Comparison of TCP variants on satellite communications

Claudio Palazzi
University of Padua
Padua, Italy
cpalazzi@math.unipd.it

Cristian Coreggioli
University of Padua
Padua, Italy
cristian.coreggioli@studenti.unipd.it

***Abstract*—TCP is one of the most used transport protocol over the Internet. It is a connection-oriented protocol initially developed for wired links. It can provide reliable delivery, congestion control and flow control mechanisms.**
In recent years, together with wired communication also wireless communication become very popular. Traditional TCP versions can be used also in wireless scenarios even if with lower performances. As TCP was initially developed for wired links, there is the need to develop TCP variants that can work with high performance on wireless links too. So, many TCP variants has been introduced like Reno, New Reno, Tahoe, Hybla. Some of them are more prone to wireless environments.
A particular wireless scenario is a satellite network communication. It is characterize by very high latency and low reliability. In this paper, a network simulator tool is used to compare three TCP variants on satellite networks by focusing on congestion windows computation and total number of received packets.

***Index Terms*—TCP New Reno, TCP Westwood, TCP Hybla, Satellite communication, NS3**

I. Introduction

Now a days, TCP is one of the most used transport protocol on the Internet together with UDP. TCP is used for connection-oriented transmissions while UDP is used for connection-less transmissions. These two protocols together comprise almost all the traffic on the Internet. They both work at transport layer in the TCP/IP stack 1.
The TCP/IP stack is also composed by other layers that are the Application, Internet and Link layer. Each of them has a specific function and a specific set of protocols.

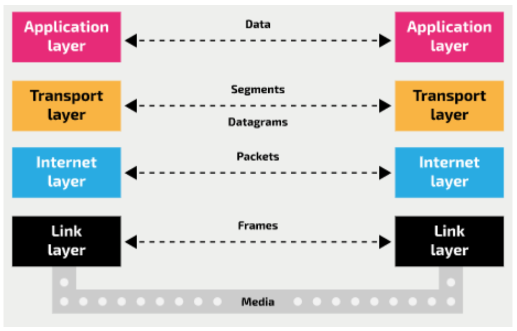


Fig. 1. TCP/IP Stack

For the purpose of this paper, only the TCP protocol is considered.

a) Contributions: In this work, we make a comparison between different TCP variants in a satellite network. The idea is to use a network simulator tool to replicate a satellite network topology and make experiments with different versions of TCP to analyze their behaviour in this particular wireless scenario. The purpose of the paper is to compare the results of the simulations and see which TCP variant is more prone to be used in a satellite wireless scenario.

Organization. The rest of the paper is structured as follows: In section II, there is an introduction to the TCP protocol with the focus on the congestion control mechanisms and a brief description of the three TCP variants that will be used for the experiments. In section III, we describe the satellite topology that aims to be replicate by a network simulator tool. In section IV, there is an introduction to the NS3 network simulator. The section V explains how the topology in section III are actually implemented in NS3 and which experiments has been done. In section VI, the results of the experiments are plotted and commented.

II. Background

A. TCP protocol

The role of a transport protocol is to provide end-to-end communication services to the applications. TCP [1] is a connection-oriented transport protocol where a connection between a Client and a Server must be established before start sending data. This process is named ”Three-way Handshake” 2 and it is described in the following steps:

1) **SYN**: the Client wants to establish a connection with a Server, so it sends a segment with a SYN (Synchronize Sequence Number) number which informs the Server that the Client wants to start a communication with the provided sequence number;
2) **SYN-ACK**: the Server responds to the Client request with SYN-ACK message. The Acknowledgement (ACK) is used to confirm the correct receive SYN packet. Together with this, the Sender choose another SYN number and sends it back to the Client, because also the Server wants to open a connection to the Client;
3) **ACK**: the Client acknowledges the response related to the received SYN message sent by the Server.

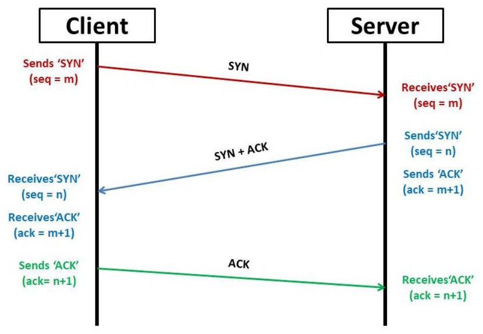


Fig. 2. Three-way Handshake

In this way two reliable connections are established, one connection goes from the Client to the Server while the other goes from the Server to the Client.
TCP provides a variety number of features, some of them are:

- **Reliable data stream**: the Sender is notified if the packets are delivered correctly or not to the Receiver;
- **Ordered segments**: all the transmitted packages are enumerate. Moreover, the TCP receiver rearrange the segments order according to the sequence number;
- **Retransmission of lost packets**: some packets may be lost, duplicate or delivered out of order due to several reasons like network congestion or particular network behaviours. Any packet that is not acknowledge is retransmitted;
- **Error-free data transfer**: corrupt packets are considered lost and retransmitted;
- **Flow control**: this mechanism limits the rate at which a Sender transfers data. This is done to ensure reliable delivery and to do not overflow the Receiver;
- **Congestion control**: they are mechanisms to ensure that a Sender does not overflow the network;

