Horizon 2020
European Union Funding
for Research & Innovation

Grant Agreement Number: 101014517
Project Acronym: AB4Rail
Project title: Alternative Bearers for Rail

DELIVERABLE D [3.5]

[Analysis of options for transport and application protocols and of their secure versions]

Project acronym:	AB4Rail
Starting date:	01-01-2021
Duration (in months):	24
Call (part) identifier:	S2R-OC-IP2-02-2020
Grant agreement no:	Number 101014517 – IP/ITD/CCA – IP2
Grant Amendments:	N/A
Due date of deliverable:	30-06-2022
Actual submission date:	22-07-2022
Coordinator:	Franco Mazzenga (Radiolabs)
Lead Beneficiary:	Romeo Giuliano (USGM)
Version:	0.1
Type:	Report
Sensitivity or Dissemination level[1]:	PU
Contribution to S2R TDs or WAs[2]	TD2.1
Taxonomy/keywords:	Adaptable Communication System; ACS; IP emulator; IP impairment models; transport protocols; application protocols; secured versions; TCP; SCTP; QUIC

This project has received funding from the Shift2Rail Joint Undertaking (JU) (now Europe's Rail Joint Undertaking, EU-RAIL) under grant agreement No. 101014517. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the Shift2Rail JU members other than the Union.

[1] PU: Public; CO: Confidential, only for members of the consortium (including Commission Services)

[2] https://projects.shift2rail.org/s2r_matrixtd.aspx

Authors Table

Name	Affiliation	Contribution
Romeo Giuliano	Università degli Studi Guglielmo Marconi (USGM)	Main contributor
Franco Mazzenga	Radiolabs (RDL)	Main contributor
Francesco Vatalaro	University of Rome Tor Vergata	Main contributor
Alessandro Vizzarri	Radiolabs (RDL)	Support to contributors

The document history table provides a summary of all the changes in reverse chronological order (latest version first).

Document history

Date	Name	Affiliation	Position/Project Role	Action/ Short Description
22 Jul. 2022	Romeo Giuliano	Università degli Studi Guglielmo Marconi (USGM)	WP leader	Analysis of the transport and application protocols and their secured versions
2 Dec. 2022	Romeo Giuliano	Università degli Studi Guglielmo Marconi (USGM)	WP leader	The updated document includes the revisions required by PO

Table of Contents

Executive Summary

This deliverable is the output of the Task 3.5 of AB4Rail project, which is dedicated the analysis of options for transport and application protocols and of the Task 3.6, concerning the analysis of security versions of the transport and application layer protocols.

Protocol performances have been assessed by means of the software emulator developed in the Task 3.3, that can reproduce the behavior of the communication bearers as seen at IP protocol level and it allows to account for the variations with time of the typical packet impairments characterizing the IP layer link such as: bandwidth, latency and packet loss rate. In particular we account for the variations with time of the available transmission capacity along the track due to variability of the modulation and coding scheme.

The analysis has concerned the secure versions of the transport protocols indicated in Del. 3.4 [**1**] with the addition of the novel QUIC protocol. The following standard application protocols: Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP) and their secure versions have been also analyzed. Analysis has been carried out in terms of achievable throughput at application level, the message delay and the download time. These are the classical key performance indicators used in the scientific literature to assess the performance of transmission protocols.

The performances of different combinations of application and transport protocols have been analyzed taking into account for different traffic categories that can be that can be traced back to the ACS traffic classes. As an example, HTTP/HTTPS application protocols are typically used in conjunction with TCP using Cubic congestion control algorithm. In AB4Rail Task 3.5 activities we have extended the study of HTTP/HTTPS over TCP with BBR congestion protocol, over the SCTP and QUIC protocols. The usage of SCTP for transporting HTTP/HTTPS is new and no results are available in the literature. The ("obsolete") SSH File Transfer Protocol (SFTP), which adopts Secure Sockets Layer (SSL) as secure layer has been gradually replaced with the FTPS adopting TLS.
From our results usage of TCP with BBR and QUIC even in conjunction with HTTP application protocol provides the best performance in terms of throughput and latency showing a substantial insensitivity to moderate packet loss.

In this deliverable we discuss the results of the research activities indicated in Task 3.6 which respond to the objective d in workstream 2, [2]. They concern the security analysis of the transport and application protocols. As demonstrated in [3] the capability of one transmission protocol (at transport or application level) to protect against one or more risks is related to the presence of specific functionalities implemented in the protocol itself.

As indicated in [2] the main output of this activity is a Table indicating the most appropriate (secure) transport and transport/application protocol to be selected for the ACS application class.

List of abbreviations, acronyms, and definitions

Acronym	Definition
3G	3rd Generation
3GPP	Third Generation Partnership Project
4G	4th Generation
5G	5th Generation
AB	Alternative Bearer
ACS	Adaptable Communication System
AI	Artificial Intelligent
API	Application Programming Interface
BS	Base Station
BBR	Bottleneck Bandwidth and Round-trip propagation time
CDF	Cumulative Distribution Function
CTA	Communication Traffic Analysis
DS	downstream
EIA	Electronic Industries Alliance
eNB	evolved Node B
ERTMS	European Rail Traffic Management System
ETCS	European Train Control System
FTP	File Transfer Protocol
FTPS	File Transfer Protocol over Transport Layer Security
GEO	Geographical Earth Orbit
GRE	Generic Routing Encapsulation
GW	Gateway
HAPS	High Altitude Platform Station
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol over Transport Layer Security
ICT	Information and Communications Technology
IETF	Internet Engineering Task Force

KPI	Key Performance Indicators
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LAN	Local Area Network
LEO	Low Earth Orbit
MA	Movement Authority
MAC	Message Authentication Code
MCS	Modulation and Coding Scheme
MEO	Medium Earth Orbit
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NA	Network Application
NAT	Network Address Translation
NG	Network Gateway
NIC	Network Interface Card
NSA	Non-Stand Alone
OA	On Board Application
OFDM	Orthogonal Frequency-Division Multiplexing
OG	On-Board Gateway
OS	Operating System
PDCP	Packet Data Convergence Protocol
P-GW	Packet Gateway
PL	packet loss probability
PLMN	Public Land Mobile Network
PMTUD	Path MTU Discovery
PR	Position Report
PRB	Physical Resource Block
PSTN	Public Switched Telephone Network
QoS	Quality of Service
QUIC	Quick UDP Internet Connections
RAN	Radio Access Network

RB	Resource Block
RDP	Remote Desktop Protocol
RLC	Radio Link Control
RRC	Radio Resource Control
RTT	Round Trip Time
SCTP	Stream Control Transmission Protocol
SFTP	Secure Shell (SSH) File Transfer Protocol
SINR	Signal-to-Interference plus Noise Ratio
SIP	Session Initiation Protocol
SNR	Signal-to-Noise Ratio
SLA	Service Level Agreement
SSL	Secure Sockets Layer
TBF	token bucket First-In First-Out
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TIA	Telecommunications Industry Association
TLS	Transport Layer Security
VoIP	Voice over IP
UDP	User Datagram Protocol
UE	User Equipment
US	upstream
VM	Virtual Machine
Wi-Fi	Wireless Fidelity

List of Figures

List of Tables

1. Introduction

This document constitutes the Deliverable D3.5 “Analysis of options for transport and application protocols and of their secure versions” according to Shift2Rail Joint Undertaking (now Europe's Rail Joint Undertaking, EU-RAIL) programme of the project titled “Alternative Bearer for Rail” (Project Acronym: AB4Rail, Grant Agreement No 101014517 — IP/ITD/CCA — IP2). On 22nd July 2020, the European Commission awarded a grant to the AB4Rail consortium of the Shift2Rail / Horizon 2020 call (S2R-OC-IP2-02-2020). AB4Rail is a project connected to the development of a new Communication System planned within the Technical Demonstrator TD2.1 of the 2nd Innovation Programme (IP2) of Shift2Rail JU: Advanced Traffic Management & Control Systems.

The IP2 “Advanced Traffic Management & Control Systems” is one of the five asset-specific Innovation Programmes (IPs), covering all the different structural (technical) and functional (process) sub-systems related to control, command, and communication of railway systems.

1.1 **Purpose and scope of the document**

The aim of this document is to identify the appropriate transport and application protocol pair(s) on realistic railway scenarios also including security. To this purpose, we analyze the selected transport protocols in the Deliverable 3.4 (i.e., TCP in its versions Cubic and bottleneck bandwidth and round-trip propagation time (BBR) congestion control algorithms, SCTP) by adding them the TLS. QUIC is also included in the analysis. In this evaluation we considered a stream between a client mounted on-board of a train and a remote server. Then we evaluate the application protocols HTTP and FPT by downloading a webpage with different characteristics and a file with several sizes.

1.2 **Document organization**

The document is organized according to AB4Rail Grant Agreement Number 101014517 (RD-1) and AB4Rail Consortium Agreement (RD-2). The document structure is the following.

In Section 2, we introduce the organization of the activities for the transport and application protocol evaluation at high-level.

In Section 3, we describe the methodology used for the analysis of the transport protocols that use TLS for the protection of end-to-end transmissions through cryptographic techniques.

In Section4, it is detailed the generation of the source traffic in terms of streams parameters, web pages for download and message characteristics. Moreover, evaluation parameters have also defined to be used for the subsequent analysis.

In Section 5, we summarize the main features of the QUIC protocol and we review the main implementations of QUIC and their related software libraries.

In Section 6 we report and discuss the results obtained by the emulator developed in Task 3.3 and described in the Deliverable 3.3 [4] of the AB4Rail project. We considered the following evaluations:

- The analysis of the secure transport protocols, which include the TLS layer;
- The analysis of the application protocols HTTP, FTP, HTTPS and FTPS;
- The analysis of the periodic short message delivery

In Section 7, we analyze the security of the protocols working at transport and application layers, by considering threats and possible defense techniques. We focused on TCP, UDP and SCTP by adding them security functionalities provided by TLS. This way application protocols as HTTP and FTP are able to transmit data from the source to the destination in a secure way.

Finally, conclusions are drawn in Section 7.

1.3 **Reference Documents**

Table 1: Reference Documents.

Document Number	**Document Description**
RD-1	AB4Rail Grant Agreement Number 101014517 – IP/ITD/CCA – IP2
RD-2	AB4Rail Consortium Agreement

2. Description and organization of activities

This document provides the results concerning the analysis of options for transport and application protocols. The document also contains the results concerning the analysis of secure version of the transport and application layer protocols.

Results presented in this Deliverable respond to objective c and d indicated in the workstream 2 in [2]. Activities have been devoted to the identification of the appropriate transport and application protocol pair(s) ensuring the required communication and characteristics capabilities for specific classes of railway application envisaged for Adaptable Communication System (ACS) applications i.e., critical and business. From the results and conclusions presented in Deliverable 3.4 [1] we restricted our analysis to the main rail applications requiring the following standard application protocols for proper operations: Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP). The message-based classes of applications that should be considered in the analysis of application and transport protocols include Session Initiation Protocol (SIP)/SDP messages for ACS signaling, European Rail Traffic Management System/European Train Control System (ERTMS/ETCS), web-based railway applications and file transfer/download applications.

The transport protocols identified and selected in the previous Task 3.4 will be considered in these activities. For each of the considered transport/application protocol pair for which a (stable) software implementation is available (on Linux) the performance evaluation will be carried out using the IP emulator/simulator developed in Task 3.3 which is described in Deliverable 3.3 [4].

Taking into account for the results obtained in the Deliverable D3.4, evaluation activities in this task have organized the activities detailed in [2] as follows. We have identified the main classes of traffic to be used for protocol assessment:

1. Transfer of data streams of varying length for the testing of the secure versions of the considered transport protocols i.e., Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP) and Quick UDP Internet Connections (QUIC)), which include the Transport Layer Security (TLS) protocol layer. It should be remarked that the analysis of transport protocols, originally confined to Task 3.4 activities, continues in this Task 3.5 because QUIC protocol which is a secure protocol by default. In this case performance comparison will be carried out

2. File transfer for testing FTP and File Transfer Protocol Secure (FTPS) application protocols to be used for file transfer-based services.

3. Download of web-pages of different sizes and complexity for testing the HTTP/HTTPS (Hypertext Transfer Protocol Secure) application protocols;

4. Periodic transmission of messages with variable length using transport protocols TCP (CUBIC+BBR), SCTP and QUIC This traffic class is important for considering the transmission delay of messages transmitted by protocols such as SIP/SDP and ERTMS/ETCS that are used for ACS and railway control/management. These messages are transmitted through the on-board ACS-GW.

To assess performances of the considered transport and application protocols, we have re-used the IP emulator developed in [Del.3.3 of AB4Rail] that was also used to obtain the results in Del. 3.4 [1].

In analyzing the performance of HTTP/HTTPS we have considered different combination of transport and application pairs. In particular, HTTP/HTTPS are typically used in conjunction with TCP implementing Cubic congestion control. In AB4Rail Task 3.5 activities we have extended the study to the HTTP/HTTPS over TCP with BBR congestion protocol, over the SCTP and QUIC protocols. The usage of SCTP for transporting HTTP/HTTPS is new and no results are available in the literature.

On the basis of the achieved results, we have determined the application/protocol pair allowing to achieve the end-to-end performance for the selected application traffic classes. Application/transport protocol analysis has been carried out in terms of: technology features such as maturity flexibility (this aspect is important for QUIC protocol which is currently under standardization), latency and the achievable throughput under different operating conditions. For what concerns traffic prioritization, the considered transport and application protocols are neutral with respect to the data flows they transport i.e., they do not implement any mechanism for traffic prioritization. These mechanisms (when and if necessary) could be implemented at application level so to manage the flows generated by the application that are transmitted using the (available) underlying transport protocol.

As evidenced in the Deliverable 3.4 [1] and even in this deliverable, the possibility of adopting well mature and widely accepted Internet Engineering Task Force (IETF) network protocols based on TCP/IP suite, as in ACS communications, allows to facilitate/simplify engineering, operational and implementation aspects. In particular, implementation complexity is drastically reduced since effective and stable implementations of these protocols are available on the market as well as on the open-source community including the Linux OS. In addition, several and widely used (open-source) tools for debugging, monitoring packet flows and collecting data for successive analysis are available.

Some of them have been used in our research activities to extract performance data from the emulator runs and to monitor end-to-end packet flows so to detect possible protocol anomalies or malfunctions. The Tshark (https://www.wireshark.org/docs/man-pages/tshark.html) (i.e., the command line version of wireshark) tool has been used to collect statistics. The aspects of monitoring packet flows are important for ACS specially to assess control plane proper operations. The main output of this first activity involving the analysis of secure transport protocols and some combinations of application/transport protocols for task is a Table indicating the most appropriate transport protocol/application pair(s) to be selected for each ACS application class that can be casted in one of the four traffic categories indicated in the previous points 1-4, that we have considered to assess performance. From the results presented in Deliverable 3.4 results will be obtained considering only the challenging mainline railway scenario. In fact, it should be remarked that from [1] we didn't observe a marked dependence of transport protocol performance on the rail scenario.

In this deliverable we also report the results of the research activities indicated in Task 3.6 and concerning the security analysis of the transport and application protocols. As indicated in [2] the analysis focuses on the following list of general technical risks indicated in EN 50159 regarding the safety-related messages for specific railway applications such as ERTMS/ETCS:

- Repetition of message
- Deletion of message
- Insertion of message
- Re-sequencing of two or more messages

- Corruption of message
- Delay of message
- Masquerade (a type of attack where the attacker pretends to be an authorized user of a system so to gain access to it or to gain greater privileges than they are authorized for).

The Task 3.6 activities respond to the objective d in workstream 2, [2]. As demonstrated in [EN 50159] the possibility for the transmission protocol (at transport or application level) to protect against one or more of the risks indicated in the previous points is strictly related to some specific functionalities implemented in the protocol itself.

Starting from this observation, to respond properly to the objective d in workstream 2 we have extended the analysis procedure reported in [EN 50159]. The extended procedure allows to achieve the security assessment of both transport and the considered application/transport protocols. In particular, we have analyzed the security aspects of:

a. TCP, User Datagram Protocol (UDP), SCTP and of their secure versions
b. QUIC transport protocols and the
c. application protocols FTP and FTPS, HTTP and HTTPS.

Analysis has been carried out with respect to the risks indicated in the previous list. The (obsolete) SSH File Transfer Protocol (SFTP), which adopts Secure Sockets Layer (SSL) as secure layer has been gradually replaced with the FTPS adopting TLS. The RaSTA in the security context is investigated in [1] and the interested reader is referred to this paper.

As in the previous case and as indicated in [2] the main output of this activity is a Table indicating the most appropriate (secure) transport and transport/protocol to be selected for the ACS application class and for each one of the considered network scenarios.

Note: The Remote Desktop Protocol (RDP) indicated in the original project proposal is a secure network communications protocol developed by Microsoft for the transfer of PC desktop contents specifically and only between client and server running Windows OS. The RDP allows network administrators to remotely diagnose problems that individual users encounter and gives users remote access to their physical work desktop computers. We believe that remote reproduction of a Windows OS desktop on a remote machine has non practical application in critical and business railway applications services that are the two categories of services and applications analyzed in AB4Rail for ACS. The RDP protocol could be of some interest when considering connectivity services oriented to train passengers that (maybe) could have some interest in remotely connecting with their workstation. In fact, RDP can be used by employees working from home or traveling who need access to their work computers. User oriented connectivity services have never been of interest in AB4Rail since they are not considered in ACS development being ACS (and in particular the on-board ACS-GW) specifically designed to offer connectivity for critical and business railway services.

3. Analysis of secure transport protocols: methodology

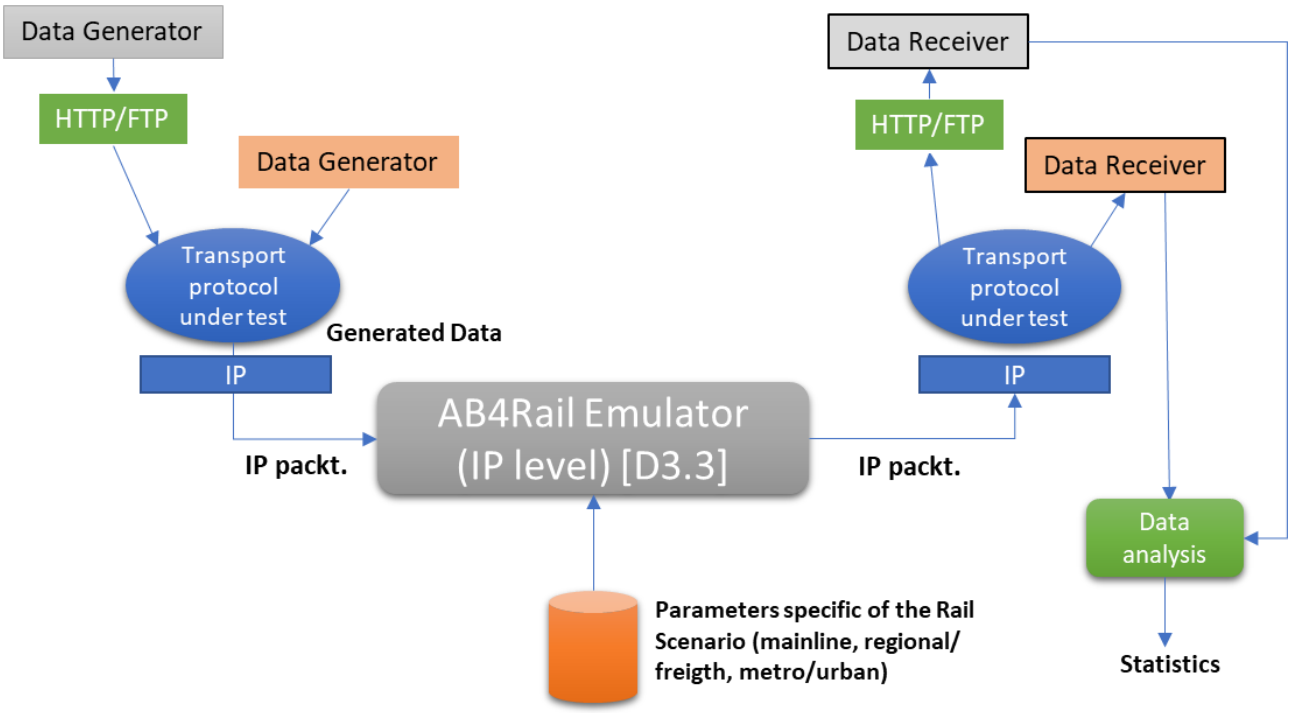
This paragraph describes the methodology used for the analysis of the transport protocols that use TLS technology (v. 1.3) for the protection of end-to-end transmissions through cryptographic techniques.

The considered transport protocols including the TLS layer are listed in the following points: [Del.of.3.4]:

1. TCP (Cubic and Bottleneck Bandwidth and Round-trip propagation time (BBR))
2. QUIC (IETF version)
3. SCTP

The principle scheme of the emulator arrangement used for the performance assessment of the transport and application/transport protocols is depicted in Figure 1.

Figure 1. High level emulator scheme used for the assessment of Transport and Application/Transport protocols



The scheme in Figure 1 is inherited from that presented in D3.4. As shown in Figure 1 the only modifications we have introduced are on the software developed for generating traffic, collecting

statistics and to perform on-board ACS GW from/to trackside ACS-GW packet communications. We consider the testing of secure versions of the transport protocols in D3.4 and the FTP/HTTP application protocols including their secure versions i.e., HTTPS/FTPS.

Even in this case communications between the on-board and the trackside ACS-GWs are considered. It is assumed the on-board ACS Gateway (ACS-GW) uses Generic Routing Encapsulation (GRE) tunnels to establish a connection with the remote ACS-GW located on the trackside side. The ACS connection emulator that interconnects the client on board the train with the server is the same used in D.34 and which was developed in Task 3.3 and whose characteristics are detailed in D3.3 [4]. This emulator implements:

a. Connection between client and server via GRE tunnel

b. Variability of the time of the QoS parameters that characterize the performance of the IP link i.e.: delay (i.e., latency), packet loss, link capacity, jitter.

