intervals measured are greater, about four times those obtained with TCP Cubic. This is symptomatic of the relative unpredictability of TCP Hybla. Moreover, TCP Hybla aggressiveness allows to recover rapidly the window size reached prior to a loss but it also tends to favour some TCP connections at the expense of the others.

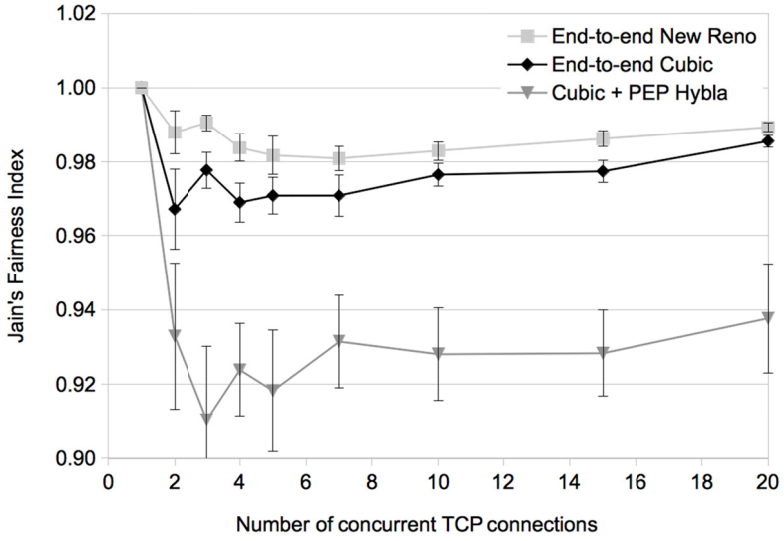

Figure 9. Mean fairness

6. Conclusions and Recommendations

The present paper has been motivated by two main goals: providing a new survey on improving TCP performances over geostationary satellite link taking into account recent high speed TCP variants, and comparing the performances provided by three approaches (namely end-to-end TCP NewReno, end-to-end TCP Cubic and split TCP connection using PEPSal) according to a set of metrics. The study is based on a configurable link emulator implemented on a testbed and a set of COTS and homemade tools involved in the different experimental steps. In the light of this study we recommend to limit the use of PEPSal for situations where very few TCP flows will be compete for the resources on the satellite link. Actually, we show that as soon as several TCP connections share the satellite link, a solution based on end-to-end TCP Cubic (currently deployed on linux boxes) should be preferred. This latter, contrary to the considered PEP approach, does not operate in violation of the TCP end-to-end paradigm and hence does not introduce vulnerability. The end-to-end TCP Cubic solution ensures a real end-to-end reliability, and gives better results in term of goodput and fairness.

Nevertheless the gain in term of observed end user performances stay really interesting with the PEPSal approach particularly in the case of a single TCP connection on the satellite link. So it would certainly be interesting to investigate the feasibility of an adaptive solution. This solution could be based on an active TCP connections counter which would allow to activate PEP solution at the ground station only when a threshold is reached.

Another further work would consist in studying the TCP friendliness of each solution that have so far been considered independently in each experiment of the present paper. Friendliness refers to the capacity to ensure a fair link capacity subdivision among competing connections that use different versions of the protocol. Actually, ETSI recommendations plan that each TCP connection via the satellite link will be able to operate in accordance with its own intrinsic needs. This means that at the same time several TCP connections might be present on a link with some based on PEPSal and others based on different end-to-end TCPs. As friendliness is also an important feature for any version of TCP protocol, this certainly motivates a new study which should address this point.

References

Allman, M. (1999). Enhancing TCP Over Satellite Channels using Standard Mechanisms. *Internet RFC 2488*.

Allman, M., Hayes, C., Kruse, H., & Osterman, S. (1997). *TCP Performance over Satellite Links*. 5th Int.

Conference on Telecommunication Systems Modelling and Design, pp. 1-13.