B. TCP Congestion Control

One of the main aspects of TCP is the Congestion Control. This technique control the rate of data entering the network, keeping the data flow below a rate that would make the network collapse. Network congestion may occur for different reasons, for example when a sender overflows the network with too many packets or when the links have an insufficient bandwidth.
The goal of TCP Congestion Control is to prevent the congestion of the network or, when it has already happened, to mitigate it efficiently. TCP uses a Congestion Window (*cwnd*) that is maintained for each TCP session and represents the maximum amount of data that can be sent into the network without being acknowledged. The Congestion Window is maintained by the sender and calculated by estimating how much congestion there is on the link.
Depending on the variant of TCP, the congestion policy implemented may change. Generally, we have the following phases:

1) **Slow Start phase**: in this phase the Congestion Window size increments exponentially after every Round Trip Time (RTT[1]) up to a threshold value called *ssthresh*;
2) **Congestion Avoidance phase**: after the *ssthresh* value, the size of the Congestion Window additive increase;
3) **Congestion Detection phase**: if a lost event occurs, TCP assumes that is due to congestion and the Congestion Window size is decreased according to specific rules. The only way a sender can guess that congestion has occurred is the need to retransmit a segment as retransmission is needed to recover a missing packet.
   Retransmission can occur in one of the two cases:
   a) *RTO expires*: if the Retransmission Timeout (RTO) expires, the *ssthresh* is reduced to half of the current window size, the Congestion Window is decreased at 1 and it starts again with Slow Start phase;
   b) *3 Duplicate ACKs*: when the sender receives three duplicate ACKs, the *ssthresh* value is reduced to half of the current window size, the Congestion Window value is set to the actual *ssthresh* value and it starts again with Congestion Avoidance phase.

The TCP policy just described can be seen in the figure 3, where the implemented version corresponds to TCP Reno.

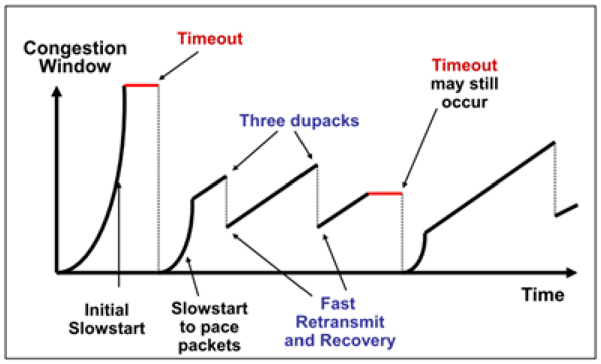


Fig. 3. TCP Reno Congestion Control

In the literature, we have many different versions of TCP where the main difference is in the approach used to compute the Congestion Window. One way of classifying different TCP versions is based on their application: high-speed variants are characterized by a more aggressive behaviour like TCP Cubic while wireless variants try to deal with problems typical for wireless scenarios like lost packets due to transmission errors, an example is TCP Hybla.
In this paper, we focus only on three versions of TCP: New Reno, Westwood, Hybla.

C. TCP Reno / New Reno

TCP Reno [2] is one of the first version of TCP, together with Tahoe. When a packet is lost, TCP Reno applies different

[1]it is the amount of time it takes for a signal to be sent plus the amount of time it takes for acknowledgement of that signal to be received.

policies according to the reason of the detected loss.
If a packet loss is detected through the **expired RTO**, it is consider as an indicator of massive congestion, so the *ssthresh* is set to half of the current windows size and the *cwnd* is set to 1. Then, Slow Start phase is applied.
On the contrary, if the sender receives **three duplicate ACKs** we take it as a sign that the segment was lost, and so Reno perform a "Fast Retransmit[2]" technique that reduce the time a sender waits before retransmitting a lost segment. After a packet loss, the *cwnd* and the *ssthresh* is reduced to half of the current windows size. Then, we continue with Congestion Avoidance phase. In this case, the duplicate ACKs are an indicator of weak congestion as the network is able to deliver other packets regardless the one lost. The TCP Reno policies are shown in figure 3.
TCP Reno performs poorly when we have multiple packet losses in one window (it may happens in wireless links). The reason is that, as it can only detect a single packet loss, if there are multiple packet losses then the first information about the lost segment comes when we receive the duplicate ACKs. But the information about the second packet which was lost will come only after the ACK for the retransmitted first segment. It is also possible that the *cwnd* is reduced many time as many packets lost are detected in one window, causing a reduce transmission speed of the TCP connection.

TCP New Reno tries to alleviate this problem introducing Partial ACKs messages. It is considered a more efficient version as it can detect multiple packet losses that occurs one close to another. It does not exit the Fast Recovery[3] phase and reset the *ssthresh* if Partial ACKs are received, it means until all the data that was sent inside the specific window are acknowledges. In this way it solves the problem of reducing multiple time the *cwnd*. TCP New Reno supports multiple retransmissions and it introduces "Partial ACK". When a sender receives a new ACK, there are two cases:

1) If it is a Partial ACK, New Reno deduces than the next segment was lost and it retransmit that segment. It does not exit from Fast Recovery phase;
2) The ACK can acknowledges all the previous segments and so it exits from Fast Recovery and the *cwnd* is set to the *ssthresh*. Then, Congestion Avoidance is applied.