As indicated in D3.4, the temporal variability of these parameters, including the link capacity, is achieved by taking into account the speed profile of the train moving along the considered section. While the train moves into the cell, the modulation coding scheme can change in accordance with the distance of the train from the receiving base station i.e., the eNB in the LTE case. In addition, as illustrated in [1] the transmission capacity available to the train is also function of the number of trains in the same cell at the same time.
The performance results presented in this Deliverable have been obtained considering the Italian railway section from Rome to Florence at high speed (mainline scenario) and a realistic train-speed vs time profile.

For the analysis of the transport and application protocols (objective of Tasks 3.5 and 3.6 of the AB4Rail project) we have implemented programs acting as traffic sources and sinks indicated in Figure 1. In particular, source software generates traffic in accordance with the required packet statistics; sink software can receive and analyze traffic to extract parameters required to achieve statistics of the performance parameters.

As shown in Figure 1 data sources can inject traffic over the application protocol layer (HTTP(S) or FTP(S)) or can directly interface with the (secure versions) of the underlying transport protocols. The Python programming language has been used to program the traffic sources and sinks shown in Figure 1. The following standard libraries, which are well tested and freely available on the Internet, have been considered in software development.

Table 2: Standard Python libraries considered in software development.

Library	Description
Python Socketserver	Framework for the fast realization of network servers based on un-secure TCP and UDP protocols using standard sockets
Python Aioquic	Pythonn implementation for the QUIC IETF protocol
Python HTTP client	It defines classes which implement the client side of the HTTP and HTTPS protocols
Python Urllib3 library	Library for the HTTP client implementation based on TCP for Python
Python HTTP server	It defines classes for implementing HTTP servers using un-secure TCP transport protocols only
Python TLS/SSL wrapper for socket objects	It provides access to TLS (often known as "Secure Sockets Layer") encryption and peer authentication facilities for network sockets, both client-side and server-side. This module uses the OpenSSL library
Python socket library	It provides access to the BSD socket interface implemented on many OSs
Ftpdlib python library	It implements FTP and FTPS server classes
Ftplib python library	It implements the FTP and FTPS client classes

For reasons that will be cleared in the next paragraph, we have also considered the LS-QUIC library implementing a well-tested and consolidated production version of the QUIC protocol.

Several Python libraries indicated in Table 2 implement classes for creating servers (sinks) and clients (source) that use IP connections based on the TCP/UDP or SCTP transport protocols without adding any functionality to ensure secure transmissions.

Secure transport protocols are obtained by adding the TLS layer to the existing protocol layer. A good part of the programming work carried out in Task 3.5 has concerned the extension of the libraries Python socket and Python HHTP in Table 2 to realize client and server software including secure functionalities and in particular to implement the protocol stacks for the following secure transport protocols:

Table 3: Considered Secure Transport/application protocols

Combination	Description
	Secure transport protocols
TLS + TCP	TCP layer can use Cubic or BBR as congestion control technique
TLS + SCTP	This is a new configuration not discussed in the current literature
QUIC	Usage of Python Aioquic library or LS-QUIC (see after) The QUIC Protocol is an important (maybe unique) example of secure transport protocols based on UDP providing and extending the features of the TCP protocol

	Secure application protocols
HTTP and HTTPS	They are transmitted over TCP (Cubic and BBR) and SCTP The motivations leading to exclude QUIC protocols are summarized in the next Section
FTP and FTPS	They are transmitted over TCP (Cubic and BBR)

In order to program the transmission software (clients and servers in Table 3) implementing the secure protocol stacks to be used in the emulator in Figure 1 we have extended the classes available in the Python libraries in Table 2.

Due to lack of detailed documentation concerning the organization of the classes defined in the several libraries in Table 2, a significant developing effort has been devoted to:

1. Read and analyse in detail the source code of the base client and server classes implementing TCP transport provided by the libraries in Table 2;
2. Identify the base server and client classes to be extended (i.e., sub-classed) to introduce the necessary modifications so to:
    a. Add the SCTP socket to the base server and client python classes
    b. To properly set the congestion control algorithm in the TCP stack i.e., the TCP BBR and CUBIC sockets;
    c. Add the TLS layer to the TCP (with CUBIC and BBR congestion control algorithms) and SCTP sockets using the Python SSL library (see Table 2);
    d. The QUIC already implements the TLS layer by default; two implementations of QUIC protocol have been considered for our evaluation (see next);
    e. Add the TLS layer to the existing HTTP server so to obtain HTTPS protocol stack
    f. Add the SCTP socket to the existing HTTP server classes and to the HTTPS server class indicated in previous point c.
3. The available FTP and FTPS classes have required no modifications for the creation of the client (source) and server (sink) entities in Table 3.

As an example, in Figure 2 we indicate the python code implementing the server class for the SCTP serves using TLS.

Figure 2. Part of python code implementing the secure server for SCTP transport protocol with TLS

```
#!/usr/bin/python3
import socketserver
import socket
import ssl
import sctp

class TCPServer(socketserver.TCPServer):
    pass

class BaseServer(socketserver.BaseServer):
    pass


# Sub-classing of TCP server in SCTPserver class - the constructor is updated
class SCTPServer(TCPServer):

    def __init__(self, server_address, RequestHandlerClass, bind_and_activate=True):

        BaseServer.__init__(self, server_address, RequestHandlerClass)

        self.socket = sctp.sctpsocket_tcp(socket.AF_INET) # 1. substitute the TCP socket with the SCTP socket
        if bind_and_activate:
            try:
                self.server_bind()
                self.server_activate()
            except:
                self.server_close()
                raise

class SSLSCTPServer(SCTPServer):
    def __init__(self, server_address, RequestHandlerClass, bind_and_activate=True):

        SCTPServer.__init__(self, server_address, RequestHandlerClass, False)

        context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
        context.load_cert_chain('server-crt.pem', 'server-key.pem')

        self.socket = context.wrap_socket(self.socket, server_side=True) # 2. add TLS layer to SCTP socket

        if bind_and_activate:
            self.server_bind()
            self.server_activate()
```

As shown in Figure 2, first we generate a new **SCTPserver** class by sub-classing the TCP server class available in the TCP python library in Table 2 (see point 1 in Figure 2, highlighted in red). The new SCTP secure (i.e., TLS based) server class **SSLSCTPServer** is then obtained by sub-classing the new **SCTPserver** class by wrapping the SCTP socket with the TLS layer (see point n.2 in Figure 2, highlighted in red) the original TCP server class defined in python library indicated in Table 2.

4. Traffic generation and parameters

4.1 Source traffic generation and assumptions

The on-board ACS-GW and the remote ACS-GW in Figure 1 exchange packets. Depending on the protocol (transport or application/transport) analysis we have considered the possibility of generating different types of traffic to/from the server and the client.

For the testing of transport protocols, we have assumed the on-board ACS-GW generates traffic directed to the trackside ACS-GW and vice-versa. We have assumed the upstream and downstream links show the same behavior in terms of time variability of the IP link parameters. This is a typical and well accepted assumption in performance analysis of full duplex communication links where it is assumed that the two separate links in the network behave in the same manner. In this case, the protocol behavior observed in one direction is also representative of that observed in the reverse direction. For further considerations in D3.4 we have also reported performance results by varying the amount of the available maximum transmission capacity (C_{max}) on the IP link to account for the asymmetric difference between uplink and downlink transmission capacity. However, as shown from results in D3.4 the final conclusions on the overall behavior and performance of transport protocols after the increase of transmission capacity are unchanged, except for the natural modification of the absolute values of the performance indicators that obviously improve.

For application/transport protocol testing including HTTP(S) and FTP(S) protocols we have considered the ACS-GW on the trackside as generating the greatest amount of traffic after the request from the client for downloading one web page (for HTTP(S)) or one file (for FTP(S)). In this case the maximum available traffic capacity on the IP link is always set to that of downlink. In this case the performance statistics are evaluated at the on-board side as shown in Figure 1 for HTTP(S) and FTP(S).

4.2 Traffic generation for transport protocol testing.

Similarly, to the approach in D3.4, for the testing of secure versions of the transport protocols including TCP (BBR+CUBIC), SCTP with TLS and QUIC we have considered the generation of data streams with different lengths of:

- 500 kbytes (short stream),
- 1 Mbytes and
- 2 Mbytes (long stream).

This allows to test the robustness of transport protocols with respect to time variable channel transmission capacity at IP layer as in the considered railway scenario. In fact, when considering the transmission assumptions in D3.4 (i.e., LTE technology with 1.28 MHz bandwidth operating in the

already existing GSM-R bandwidth) the maximum achievable (bit rate) transmission capacity under the most favourable transmission conditions (i.e., modulation-coding-scheme (MCS) with 64 QAM with 4/5 coding) is about 5.2 Mbps. When the train moves in the cell the available transmission capacity can vary during the transmission of the stream due to variability of the achievable MCS to account for the distance of the train from the eNB. Accounting for transmission capacity variability with time allows to evidence the ability of the transport protocol to rapidly adapt to time varying channel conditions even in the presence of packet loss (that can be set in the emulator). The procedure we have considered to evaluate the protocol performance such as latency and throughput is that used by OOKLA (https://www.ookla.com/, https://www.speedtest.net/it).

For further testing of transport protocols, we have considered variable bit rate sources sending short time-separated messages of different lengths. The lengths of the considered messages are:

- 1 kbyte
- 10 kbyte

In the 1 kbyte case the length of the packet is smaller than the typical MTU of the underlying transport protocol layer (in the TCP case the typical MTU is about 1460 bytes). This case can account for SIP/SDP and ERTMS/ETCS messages exchange where the entire message can be contained in one MTU. In the second case the length of the message is larger than MTU and the loss of one IP packet could lead transport packet to require retransmission.
Packets from the on-board ACS-GW to the trackside ACG-GW are generated on a periodic basis.

4.3 Traffic generation for application/transport protocols analysis

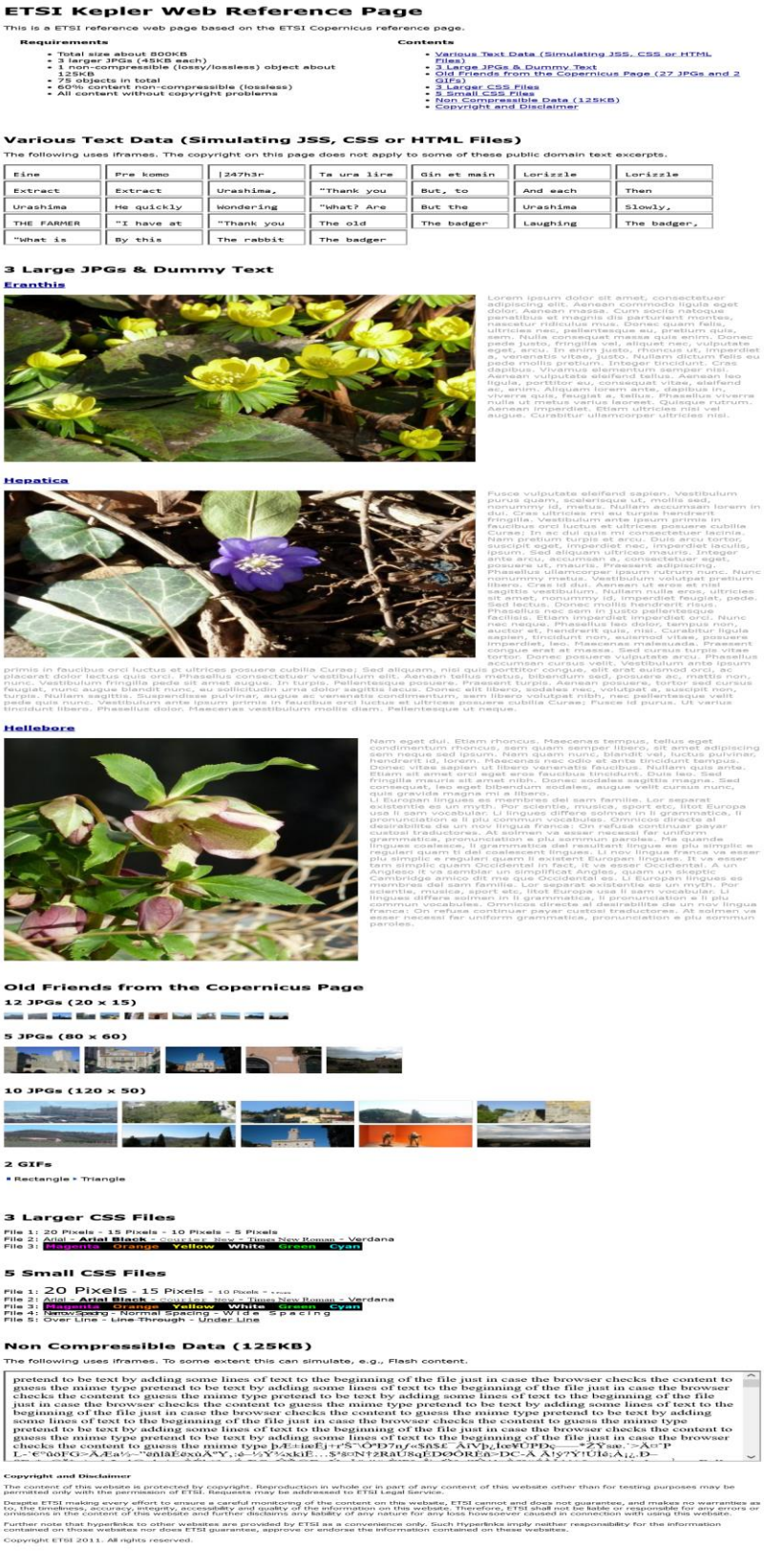
For the analysis of HTTP(S) protocol we have considered the generation of traffic from the server to the client consisting in the retrieval of one web page from the server (i.e., this is the typical usage of HTTP(S) protocol):

1 Download of web-pages from a web-server for the testing of HTTP(S) application/transport protocols based on TCP (CUBIC or BBR) and SCTP transport protocols; the considered web pages have different dimensions. The first one is the ETSI reference page (https://www.akostest.net/kepler/) of about 1 Mbyte that contains the main index.htm file of about 17 kbytes and several small files of some kbytes (many files are below 1 kbyte) of images and text.

2 The second web page is the main page of an Italian daily newspaper that has been downloaded and saved from the corresponding website. The overall size of the pages is about 5 Mbytes. The main .htm file is of 580 kbytes and the other several short files whose sizes varies from few kbytes to hundreds of kbytes. The download of web page consists in downloading the main htm file and all the associated files to the web page.

The snapshots of the two web pages i.e., the ETSI page and the Newspaper page are depicted in Figure 3 and Figure 4.

Figure 3. Snapshot of the Kepler ETSI reference HTML page to be used for testing HTTP(S) protocols.



ETSI Kepler Web Reference Page

This is a ETSI reference web page based on the ETSI Copernicus reference page.

Requirements

- Total size about 800KB
- 3 larger JPGs (45KB each)
- 1 non-compressible (lossy/lossless) object about 125KB
- 75 objects in total
- 60% content non-compressible (lossless)
- All content without copyright problems

Contents

- Various Text Data (Simulating JSS, CSS or HTML Files)
- 3 Large JPGs & Dummy Text
- Old Friends from the Copernicus Page (27 JPGs and 2 GIFs)
- 3 Larger CSS Files
- 5 Small CSS Files
- Non Compressible Data (125KB)
- Copyright and Disclaimer

Various Text Data (Simulating JSS, CSS or HTML Files)

The following uses iframes. The copyright on this page does not apply to some of these public domain text excerpts.

Eine	Pre komo	|247h3r	Ta ura lire	Gin et main	Lorizzle	Lorizzle
Extract	Extract	Urashima,	"Thank you	But, to	And each	Then
Urashima	He quickly	Wondering	"What? Are	But the	Urashima	Slowly,
THE FARMER	"I have at	"Thank you	The old	The badger	Laughing	The badger,
"What is	By this	The rabbit	The badger			

3 Large JPGs & Dummy Text

Eranthis

Hepatica

Hellebore

Old Friends from the Copernicus Page

12 JPGs (20 x 15)

5 JPGs (80 x 60)

10 JPGs (120 x 50)

2 GIFs

Rectangle Triangle

3 Larger CSS Files

File 1: 20 Pixels - 15 Pixels - 10 Pixels - 5 Pixels
File 2: Arial - Arial Black - Courier New - Times New Roman - Verdana
File 3: Magenta Orange Yellow White Green Cyan

5 Small CSS Files

File 1: 20 Pixels - 15 Pixels - 10 Pixels - 5 Pixels
File 2: Arial - Arial Black - Courier New - Times New Roman - Verdana
File 3: Magenta Orange Yellow White Green Cyan
File 4: Narrow Spacing - Normal Spacing - Wide Spacing
File 5: Over Line - Line-Through - Under Line

Non Compressible Data (125KB)

The following uses iframes. To some extent this can simulate, e.g., Flash content.

pretend to be text by adding some lines of text to the beginning of the file just in case the browser checks the content to guess the mime type pretend to be text by adding some lines of text to the beginning of the file just in case the browser checks the content to guess the mime type pretend to be text by adding some lines of text to the beginning of the file just in case the browser checks the content to guess the mime type pretend to be text by adding some lines of text to the beginning of the file just in case the browser checks the content to guess the mime type pretend to be text by adding some lines of text to the beginning of the file just in case the browser checks the content to guess the mime type pretend to be text by adding some lines of text to the beginning of the file just in case the browser checks the content to guess the mime type þÆtïæÉjrrŠ"Ó°Ð7n/«šâ$£ ÁíVþ¿Ïœ¥ÜPÐç——*ŽŸ»w.`>Ãœ'Þ L-€°ùöFG>Ã£a½—"ëñlaÉ¢¤ÃºY.,é–½Ÿ¾xkiÉ…$*8¤Ñ†2RäÜ8qÉDÒÖRÉã~DC-Â Ã¡ÿ7¥ÜÏë:Ã¡¿.Ð—

Copyright and Disclaimer

The content of this website is protected by copyright. Reproduction in whole or in part of any content of this website other than for testing purposes may be permitted only with the permission of ETSI. Requests may be addressed to ETSI Legal Service.

Despite ETSI making every effort to ensure a careful monitoring of the content on this website, ETSI cannot and does not guarantee, and makes no warranties as to, the timeliness, accuracy, integrity, accessibility and quality of the information on this website. Therefore, ETSI shall not be liable or responsible for any errors or omissions in the content of this website and further disclaims any liability of any nature for any loss howsoever caused in connection with using this website.

Further note that hyperlinks to other websites are provided by ETSI as a convenience only. Such Hyperlinks imply neither responsibility for the information contained on those websites nor does ETSI guarantee, approve or endorse the information contained on these websites.

Copyright ETSI 2011. All rights reserved.

Figure 4. Snapshot of the newspaper reference HTML page to be used for testing HTTP(S) protocols.

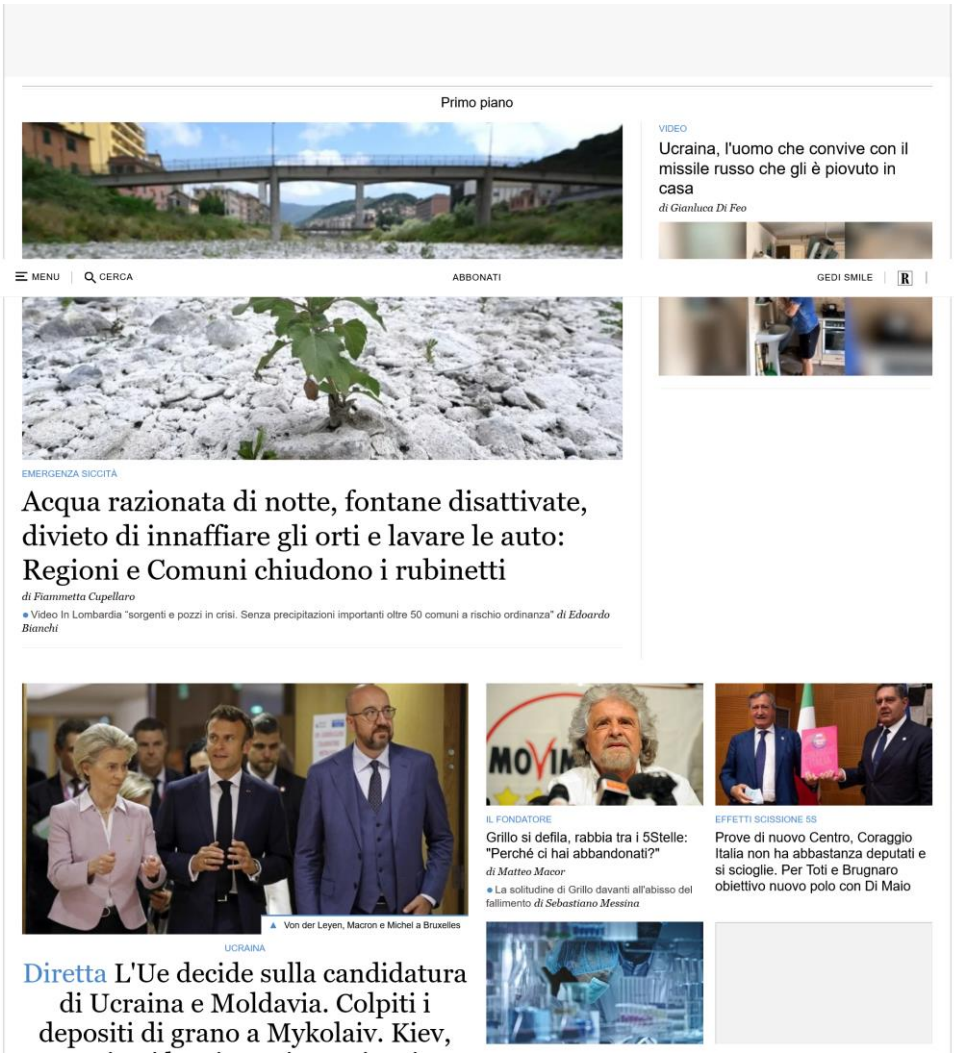


Figure 4 is the snapshot of the Newspaper HTML page. The page contains a lot of pictures of variable size (from 1 kbyte to hundreds of kbytes), text files, javascript files, html files.

