ALMA MATER STUDIORUM – UNIVERSITY OF BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

Department of Computer Science and Engineering

Master's degree in Computer Engineering

MASTER'S THESIS

in

Protocols And Architectures For Space Networks M

Performance evaluation of Licklider Transmission Protocol over Free Space Optical communication testbeds

Candidate:
Antony Zappacosta

Supervisor:
Prof. **Carlo Caini**
Co-supervisor:
Dr. **Tomaso De Cola**

Academic Year 2021/2022

Abstract

Optical communication systems can reach transmission data rates of several Gbps but in near-Earth scenarios they are also affected by link intermittency and disruption, so that the use of the Delay-/Disruption-Tolerant Networking architecture and related protocols, such as the Licklider Transmission Protocol (LTP), seems particularly promising. For this reason, the German Aerospace Center (DLR), in accordance with Unibo, decided to devote this thesis to this topic.

First, the Institute of Communication and Navigation of DLR provided us with tracks representing the dynamic state of an FSO channel between a Low Earth Orbit satellite and a ground station. Starting from these, we applied a Reed Solomon (255, 223) error correcting code and further processing to obtain "erasure vectors" (EV) representing the binary state (on/off) of the channel at a sample rate of 10 kHz.

To emulate the channel in accordance with the EV traces, we developed "detemu". Then, by using DTNperf, detemu and DTNME (the bundle protocol implementation by NASA MSFC), we performed a series of tests with LTP "red" (reliable). We developed a Python program called "LTP performance analyser" to automatically compute statistics about session durations, retransmission cycles and penalization times.

Preliminary results showed that the session duration was significantly decreased by setting the retransmission time out time to 100ms instead of 1.1s, with a corresponding very significant goodput increase for a given session parallelism. We also found that by increasing the parallelism above a threshold, depending on channel characteristics and other factors, there is no advantage in terms of goodput, even if the full channel utilization is not reached, as explained in the thesis. Summarizing, LTP proved to be extremely robust to high loss ratios and an excellent match to FSO links in near-Earth scenarios.

Table of contents

1. INTRODUCTION

1.1 CHALLENGED NETWORKS

As of 2022, there are 4.95 billion people actively using Internet; and each user, on average, spends 6 hours and 56 minutes online each day [1]. It is therefore true that this world-spread computer network holds crucial importance in supporting tons of human activities, being these related to work, social life or leisure time. However, the way Internet is designed to work on Earth is not suitable in some environments like space, extreme regions, submarine or military ambiences, etc. In the literature, these scenarios are referred to as "challenged networks", usually featuring at least one of the following challenges:

- Long propagation delays due to huge distances
- Link intermittency
- Network partitioning
- High packet error rates (PERs) due to channel impairments
- Significant transmission rate asymmetry

While the common TCP/IP architecture used on our planet is built upon several hypothesis:

- Round trip times (RTTs) are relatively small
- Existence of a continuous path between source and destination for the whole duration of the transmission
- PERs are small and mostly due to congestion
- All nodes support TCP/IP protocols

In the last decades, even more attention has been dedicated to space research and exploration: almost all countries share the belief of addressing space challenges to expand technology, create new industries and help to foster peaceful connection with other nations. As an example, Earth observation from space has demonstrated its growing scientific, social, economic and political importance by contributing to a better understanding and regular monitoring of our planet and its environment, by supporting a wide range of applications and by providing essential data for geopolitical purposes. [2]

Therefore, the strong need of collecting and exchanging data between Earth and all the devices (satellites, spacecrafts, rovers, etc.) spread on the Solar System and beyond, led the scientific community to define the Delay-/Disruption Tolerant Networking (DTN) Architecture.

1.2 DELAY-/DISRUPTION TOLERANT NETWORKING (DTN)

Since nineties, Vinton Cerf and scientists at NASA Jet Propulsion Laboratory started working on the design of an Interplanetary Internet (IPN), sharing the need of adapting TCP/IP protocols to the space environment. With time, the research group IPNRG (Interplanetary Network Research Group) realized that space channels impairments are common to other environments like extreme regions or built-up networks for disaster recovery. All together, these types of networks were referred to as "challenged networks" and the draft "Delay-/Disruption Tolerant Network Architecture: The evolving Interplanetary Internet" was published. The evolution of this document led to the publication of RFC 4838, which is the reference definition of the DTN architecture [3].