D. TCP Westwood

TCP Westwood [3] is a different TCP variant that improves the performances of TCP Reno especially in wireless networks. It uses end-to-end principle that it does not require inspection of TCP segments at intermediate nodes. The key idea is to continuously estimate, at the TCP sender, the packet rate of the connection by averaging the rate of returning ACKs. The estimated connection rate is then used to compute Congestion Window and Slow Start Threshold used after a congestion occurs (after three duplicate acknowledgments or after an expired timeout).
Westwood comes from the idea that if a connection is currently achieving a given rate, then it can safely use the window corresponding to that rate without causing congestion in the network. It attempts to select the *ssthresh* and a *cwnd* which are consistent with the effective bandwidth used at the time of detected congestion. This approach makes Westwood more robust to sporadic losses, especially in wireless domain.
We note that the wireless domain is not what was TCP initially developed for. However, as wireless is very spread now a days, we need TCP protocols that can work efficiently also in such domain.

E. TCP Hybla

Traditional TCP versions like Tahoe, Reno, New Reno perform poorly in connections with high link errors and long propagation delay, like satellite radio link. They are very disadvantaged because of the long Round Trip Time (RTT). In case of satellite or wireless environment, as the TCP protocol was designed to recover only from congestion situations, losses of TCP segments due to possible errors on the network are erroneously associated to congestion problems, causing an inappropriate activation of the Congestion Avoidance mechanisms. Moreover, the long RTTs makes a reduction in the *cwnd* that result in a throughput degradation (together with unfair sharing of available resources).
TCP Hybla [4] tries to alleviate such problems. The idea is to remove the dependency of the RTT in computing the Congestion Window. The purpose of TCP Hybla is to obtain for long RTT connections the same transmission rate of a TCP connection with a lower RTT. It can be shown that this goal can be achieved by making the throughput independent from the RTT. Such independence is obtained by a special coefficient. This coefficient is used to calculate both the Slow Start Threshold and the Congestion Window. With Hybla, the performances of TCP, especially for links that have a long RTT like satellites, should improve drastically compared to original TCP versions.

III. Satellite communications

The purpose of this paper is to use a network simulator to make experiments on satellite communication 5 and analyze the performances of previously described TCP variants. Three topology of satellite communications has been identified:

1) **Low Earth Orbit (LEO)**: it is an orbit that is relatively close to Earth's surface, the altitude is less than 1000 km. It is the orbit most commonly used for satellite imaging, as being near the surface allows it to take images of higher resolution. To reach a global coverage, thousands of LEO satellites are needed;
2) **Geostationary Orbit (GEO)**: it is used by satellites that need to stay constantly above one particular place over Earth, such as telecommunication satellites. This way, an antenna on Earth can be fixed to always stay pointed

[2]when the sender receives three duplicate ACKs, it assumes that the packet is lost and it retransmit that packet without waiting for a retransmission timer to expire.

[3]sequence of operation that TCP Reno applies after receiving 3 duplicate ACKs.

towards that satellite without moving. It can also be used by GPS, military purposes and weather monitoring satellites because they can continually observe specific areas to see how the weather change. Satellites in GEO cover a large range of Earth so only few satellites (three equally-spaced) can provide almost a global coverage. The altitude of a GEO satellite is 35,786 km;

3) **Medium Earth orbit (MEO)**: it comprises a wide range of orbits anywhere between LEO and GEO. It is very commonly used by navigation satellites, like the European Galileo system 4.

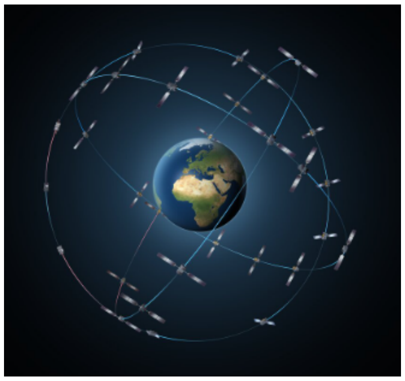
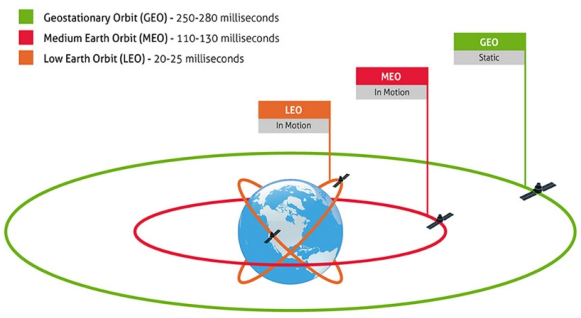


Fig. 4. Galileo constellation

Galileo is a global navigation satellite system created by the European Union. It uses a constellation of multiple satellites to provide coverage across large parts of the world all at once. The range of applications of Galileo is very big, examples are tracking jets or getting directions to your smartphone.

The figure 5 shows a graphical representation of the different orbits and the corresponding satellites.



🟩 Geostationary Orbit (GEO) - 250-280 milliseconds
🟥 Medium Earth Orbit (MEO) - 110-130 milliseconds
🟧 Low Earth Orbit (LEO) - 20-25 milliseconds
GEO
Static
MEO
In Motion
LEO
In Motion

Fig. 5. LEO - MEO - GEO

IV. Introduction to NS3 simulator

NS3 is a network simulator tool based on C++ that can be used to perform a variety number of experiments in a network domain. With NS3 you can create virtual nodes (like computers in real life) and install on them applications, Internet protocols and other services.