For the download of each file in the web-page (e.g., images, html files, text files, javascript and json files etc.) a new HTTP(S) connection was opened and successively closed after successful file download.

For FTP(S) application/transport protocol over TCP (CUBIC or BBR) testing we have considered the download of data files of different sizes. The sizes of the files vary from 500 kbytes, 1 Mbytes and 2Mbytes.

4.4 Performance parameters considered for performance assessment

From results in D.3.4 the performance parameters that have been considered for assessment of transport and application/transport protocols are indicated in the following list:

1. Achievable throughput (TH) at the receiver side. The following statistics have been considered:
   a. The TH evaluated over the total transmission time i.e.

$$TH = \frac{\text{Total Transmitted data}}{\text{Entire duration of transmission}}$$

   b. The cumulative distribution function (cdf) of TH; this is obtained after repeating emulation several times (i.e., up to 100 repetitions) under the same operating conditions i.e., mean of latency and packet loss and for varying available transmission capacity

   c. Statistics of the total transmission time and the statistics of TH

2. In the case of application/transport protocol assessment we have considered the statistics of the download time of the single web page (HTTP(S)) and the file download time (FTP(S)) such as the CDF (Cumulative Distribution Function) and its average.

3. In the case of short message transmission, the cdf of the transmission delay has been evaluated for the variable lengths of the transmitted messages.

All data have been collected considering different operating scenario conditions including (randomly varying) one-way latency around the mean of 25, 50 and 150 ms (the mean round trip time is twice the average single link latency). One-way latency has been varied of ±5ms around the mean in the 25ms and 50ms cases while variation has been increased to ±10ms in the case of 150ms mean. Variable one way packet loss, 0, 0.1,0.3,0.5, 1.0, 2.5 has been included in emulation.

Note: for enabling TLS layer we have first created a private certification authority to authenticate client and server. The CA certificate is *ca-crt.pem*. Then we have generated the public and private key and the authenticated certificates for the client (e.g., *client1-crt.pem*, *client1-key* and *client1-csr.pem*) and for the server (*server-crt.pem*, *server-key* and *server-csr.pem*). These files are provided to the client and to the server to enable TLS operations during emulation run. The openSSL library has been used for this purpose.

5. QUIC transport protocol analysis

QUIC is a general-purpose transport layer network protocol initially designed at Google, implemented, and deployed in 2012, announced publicly in 2013 as experimentation broadened, and described at an IETF meeting. Google suggested this as a user-level protocol running over UDP instead of TCP, thus removing the necessity for the TCP protocol's initial handshake function. It runs its encryption scheme, which is comparable to TLS, combines link establishment and key agreement into 1 RTT only.

Unlike what happens with TCP, the QUIC protocol allows communications only in encrypted form. Since un-encrypted communication forms in QUIC are forbidden by design, privacy and security are inherently part of QUIC data transfers. This is important for cybersecurity but it may also represent a useless overhead when encryption is not strictly required. But the real breakthrough of QUIC consists in the time required to establish a secure connection when compared to TCP + TLS since the overhead during connection setup is reduced, see Deliverable 3.1 in [5].

5.1 **Current status of QUIC development and deployment**

The IETF started to work on QUIC not from scratch. In 2012, Google designed its own version of QUIC and then deployed it both in its popular Chrome browser and most of its services, including YouTube and search. This allowed them to observe the protocol in action and tweak its design before submitting it to the IETF for consideration in 2016. The IETF QUIC Working Group took Google’s documents as input, and has created a set of drafts that used them as a starting point.

After IETF several aspects of the protocol have been changed. The biggest change is in how encryption is negotiated. Google QUIC's bespoke encryption handshake was new to many, whereas Transport Layer Security (TLS) is more widely understood, has more features, and is much more widely supported in both implementations, and deployment. Considering the investment the community has in TLS research, security analysis, implementation, and deployment, the QUIC Working Group was chartered to use it as the basis of encryption in QUIC. In this case, when the QUIC handshake starts, the TLS handshake takes place inside of the QUIC frames, so that the peers can authenticate each other and derive session keys for encryption. Once that takes place, those keys are used to encrypt the QUIC frames. QUIC also has unidirectional as well as bidirectional streams, to aid in composing different types of applications on it.

HTTP over QUIC has changed as well. Besides explicitly separating it out into a separate document, Google's QUIC used HTTP/2's header compression scheme, HPACK. However, HPACK dictionaries track their state by assuming that ordering of messages on different streams is guaranteed by TCP–something that QUIC doesn't provide. So, we've designed a new, QUIC-specific header compression scheme, QPACK.

There are few downsides to the QUIC protocol. It improves web communications and reduces latency, but it's still in its experimental stages. It's not widely adopted by other websites or web servers, nor is it supported by cybersecurity tools such as firewalls [6]. Currently supported QUIC

versions are v1, Internet-Draft versions 29, and 27; and the older "Google" QUIC versions Q043, Q046, an Q050. Firewalls pass HTTP and HTTPS traffic through a web protection module, which performs malware scanning. But what happens if the connection is made via QUIC? Well, the browser and supporting web servers do recognize it as a QUIC connection, but the device you are browsing on may not. It treats it like simple UDP traffic, which doesn't get sent to your firewall's web protection module. The Official IETF document of QUIC is still an Internet Draft.

5.2 **QUIC available implementations**

5.2.1 Python Aioquic library

Aioquic is the python library for the QUIC network protocol in Python. We have started our investigations considering the Aioquic library. This library implements a QUIC protocol stack based on the the New Reno congestion controller and as shown in the following this may lead to reduced QUIC performance when compared to TCP using BBR congestion control. Unfortunately, it is not easy to change the congestion control algorithm in Aioquic without re-writing many parts of the available python source code. It is out of the scope of AB4Rail project to extend aioquic library to include BBR or whatever other congestion control strategy. Aioquic features a minimal TLS v1.3 implementation, a QUIC stack and an HTTP/3 stack. QUIC was standardised in RFC 9000 RFC 9001 [7] [8], but HTTP/3 standardisation is still ongoing. aioquic closely tracks the specification drafts and is regularly tested for interoperability against other QUIC implementations.
For this reason in our investigation, we have also considered other (development) QUIC libraries which are freely available on the Internet. The most important seems to be **LS-QUIC**. It is written in C and implements QUIC protocol using Cubic and BBR congestion algorithms.

5.2.2 LS-QUIC library

LiteSpeed QUIC (LSQUIC) Library is an open-source implementation of QUIC and HTTP/3 functionality for servers and clients. LSQUIC is: fast, flexible and (very important) production-ready. Currently supported QUIC versions are v1, Internet-Draft versions 29, and 27; and the older "Google" QUIC versions Q043, Q046, an Q050. It should be not very difficult (in principle) to embed LS-QUIC into products using common network programming. LS-QUIC does not use sockets to receive and send packets; that is handled by the user-supplied callbacks. The library also does not mandate the use of any particular event loop. Instead, it has functions to help the user schedule events.

The various callbacks and settings are supplied to the engine constructor. LS-QUIC keeps QUIC connections in several data structures in order to process them efficiently. Connections that need processing are kept in two priority queues: one holds connections that are ready to be processed (or "ticked") and the other orders connections by their next timer value. As a result, no connection is processed needlessly. For more detail the reader is referred to [9].

For our purposes we have used the demo programs (with many options that can be set) provided by LS-QUIC authors that implement client and server software to:

a. Test and evaluate performance of QUIC protocol for data transfer (i.e., QUIC operations as basic transport protocol);

b. Test and evaluate QUIC performance for HTTP/3 data transfer (i.e., QUIC supporting HTTP data transfer).

6. Performance Results

In this section we discuss and analyze the results obtained from emulation for the considered transport and application protocols. We reported also results for secure version of the HTTP and FTP. Moreover, we considered the delivery of short messages.

Results are organized in three main subsections:

1. Results for transport layer protocols
2. Results for application layer protocols, and
3. Results for short message delivery

For each of them we also provide an analysis and some comments on the results.

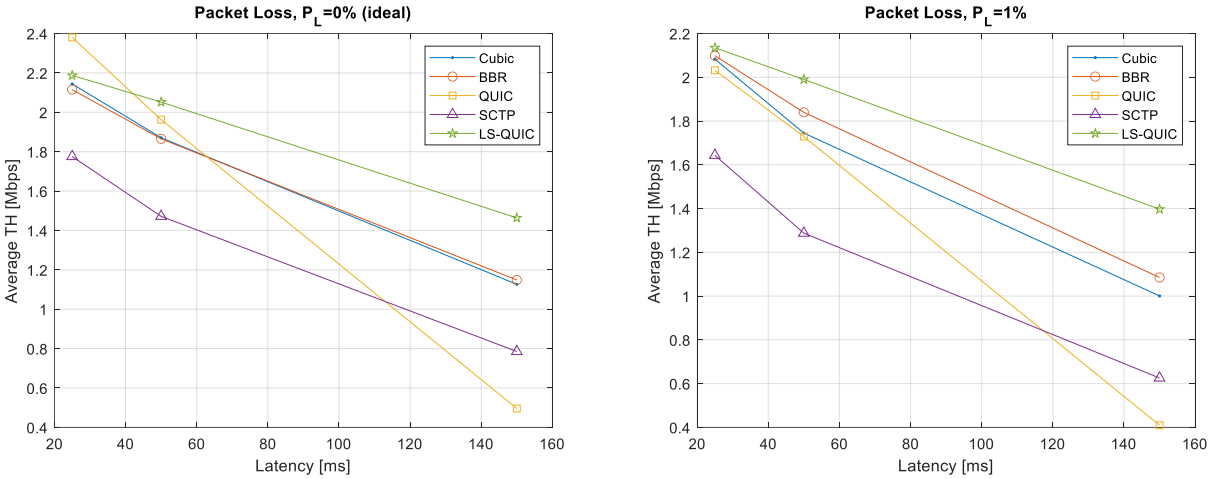
6.1 **Secure versions of the transport protocols**

In this section we considered the following transport protocols: TCP-CUBIC, TCP-BBR, QUIC, SCTP and LS-QUIC. As outlined in the previous Section performance are obtained considering the transfer of blocks of data of variable length ranging from 500 kBytes to 2 Mbytes. This method of assessing performance of transport protocol is commonly used in the real networks to measure performance of the link in real time.

In Figure 5 we reported the average Throughput as a function of the channel latency for the ideal case (PL=0%) in (a) and for a packet loss of 1% in (b). TCP-CUBIC and TCP-BBR show similar performance (slightly worst for CUBIC in case of PL=1%), having from approximately 2 Mbit/s for 25 ms to 1 Mbit/s for 150 ms. SCTP has the worst performance in all cases (in terms of packet loss and channel latency).

QUIC has the best performance in ideal case and for low channel latencies (25 ms) but it rapidly degrades showing lower performance than SCTP for latency of 150 ms. This is due to the available Python Aioquic software implementation which adopts the congestion control algorithm of New Reno type. On the contrary, LS-QUIC, which implements an adaptive congestion control algorithm, properly selecting BBR or CUBIC, overperforms the other transport protocols. Moreover, as in the case of TCP, since LS-QUIC adopts BBR, QUIC performance is slightly dependent by the packet loss experienced in the communication channel, showing performance in the ideal and PL=1% cases which are close. These results, once again, evidence the importance of the proper selection of the congestion control algorithm in the transport protocol (secure or not secure) to achieve performance.

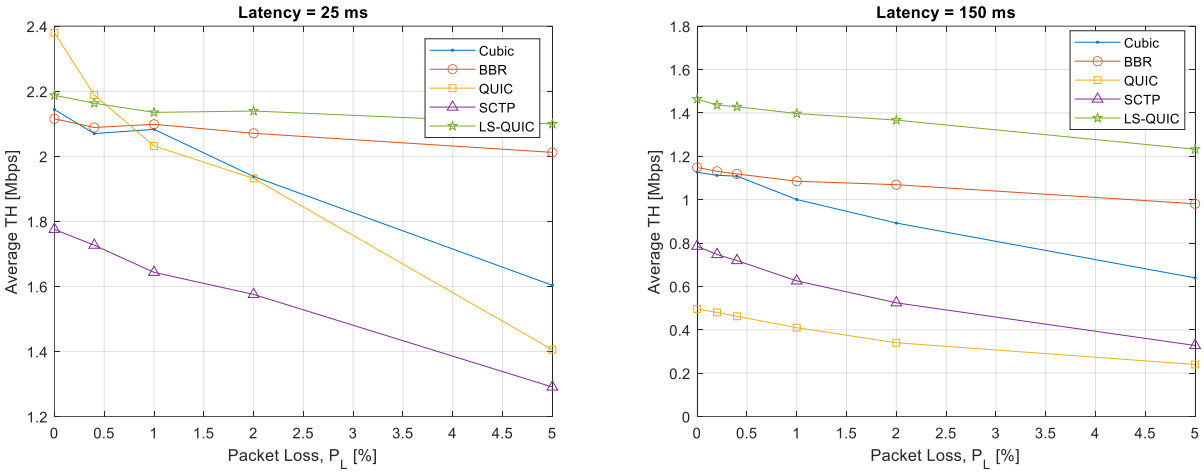
Figure 5. Average Throughput vs channel latency for: (a) PL=0% (ideal case) and (b) PL=1%.



(a) (b)

In Figure 6, the average Throughput is reported as a function of the packet loss variation in the communication channel. We considered two channel latency: 25 ms in Figure 6a and 150 ms in Figure 6b. TCP-BBR and LS-QUIC show the best performance as they are slightly dependent by the packet loss both for low latency and higher latency. This is due to the BBR congestion control algorithm adopted in both transport protocols. TCP-CUBIC has a good behavior (i.e., similar to that of TCP-BBR) for low packet loss values but it worsens for higher packet loss. The SCTP has lower performance with respect to TCP-BBR and TCP-CUBIC in all cases (packet loss and latency). Instead, QUIC provides the best performance for low latency and low packet loss (Aioquic and LS-QUIC), while QUIC (aioquic) sensibly degrades when latency increases. Instead, LS-QUIC outperforms other protocols. This behavior is mainly due to the selected implementation of the congestion control algorithm. In particular, for QUIC, the Aioquic adopts New Reno congestion control algorithm and for latency of 150 ms the throughput is lower than SCTP. Instead, LS-QUIC adopts BBR and, depending on the estimated RTT, LS-QUIC can adaptively switch to Cubic.

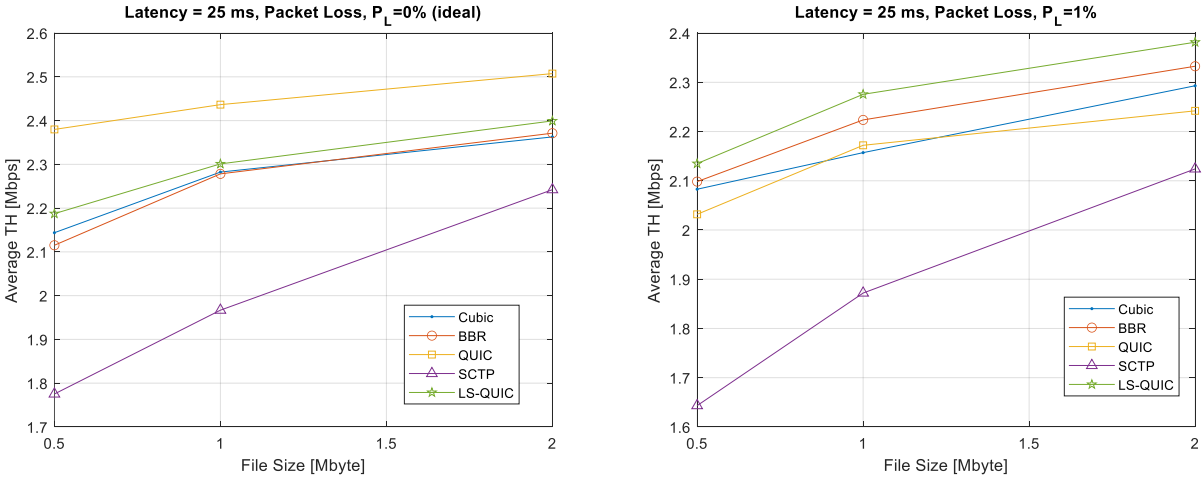
Figure 6. Average Throughput vs packet loss for (a) channel latency = 25 ms and (b) channel latency = 150 ms.



(a) (b)

In Figure 7 we reported the average Throughput as a function of the transmitted file for a channel latency of 25 ms. We considered two packet loss: 0% (ideal) in Figure 7a and 1% in Figure 7b.
As for the other plots, QUIC outperforms the other transport protocols in case of ideal transmission (i.e., PL=0%) but it degrades under the experienced Throughput of TCP-BBR, TCP-CUBIC and LS-QUIC for PL=1%. In both cases, SCTP has the lowest Throughput on average as also indicated in Del. 3.4 [1].

Figure 7. Average Throughput vs file size for a channel latency of 25 ms and for (a) PL=0% (ideal case) and (b) PL=1%.

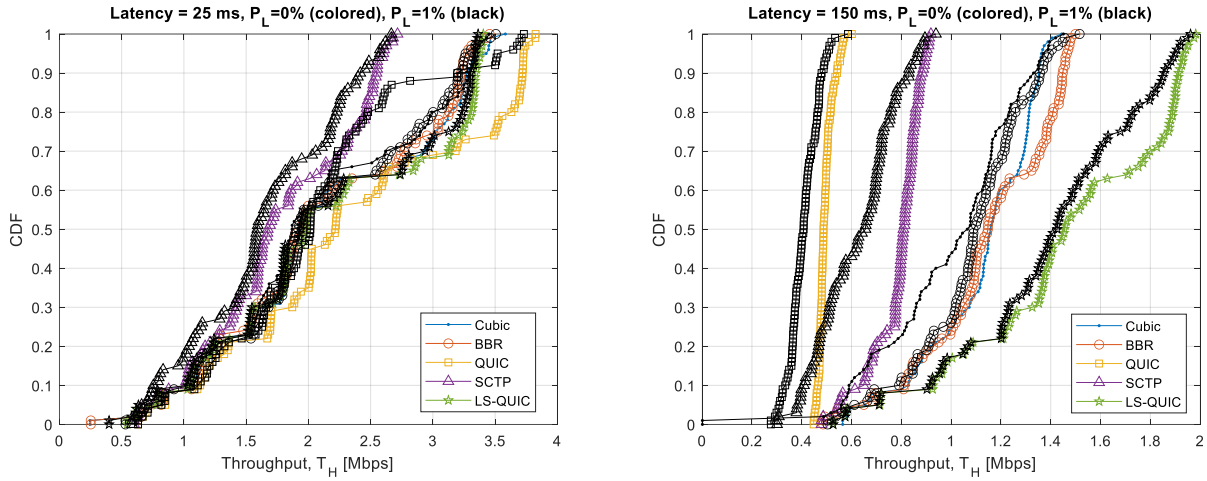


(a) (b)

Finally, in Figure 8 we reported the Cumulative Distribution Functions (CDFs) for a latency of 25 ms

in Figure 8a and a latency of 150 ms in Figure 8b. For all considered transport protocols the ideal transmission case (PL=0%) has a higher Throughput. This phenomenon is more evident in case of higher latency. In this case, LS-CUBIC has the best behavior with respect to the others, while TCP-BBR and TCP-CUBIC experience a similar Throughput. Then, SCTP and QUIC (Aioquic) show the worst performance and this is due to the implemented congestion control algorithm.

Figure 8. CDF of the throughput TH for transport protocols for ideal case (PL=0%, colored curves) and PL=1% (black curves): latency = 25 ms (a) and latency = 150 ms (b).



(a) (b)

6.2 **Application protocols**

In this section we consider and we analyze the performance of the two most popular application protocols and their secure versions: HTTP, HTTPS, FTP and FTPS. Performance results are reported in the following four subsections, respectively. Differently from the case of transport protocols in this case we considered as main performance parameter the download time. In the analysis, we evaluated the download of a webpage of a file with different sizes. The SSH File Transfer Protocol (SFTP), which adopts Secure Sockets Layer (SSL) as secure layer has been gradually replaced with the FTPS adopting TLS, which has been considered in Task 3.5 activities. In the following results have been obtained TLS v1.3.