1.2.1 DTN generalities

The DTN architecture embraces the concept of occasionally-connected networks that may suffer frequent partitions, intermittent connectivity, large-delay environments and possible high PER. Furthermore, different nodes of the transmission may apply divergent set of protocols.
For these reasons, some Delay-/Disruption Tolerant Networking features are [4]:

- Variable-length (possibly long) messages (not streams or limited-sized packets)
- A naming syntax that supports wide range of naming and addressing conventions to enhance interoperability
- Using storage within the network to support store-and-forward over multiple paths
- Provide security mechanisms that protect the infrastructure from unauthorized use
- Provide coarse-grained classes of service, delivery options, and a way to express the useful lifetime of data to allow the network to better deliver data in serving the needs of applications

An application running on top of the DTN infrastructure should consequently be designed to:

- Minimize the number of round-trip exchanges
- Cope with restarts after failure while network transactions remain pending
- Inform the network of the useful life and relative importance of data to be delivered

1.2.2 Bundle Protocol

The DTN architecture relies on the insertion of a specialized layer between Application and lower layers (e.g. Transport): the Bundle Layer, regulated by its corresponding Bundle Protocol. At this level the Protocol Data Unit (PDU) is called "bundle" and the main purpose of the BP is to realize a store-(carry)-and-forward switching transfer mode: a bundle is first received, stored on a local database, sometimes carried (as part of its path to the destination can be physically accomplished by the movement of the device storing it) and then transmitted, when possible, to the next hop. Bundles may be removed from the local database once sent (always removed when the bundle lifetime expires).

Bundle Protocol has been initially specified in RFC 5050 [5] and, recently, its 7th version (BPv7) has been standardized in the new RFC 9171 [6]. With the usage of BP, Transport is no more end-to-end: this semantic holds inside the DTN hop only, with two advantages: first, it makes recovery more efficient, as the transport layer retransmissions operate on one DTN hop, i.e. with shorter RTT; second, it enables the use of a different transport protocol on each hop, which is essential to match the different characteristics of space and terrestrial links. [7]

Thanks to BP, applications become independent of the underlying transport protocol. In turn, BP can manage the heterogeneity of these lower layer protocols harnessing an additional thin layer inserted between BP and Transport: the Convergence Layer Adapter (CLA), which provides an abstract interface to specific transport protocols [8]. As said, different transport protocols can be used in different DTN hops: this feature is crucial in space systems where TCP cannot be employed over interplanetary legs and it is replaced by the Licklider Transmission Protocol (LTP), which is the main subject of this thesis work.