NS3 allows to create different type of connections, for example, a Point-To-Point network represents a physical link that connects only two nodes. The link has a number of attributes that can be set like data rate and delay. If we extend the P2P network between an arbitrary number of nodes we have a so called CSMA network (in essence, it is a bus network with many nodes). Moreover, we can create a wireless network, made by a router and a number of connected nodes decided by the administrator.

The NS3 simulator may be useful to test different protocols in specific networks or make simulation before actually develop and install a real network.

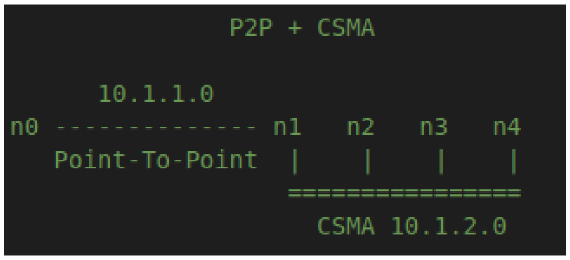
A. Building topology

The first step to perform when you are developing a project in NS3 is to understand the topology you need to implement. In this section we describe the main steps needed to build a general topology using NS3 simulator. A large variety of topology can be implemented, however, the general steps are similar even if the topology we want to create are different. The main steps are the following:

1) First, we create the ”nodes”, that are instances that can represent different objects like a PC, router, etc. It is done by instantiating the class *NodeContainer*;
2) Then we create the ”channel” between the nodes. It is the representation of the network you want to create. More precisely, this corresponds to peripheral cards (Network Interface Card) and network cables, in reality. In NS3, they are usually coupled together. As previously stated, examples of channels are Point-To-Point, CSMA, WiFi;
3) In previous steps we have defined the ”nodes” and the ”channel” but they are not connected yet. To connect it we use the class *NetDeviceContainer* that allows to attach the channel created at point 2 with the nodes created at point 1. Peripheral card are installed on each node and the required connection is established;
4) At this point we have node and device configured but we do not have any protocol stack installed on the nodes. So, we need to install Internet rules on the nodes using the class *InternetStackHelper*. It allows to install all the Internet protocols (IP/TCP/UDP functionalities) on each node;
5) Now, in order to let the nodes communicates we need to assign IP addresses. Both IPV6 and IPV4 addresses can be used. As an example, if we decide to assign IPV4 address we use the class *Ipv4AddressHelper*;
6) As last step, the routing table need to be populated using the class *Ipv4GlobalRoutingHelper*.

The above steps are needed to create an arbitrary network topology in NS3 that can be customize by the administrator.

In the following figure 6 there is an example of topology that combine a P2P and CSMA network. More precisely, between the nodes n0 and n1 there is a Point-to-Point network, while between the nodes n1, n2, n3, n4 there is a CSMA network that, in essence, corresponds to a bus link. The P2P network is identified by the address 10.1.1.0 while the CSMA by the address 10.1.2.0.



```
          P2P + CSMA

    10.1.1.0
n0 -------------- n1   n2   n3   n4
   Point-To-Point  |    |    |    |
                  ================
                    CSMA 10.1.2.0
```

Fig. 6. P2P + CSMA topology example

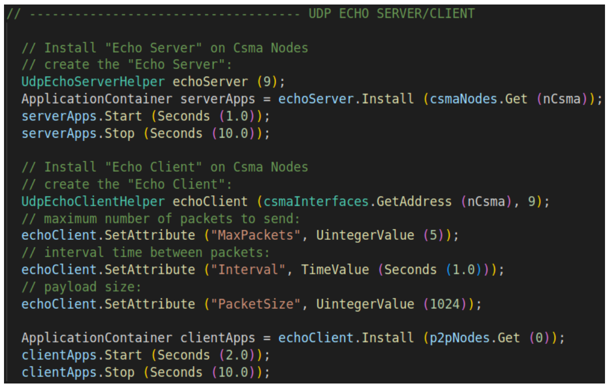
B. Applications

Ones the topology we want to simulate is ready, we need applications that generates traffic so that we can perform analysis on the network. NS3 allows to create different type of applications like Client-Server, UDP or TCP custom applications. Examples are:

- **Echo Server application 7**: an Echo server is an application that connects a Client and a Server. The Client sens a message to the Server and the Server receives the message and it echos back to the Client. This application can be based both on TCP or UDP transport protocols. If it is UDP based, there is no need to establish a connection and no guarantee of message delivery. The packets are sent without taking care of possibly lost packets. On the contrary, if it is TCP based, Client and Server first need to establish a connection 2 and then they can exchange messages. TCP also implements congestion control mechanisms for lost packets;
- **On-Off application**: it is application that generates traffic according to an On-Off pattern. It alternates two states, On and Off. The duration of such states can be decided by the administrator. The application generates traffic only in the On state. The generated traffic is characterized by specific “data rate” and “packet size”. Also this application can be based on UDP or TCP;
- **Bulk Application**: the idea of a Bulk application is different, it aims to send as much traffic as possible trying to fill the bandwidth. It is made by a traffic generator that sends data as fast as possible up to a certain value specified by the user or until the application is stopped. This type of application is usually based on TCP;
- **Custom TCP/UDP application**: it is an application that is not pre-defied in NS3. You need to manually create all the components and it can be useful if you want to hook some methods to analyze specific results. An example is attach a function to obtain the Congestion Window values.