Bisio, I., Marchese, M., & Mongelli, M. (2009). Performance Enhanced Proxy Solutions for Satellite Networks: State of the Art, Protocol Stack and Possible Interfaces. *Personal Satellite Services, 15*, 61-67. http://dx.doi.org/10.1007/978-3-642-04260-7_8

Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., & Weiss, W. (1998). RFC 2475: An Architecture for Differenctiated Services. *Internet RFC 2475*.

Border, J., Kojo, M., & Griner, J. (2001). Performance enhancing proxies intended to mitigate link-related degradations. *Internet RFC 3135*.

Brakmo, L. S., O'Malley, S. W., & Peterson, L. L. (1994). *TCP Vegas: New techniques for congestion detection and avoidance*. Proceedings of the conference on Communications architectures, protocols and applications, pp. 24-35, August 31 - September 02, 1994, London, United Kingdom. http://dx.doi.org/10.1145/190809.190317

Caini, C., Cornice, P., Firrincieli, R., & Lacamera, D. (2008). A DTN Approach to Satellite Communications. *IEEE Journal on Selected Areas in Communications, 26*(5), 820-827. http://dx.doi.org/10.1109/JSAC.2008.080608

Caini, C., Cruickshank, H., & Marchese, M. (2011). Delay- and Disruption-Tolerant Networking (DTN): An Alternative Solution for Future Satellite Networking Applications. *Proceedings of the IEEE, 99*(11), 1980-1997. http://dx.doi.org/10.1109/JPROC.2011.2158378

Caini, C., & Firrincieli, R. (2004). TCP Hybla: a TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking, 22*, 547-566. http://dx.doi.org/10.1002/sat.799

Caini, C., Firrincieli, R., & Lacamera, D. (2006). PEPsal: a Performance Enhancing Proxy designed for TCP satellite connections. *In: Vehicular Technology Conference*. http://dx.doi.org/10.1109/GLOCOM.2006.474

Caini, C., Firrincieli, R., & Lacamera, D. (2007). PEPsal: a Performance Enhancing Proxy designed for TCP satellite connections. *In: IEEE AE SYSTEMS MAGAZINE*. http://dx.doi.org/10.1109/MAES.2007.4301030

Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. Y., & Wang, R. (2002). TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks, 8*(5), 467-479. http://dx.doi.org/10.1023/A:1016590112381

Chitre, P., Karir, M., & Hadjitheodosiou, M. (2004). TCP in the IPSEC Environment. *In: AIAA International Communications Satellite Systems Conference and Exhibit (ICSSC)*. http://dx.doi.org/10.2514/6.2004-3207

Cruickshank, H., Mort, R., & Berioli, M. (2008). PEP Architecture for Broadband Satellite Multimedia (BSM) Networks. *Workshop on Satellite PEPs Current Status and Future Directions*.

Davern, P., Nashid, N., & Sreenan, C. J. (2011). HTTPEP: a HTTP Performance Enhancing Proxy for Satellite Systems. *International Journal of Next-Generation Computing, 2*(3), 242-256.

ETSI. (2009). Satellite Earth Stations and Systems (SES); Broadband Satellite Multimedia (BSM); Performance Enhancing Proxies (PEPs). *In: ETSI TS 102 676 V1.1.1*.

ETSI. (2011a). Digital Video Broadcasting (DVB), Second Generation DVB Interactive Satellite System, Digital Video Broadcasting (DVB), Second Generation DVB Interactive Satellite System; Part 3: Higher Layer for Satellite standard RCS2. *In: ETSI TS 101 545-3*.

ETSI. (2011b). Digital Video Broadcasting (DVB), Second Generation DVB Interactive Satellite System, Part 1: Overview and System Level specification. *In: ETSI TS 101 545-1*.

ETSI. (2011c). Digital Video Broadcasting (DVB), Second Generation DVB Interactive Satellite System, Part 2: Lower Layers for Satellite standard RCS2. *In: ETSI TS 101 545-2*.

Giambene, G., & Hadzic, S. (2008). Cross-Layer PEP-Spoofer Approach to Improve TCP Performance in DVB-RCS Networks. *In: Workshop on Satellite PEPs, Current Status and Future Directions, ESTEC, Noordwijk*.