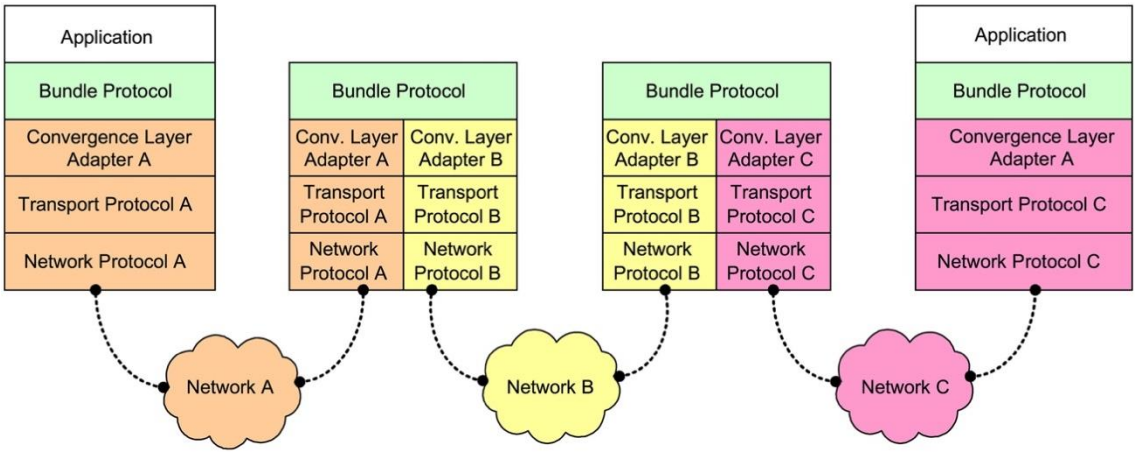

***Figure 1**:example of interconnected DTN nodes*

One or more DTN nodes are uniquely identified by an Endpoint Identifier (EID), which serves like an address expressed in the Uniform Resource Identifier (URI) standard: the first part determines the scheme and the second one to the specific endpoint. The only used schemes in the DTN domain are the "ipn" and the "dtn" ones (examples are "ipn:103.2000" and "dtn://gereleo.dtn"). In this thesis' activities both schemes are used (depending on the BP implementation being used).

1.2.3 DTN contacts

In a DTN-based space network, connection between nodes may routinely appear and disappear, and there may never be continuous connectivity from a bundle's source all the way through to its destination [9]. As an example, a Moon's orbiting satellite collecting data (e.g. pictures) of the non-visible side of the celestial body must wait until it moves in a connection-available position with an Earth's ground station to deliver its sampled data.

The time interval in which one DTN node can transmit data at a given nominal speed to a neighbour node is called “contact”. Note that contacts are unidirectional in space, due to possible long delays and rate asymmetry.

DTN contacts are classified as “scheduled” or “opportunistic”.

Scheduled contacts are pre-arranged opportunities of data transmission between two or more nodes. They are well-planned in advance and are based on known patterns of DTN nodes mobility. Consequently, each DTN node is aware of the time instants at which contacts from it to other nodes are available, as well as the nominal transmission speed (data rate) for each contact. This knowledge is usually available at each DTN node, so that this information can be exploited by routing algorithms, such as CGR/SABR to find the best route to destination (i.e. the best sequences of contacts to get to the destination node in the minimum time) [SABR CCSDS blue book] [10] [11].

As an example, considering the following network topology of three nodes:

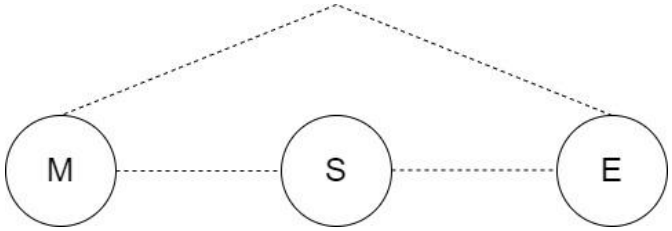

Figure 2:example of contacts between DTN three nodes

The purpose of M is to deliver 200 Mbytes of data to E and every node of the network owns the following information:

Table I

Contact	Sender	Receiver	From time (s)	Until Time (s)	Rate (kbps)
1	M	S	500	700	2000
2	S	E	800	1000	2000
3	M	E	1100	1900	2000

Therefore, there are two routing solutions to achieve the transmission from M to E:

- First transfer 50Mbyte from M to S using contact #1; then S forwards these data to E using contact #2; finally, M sends the remaining 150 Mbytes to E using contact #3
- M transmits 200 Mbytes to E using contact #3

Although both the options achieve the transmission of all data from M to E, DTN routing is aimed at finding the sequence of contacts resulting in the earliest final arrival time, regardless

of the number of contacts which have been employed. Accordingly, the first solution would be used.

On the other hand, **opportunistic contacts** are unplanned opportunities of data transmissions between two or more nodes. These contacts are typically shorter and may occur due to chance encounters.

Given the scheduled nature of Licklider Transmission Protocol, only scheduled contacts are considered in this thesis work.

1.2.4 Bundle Protocol implementations

The main open-source implementations of BP are:

- ION (Interplanetary Overlay Network): ION is a Bundle Protocol implementation written in C language by NASA Jet Propulsion Laboratory (JPL), and specifically designed for space environments (e.g. to run on board of robotic spacecrafts). The bundle storage is based on the Simple Data Recorder (SDR), which is a mechanism already running in spacecrafts: it can be configured to store data on disk, in memory or to use both. A relevant feature is that SDR supports a transaction mechanism that ensures database (DB) integrity in case of failing DB operations.
  Being mainly modelled for space and interplanetary communications, it is designed to optimize transmission bandwidth, storage and CPU utilization. Furthermore, a great attention is given to maintenance: a node could be extremely far from Earth, therefore is severely important to reduce risks of fault.
  In this thesis ION version 4.1.2 has been used in some tests [12]
- DTN2/DTNME: DTN2 is the reference implementation of Bundle Protocol, originally developed by Intel and now managed by NASA Marshall Space Flight Centre (MSFC). It is written in C++ language and its use-cases are not restricted to space applications. DTN2 features two different built-in modules for bundle storage: a memory-based storage and a disk-based storage relying on the Berkeley DB library.
  In the last years DTN2 has been revised and extended in some of its classes, and new features, commands and capabilities have been added to the original project. The resulting implementation has been renamed DTNME (DTN Marshall Enterprise Implementation). Like DTN2, DTNME is fully interoperable with ION. [13]
  DTNME version 1.2.0 is the BP implementation generally used in this thesis. [14]
- IBR-DTN: IBR-DTN is a C++ implementation of the Bundle Protocol specifically designed for embedded systems. It is developed by the University of Braunshweig and it is particularly suitable for being used on mobile technologies and network sensors. Like ION, it allows bundles to be stored on disk, RAM or on both of them [13].
  Due to its main applicative target, IBR-DTN has not been used in this thesis work.

1.2.5 Unified API

Unified API [15]is a library developed and maintained by the University of Bologna which abstracts the specific underlying BP implementation APIs. By decoupling DTN applications from specific BP APIs, it makes possible to run these applications on top of whichever BP

implementation, which is particularly interesting when dealing with multiple implementations, as in this thesis. Moreover, the Unified API directly tackles some of the most challenging BP aspects, such as parameter parsing or registration under either the ipn or dtn scheme, thus simplifying the development of DTN applications. [16]

This library is part of the DTNsuite [17], which consists of a set of DTN applications based on the Unified-API, among which there is DTNperf, widely used in tests.

1.2.6 DTNperf

DTNperf is a software developed at the University of Bologna aimed at assessing DTN performance. It is conceived to be a sort of DTN equivalent of the Iperf tool, widely used to test TCP and UDP performance. [18]

From version 3, DTNperf provides three operating modes: client, server and monitor. The client generates and sends bundles, the server receives them, and the monitor collects bundle status reports, which are informative bundles generated by the BP itself and sent to the "report-to" EID [3].

The client of DTNperf is able to send bundles in three different ways:

- Data mode: the client sends as many bundles of a specified payload size as the amount of fixed data to be sent; in this case the payload content of the bundles is meaningless
- Time mode: the client sends as many bundles of a specified payload size for a fixed number of seconds; in this case the payload content of the bundles is meaningless
- File mode: the client sends a file to the server in one or more bundles of fixed dimension; in this case the payload content of the bundles coincides with the segments of the file being sent

Two alternative congestion control policies are possible:

- Window-based: a window specifies the maximum number of bundles in flight (each sent bundle must be confirmed by the server through a bundle ACK)
- Rate-based: the amount of transmitted data per second must match a user-specified rate. This is particularly useful when the client must steadily generate traffic on the channel, independently from the delivery outcome of previous sent bundles

DTNperf is currently at version 4, being re-engineered to enjoy a modular and more-extendible code, as well as the latest features included in Unified API. [19] [20]

1.3 LICKLIDER TRANSMISSION PROTOCOL (LTP)

Transport protocols usually engaged in TCP/IP do not suit challenged networks requirements. For example, TCP and UDP cannot be employed in transmissions between

Earth and Jupiter's moon Europa, of which RTTs run between 66 and 100 minutes. Consequently, since 2008, the Licklider Transmission Protocol (name in honour of Joseph Licklider) standardization begun and, over the years, it became the convergence layer of choice when dealing with space links. The major characteristics of this protocol are [21] [22] [23]:

- Unidirectional data flow to cope with possible channel asymmetry (the reverse channel is used for signaling only)
- It offers both reliable and unreliable service: a "red" service intended to be reliable (acknowledged transmission) and a "green" service intended to be unreliable (non-acknowledged) transmission
- Lowest "chattiness" level: no connection-establishment phase (by contrast to TCP 3-way handshake) avoiding inconvenient delays (due to high RTTs) and link underutilisation
- Energy efficiency, as it only transmits if a link to the next hop is available
- Rate based transmission speed (not based on feedback)
- Bundles passed by BP are encapsulated in LTP "blocks", to be transmitted by independent LTP "sessions" running in parallel to fill the Bandwidth-Delay Product
- An LTP block is split into a number of LTP "segments", each passed to the underlying protocol
- In contrast to TCP, LTP acknowledgments ("report segments") are only triggered by data segments flagged as "checkpoints"