C. Analysis

Ones we have installed one or more applications on the previously defined topology, we are ready to make experiments



```
// ---------------------------------- UDP ECHO SERVER/CLIENT

  // Install "Echo Server" on Csma Nodes
  // create the "Echo Server":
  UdpEchoServerHelper echoServer (9);
  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  // Install "Echo Client" on Csma Nodes
  // create the "Echo Client":
  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  // maximum number of packets to send:
  echoClient.SetAttribute ("MaxPackets", UintegerValue (5));
  // interval time between packets:
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  // payload size:
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));
```

Fig. 7. Echo Server Application NS3 code

and analyze the results. NS3 provides different features that can be used to retrieve information from the simulations. The main features used to analyze the results are:

- **NetAnim**: it stands for Network Animator. It is a software tool able to provide a graphical representation of the implemented topology where you can see the sending and received packets on the way. It is an animated version of how network looks in real and how data are transferred from one node to other. Moreover, you can get other information like the addresses of all the nodes, the time at which packets are sent/received, what is the next hop for the packet. It is useful to have a graphical representation of the implemented topology. An example in shown in figure 8;

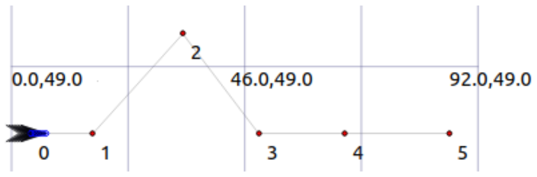


Fig. 8. NetAnim tool example

- **Flow Monitor**: it is a tool to monitor the performance of your simulation. By writing the corresponding code inside an NS3 script, it can generate an ”.xml” file with all the flows of packets together with the corresponding information like source address, destination address, packet size, etc. Then, this file must be properly process to be able to read the statistics;
- **ASCII trace**: NS3 provides a ”tracing system” mechanism that allows to trace every packet during the simulation time. By using the class *AsciiTraceHelper*, a new ”.tr” file is generated with all information about packet movement;
- **Pcap trace**: similarly to ASCII trace, NS3 enable the possibility to generate a ”.pcap” file with all information of the packets (e.g., Sequence number, Source IP, destination IP, etc) that can be processed by tools like Wireshark;

V. Test-bet

A. LEO, MEO implementation

Up to now we have seen how to build a generic topology, which kind of application we can install on it and some tools to analyze the results of the simulations.
In this section we implement two of the satellite topology described at the beginning of the paper and we perform some experiments in order to test the TCP variants described in section II.

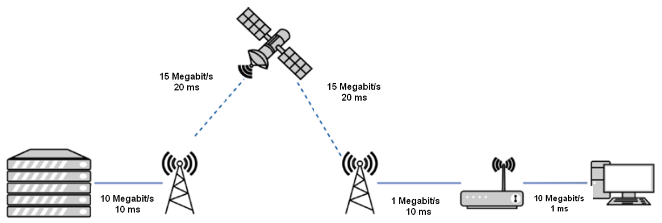


Fig. 9. LEO network topology

In order to simulate a satellite communication in NS3 we implement the topology shown in figure 9, where at each link corresponds a Point-To-Point connection with a specific data rate and delay as written in the figure 9. To differentiate between LEO and MEO, we change the delay on the links that connects the satellite to the antenna's. For LEO the delay is set to 20ms, for MEO is set to 90ms. Such delay aims to represent the different altitude of the satellites. Moreover, we introduce an error on the packet lost in the link that connects the antenna to the satellite. This is done to simulate lost packets as in reality we suppose that as we would have a wireless communication with potentially high RTT, we may lost some packets on the way.
Then, to generate traffic we implement a custom TCP application because we want to hook a function to study the Congestion Window for each TCP variant and the number of received packets.

B. Experiments

In order to see the behaviour of the different TCP variants, we perform some experiments on LEO and MEO satellite topology. For each of them we build simulations with TCP New Reno, Westwood and Hybla as transport protocol. The simulations are the following:

- **Exp1**: implement an application that sends data from the Server to the Client. All the links has no error on packet lost;
- **Exp2**: implement the same application but including an error on packet lost in the link connection between the antenna and the satellite. Insert an error of 0.1% to simulate some packet lost;

VI. Results

All the experiments are made with a packet size of 536 bytes, an application data rate of 9 Mbps and the same network topology.

A. LEO simulations

We consider here only the experiments regards the LEO satellite communication. The experiments for LEO are done with a simulation time of 180s. All the links should deliver the packets correctly as we do not insert any packet lost error rate. However, as the application data rate is 9 Mbps and the bandwidth of the fourth link is only 1 Mbps, we obtain that after a certain amount of time the channel will be full, so it inevitably lost some packets. Indeed, we are creating congestion in the network and so the TCP Congestion control mechanism is triggered on. This behaviour can be seen from the Congestion Window plot in figure 10. We can notice that for Hybla we obtained five events (packet loss), for New Reno just two and a half and with Westwood almost two events. Of course the simulation time is the same so this behaviour is due the the technical details of each TCP version. On the other hand, if we analyze the Received packets plots we can see that the values of TCP New Reno and Hybla are the same while with Westwood the received packets are lower. The exact numbers are provided in the following table.
Let's now analyze figure 11, where we introduce an error on 0.1% of lost packets in the link from the antenna to the satellite. Now, many peaks can be seen from the plots of the Congestion Windows as the TCP needs to retrasmit more lost packets. As previously mentioned in the paper, each TCP variants differs in the way of computing the Congestion Window. We can notice the different behaviours from the plots in figure 11. The most aggressive behaviour is with TCP Westwood, as the *cwnd* becomes closed to zero at every event (lost packet). The *cwnd* plot of New Reno is similar to Westwood, however, after an event the values are not dropped to zero but to a higher value. With Hybla instead, the general values are higher compare to the other two variants and, indeed, this is the variant that allows to transfer the highest number of packets in the same amount of time. The specific values are shown in the table below.
By this simulation, TCP Hybla is the most stable version in this scenario.