6.2.1 HTTP

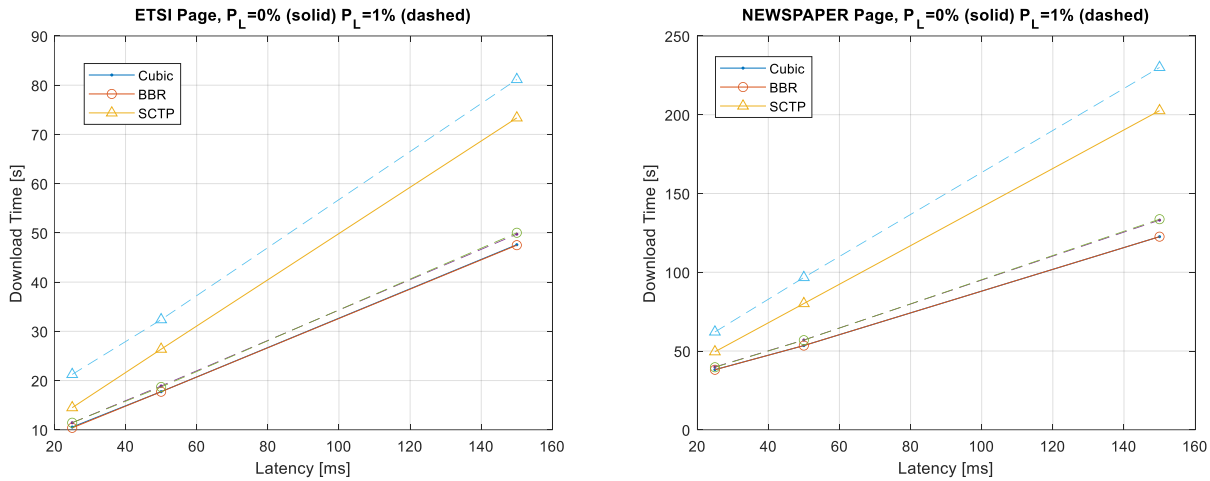
For simple HTTP we evaluated the performance of HTTP stacked over one of the following transport protocols: TCP-CUBIC, TCP-BBR and SCTP. For the moment, secure layer is not considered. For analysis purposes we have implemented the client application running on the on-board user terminal invoking the download of a webpage from a remote server. The downloading of a webpage involves the sequential download of several files including the reference web page (i.e., the first page download after the request e.g., the typical index.htm page) and all the files (images, text, javascript, etc.) which are indicated in the first downloaded page. Typically, one browser opens more simultaneous TCP connections to speed up the download. In our case we have preferred to consider only one TCP or QUIC connection at time so to have a clear picture of the application transport protocol performance in the case of a transmission channel where the available transmission capacity varies with time. In fact, we assume the user terminal is on-board moving from Rome to Florence in the mainline. The impairments of the wireless channel as seen at IP layer vary in accordance with the train mobility pattern (i.e., the train-speed vs time profile as indicated in Del. 3.4) and the radio cell coverage.

For this analysis we considered two types of webpages:

- The ETSI webpage – this is a test page issued by ETSI to test applications running over mobile devices;
- The webpage of an (Italian) daily newspaper available on the Internet.

In Figure 9 we indicate the Download Time (DT) required to download the two considered webpages. The DT depends on the channel latency ranging from 25 ms to 150 ms and increases with packet loss. The ideal case (i.e., the reference case) obtained for PL=0%, is reported in solid lines while the case corresponding to PL=1%, is indicated in dashed lines. In Figure 9a we show the DT of the ETSI page, while in Figure 9b it is reported the NEWSPAPER webpage. As expected for all transport protocols the DT increases as the channel latency increases and for higher packet loss. Moreover, the NEWSPAPER page experiences a higher DT with respect to the ETSI page due to larger amount of data to be transferred by the server to the user terminal. In both cases, TCP-CUBIC and TCP-BBR have very similar performance showing a DT ranging between 10 s and 50 s for the ETSI page and between 38 s and 2m15s for NEWSPAPER page. SCTP shows the worst performance, which further degrades with PL = 1%.

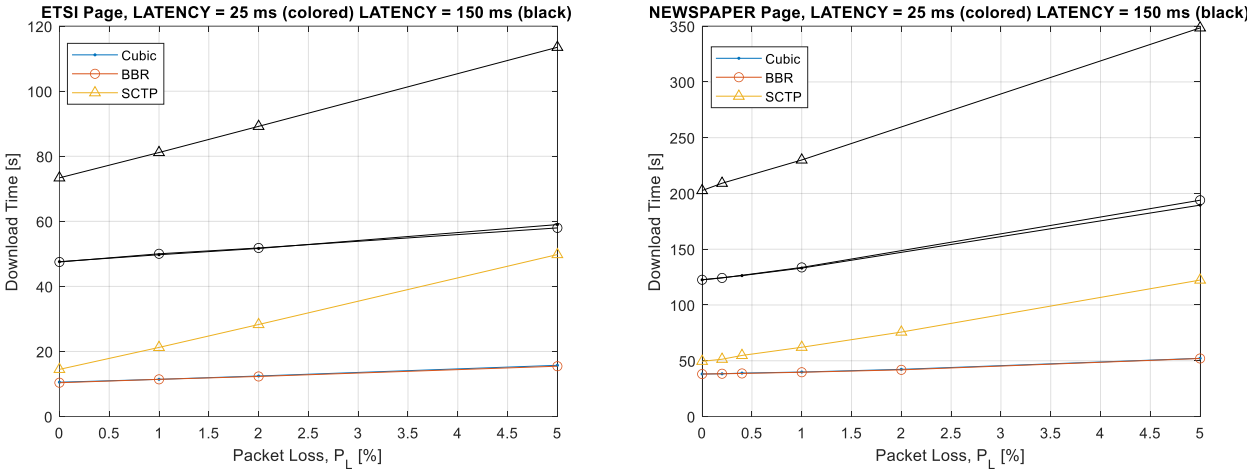
Figure 9. Average Download Time vs channel latency for PL=0% (ideal case) and PL=1%: (a) ETSI page; (b) NEWSPAPER page.



(a) (b)

In Figure 10 we reported the Download Time of the two webpages (ETSI in Figure 10a and NEWSPAPER in Figure 10b) as a function of packet loss. We also considered two values for the latencies i.e., 25 ms (colored lines) and 150 ms (black lines). Even in this case, TCP-CUBIC and TCP-BBR have similar performance and they do not show a significant degradation in case of the increase of the packet loss having a DT between 10 s and 15 s (for ETSI page) and between 38 s and 52 s (for NEWSPAPER page) for low channel latency (i.e., 25 ms). In case of higher channel latency (i.e., 150 ms) they experience a higher DT but the degradation with the increase of packet loss is not so marked. Even in this case, SCTP shows the worst performance, which degrades sensitively from ideal case (PL=0%) to PL=5% passing from 1'13" to 1'53" (ETSI page) and from 3'22" to 5'48" (NEWSPAPER page).

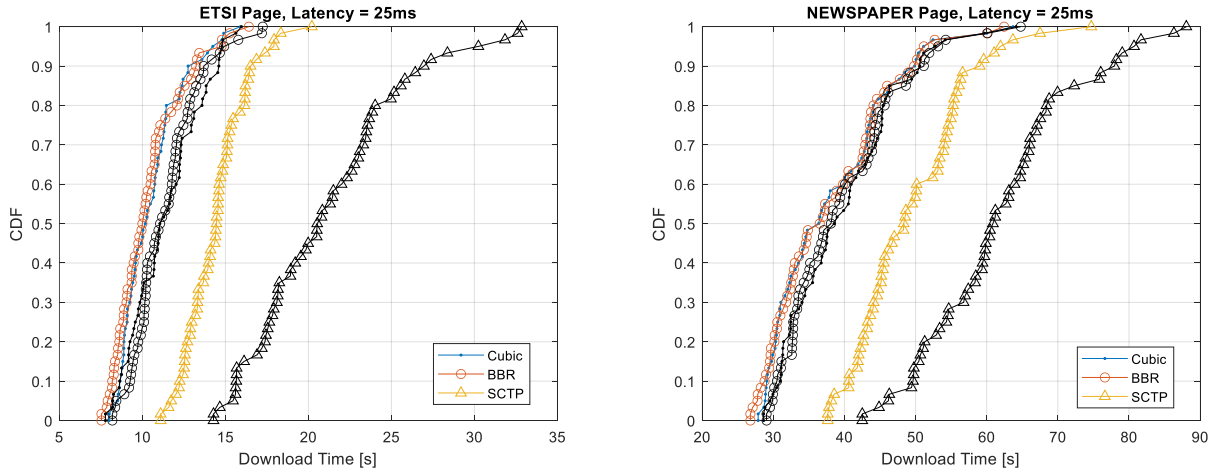
Figure 10. Average Download Time vs packet loss for a channel latency = 25 ms (colored curves) and a channel latency = 150 ms (black curves): ETSI page (a), NEWSPAPER page (b).



(a) (b)

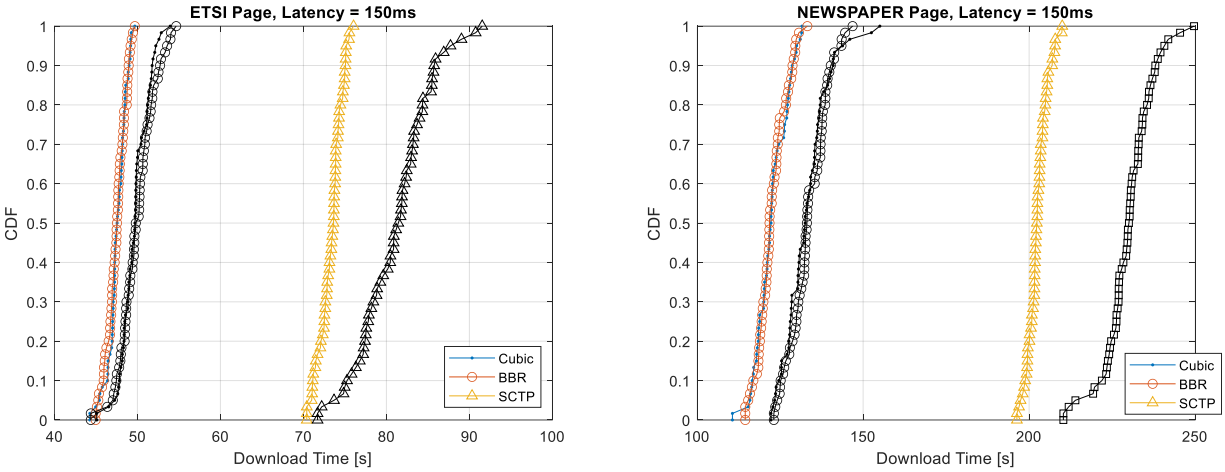
In Figure 11 and in Figure 12 we reported the CDF of the Download Time for TCP-CUBIC, TCP-BBR and SCTP for the two webpages for low channel latency (i.e., 25 ms) and high channel latency (i.e., 150 ms), respectively. Moreover, we reported the ideal case (i.e., PL=0%, reported in colored lines) and PL=1%, reported in black lines.
The CDF confirms the behavior of the average DT reported in Figure 9 and Figure 10 for the three transport protocols. CDFs give a higher sensitiveness to the performance for each measured DT.

Figure 11. CDF of the download time for HTTP for latency = 25 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): ETSI page (a) and NEWSPAPER page (b).



(a) (b)

Figure 12. CDF of the download time for HTTP for latency = 150 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): ETSI page (a) and NEWSPAPER page (b).



(a) (b)

6.2.2 HTTPS

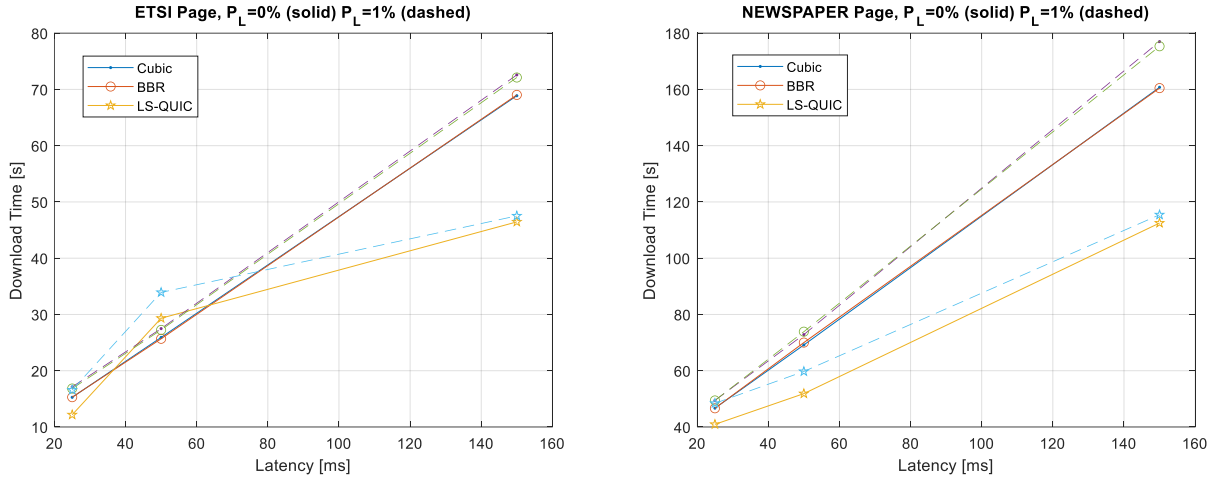
In this subsection we considered the transmission of two webpages (the ETSI and the NEWSPAPER pages), but in this case we used the HTTPS application protocol. Then, we considered the transport protocols implementing also the TLS for securing their data transmission between the server and the client: TCP-CUBIC, TCP-BBR, SCTP and LS-QUIC.

In Figure 13 it is reported the Download Time (DT) required to download all the considered webpage. The DT is as a function of the channel latency ranging from 25 ms to 150 ms and for two packet losses: ideal case (i.e., PL=0%, reported in solid lines) and PL=1%, reported in dashed lines. In Figure 13a it is reported the DT of the ETSI page, while in Figure 13b it is reported the NEWSPAPER webpage. Also in this case, as expected for all transport protocols the DT increases with the channel latency increases and with packet loss. Moreover, the NEWSPAPER page experiences a higher DT with respect to the ETSI page due to its larger amount of data to be transferred by the server to the user terminal.

TCP-CUBIC and TCP-BBR have very similar performance. They do not show any big variation with respect to the non-secure cases in Figure 9a and in Figure 9b. With HTTPS for TCP-CUBIC and TCP-BBR the DT ranges between 15 s and 70 s for the ETSI page and between 46 s and 2 min. 45 s for NEWSPAPER page.

Concerning the LS-CUBIC, it presents higher performance with respect to TCP (both CUBIC and BBR versions), more marked for a channel latency of 150 ms, showing in this case a value of 46 s (ETSI page) and 1 min 52 s (NEWSPAPER page). Moreover, its average DT is slightly affected by the packet loss.

Figure 13. Average Download Time vs channel latency for: (a) PL=0% (ideal case) and (b) PL=1%.

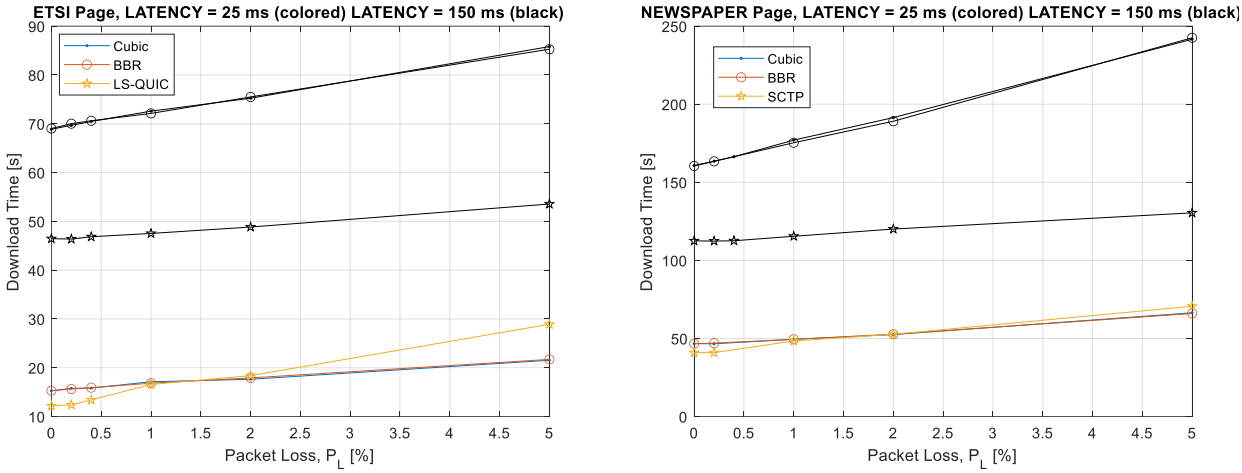


(a) (b)

In Figure 14 we reported the Download Time of the two webpages (ETSI in Figure 14a and NEWSPAPER in Figure 14b) as a function of the packet loss. We also considered two latencies: 25 ms (colored lines) and 150 ms (black lines). All three application protocols give similar performance for low latency, even if LS-QUIC presents slightly variable performance when the downloaded page is reduced.

In case of latency = 150 ms, LS-QUIC outperforms TCP-CUBIC and TCP-BBR experiencing a reduction of DT between 20 s and 30 s for the ETSI page and between 48 s and 110 s for the NEWSPAPER page. This fact can be explained by remembering that QUIC has been explicitly optimized by Google to speed up download of web-pages (i.e., the HTTP/3 has also been introduced with QUIC). To this purpose the QUIC considers UDP protocols to avoid TCP overhead and, most important, reduces the time required by the TLS to exchange the authentication information between client server. The reduction of this time is of importance especially in the case of communication links characterized by large latency.

Figure 14. Average Download Time vs packet loss for a channel latency = 25 ms (colored curves) and a channel latency = 150 ms (black curves): ETSI page (a), NEWSPAPER page (b).

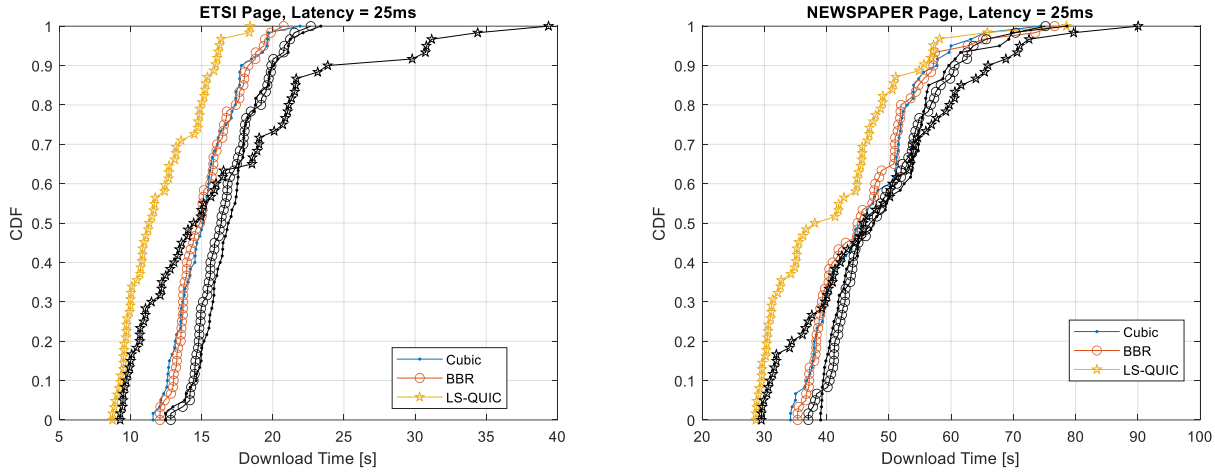


(a) (b)

In Figure 15 and in Figure 16 we have reported the CDF of the Download Time for TCP-CUBIC, TCP-BBR and LS-QUIC for the two webpages in low and high channel latency (i.e., 25 ms and 150 ms), respectively. Moreover, we have reported the reference ideal case (i.e., PL=0%, in colored lines) and PL=1%, (black lines).

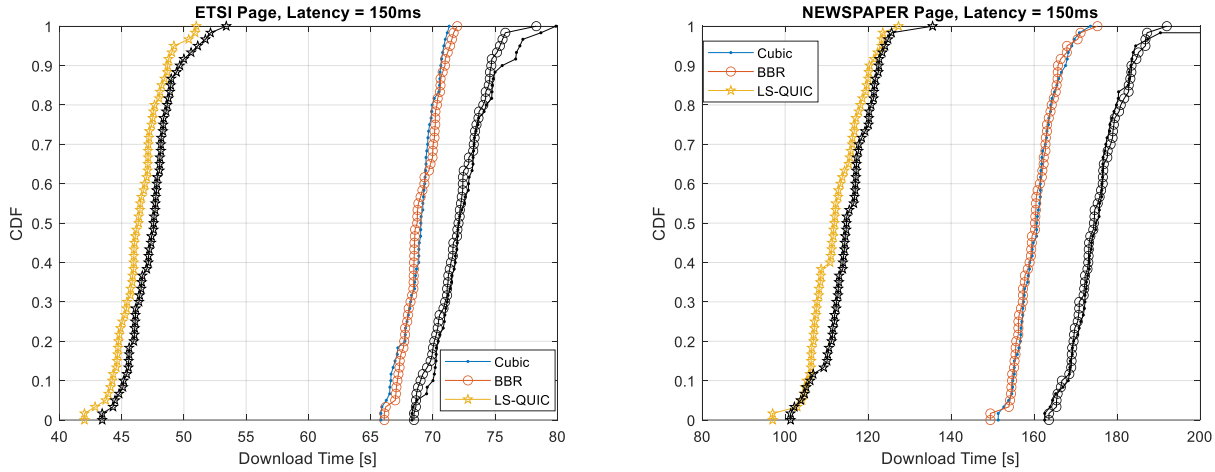
For latency of 25 ms and for the newspaper webpage, LS-QUIC has similar performance of TCP-CUBIC and TCP-BBR. In case of small web page (i.e., the ETSI page), LS-QUIC outperforms the other transport protocols in the ideal channel case, while for large latencies and PL=1% the LS-QUIC implementation may have some un-fair behavior which is related to the time interval of the probe used to estimate the RTT which is typically set to 200ms and, as in our case, it is lower than the true (average) RTT of 300ms (i.e., the average RTT is about two times the latency of 150ms). This fact has been observed experimentally in [10] and it is related to the implementation of BBR in the LS-QUIC. Some modifications to LS-QUIC implementation are currently under study for solving this problem even considering an evolution of BBR congestion control. This implementation issue leads to a loss of performance of QUIC as shown in Figure 16 the DT is greater than about 3 s with respect to TCP-CUBIC and TCP-BBR.

Figure 15. CDF of the download time for HTTP for latency = 25 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): ETSI page (a) and NEWSPAPER page (b).



(a) (b)

Figure 16. CDF of the download time for HTTP for latency = 150 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): ETSI page (a) and NEWSPAPER page (b).