From now on, the following definitions are considered [22]:

- Block: an array of contiguous bytes of application data handed down by the upper layer protocol (typically Bundle Protocol) to be transmitted from one LTP client service instance to another
- Red-part: part of the block to be transmitted reliably, i.e. subject to acknowledgment and retransmission
- Green-part: part of the block to be transmitted unreliably, i.e. not subject to acknowledgments or retransmissions
- Session: a thread of LTP protocol activity conducted between two peer engines for the purpose of transmitting a block. Data flow in a session is unidirectional; data traffic flows from the sending peer to the receiving peer, while data-acknowledgment traffic flows from receiving peer to the sending peer. In other words, the process of transferring a block
- Segment: the unit of LTP data transmission activity. It is the data structure transmitted from one LTP engine to another in the course of a session. Each LTP segment is one of the following types: data segment, report segment, report-acknowledgment segment, cancel segment, cancel-acknowledgment segment
- Reception claim: an assertion of reception of some number of contiguous bytes of application data characterized by the offset of the first received byte and the number of consecutive contiguous octets received (starting the offset)

- End of Block (EOB): the last data segment transmitted as part of the original transmission of a block. This data segment also indicates that the segment's upper bound is the total length of the block (in bytes)
- End of Red-Part (EORP): the segment transmitted as part of the original transmission of a block containing the last byte of the block's red-part. This data segment also indicates that the segment's upper bound is the length of the block's red-part (in bytes)
- Checkpoint (CP): a data segment soliciting a reception report from the receiving LTP engine. The EORP segment must be flagged as a checkpoint, as must the last segment of any retransmission; these are "mandatory checkpoints". All other checkpoints are "discretionary checkpoints"

Each LTP segment comprises a header, zero or more octets of content and zero or more octets of trailer. An LTP segment header comprises three data items: a single-octet control byte, the session ID, and the Extensions field. The session ID comprises the session originator (EID of the sender) and the session number (which is suggested to be sequential [24]) to uniquely identify, among all transmissions between the sender and the receiver, the session of which the segment is one token.

LTP segments are of four general types depending on the nature of the content:

- Data segments flow from the sender to the receiver and carry client service (application) data
- A report segment (RS) flows from the receiver to the sender and carries data reception claims together with the upper and lower bounds of the block scope to which the claims pertain. It also includes two serial numbers to support efficient retransmission
- A report-acknowledgment (RA) segment flows from the sender to the receiver and acknowledges reception of a report segment. It carries the serial number of the report being acknowledged
- Session management segments may be generated by both the sender and the receiver and are of two general sub-types: cancellation and cancellation-acknowledgment. A cancellation segment initiates session cancellation procedure at the peer and carries a single byte reason-code to indicate the reason for session cancellation. Cancellation-acknowledgment (CA) segments merely acknowledge reception of a cancellation segment and have no content.

1.3.1 LTP monochrome blocks

As per [22] and [24], an LTP block could contain either Red and Green parts, i.e. a set of segments to be transmitted, respectively, reliably and unreliably. However, more than ten years' experience of DTN researchers showed tremendous advantages in using monochrome blocks, i.e. using a block which is either fully Green or Red, and not miscellaneous. Accordingly, implementation complexity could be reduced and a session would assume the colour of the block, giving a clear one-to-one mapping between the required QoS and the colour of the session for sending a bundle. For example, if a bundle should be transmitted

reliably, a Red session should start and the bundle should be inserted into a Red block. Conversely, if the unreliable service is required, the new session should be Green. For these reasons, CCSDS on-going standardization (LTPv2) will conceive monochrome blocks only [25].

From now on, the terms "Red session" and "Green session" will be referred to as session in which a block is transmitted reliably or unreliably, respectively.