Hemminger, S. (2005). Network Emulation with NetEm. *In: Linux Conf Au*.

Hubert, B., Graf, T., Maxwell, G., van Mook, R., van Oosterhout., Schroeder, P. B., … Larroy, P. (2003). *Linux advanced routing and tra#c control HOWTO rev1.43*.

IRTF. (2011). *The Transport Modeling Research Group*. Retrieved from

http://trac.tools.ietf.org/group/irtf/trac/wiki/tmrg

Jacobson, V., & Karels, M. J. (1988). *Congestion Avoidance and Control*. Proceedings of the Sigcomm '88 Symposium, Stanford, pp. 314-329. http://dx.doi.org/10.1145/52325.52356

Jain, R., Chiu, D., & Hawe, W. (1984). A Quantitative Measure of Fairness And Discrimination for Resource Allocation in Shared Computer System. *DEC Technical Report 301*.

Kapoor, A., Falk, A., Faber, T., & Pryadkin, Y. (2005). Achieving faster access to satellite link bandwidth. *In: INFOCOM*. http://dx.doi.org/10.1109/INFCOM.2005.1498578

Liu, S., Basar, T., & Srikan, R. (2008). TCP Illinois: A loss and delay-based congestion control algorithm for high-speed networks. *Performances Evaluation, 65*, 417-440. http://dx.doi.org/10.1016/j.peva.2007.12.007

Marcondes, C., Matthews, J., Chen, R., Sanadidi, M. Y., & Gerla, M. (2008). A Cross-Comparison of Advanced TCP Protocols in High Speed and Satellite Environments. *Advanced Satellite Mobile Systems, 2008. ASMS 2008. 4th.*, pp. 179-185. http://dx.doi.org/10.1109/ASMS.2008.38

Mathis, M., Mahdavi, J., Floyd, S., & Romanow, A. (1996). TCP Selective Acknowledgement Options. *Internet RFC 2018*.

Partridge, C., & Shepard, T. J. (1997). TCP/IP Performance over Satellite Links. *IEEE Network, 11*(5), 44-49. http://dx.doi.org/10.1109/65.620521

Rhee, I., & Xu, L. (2005). CUBIC: A New TCP Friendly High Speed TCP Variant. *CM SIGOPS Operating Systems Review - Research and developments in the Linux kernel, 42*(5), 64-74. http://dx.doi.org/10.1145/1400097.1400105

Roseti, C., Luglio, M., & Zampognaro, F. (2010). Analysis and Performance Evaluation of a Burst-Based TCP for Satellite DVB RCS Links. *IEEE/ACM Transactions on Networking, 18*(3), 911-921. http://dx.doi.org/10.1109/TNET.2009.2033272

Sourceforge.net. 2012 (April). *PEPsal: Project Web Hosting - Open Source Software*. http://pepsal.sourceforge.net/

Tan, K., Song, J., Zhang, Q., & Shridaran, M. (2006). Compound TCP: A Scalable and TCP Friendly Congestion Control for High Speed Networks. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1-12. http://dx.doi.org/10.1109/INFOCOM.2006.188

Trivedi, S., Jaiswal, S., Kumar, R., & Rao, S. (2010). Comparative performance evaluation of TCP Hybla and TCP Cubic for satellite communication under low error conditions. *In: IMSAA 2010 IEEE*. http://dx.doi.org/10.1109/IMSAA.2010.5729424

Xu, L., Harfoush, K., & Rhee, I. (2004). Binary increase congestion control (BIC) for fast long-distance networks. *INFOCOM, 4*, 2514-2524. http://dx.doi.org/10.1109/INFCOM.2004.1354672

Notes

Note 1. Tcpdump http://www.tcpdump.org/

Note 2. tcptrace http://www.tcptrace.org/

Note 3. wget http://www.gnu.org/software/wget/

Note 4. wput http://wput.sourceforge.net/

Note 5. Boost thread library http://www.boost.org/doc/libs/1_49_0/doc/html/thread.html