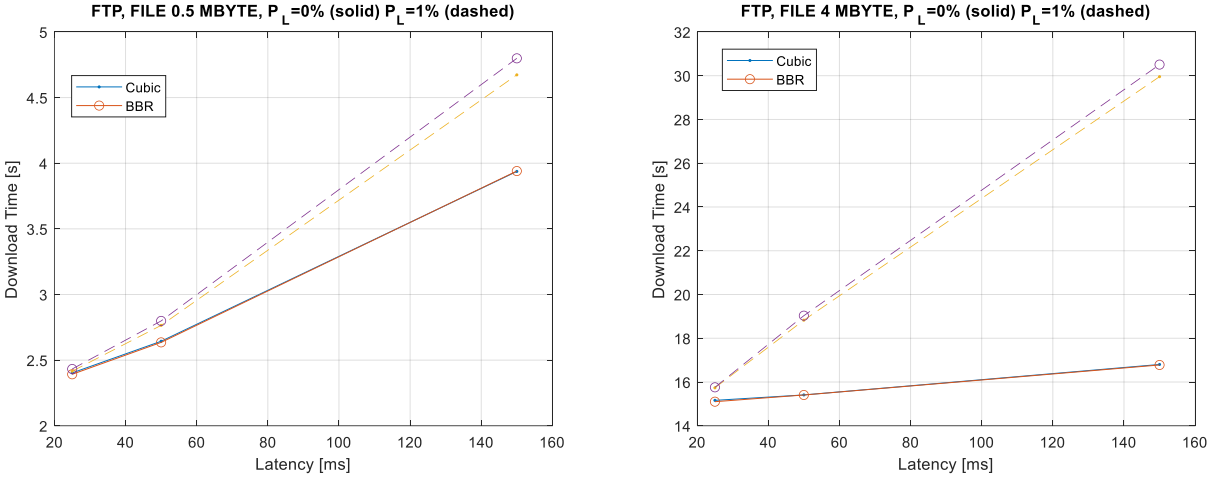
(a) (b)

6.2.3 FTP

In this subsection, we reported the Download Time required by a client in a train to download a file from a remote server. for this analysis we considered four files with different sizes of 0.5 Mbyte, 1 Mbyte, 2 Mbyte and 4 Mbyte. We considered in this analysis only TCP-CUBIC and TCP-BBR, that are currently adopted for the FTP service.

In Figure 17 it is reported the Download Time (DT) as a function of the channel latency ranging from 25 ms to 150 ms and for two packet losses: ideal case (i.e., PL=0%, reported in solid lines) and PL=1%, reported in dashed lines. In Figure 17a it is reported the DT of the smallest file (i.e., 500 kbyte), while in Figure 17b it is reported the biggest file (i.e., 4 Mbyte). Performance are similar and as expected for both transport protocols the DT increases as the channel latency increases and for higher packet loss. From Figure 17 we can also note that similar performance are obtained in ideal and with a channel with a packet loss of 1% in case of low channel latency (i.e., 25 ms). On the contrary the degradation from ideal to a lossy channel in case of latency of 150 ms, passing from 4 s to about 4.7 s for the small size file and from 17 s to about 30 s for a large size file.

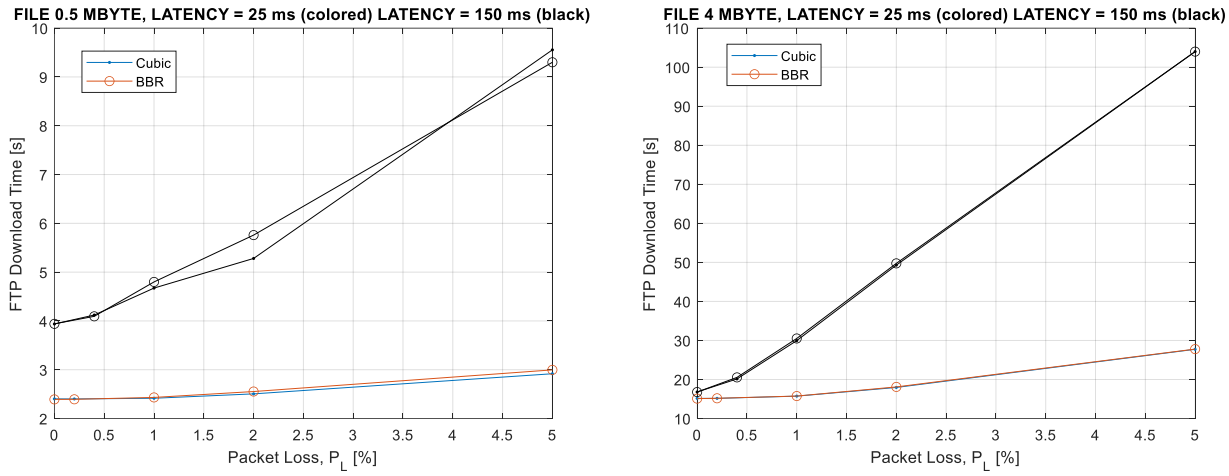
Figure 17. Average Download Time vs channel latency for PL=0% (ideal case, solid lines) and PL=1% (dashed lines): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).



(a) (b)

In Figure 18 we reported the Download Time of the two files (500 kbyte) in Figure 18a and 4 Mbyte in Figure 18b) as a function of the packet loss. We also considered two latencies: 25 ms (colored lines) and 150 ms (black lines). Also in this case, TCP-CUBIC and TCP-BBR have similar performance. They do not particularly degrade in case of the increase of the packet loss having a DT between 2.5 s and 3 s (for 0.5 Mbyte file) and between 17 s and 27 s (for 4 Mbyte file) for low channel latency (i.e., 25 ms). In case of higher channel latency (i.e., 150 ms) they experience a higher DT.

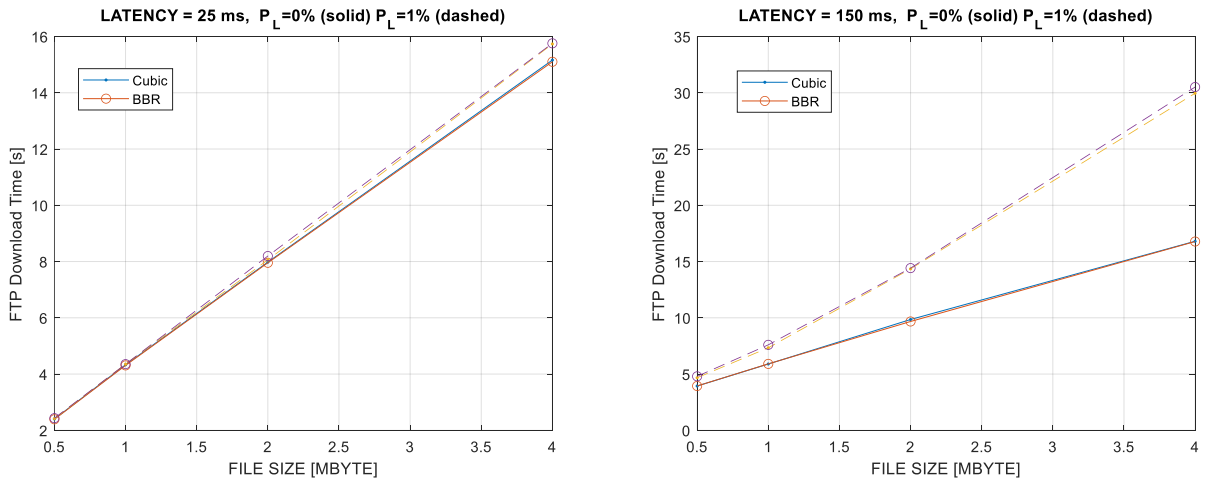
Figure 18. Average Download Time vs packet loss for a channel latency = 25 ms (colored curves) and a channel latency = 150 ms (black curves): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).



(a) (b)

In Figure 19 we reported the Download Time as a function of the file size for latency = 25 ms (Figure 19a) and latency = 150 ms (Figure 19b), for two packet losses: ideal case (i.e., PL=0%, reported in solid lines) and PL=1% (lossy channel), reported in dashed lines. TCP-CUBIC and TCP-BBR have similar performance. As expected, the DT increases as the file size increases. However, for latency = 25 ms ideal and lossy channel show the same behavior slightly depending on the file size. In case of latency = 150 ms, the degradation from an ideal to a lossy channel increase as the file size increases passing from DT = 17 s to DT = 30 s for 4 Mbyte.

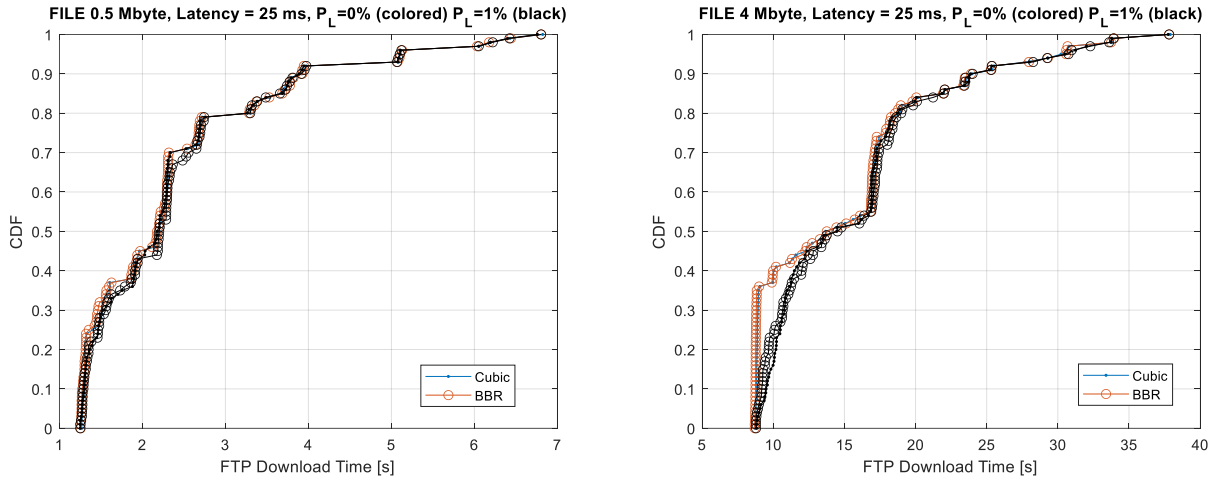
Figure 19. Average Download Time vs file size for PL=0% (ideal case, solid lines) and PL=1% (dashed lines): channel latency of 25 ms (a) and channel latency of 150 ms (b).



(a) (b)

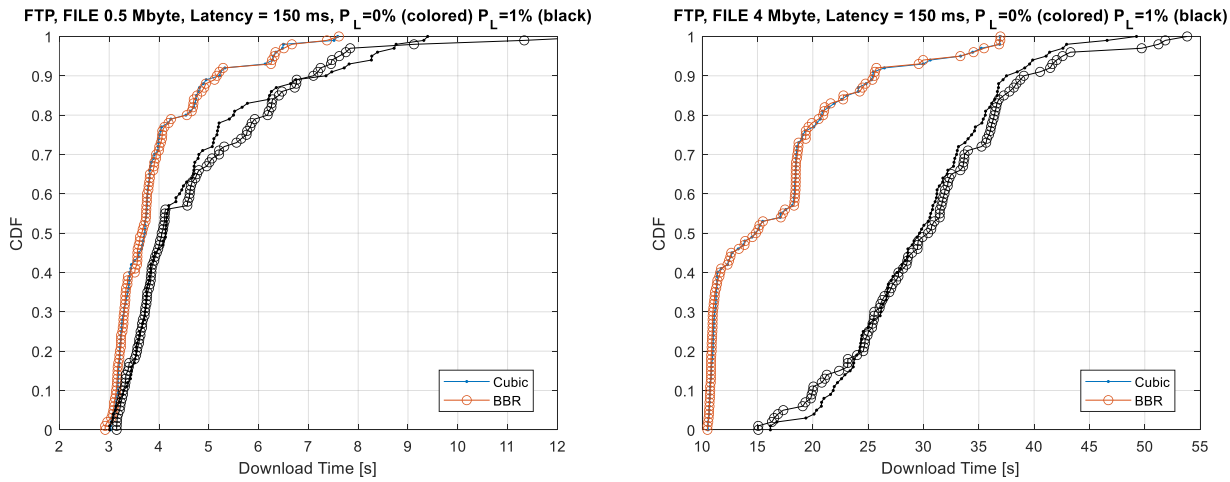
In Figure 20 and in Figure 21, we reported the CDF of the Download Time for TCP-CUBIC and TCP-BBR for the file of 500 kbyte (Figure 20a and Figure 21a) and the file of 4 Mbyte (Figure 20b and Figure 21b) for low channel latency (i.e., 25 ms) and high channel latency (i.e., 150 ms), respectively. Moreover, we reported the ideal case (i.e., PL=0%, reported in colored lines) and PL=1%, reported in black lines.
Higher Download Time are obtained in case of latency 150 ms lossy channel (PL = 1%) with respect to an ideal channel (PL = 0%).

Figure 20. CDF of the download time for FTP for latency = 25 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).



(a) (b)

Figure 21. CDF of the download time for FTP for latency = 150 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).



(a) (b)

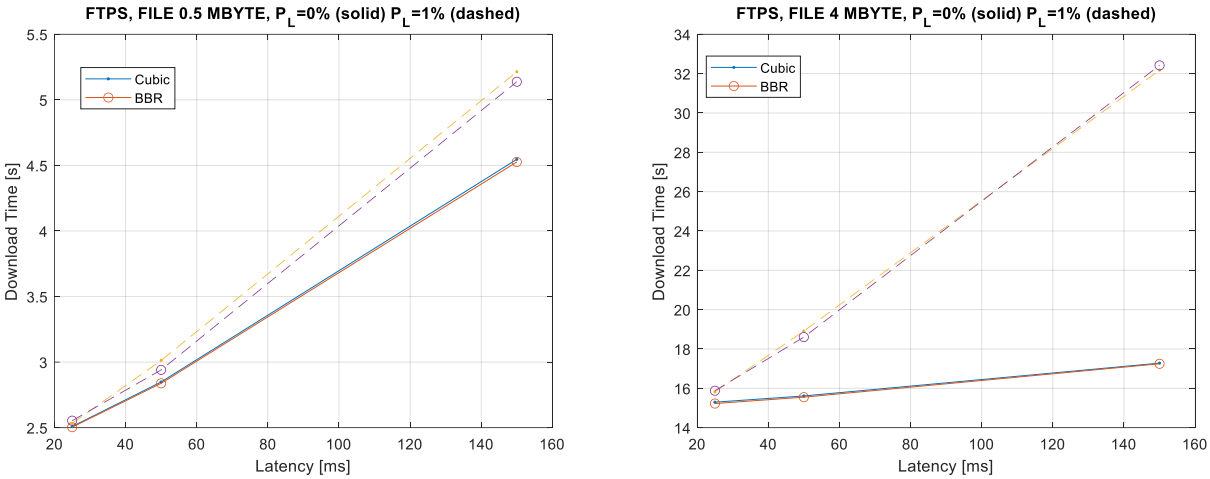
6.2.4 FTPS

In this subsection, we reported the Download Time required by a client in a train to download a file from a remote server using the FTPS application protocol. For this analysis we considered four files with different sizes of 0.5 Mbyte, 1 Mbyte, 2 Mbyte and 4 Mbyte. We considered in this analysis only TCP-CUBIC and TCP-BBR. To the best of authors' knowledge, at the moment of this writing there are no software implementations of FTP over QUIC nor it is envisaged to use QUIC for FTP transmissions. However, performance of FTP over QUIC should be not so different from the performance achieved considering TCP with BBR and TLS for transferring large files (see next). In fact, as shown in Figures 6-8 LS-QUIC and TCP with BBR almost achieve similar performance. In addition, it should be remarked that the main objective of the FTP layer, being an application protocol, is to implement additional functionalities allowing the application above the FTP to better manage the file transfer i.e., control of errors during the file transfer, the possibility of resuming transfer starting from the last byte that has been correctly received etc. Apart of the overhead introduced by the FTP layer, the transmission of files as well as the achievable throughput/file transfer rate is regulated by the operations of the underlying transport protocol.

In Figure 22 it is reported the Download Time (DT) as a function of the channel latency ranging from 25 ms to 150 ms and for two packet losses: ideal case (i.e., PL=0%, reported in solid lines) and PL=1%, reported in dashed lines. In Figure 22a it is reported the DT of the smallest file (i.e., 500 kbyte), while in Figure 22b it is reported the biggest file (i.e., 4 Mbyte). Performance of the two TCP versions are similar and as expected for both transport protocols the DT increases as the channel latency increases and for higher packet loss. From Figure 22 we can also note that similar performance are obtained in ideal and with a channel with a packet loss of 1% in case of low channel latency (i.e., 25 ms). On the contrary the degradation from ideal to a lossy channel in case of latency of 150 ms, passing from 4 s to about 4.7 s for the small size file and from 17 s to about 30 s for a large size file.

Comparing them with the classical FTP application transfer protocol, it is noted a small increase of the DT due to the TLS overhead inserted in the packets of the secured FTP version and the additional time required for initial TLS handshaking that can be more marked with the increase of channel latency. However, in the case of large file transfer this additional time can be negligible with respect to the overall time required for the file download. As an example, the DT increases from 3.9 s to 4.6 s for latency = 150 ms, lossless channel and file size = 500 kbyte.

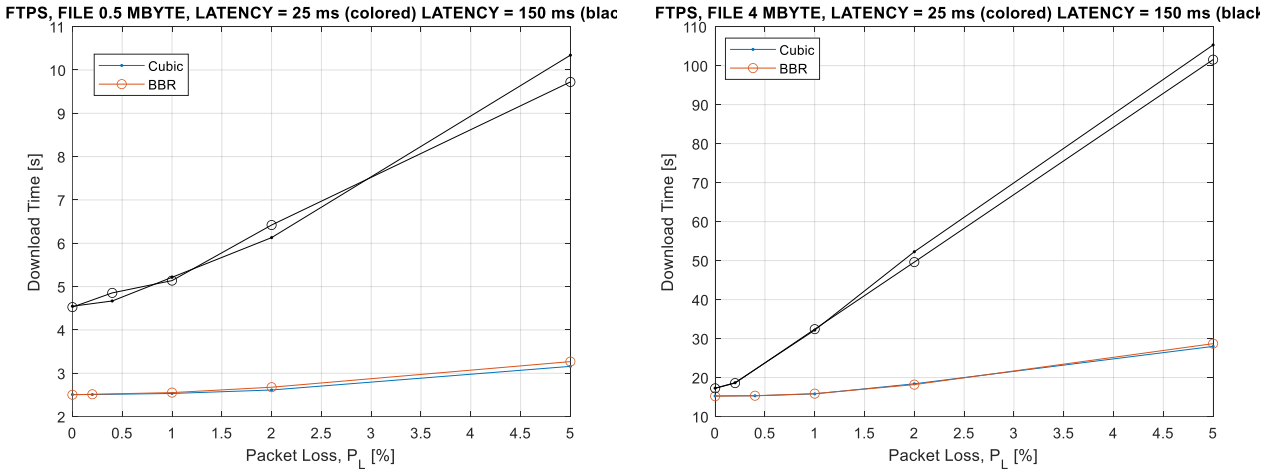
Figure 22. Average Download Time vs channel latency for PL=0% (ideal case, solid lines) and PL=1% (dashed lines): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).



(a) (b)

In Figure 23 we reported the Download Time of the two files (500 kbyte in Figure 23a and 4 Mbyte in Figure 23b) as a function of the packet loss. We also considered two latencies: 25 ms (colored lines) and 150 ms (black lines). Also in this case, TCP-CUBIC and TCP-BBR have similar performance. Considerations for this behavior are similar to those concerning the classical FTP case.

Figure 23. Average Download Time vs packet loss for a channel latency = 25 ms (colored curves) and a channel latency = 150 ms (black curves): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).

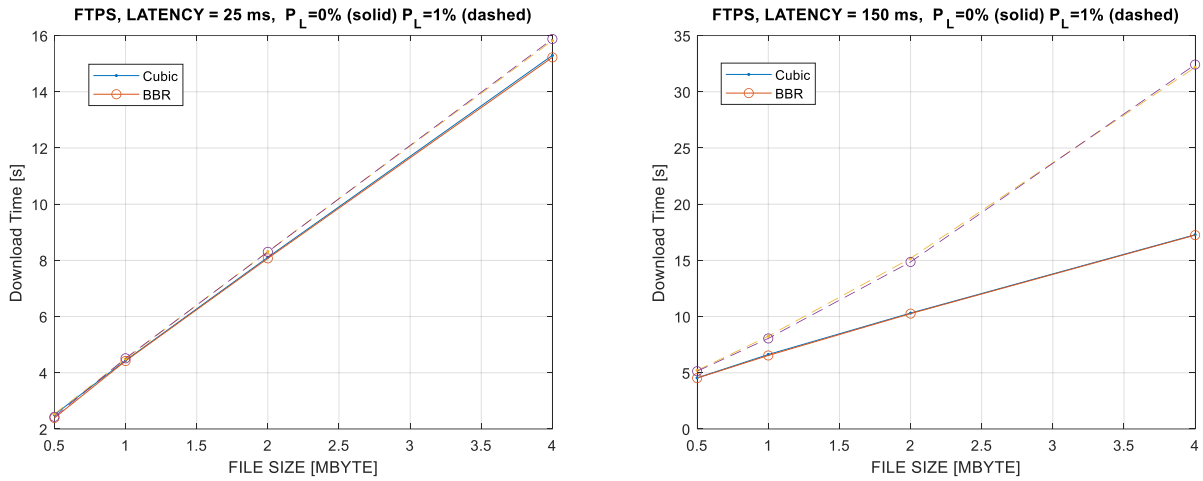


(a) (b)

In Figure 24 we reported the Download Time as a function of the file size for latency = 25 ms (Figure 24a) and latency = 150 ms (Figure 24b), for two packet losses: ideal case (i.e., PL=0%, reported in

solid lines) and PL=1% (lossy channel), reported in dashed lines. TCP-CUBIC and TCP-BBR have similar performance. As expected, the DT increases as the file size increases. However, for latency = 25 ms ideal and lossy channel show the same behavior slightly depending on the file size. In case of latency = 150 ms, the degradation from an ideal to a lossy channel increases as the file size increases passing from DT = 17 s to DT = 30 s for 4 Mbyte.
FTPS slightly has worsen performance with respect to FTP above all for file size = 4 Mbyte, for latency = 150 ms and PL=1% passing from 30 s (Figure 19b) to 32 s (Figure 24b).