LEO Received Packets

	New Reno	Westwood	Hybla
0% pkt error	37.834	36.231	37.834
0.1% pkt error	31.177	21.562	37.054

B. MEO simulations

In this section we consider the topology of MEO satellites. We recall that to simulate a MEO network we increase the delay of the satellite links to 90ms, instead of 20ms for LEO. In the case of no error rate in packet lost, to obtain a behaviour

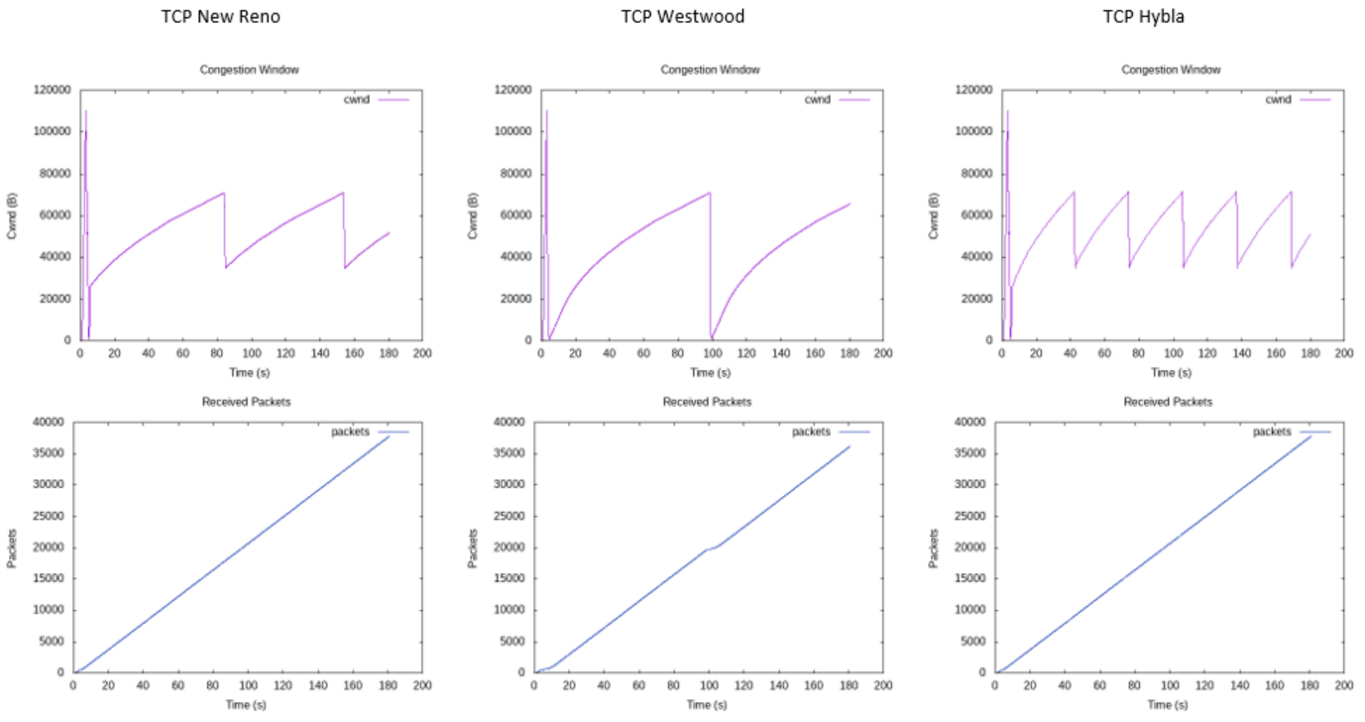


Fig. 10. LEO simulations - 0% pkt lost error

that is closed to the one in figure 10, we need to increment the simulation time up to 360s. This is reasonable as we have incremented the delay in the satellite links and the RTT increased. So, all the experiments in MEO are done with 360s of simulation time.
Also in this case, the total number of received packet of Hybla and New Reno are almost the same while Westwood has a lower value. The specific numbers are in the table below. The congestion window plots are very similar to the ones of LEO as the behaviour of each TCP variant is always the same, the difference is the received packets. As we increased the simulation time, we obtain that more packets are received with LEO but if we would fix the simulation time equal for both LEO and MEO we would obtain that less received packets with MEO as the RTT is higher.