Figure 24. Average Download Time vs file size for PL=0% (ideal case, solid lines) and PL=1% (dashed lines): channel latency of 25 ms (a) and channel latency of 150 ms (b).

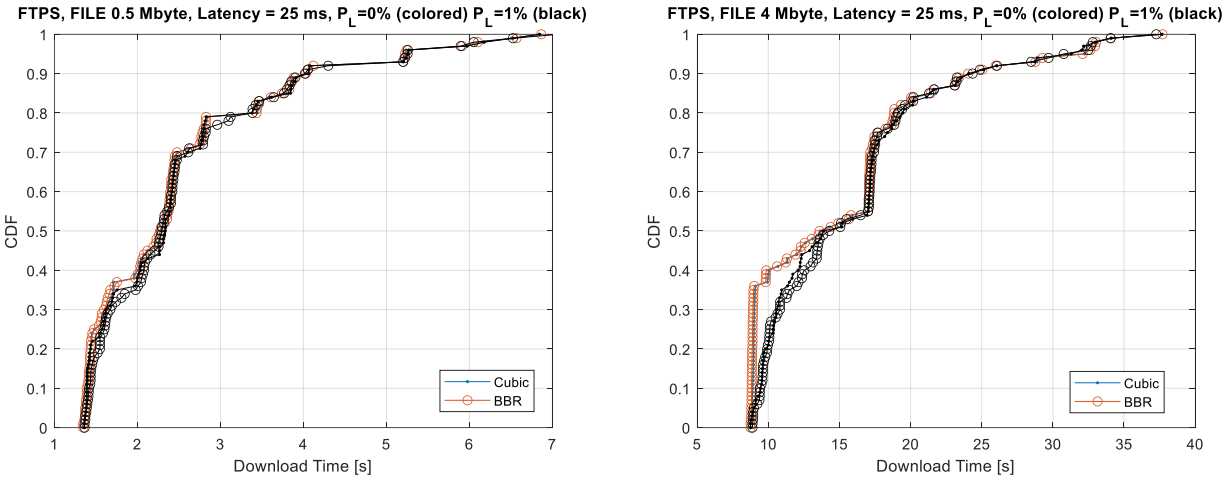


(a) (b)

In Figure 25 and in Figure 26, we reported the CDF of the Download Time of FTPS for TCP-CUBIC and TCP-BBR for the file of 500 kBYTE (Figure 25a and Figure 26a) and the file of 4 MBYTE (Figure 25b and Figure 26b) for low channel latency (i.e., 25 ms) and high channel latency (i.e., 150 ms), respectively. Moreover, we reported the ideal case (i.e., PL=0%, reported in colored lines) and PL=1%, reported in black lines. Higher DTs are obtained in case of latency 150 ms lossy channel (PL = 1%) with respect to an ideal channel (PL = 0%).

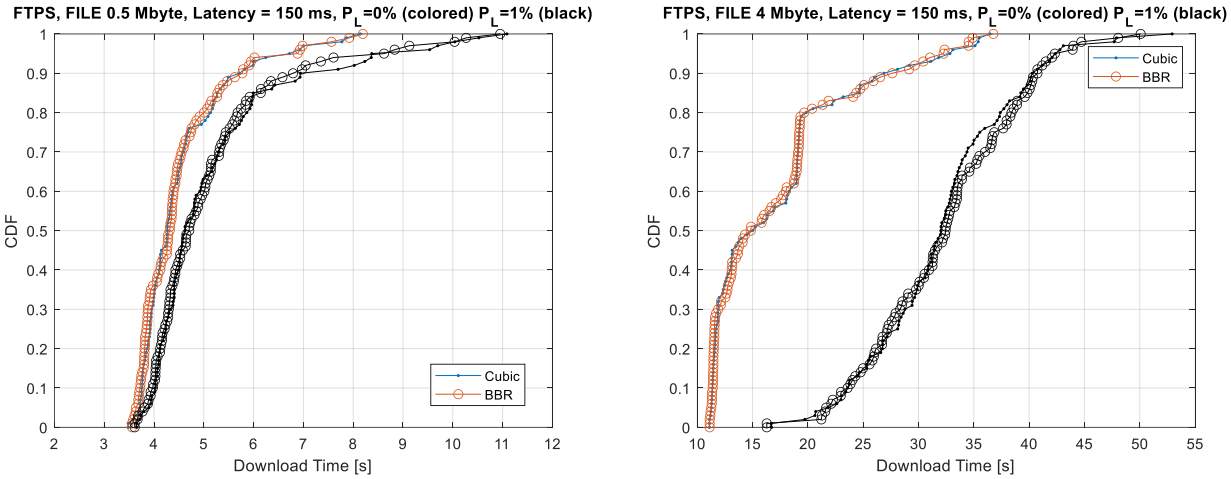
TCP-CUBIC and TCP-BBR show similar performance. Anyway, also in this case FTPS has a small degradation with respect to classical FTP mainly due to additional TLS overhead. As an example, we can refer to the starting point of Figure 26a for FTPS, which is equal to 3.5 s, and it is 0.5s higher than the starting point of the corresponding FTP case (see Figure 21a), which starts at 3 s.

Figure 25. CDF of the download time for FTP for latency = 25 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).



(a) (b)

Figure 26. CDF of the download time for FTP for latency = 150 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves): file size = 0.5 MBYTE (a); file size = 4 MBYTE (b).



(a) (b)

6.3 **Periodic Short Messages Delivery**

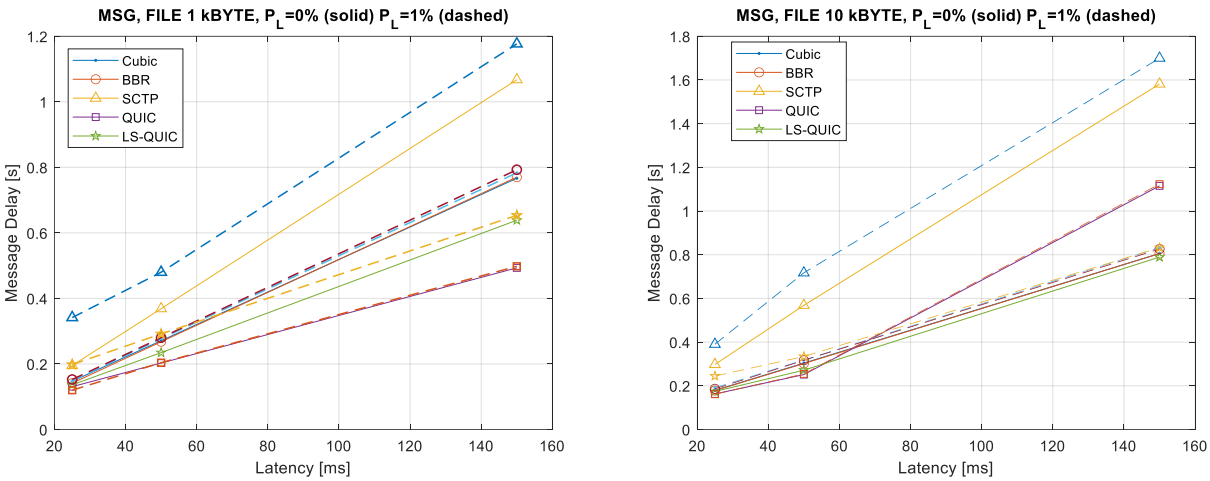
In this section we analyze the performance of the considered transport protocols in the case of delivery of periodic short messages. This study case is of great importance since it can be associated to the class of ACS services concerning signaling using short messages as in the ETRMS/ETCS case or SIP/SDP. We considered two lengths of the messages to be sent: 1 kbyte and 10 kbyte. The following secure versions of transport protocols are considered: TCP-CUBIC, TCP-BBR, SCTP, QUIC with Aioquic and LS-QUIC implementations.

The emulator developed in Task 3.3. is used for performance assessment (see Figure 1). We assume the transmitting application is on board and transmit periodic (short) messages to a remote server. One new message is transmitted every 3 seconds. The connection from the terminal to the server is opened before starting transmission and closed when message transmission ends. To evaluate performance we have considered two different message delays: the first delay includes the time interval required for the initial TLS handshake carried out before message transmission and the time interval between the transmission and reception of the message after the TLS handshake is completed. Obviously, by definition the first time interval is the sum of the TLS handshake time and that required for message reception. This second time interval is of importance to assess the performance of message transmission when non-secure version of the transport protocols are used since in this case TLS handshake is not present.

6.3.1 Message Delay including TLS handshaking time

In Figure 27 it is reported the average Message Delay (MD) required to send the short messages. As expected, the overall MD is as a function of the channel latency ranging from 25 ms to 150 ms. We have analyzed MD ecven considering packet loss: ideal case (i.e., PL=0%, reported in solid lines) and PL=1%, reported in dashed lines. In Figure 27a it is reported the MD of the message of 1 kbyte, while in Figure 27b it is reported the message of 10 kbyte. As expected for all transport protocols the MD increases as the channel latency increases as well as the packet loss increases. Obviously, the message with a greater size takes a larger delay. QUIC outperforms all the other transport protocols for small-size files and for 10 kbyte-size in case of low latencies (lower than 60 ms), while it has an MD great then both versions of the TCP for latency = 150 ms. TCP-CUBIC and TCP-BBR have very similar performance showing a MD ranging between 150 ms and 800 ms for 1 kbyte-file size and between 185 ms and 830 ms for 10 kbyte-file size for a lossless channel. SCTP shows the worst performance, which further deteriorates for PL = 1%. For QUIC and TCPs performance do not degrade significantly for the lossy channel with respect to the lossless channel.

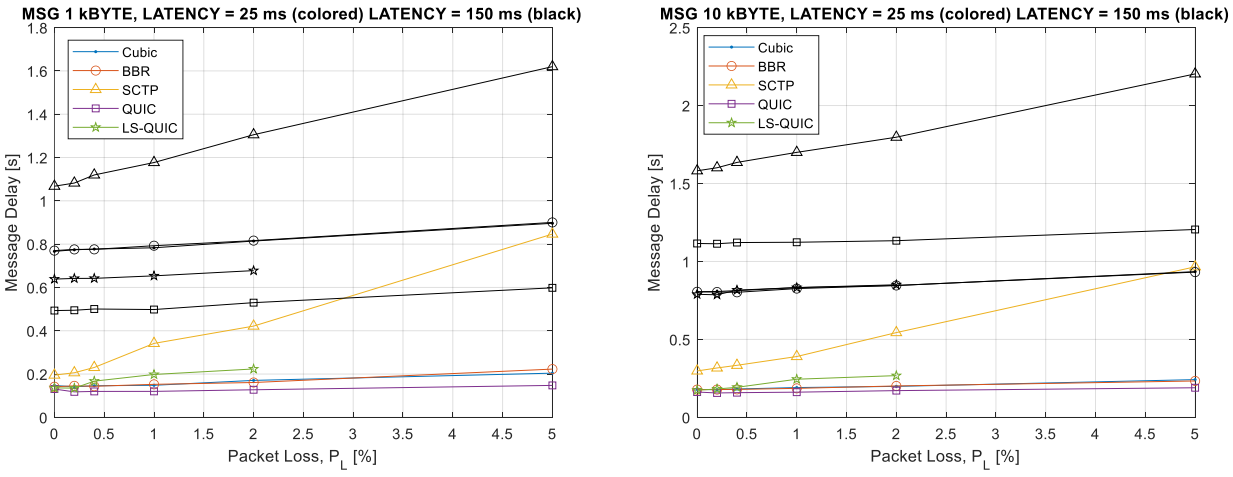
Figure 27. Average Message Delay vs channel latency for PL=0% (ideal case) and PL=1%: (a) 1 kbyte-file size; (b) 10 kbyte-file size.



(a) (b)

In Figure 28 we reported the average Message Delay of the two files (1 kbyte-file size in Figure 28a and 10 kbyte-file size in Figure 28b) as a function of the packet loss. We also considered two latencies: 25 ms (colored lines) and 150 ms (black lines). TCP-CUBIC, TCP-BBR and QUIC have similar performance for latency = 25 ms and they do not particularly degrade in case of the increase of the packet loss having a MD between 140 ms and 220 ms (for 1 kbyte-file size) and between 175 ms and 230 ms (for 10 kbyte-file size). In case of higher channel latency (i.e., 150 ms), QUIC has a better behavior with respect to TCP-CUBIC and TCP-BBR for 1-kbyte file, while it experiences a higher MD for 10-kbyte file.
Also in this case, SCTP shows the worst performance, which degrades sensitively from ideal case (PL=0%) to PL=5% passing from 195 ms to 850 ms (1 kbyte-file size) and from 300 ms to 970 ms (10 kbyte-file size) for latency = 25 ms.

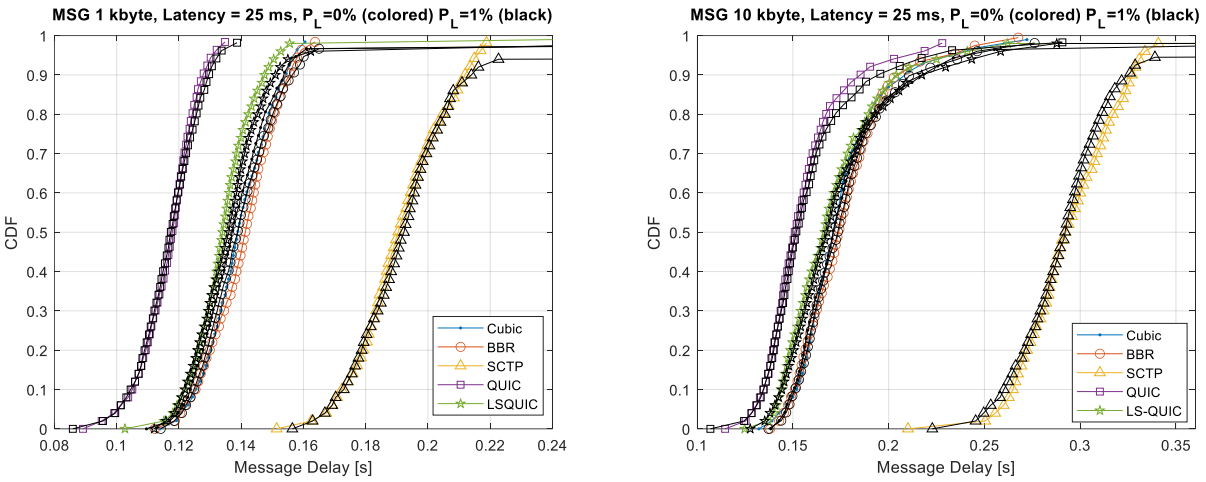
Figure 28. Average Message Delay vs packet loss for a channel latency = 25 ms (colored curves) and a channel latency = 150 ms (black curves): (a) 1 kbyte-file size; (b) 10 kbyte-file size.



(a) (b)

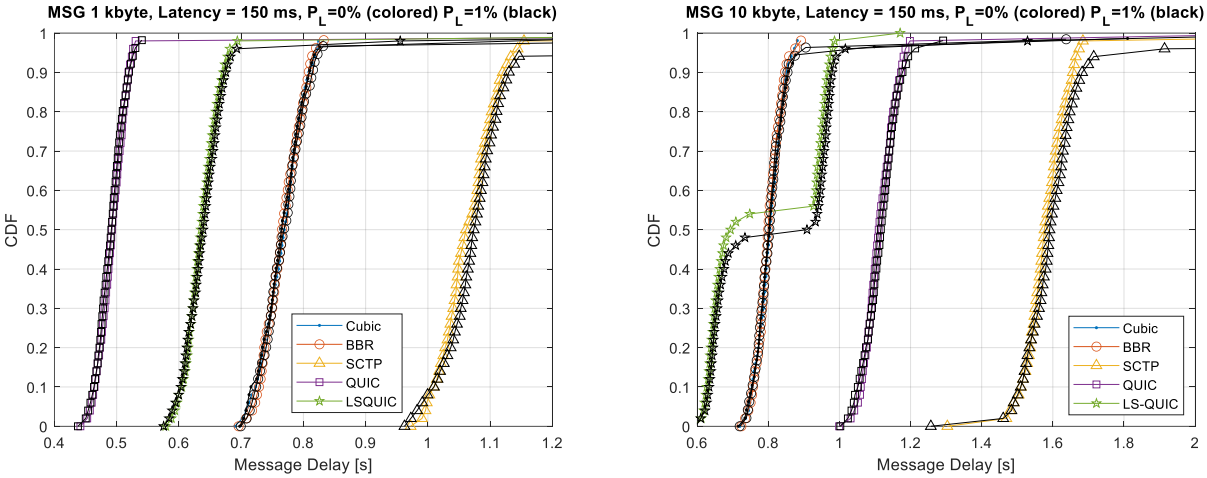
In Figure 29 and in Figure 30 we have reported the CDF of the Message Delay for TCP-CUBIC, TCP-BBR, SCTP and QUIC of the two file sizes (1 kbyte and 10 kbyte) for low channel latency (i.e., 25 ms) and high channel latency (i.e., 150 ms), respectively. Moreover, we reported the ideal case (i.e., PL=0%, reported in colored lines) and PL=1%, reported in black lines.
The CDF confirms the behavior of the average MD reported above for the three transport protocols. CDFs give a higher sensitiveness to the performance for each measured MD. In fact, it is possible to note that the MD for TPC-CUBIC and TCP-BBR ranges close to latency = 25 ms (Figure 29a) and latency = 150 ms (Figure 30a) for 1 kbye case. The packet loss hardly affects their performance due to the small size of the sent messages. MD increases as the file size increases (see Figure 29b and Figure 30b).

Figure 29. CDF of the Message Delay of Message delivery service for latency = 25 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves):
(a) 1 kbyte-file size; (b) 10 kbyte-file size.



(a) (b)

Figure 30. CDF of the Message Delay of Message delivery service for latency = 150 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves):
(a) 1 kbyte-file size; (b) 10 kbyte-file size.



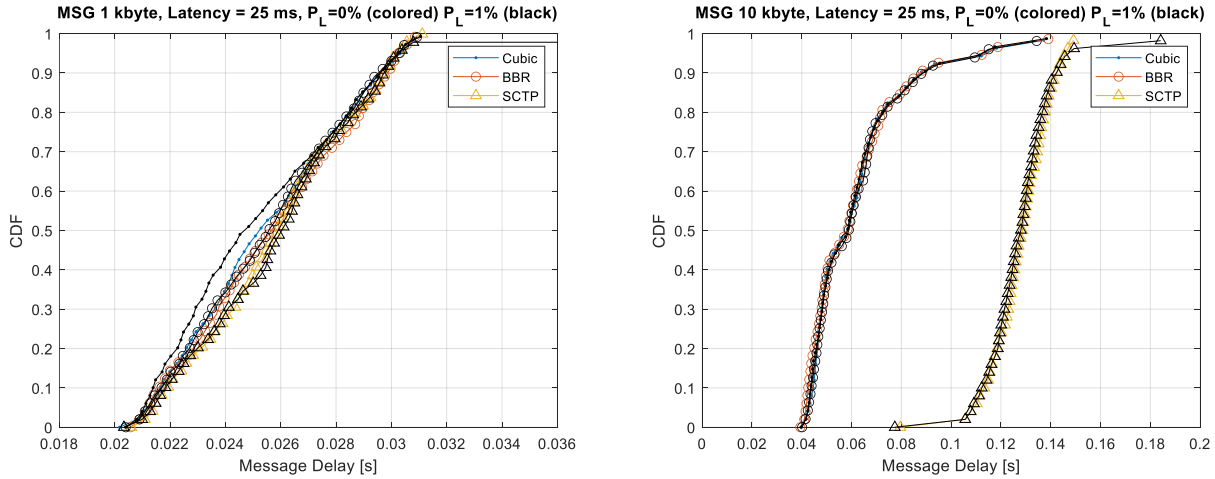
(a) (b)

For the LS-QUIC case we can observe a bi-modal behavior of the protocol in the case of delay of 150ms and large messages that can lead to a degradation of QUIC performance i.e., an increase of MD with respect to TCP-BBR with TLS for a percentage of packets. This behavior is not easy to explain but we should remember that LS-QUIC is set by default to send an RTT probe every 200ms which is lower than the RTT of about 300ms. Instead, the TCP-BBR send one probe every 8 RTT (typically). This leads to enqueuing remaining packets to be transmitted packets due to the delay in remining in the draining phase of BBR (i.e., in the drain status the packet transmission is stopped and it can restart only after we have received the ACK of the transmitted packets). This can explain the bimodal behavior of LS-QUIC shown in Figure 30. The un-fair behavior of BBR has been studied in [10].

6.3.2 Message Delay without the TLS negotiation time

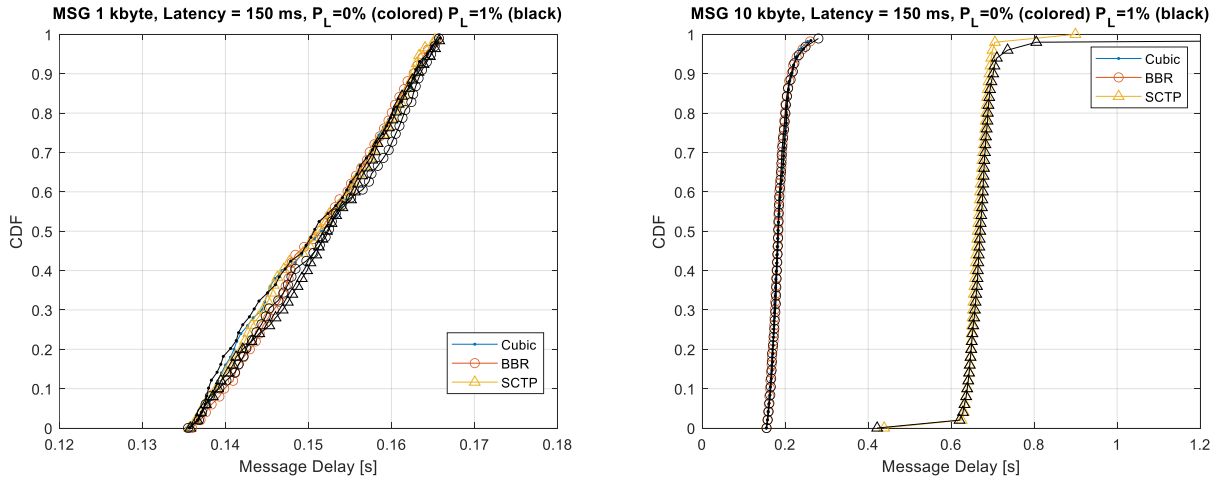
In this Section we provide results concerning the MD without including the time interval required for initial TLS handshaking. As previously outlined, these results can be representative of performance non-secure protocols even though the results do not include the delay due to protocol initial synchronization phase. In this case, only results for TCP with BBR and Cubic and SCTP are reported. In fact, it makes no sense to strip the TLS layer from QUIC.

Figure 31. CDF of the Message Delay of Message delivery service for latency = 25 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves):
(a) 1 kbyte-file size; (b) 10 kbyte-file size.



(a) (b)

Figure 32. CDF of the Message Delay of Message delivery service for latency = 150 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves):
(a) 1 kbyte-file size; (b) 10 kbyte-file size.



(a) (b)

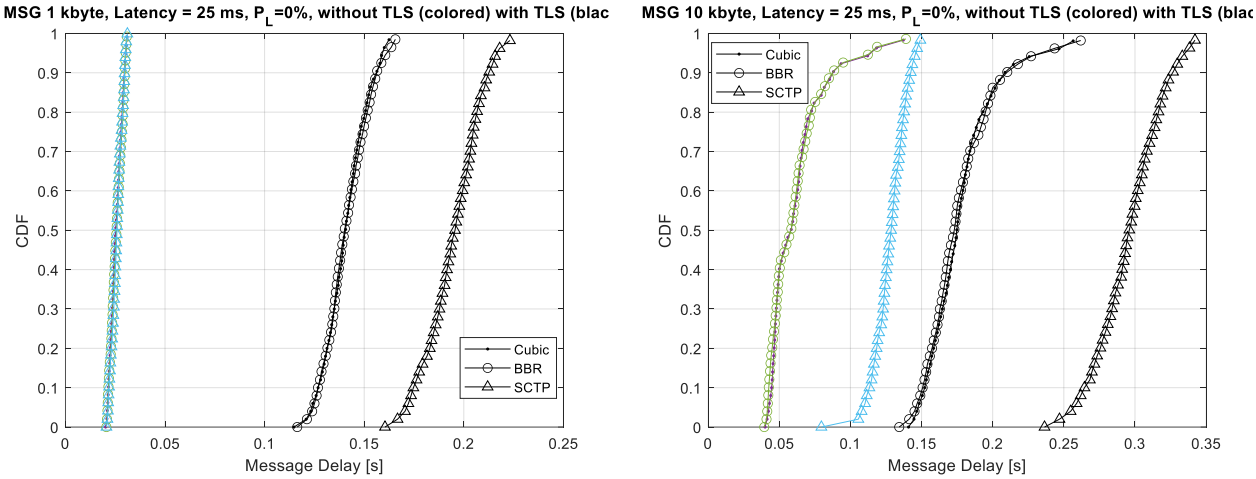
As shown in Figure 31 and Figure 32 the absence of TLS handshaking allows to drastically reduce the MD. In Figure 31 corresponding to latency of 25 ms the minimum MD is 20 ms. This is not a surprise since it should be remined that latency in the emulator is randomly generated at each iteration in accordance with a uniform distribution around its mean of 25ms±5ms. Similar considerations apply to the results indicated in Figure 32 corresponding to different mean latency of 150 ms.
In case of messages of 1 kbyte, TCP-BBR, TCP Cubic and SCTP show similar performance, independently of the channel latency, while in case of messages of 10 kbyte SCTP degrades with respect to both versions of the TCP.

6.3.3 Comparison of message delay with and without TLS handshake

In this section we compare the achievable MD for short message transmission in the absence or presence of TLS negotiation. For brevity, only the CDFs of MD are reported.

In Figure 33a and Figure 33b we reported the CDF of the Message Delay for TCP-CUBIC, TCP-BBR, SCTP, QUIC and LS-QUIC for low channel latency (i.e., 25 ms), and for the file size of 1 kbyte, and the file size of 10 kbyte, respectively. Moreover, we reported the CDF of MD, which includes the TLS setup time for each message (black lines) and the simplified case, where TLS has not been included in the setup reported in colored lines. It is evident the increased delay due to the TLS setup. Note that in case of non-secure transmission (no TLS setup) and for the file size of 1 kbyte, delays are similar to the channel latency and similar for all transport protocols due to the reduced size of the file. In case of the file size of 10 kbyte, SCTP without TLS (triangle cyan in Figure 33b) degrades.

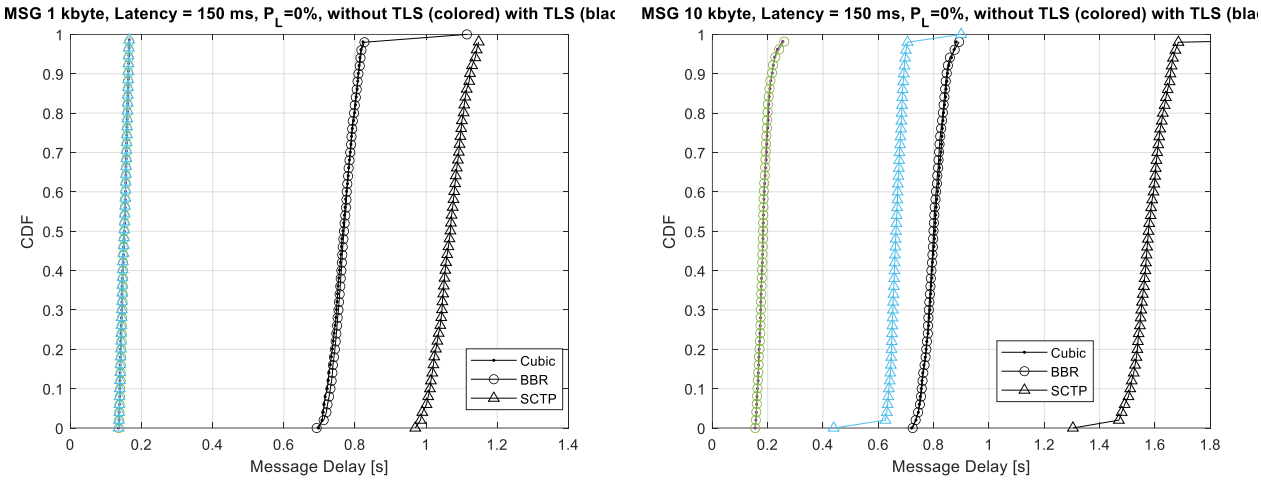
Figure 33. CDF of the Message Delay of Message delivery service for latency = 25 ms and for ideal case (PL=0%), colored curves) and PL=1% (black curves):
(a) 1 kbyte-file size; (b) 10 kbyte-file size.



(a) (b)

Similar considerations can be noted in case of for latency = 150 ms, whose results are reported in Figure 35 (Figure 35a for 1 kbyte and Figure 35b for 1 kbyte).

Figure 34. CDF of the Message Delay of Message delivery service for latency = 150 ms and for ideal case (PL=0%, colored curves) and PL=1% (black curves):
(a) 1 kbyte-file size; (b) 10 kbyte-file size.



(a) (b)

7. Analysis of security threats and possible defense techniques

In this section we analyze the security aspects of the considered transport and application protocols especially when message transmission is considered. The security aspects of the RASTA protocol have been analyzed in the D3.4 taking results from the available literature. In this Section we are focusing on TCP, UDP, SCTP even with TLS and QUIC. Usage of transport protocols adopting TLS allow the application protocols, HTTP and FTP, to transmit data from the source to the destination in a secure way.

7.1 **Most common threats and countermeasures envisaged in message transmissions**

The most common threats to be considered in the case of rail applications are reported in the following list, (from [3]):

- ***Repetition.***
  - The attack is performed by storing a packet by the attacker, who transmit it subsequently, thus giving to the application wrong information (e.g., not updated train position), maybe in different situation, for example, injecting a message collected when the train is at 250 km/h and replaying it when the train is at 40 km/h or vice versa;

- ***Deletion.***
  - The attacker intercepts a packet and deletes it, leaving the destination without the information inserted in the packet, for example cancelling the message “immediately stop”;

- ***Insertion.***
  - The attacker inserts a message thus providing to the train wrong information, for example allowing its speed to be 250 km/h in rail slots where it is not safe;

- ***Re-sequencing.***
  - The attacker intentionally (or a hardware failure unintentionally) modifies the order of the messages, thus for example changing the meaning of the information sent by the train to the remote-control center or vice versa;

- ***Corruption.***
  - The message has been modified intentionally or unintentionally, providing the recipient with wrong information such as speed at 250 km/h is available in the part of the rail;

- ***Delay.***
  - Messages are delayed above the maximum allowed delay causing for example train stop due to missing movement authorization. In this case, the attacker injects a large number of messages delaying or stopping the service. This can happen also by an overload of the network not able to dispose all packets.

- ***Masquerade.***
    - An attacker pretends to be an allowed sender or receiver getting access to safety data.

Note that some attacks such as repetition, deletion and corruption can occur also due a hardware failure.

The most common countermeasures to be considered in the case of rail applications are reported in the following list, (from [3]):

- ***Sequence number of packets***
    - It is a number added to each packet or message inserted by the source and incremented at each sent message. Of course, transmitter and recipient should agree the first sequence number, the allowed interval and the incrementation mode.

- ***Time stamp***
    - Similarly to the previous field, the value of the time has been added to the message header, allowing the recipient to know the timeliness of the received message. Sometimes, only the last significant values of the local clock can be inserted in the message due to the narrowness of the available bits. This information can be exploit jointly with the sequence number or alone. Nevertheless, it may be difficult to manage since the clocks in the elements of the network may significantly differ reducing its effectiveness.

- ***Time out***
    - It is defined as the maximum amount of time within two consecutive messages whose exceeding an error should be assumed in the communication between transmitter and recipient. There are two main cases. In the first one, the interval between two consecutive messages is measured at the recipient. Then, if it exceeds the agreed threshold the communication is considered with error. In the second case, the source transmits a message and it waits for its acknowledgment. In case, it arrives exceeding the maximum agreed time interval (thus exceeding the threshold) the communication is considered in timeout and should be resumed.

- ***Source and destination identifiers***
    - The header of the message can be provided by the identification of the transmitter or of the recipient or both in order to be sure to provide the message content to correct (and safety) machine process.

- ***Feedback message***
    - The use of feedback messages such as acknowledgements or messages for handshakes during initial phase sent by the recipient to the transmitter may improve the safety process or simply providing a confirmation to have correctly received a message. The missing of this type of messages can enable the transmitter to take proper actions or countermeasures.

- ***Identification procedure***
    - It means defining (or having) a procedure that allows to know that whoever you are communicating with is actually who you claim to be; the procedure can concern the sender/recipient identifiers or provide also that the behavior is that it is expected.

- ***Safety code***
  - Due to transmission errors, a safety code under the control of the safety-related process is required additionally to detect message corruption (i.e., its integrity), caused by air interface of by hardware failure or other external influences.

- ***Cryptographic techniques***
  - The adoption of cryptographic codes can be used to provide confidentiality or integrity or both.

Starting from the above classification of the main threats in a rail environment in [3] it is clearly outlined that a number of mechanisms should be implemented in the transmission protocol in order to counter-act the previous threats. One protocol can implement one or more of these mechanisms that are listed in the following list.

In [3] a specific matrix related to the techniques to provide defenses against the threats envisaged is reproduced and reported in Table 4.

Table 4: Matrix to report defending technique (in blue) to counteract the security threats (in red) [3].

	Defenses							
Threats	**Sequence number**	**Time stamp**	**Time-out**	**Source and destination identifiers**	**Feedback message**	**Identification procedure**	**Safety code**	**Cryptography techniques**
Repetition	X	X						
Deletion	X							
Insertion	X			X	X	X		
Re-sequencing	X	X						
Corruption							X	X
Delay		X	X					
Masquerade					X	X		X

7.2 **Analysis of transport protocols to counteract the security threats**

Starting from the indications reported in previous Section, in the following we analyze the considered transport and application protocols from the security point of view. In particular, by extending the concepts in [3] we evaluate if the considered transport protocols can counteract one, more or all of the security threats indicated in Table 4. The considered transport protocols are: TCP, SCTP with and without TLS, the UDP or better its "QUIC" version.

The analysis of the security aspects of the considered protocols start with a short review of the main features of the considered protocols. In particular, we analyze the fields inside their header and we summarize the procedures that are important from the point of view of security. Our main findings are then summarized in a final Table.

TCP – short analysis

Before starting with the TCP analysis, it should be noted that for security purposes the selection of congestion control algorithm is not important. Thus, the following discussion includes both TCP-BBR and TCP-cubic version, as the header format is the same.

Based on the header format in RFC 793 [11], RFC 5681 [12], RFC 8312 [13], TCP has the following fields:

a. sequence number,
b. acknowledgment number,
c. source and destination ports (for the identification of the communicating processes),
d. checkSum field.

The CheckSum of the TCP is calculated by taking into account for the content of the TCP Header, the TCP body and Pseudo IP header. In order to avoid any possible crosslayer (i.e., use at TCP layer of data only available at IP layer) the fields of the Pseudo IP header (12 bytes) are the IP address of the source (4 bytes), the IP address of the destination (4 bytes), the TCP segment length (2 bytes), the transport protocol (stating the type of the protocol used, thus TCP in this case) (1 byte) and a fixed field of 1 byte. This allows the TCP to avoid any mis-addressing of the segments. Similarly, the Checksum is even calculated for UDP.

TLS secure layer – short recap

As specified in RFC 5246 [14] and RFC 8446 [15], TLS is a protocol layer aiming of providing a secure channel between two communicating peers. TLS provides:

1. authentication,
2. confidentiality and
3. integrity

This is achieved through two main components: the handshake protocol and the record protocol. The only requirement from the underlying (non-secure) transport protocol is a reliable, in-order data stream delivery. It means that TLS can be applied above the TCP and SCTP but not above the UDP. Moreover, it is integrated by default on the QUIC protocol to provide these functionalities to UDP transport layer.

The handshake protocol is used to authenticate the communicating entities, to negotiate cryptographic modes and parameters, and to establish shared keying material. After parameters and keys have been agreed through the handshake protocol, the record protocol protects the exchanged traffic between the two communicating entities. Two 64-bit sequence numbers are used for each record sent or received and incremented by one periodically. The client is able to know the identity of the server whom is connecting to and authenticate it, through the TLS handshake. Optionally, the server can know and authenticate the client identity.

In order to simplify the handshake procedure, TLS defined two versions: 0-RTT and 1-RTT. The second one is the handshake adopted for the first time. Then, the client is unable to send protected application data until it has agreed all parameters and keying materials sent by the server. 0-RTT is used for a session resume, where some parameters can be retrieved by the previous connection, such as the last used key. Anyway, in this case (i.e., with 0-RTT handshake) a simpler replay attack can be applied. Finally, as described in RFC6962 [16] [17], a set of extension values for the CertificateEntry can be defined in the "extension" format, thus for example providing a mechanism for a server to send a Signed Certificate Timestamp (SCT). In TLS, certificates may expire or may be no longer valid.

SCTP protocol

Based on RFC 3286 [18], RFC 4960 [19], RFC 9260 [20] the SCTP implements the following functionalities.

a. The multi-streaming function provided by SCTP allows to partition data into multiple streams. Each payload DATA namely "chunk" uses a Transmission Sequence Number for the transmission of messages and the detection of message loss. In addition, it uses the Stream ID/Stream Sequence Number pair to determine the sequence of delivery of received data.
b. The received SCTP data "chunks" are acknowledged according to the TCP's Selective ACK procedure, in order to provide notification of duplicated or missing data chunks.

SCTP implements the flow control and the congestion control based on the TCP functionalities in RFC 2581 and RFC 5681 [21], [22]. Nevertheless, the application can specify a lifetime for data to be transmitted in order to properly manage messages that are time-sensitive. Then, the lifetime has expired and the data has not yet been transmitted, it can be discarded for timeout.

According to RFC 9260 [20] the SCTP format includes the following fields:

- Source and Destination Port Numbers (2 bytes, each), that are used by the receiver in combination with the source/destination IP addresses to identify the association to which this packet belongs and to properly forward the packet to the correct application.
- Verification Tag (4 bytes), that is used by the receiver to validate the sender of this packet.
- 32-bit checksum based on Cyclic Redundancy Check (CRC) for protecting SCTP packets against bit errors and mis-delivery of packets.

Finally, the STCP employs one mechanism to improve security, with respect to TCP and UDP. It is based on cookies exchanged during the initialization to provide protection against synchronization attacks. The synchronization attack is implemented in the connection establishment phase. An attacker sends a massive number of SYN requests to a server in order to establish connections, often using fake IP addresses. The server responds to the requests with a SYN-ACK message and by assigning a port for the connection. Since the client IP are fakes, the third message in the establishing phase is not received, blocking after a bit the server. In SCTP this is overcome by enabling the server to send a cookie (INIT-ACK) in response to the connection establishment request (INIT message).

The cookie is a text file with small blocks of data, containing a hashing (i.e., a Message Authentication Code, MAC) and other information to establish the connection such as a time stamp and the life span of the state cookie. In this case the server does not allocate any resource (e.g., port, memory) since it receives a valid answer to the INIT-ACK, namely COOKIE ECHO, containing the requested information in the cookie parameters from the previous INIT ACK. Then, the resource allocation is delayed during association setup until the client's identity can be verified using a cookie exchange mechanism, thus reducing also the possibility of Denial-of-Service attacks.

UDP protocol

As specified in RFC 768 [23], UDP is very simple transport protocol. Its header format is very simple and provides the following functionalities:

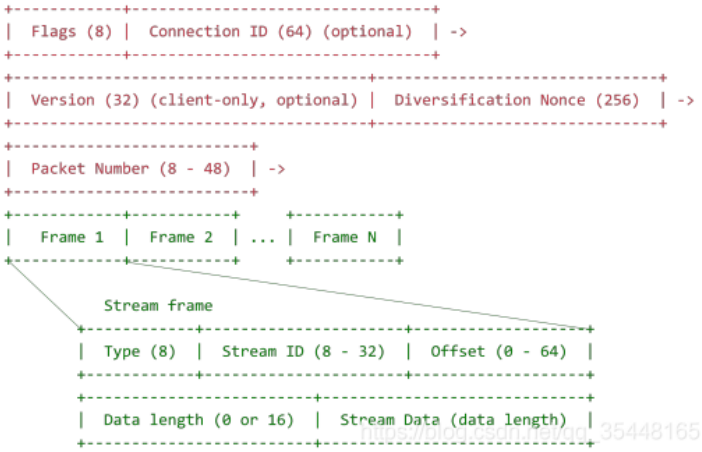
- Source and destination ports to identify the source and recipient applications, respectively,
- The length of the datagram, including header and payload
- The CheckSum, which implements the same TCP functionalities thus providing mis-delivery of datagrams.

For these reasons, UDP is not considered alone for the purpose of the document but only jointly with QUIC, where extra (and important) functionalities are added.

QUIC protocol

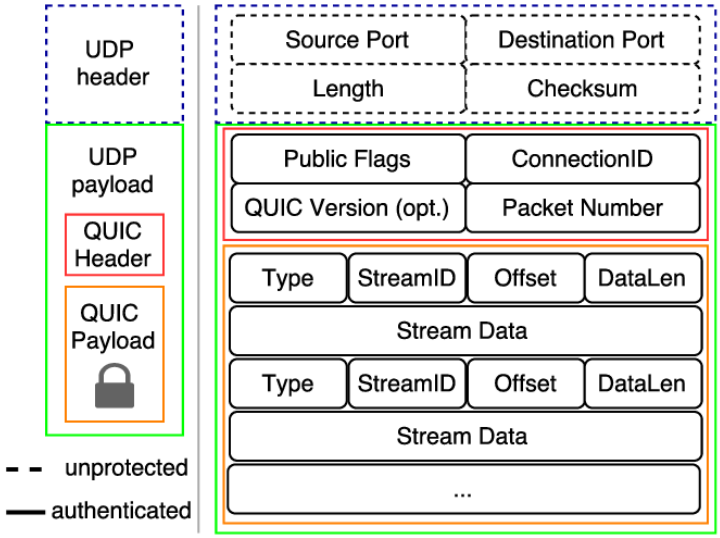
The QUIC protocol is described in RFC 9000 [7], RFC 9001 [8], RFC 9002 [24]. QUIC established a connection between a client and server through a handshake that negotiates the TSL parameters, thus guaranteeing authentication, confidentiality and integrity. When the handshake is properly performed, a connection is established (and a connection ID is available). Data are exchanged using several ordered byte-based streams. Application data are segmented into frames and multiplied in a single or more than one parallel streams, each one identified by a stream ID. Transmitted packets have a flexible structure and can aggregate frames belonging to different streams reducing the head-line-of-blocking. An example of the packet organization is reported in Figure 35.