MEO Received Packets

	New Reno	Westwood	Hybla
0% pkt error	75.663	59.955	75.527
0.1% pkt error	23.825	15.363	36.052

VII. Conclusion

In this paper three TCP variants has been analyzed and tested in satellite scenarios. The total number of received packets and the Congestion Window values has been plotted in all different simulations.
From all the simulations, TCP Hybla is the protocol that performs better in both LEO and MEO satellite scenarios. There is a high discrepancy compared to other two variants in terms of number of received packets. With Hybla much more packets can be sent in the same amount of time. This is due to the fact that Hybla aims to increase the performance even with high RTT.
On the contrary, the behaviour of the Congestion Window of TCP New Reno and Westwood is similar, even if, in the case of packet lost, the total number of received packet of New Reno is higher.
From the Congestion Windows plots, the behaviour of TCP Westwood is the most aggressive as at each packet lost, the value of the *cwnd* is dropped. It is also confirmed by the lowest values of received packets shown in the previous tables.

References

[1] Santosh Kumar, Sonam Rai. 2012. Survey on Transport Layer Protocols: TCP & UDP.
[2] A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas.
[3] Mario Gerla, M.Y.Sanadidi, Ren Wang, Andrea Zanella. 2002. TCP Westwood: congestion window control using bandwidth estimation.
[4] Carlo Caini, Rosario Firrincieli. 2004. TCP Hybla: a TCP enhancement for heterogeneous networks.

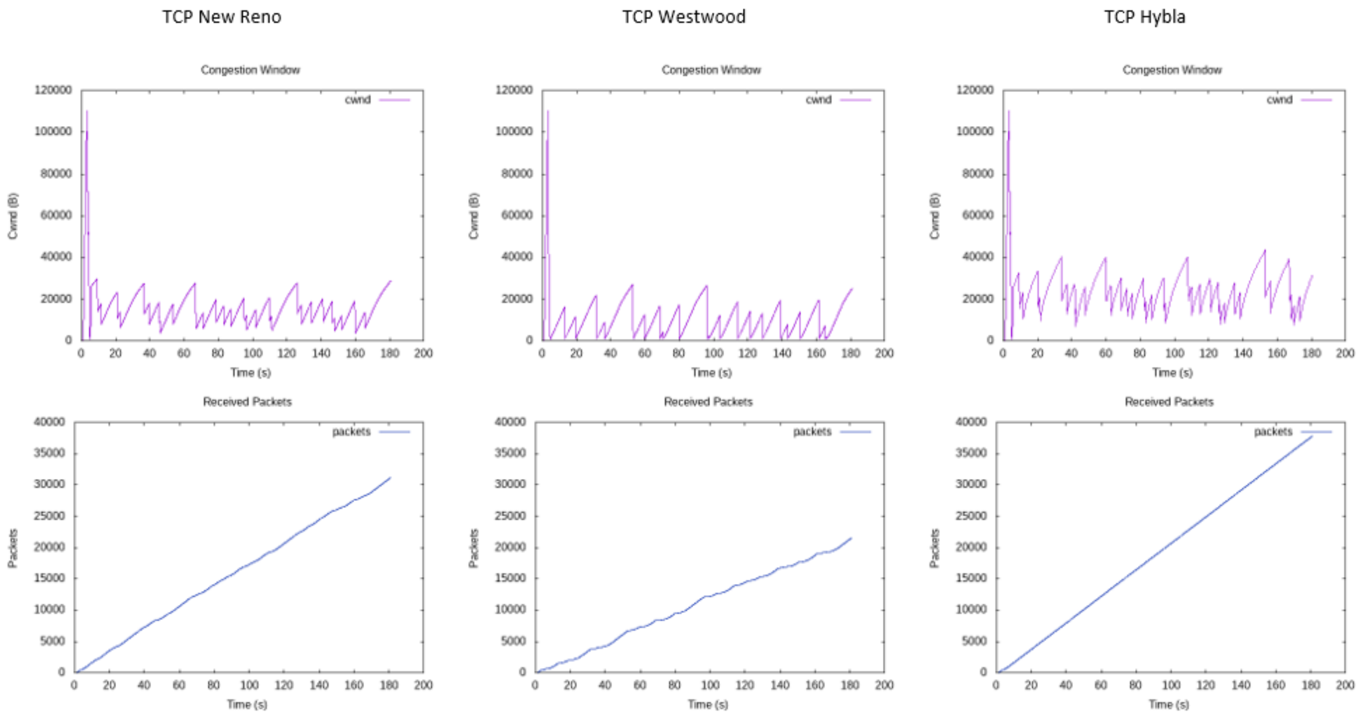


Fig. 11. LEO simulations - 0.1% pkt lost error

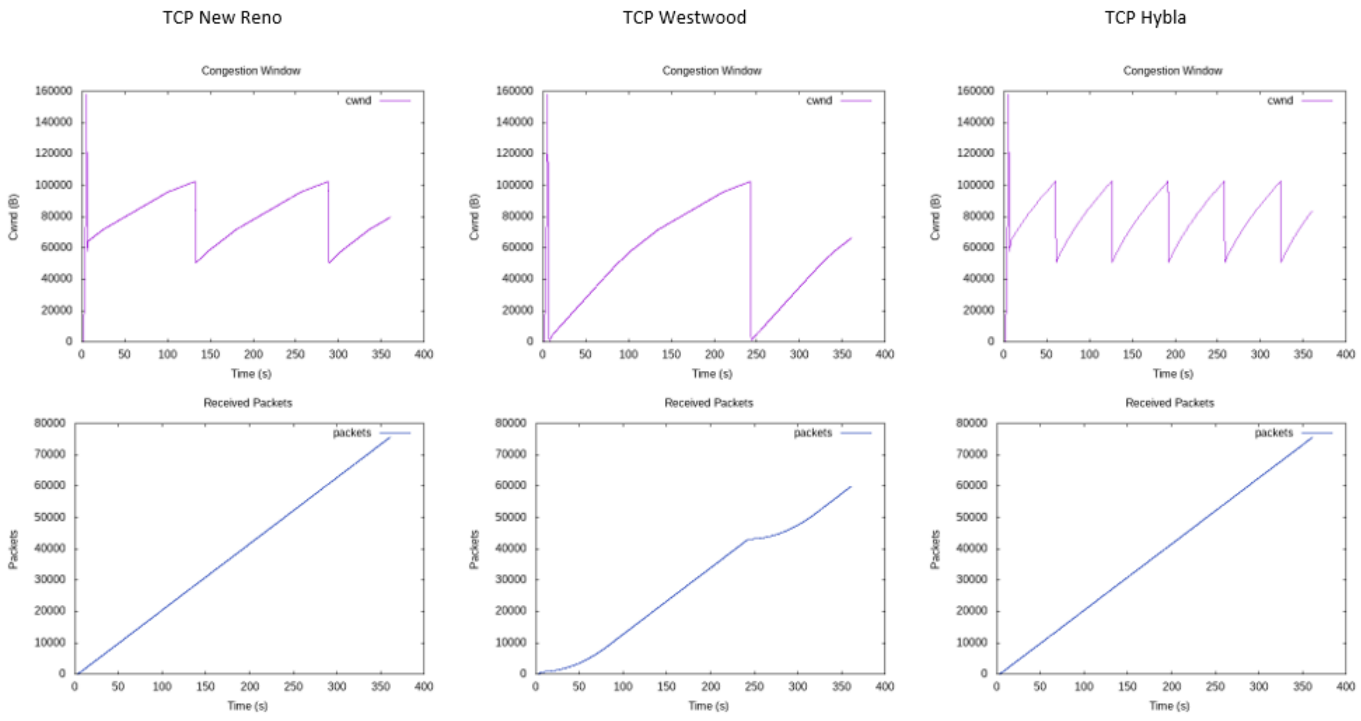


Fig. 12. MEO simulations - 0% pkt lost error

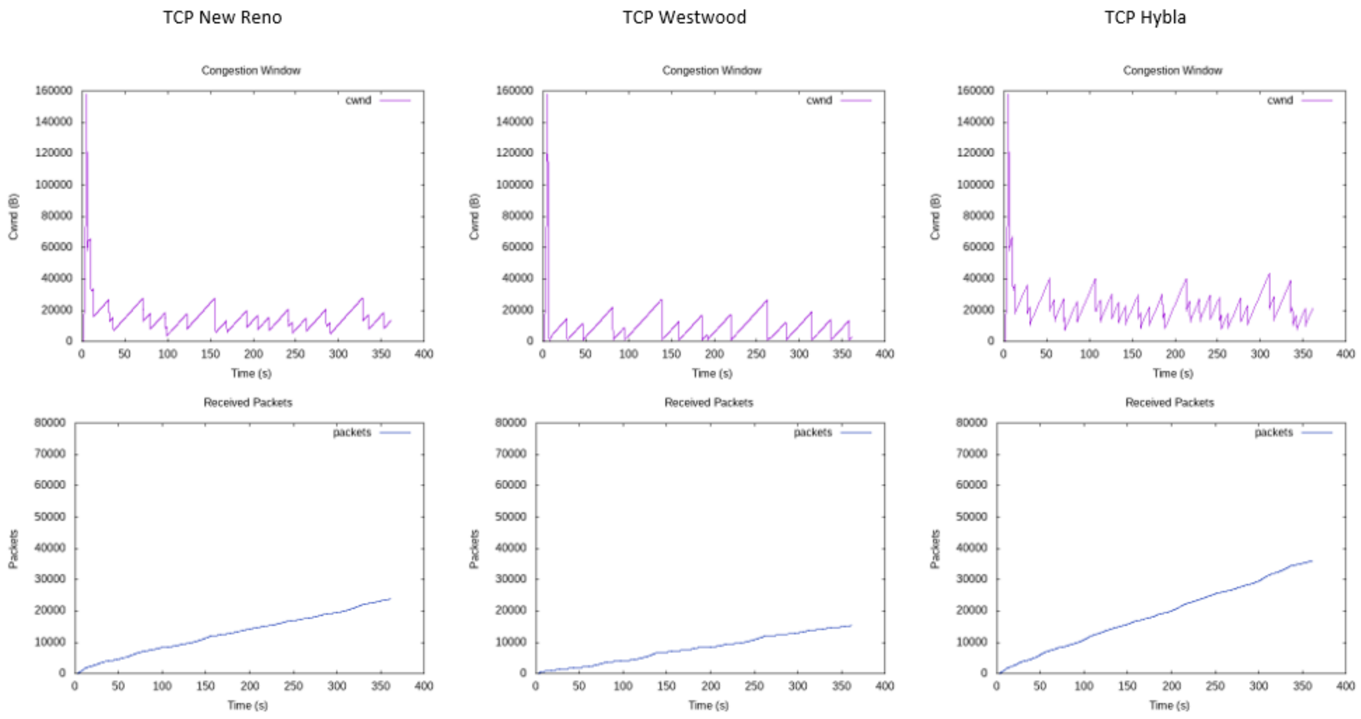


Fig. 13. MEO simulations - 0.1% pkt lost error