Figure 35: Principle of the header format of QUIC taken from [7], [25]



QUIC adopts UDP as transport protocol and as said before it adopts the TLS to provide a secure connection between endpoints. In Figure 36 it is reported the encapsulation of the QUIC packet, which is formed by a header and some frames, into the UDP segment.

Figure 36: Example of encapsulation of QUIC packet into an UDP payload [26].



From Figure 35 and Figure 36, QUIC packet is equipped with the sequence number to provide retransmission and so to guarantee reliability. Each endpoint acknowledges all the packets it receives and processes. When a connection is established, endpoints agree the value of the parameter *max_idle_timeout*. When it is reached, the connection is closed for timing out.

During the connection establishment, the address validation for both endpoints is also guaranteed since receipt of a packet protected with agreed keys confirms that the client received the Initial packet from the server.

7.3 **Summary and findings for security in transport protocols**

In the following table we summarize the main results that have emerged by the security analysis of the transport protocols illustrated in the previous Section. For each one of the considered protocols we have indicated their ability to support the countermeasures required to counteract one, more or all of the security threats listed in the previous Section.

Table 5: secure functionalities supported by transport protocols to counteract the security threats.

	TCP	**TCP+TLS**	**SCTP**	**SCTP+TLS**	**UDP**	**QUIC**
Sequence number	Yes	Yes	Yes		No	Yes
Time stamp	No	[RFC6962] provides a mechanism for a server to send a Signed Certificate Timestamp (SCT)	No	[RFC6962] provides a mechanism for a server to send a Signed Certificate Timestamp (SCT)	No	No
Time out	Yes	Both TCP connection and TLS certificates may expire	Yes	Both SCTP connection and TLS certificates may expire	No	Yes, after max_idle_timeout the QUIC connection is silenced
Source and destination identifiers	Yes	It uses the ID ports of the TCP	Yes	It uses the ID ports of the SCTP	Yes	Yes, it uses tuple of IP version, IP address and UDP port number that represents one end of a network path.
Feedback message	Yes, ACKs are required	Yes, TLS uses feedback messages of TCP	Yes	Yes, TLS uses feedback messages of SCTP	No	Yes, endpoints acknowledge all packets they receive and process.
Identification procedure	No	Yes, by providing authentication. After completing the TLS handshake, the client will have learned and authenticated an identity for the server, and the server is optionally able to learn and authenticate an identity for the	No, but further protection mechanisms are implemented during the initialization phase.	Yes, by providing authentication. After completing the TLS handshake, the client will have learned and authenticated an identity for the server, and the server is optionally able to learn and authenticate an identity for the client	No	It uses TLS functionalities. Moreover, address validation for both endpoints during the handshake phase.

		client				
Safety code	Only on CheckSum field	TLS uses the CheckSum of TCP	Validation tag and 4-bytes checksum		Only on CheckSum field	It uses the CheckSum field of UDP
Cryptographic techniques	No cryptography technique is implemented. Nevertheless, sequence number randomization may prevent IP spoofing	Yes, ciphering and integrity based on MAC	No			Yes, packets have confidentiality and integrity protection. Moreover, it verifies the identity using cookies exchange during association setup, reducing the possibility of DoS, MitM, masquerade

In some cases, the motivations justifying the assertion have been inserted in the Table 5.

Before concluding this section, it is noteworthy to observe that more than one event can generate the same threat. In the following Table 6 (which has been taken and copied from [3]) the hazardous events are related to the generated threat indicated in Table 4.

Table 6: Relationship between hazardous events and threats (Table A.1 in [3])

Hazardous Events	Threats						
	Repetition	Deletion	Insertion	Re-sequencing	Corruption	Delay	Masquerade
HW systematic failure	X	X	X	X	X	X	
SW systematic failure	X	X	X	X	X	X	
Cross-talk		X	X		X		
Wires breaking		X			X	X	
Antenna misalignment		X			X		
Cabling errors		X	X		X	X	
HW random failures	X	X	X	X	X	X	
HW ageing	X	X	X	X	X	X	
Use of uncalibrated instruments	X	X	X	X	X	X	
Use of unsuitable instruments	X	X	X	X	X	X	
Incorrect HW replacement	X	X	X	X	X	X	
Fading effects		X		X	X	X	
EMI		X			X		
Human mistakes	X	X	X	X	X	X	
Thermal noise		X			X		
Magnetic storm		X			X	X	
Fire		X			X	X	
Earthquake		X			X	X	
Lightning		X			X	X	
Overloading of TX system		X				X	
Wire tapping	X	X	X	X	X	X	
HW damage or breaking		X			X	X	
Unauthorized SW modifications (a)	X	X	X	X	X	X	X*
Transmission of unauthorized messages (a)	X		X				X*
* In this case the message is fraudulent from the beginning; a strong defense is needed, for example the use of a key.							

Starting from Table 6 it is possible to evaluate the robustness of the considered transport protocols to

the security threats. In particular, in Table 7 we have indicated whether one transport protocol implements countermeasure able to counteract the effect of each one of the threats (in the row in Table 7). As an example, the TCP has a sequence number in its header format (but not a timestamp) thus guaranteeing robustness against the repetition threat.

Table 7: Implemented countermeasures by transport protocols

Threats	TCP	TCP+TLS	SCTP	SCTP+TLS	UDP	QUIC
Repetition	X	X	X	X		X
Deletion	X	X	X	X		X
Insertion	X	X	X	X	X	X
Re-sequencing	X	X	X	X		X
Corruption	X	X	X	X	X	X
Delay	X	X	X	X		X
Masquerade	X	X	X	X	X	X

Intersecting the data in Table 5, Table 6 and Table 7 allows to assess if any of considered transport protocol is robust to the possible set of hazardous events. To this purpose, results of this analysis are summarized in Table 8 where we have related the set of hazardous events and the transport protocols.

Table 8: Robustness of considered transport protocols against hazardous events.

	Repetition	**Deletion**	**Insertion**	**Re-sequencing**	**Corruption**	**Delay**	**Masquerade**
HW systematic failure	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
SW systematic failure	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Cross-talk		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC		TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC		
Wires breaking		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Antenna misalignment		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC		

Cabling errors		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC		TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
HW random failures	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
HW ageing	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Use of uncalibrated instruments	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Use of unsuitable instruments	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	

Incorrect HW replacement	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Fading effects		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
EMI		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC		
Human mistakes	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Thermal noise		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC		

Magnetic storm		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Fire		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Earthquake		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Lightning		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Overloading of TX system		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC				TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	

Wire tapping	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
HW damage or breaking		TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC			TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	
Unauthorized SW modifications (a)	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC	TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC
Transmission of unauthorized messages (a)	TCP, TCP+TLS, SCTP, SCTP+TLS, QUIC		TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC				TCP, TCP+TLS, SCTP, SCTP+TLS, UDP, QUIC

7.4 Comments on security aspects of HTTP and FTP and their secure versions

HTTP

HTTP is used for communications over the Internet, in general adopted by web browsers to retrieve web pages from a remote server (https://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html).

The client uses simple messages and methods to facilitate applications to get the data from the server. Case sensitive examples are reported in Table 9, while others are in https://www.tutorialspoint.com/http/http_requests.htm.

Table 9: Examples of HTTP 1.1 methods

Method	Description
GET	To retrieve information from the given server using a given URI
POST	To send data to the server (e.g., customer information, file upload) using HTML
PUT	Replaces all the current representations of the target resource with the uploaded content
DELETE	Removes all the current representations of the target resource given by URI
CONNECT	Establishes a tunnel to the server identified by a given URI

HTTP commands are sent in plaintext and anyone monitoring the connection can read them. Then, any data sent through this protocol such as a password, a credit card number, or any other data entered into a form can be seen by others.

In addition to the sniffing of transmitted data using HTTP, other threats are:

- The possibility to access other files in the server by exploiting a sniffed URI from another request. An attacker can slightly modify the sniffed URI and retrieve other file on the same server without having the permission. A possible solution can be the disaggregation of the path name and the file.
- Another threat is based on the deliberate mis-association of the IP address and the DNS (Domain Name System) name. This attack is called DNS spoofing and can be counteract by updating the DNS information instead of using those stored in the local cache.
- Similarly to the previous one, the same request can be valid for more than one servers in case an organization has multiple locations. Then, one server can be accessed instead of the one contact by the request. The problem in this case can be that resources can be overwritten in the wrong server.
- When a client accesses a server with credentials for its authentication, the browser can store them indefinitely. Time outs and expiring passwords should be used to mitigate this threat.

- In addition to remote servers, data can be stored in proxy servers and local caches to improve the time responsiveness. Then, they can store both sensitive user data and organization data as they behave as a men-int-the-middle in the communication, then being appealing by attackers. Further protection should be implemented by the proxy and cache providers. Anyway, it is not usual in the rail application to have a proxy.

The inclusion of the TLS protocol layer above the transport protocol allow to encrypt the payload and then to avoid many of the threats listed in the previous points.

FTP

Similarly to HTTP, FTP has a client-server architecture and it is used by applications to easily transfer files between computers over the Internet. Commands are simple and, even in this case, are transmitted in plain text. Examples of command lines are [27]:

Table 10: Examples of FTP commands

Command	Description
RETR	Get file from the remote computer
STOR	Accept data and store as a file
RNFR, RNTO	Renames a file
QUIT	Exits from FTP
DELE	Deletes a file

A list of other FTP commands are available in [27]. Other FTP commands for Windows server are available at: https://www.serv-u.com/ftp-server-windows/commands, while for IBM systems at https://www.ibm.com/docs/en/aix/7.2?topic=f-ftp-command.

FTP is considered a non-secure protocol because it relies on clear-text usernames and passwords for authentication and the data transferred is not encrypted (https://www.integrate.io/blog/5-tips-on-avoiding-ftp-security-issues/). It makes FTP vulnerable to malicious techniques such as packet sniffing, spoofing attacks, and brute force attacks.

Possible solutions are reported in the following.

- The adoption of a transport layer which implements a security layer is the most popular solution as they create an encrypted connection between the client and the server. Similarly to HTTP, the usage of TCP and TLS enable to create the FTPS and then to solve many of the problems related to the threats similar to that of HHTP.

- Instead of using TLS, other solutions are available. One of them is the Secure Socket Layer (SSL), and SSH File Transfer Protocol (also known as Secure File Transfer Protocol or SFTP); [28]

- Another solution exploits the security functionalities provided at application level by common cloud storage services such as Google Cloud Storage or Microsoft OneDrive. Nevertheless, in this case rail applications should be deployed in external and public clouds, that in some cases could not be accepted by rail operators;

- In the transmission the file could be encrypted its self, overcoming the plain transmission over unsecure channel. Then, the file should be encrypted at the server and decrypted at the client. Some delays con be increased;

- Finally, an IP blacklist can be considered to reduce the malicious nodes to eavesdrop the FTP file or a whitelist to restrict the access to the server, containing the FTP file.

7.5 **Conclusions on security analysis**

Concerning the transport protocols, we noted that the usage of the most popular IETF transport protocols and the corresponding secured versions provide a good robustness to many or all of the threats listed in Table 7. The only protocol presenting major problems is UDP where simplicity is obtained at the expense of reliability and security. TCP and SCTP allows to secure transmissions against threats. In addition, when TLS is used, both in conjunction with TCP and SCTP or in an integrated manner in the QUIC, all the security problems are solved,

Concerning the application protocols stacked above the transport layer, HTTP and FTP do not implement any security mechanism. Generally, security issues at this level are solved by providing a security layer below them. To this purpose the addition of TLS above the transport protocol such as TCP or the usage of a transport protocol already embedding TLS such as QUIC, allows to create a secure version of the two application protocols i.e., the HTTPS and the FTPS. Theoretically, in same specific cases, security layers or algorithms could be added even at application level (i.e., above the HTTP and FTP). Examples are those of the banks in case of their clients have to access to their bank account and make economic transactions. These cases are very specific and usually require to have a dedicated user equipment.

8. Conclusions

This deliverable responds to the objective a in workstream 2, and it is devoted to identification of the appropriate transport protocols ensuring the required communication and characteristics capabilities in the application development stage as well as of security aspects.

To assess the performance of the secure version of the transport and of some combinations of application and transport protocols we have used the software emulator developed in the Task 3.3 which allows to reproduce the behavior of the communication bearers at IP protocol level. Then, in this case we used it to evaluate the performance of the transport protocols: TCP cubic, TCP BBR, UDP and SCTP and the recent protocol QUIC.

Performances have been evaluated in terms of the statistics (i.e., cumulative distribution function, mean, standard deviation etc.) of:

1. The achievable throughput,
2. The download time,
3. The message delay.

Analysis has been carried out in mainline railway scenario in different operating conditions i.e., variable latency, time variability of the available transmission capacity and packet loss.

In Table 11 we summarize and comment the secure versions of the considered transport protocol(s) to be selected for each ACS application class. From results presented in previous Sections we observe that transport protocol performance are practically independent from the selected railway scenario (e.g., mainline and regional).

Table 11: Summary of transport and application protocols for ACS application classes.

ACS Application class	Transport/Application protocol	Note
Signaling	SCTP+TLS, TCP BBR+TLS, TCP Cubic+TLS and QUIC	They similar performance in case of low latency. TCP BBR, TCP Cubic and QUIC present a reduced message delay for higher latency introduced by the channel. Nevertheless, as shown in Del. 3.4, in the case the packets are enqueued (we have an additional delay due to queue) SCTP experiences lower latency (even though throughput is also reduced).
Critical Voice	UDP(*), TCP BBR, TCP-BBR+TLS	VoIP based voice services use TCP to establish an initial connection and for signaling. In this case TCP+BBR could be a

		good choice. (*)UDP is the default (and mandatory) choice for transferring voice data when VoIP is (obviously) considered. It could be of some interest to test the usage of TCP with BBR for voice data transfer in low PL scenarios. However, this choice would mean to abandon the VoIP standard. TCP+TLS could also be used for the initial connection (for securing signaling data) transporting signaling as well as for transporting SIP packets; in fact, TCP-BBR +TLS provides lower delay in all the considered conditions.
Critical Data	TCP BBR, QUIC	They experience higher throughput above all in lossy communication channel
	HTTP, HTTP/3, FTP HTTPS, FTPS	In the case of low requested latency, sending in clear text can be provided (through the http and FTP protocols)
Critical Video	UDP*, QUIC, TCP BBR	*Even though in high lossy environments UDP should be avoided, typically for video data transmissions UDP; however, UDP is not used alone; in fact, the typical protocol used for video transmissions includes RTP/RTPC protocols. TCP BBR could be used because is less sensitive to loss in terms of latency mainly for signaling such as RTPC (when UDP is not used even for RTPC) and especially for SIP protocol (if and when it is used for). The possibility of stacking RTP/RTPC over QUIC is currently under study and seems to be a promising solution. At the moment of this writing only preliminary proposal exist and no software implementations of QUIC+RTP/RTPC are available.
Non-critical Data	TCP BBR, TCP cubic	Most of the traffic uses TCP and both TCP variants could be used provided coexistence issues are taken into account.
	HTTP, HTTP/3, FTP HTTPS, FTPS	In the case of low requested latency, sending in clear text can be provided (through the http and FTP protocols)
Internet Connectivity	QUIC, TCP BBR, TCP cubic	Most of the traffic uses TCP. UDP is considered for some applications protocols that use it as default transport protocol.
	HTTP, HTTP/3, FTP HTTPS, FTPS	Depending on the service types and their particular applications

In case of lossy channel, results show that in every scenario the TCP BBR and QUIC protocols offer the better performance since they are:

a. able to track the available transmission channel capacity which in a rail scenario can vary with time,
b. resistance (i.e., practical insensitivity) to packet loss; in fact, we have observed that in all cases since the CDF of the TH at PL=1% are similar to that obtained at PL=0%.

The QUIC protocol is of great interest since it provides robust performance in many (all) the considered scenarios. However, at the moment of this writing it suffers of many drawbacks that can limit its diffusion especially in the ACS environment. QUIC is currently under standardization and revision. Actually at least two versions exist of QUIC (i.e., the Google version and the IETF version). Some implementations of QUIC are available on the open-source community and the LS-QUIC is suggested for being used in a production environment. In addition, QUIC is not integrated in the kernel of the most important OSs such as Linux and many times the existing servers do not implement or support QUIC.

In the second part of this deliverable, to be compliant with Task 3.6 activities, we have analyzed security aspects of the considered transport and application protocols. We have observed that the robustness against the several threats listed in Table 5 of the considered application protocols is mainly related to the characteristics of the underlying transport protocol. Obviously, additional security mechanisms could be added at application level i.e., above the application protocol.
Even in this case TCP and QUIC implement all the security mechanisms able to successfully counteract many of the threats indicated in Table and, in the QUIC case, also to guarantee security. The addition of the TLS layer to TCP allows to guarantee security against the considered threats.

To summarize the main findings in the WP3 activities we should start by observing that ACS system has been conceived, designed and build around the IETF protocols. For this reason, we believe that railway developers of ACS should orient their choice on well accepted IETF transport protocols such as TCP and also UDP for very limited secondary applications (refer to Table 11indicating the ACS traffic classes). In particular from the analysis carried out in the AB4Rail WP3 activities it has been clearly emerged that BBR congestion control should be the preferred choice so to counteract packet loss effects and to guarantee the maximum achievable throughput. However, as indicated in Del.3.4 [1] the BBR could suffer of coexistence problems with other TCP links using different congestion control strategies such as Cubic. For this reason, it is important that the on-board ACS-GW as well as the ACS-GW at the trackside guarantee that all the TCP connections sharing the same ACS tunnel use the same congestion control algorithm so to avoid un-desired capacity starving phenomena.

9. References

[1] R. Giuliano, F. Mazzenga, A. Vizzarri, "Identification of transport protocol for railway applications", AB4Rail project, Deliverable 3.4, 29 Apr. 2022

[2] AB4Rail "ALTERNATIVE BEARERS FOR RAILWAY", SECTIONS 1-3, technical proposal annex, GA 101014517, Nov. 2020, ID: S2R-OC-IP2-02-2020.

[3] "Railway applications – Communication, signalling and processing systems – Safety-related communication in transmission systems", European Standard EN 50159, 1 Sep. 2010.

[4] A. Vizzarri, F. Vatalaro, F. Mazzenga, R. Giuliano, "IP Emulator and scenarios definition", AB4Rail project, Deliverable 3.3, 6 Nov 2021.

[5] A. Vizzarri, F. Vatalaro, F. Mazzenga, R. Giuliano, "Review of ACS, of existing transport protocols, application protocols, railway applications", AB4Rail project, Deliverable 3.1, 25 May 2021.

[6] "What is QUIC?" 30 Sep. 2020, https://nordvpn.com/it/blog/what-is-quic-protocol/

[7] M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, ISSN: 2070-1721, May 2021, https://datatracker.ietf.org/doc/html/rfc9000, https://www.rfc-editor.org/rfc/rfc9000#name-error-handling, https://www.rfc-editor.org/rfc/rfc9000#name-security-considerations

[8] Thomson, M., and S. Turner, "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>, https://quicwg.org/base-drafts/rfc9001.html#aead, https://quicwg.org/base-drafts/rfc9001.html

[9] "QUIC and HTTP/3 Library", https://www.litespeedtech.com/quic-http3-library

[10] Belma Turkovic, Fernando A. Kuipers and Steve Uhlig, "Fifty Shades of Congestion Control: A Performance and Interactions Evaluation", Computer Science, 9 March 2019.

[11] "Transmission Control Protocol", RFC 793, https://datatracker.ietf.org/doc/html/rfc793

[12] "TCP Congestion Control", RFC 5681, https://datatracker.ietf.org/doc/html/rfc5681

[13] "CUBIC for Fast Long-Distance Networks", RFC 8312, https://datatracker.ietf.org/doc/html/rfc8312

[14] "The Transport Layer Security (TLS) Protocol, Version 1.2", RFC 5246, https://datatracker.ietf.org/doc/html/rfc5246

[15] "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, https://datatracker.ietf.org/doc/html/rfc8446

[16] "Certificate Transparency", RFC 6962, https://datatracker.ietf.org/doc/html/rfc6962

[17] Chen, S., Jero, S., Jagielski, M. et al. "Secure Communication Channel Establishment: TLS 1.3 (over TCP Fast Open) versus QUIC", J Cryptol 34, 26 (2021). https://doi.org/10.1007/s00145-021-09389-w

[18] "An Introduction to the Stream Control Transmission Protocol (SCTP)", RFC 3286, https://datatracker.ietf.org/doc/html/rfc3286

[19] "Stream Control Transmission Protocol", RFC 4960, https://datatracker.ietf.org/doc/html/rfc4960

[20] "Stream Control Transmission Protocol", RFC 9260, https://datatracker.ietf.org/doc/rfc9260/

[21] "TCP Congestion Control", RFC 2581, https://datatracker.ietf.org/doc/html/rfc2581

[22] "TCP Congestion Control", RFC 5681, https://datatracker.ietf.org/doc/html/rfc5681

[23] "User Datagram Protocol", RFC 768, https://datatracker.ietf.org/doc/html/rfc768

[24] "QUIC Loss Detection and Congestion Control", RFC 9002, https://datatracker.ietf.org/doc/html/rfc9002

[25] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., "The quic transport protocol: Design and internet-scale deployment," in Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 183–196, ACM, 2017.

[26] Sharma, A., Kamthania, D. (2022). QUIC Protocol Based Monitoring Probes for Network Devices Monitor and Alerts. In: Singh, U., Abraham, A., Kaklauskas, A., Hong, TP. (eds) Smart Sensor Networks. Studies in Big Data, vol 92. Springer, Cham. https://doi.org/10.1007/978-3-030-77214-7_6

[27] "File Transfer Protocol (FTP)", RFC 959, https://www.rfc-editor.org/rfc/rfc959

[28] "An overview of the SSL or TLS handshake", IBM document, 30 Jun. 2022, https://www.ibm.com/docs/en/ibm-mq/7.5?topic=ssl-overview-tls-handshake