1.3.2 Green session

A Green session is conceived to be an LTP session in which the block is unreliably transferred: when the transmission starts, all data segments are sent, the last being flagged as EOB. As soon as the EOB is sent, the LTP sender notifies BP that it has successfully terminated the transmission session and, consequently, the BP sender may delete the bundle from its DB.

1.3.3 Red session

Segments involved in a Red session must be reliably delivered to destination on the basis of an ARQ mechanism (like TCP).

Case no losses. As per the image below, all segments are sent and the last one is flagged as EOB and CP. The reception of a Checkpoint (CP) triggers a Report Segment (RS) from the receiver node; the RS confirms all data has been received and triggers a Report Acknowledgment from the sender node which confirms the RS has been received and the session is closed. [25]

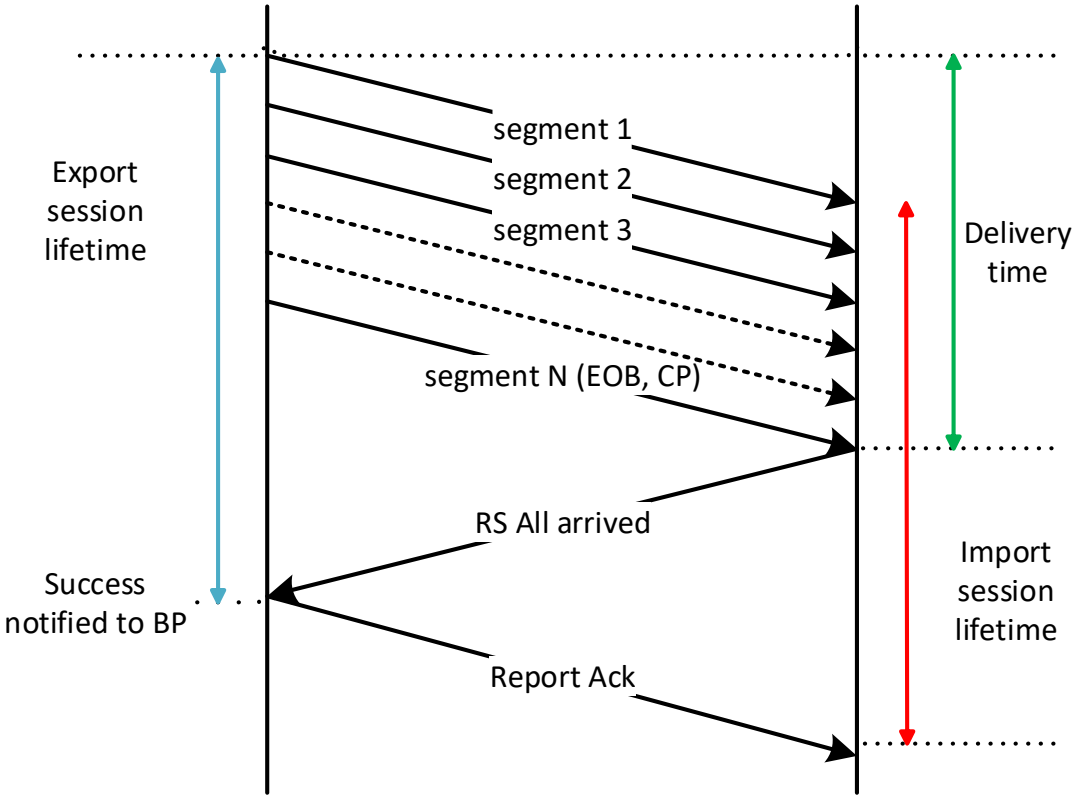

***Figure 3**:example of LTP transmission without losses*

The LTP payload is delivered in ½ RTT, which is the theoretical minimum in absence of losses.

Case losses on data segments. As per the image below, all segments are sent and the last one is flagged as EOB and CP. Segments 2 and 3 are lost due to channel impairments. All segments but 2 and 3 arrive to the receiver node; the CP triggers a RS from the receiver node in which segments 2 and 3 are claimed. The reception of the RS triggers a RA and retransmissions of segment 2 and 3 from the sender side. In particular, the current last segment (3) is flagged as checkpoint. This time all the segments are received correctly and the arrival of the CP (segment 3) triggers a RS from the receiver side confirming all data has been received. The arrival of the RS triggers a RA from the sender node confirming that the RS has been correctly received and the session is closed.

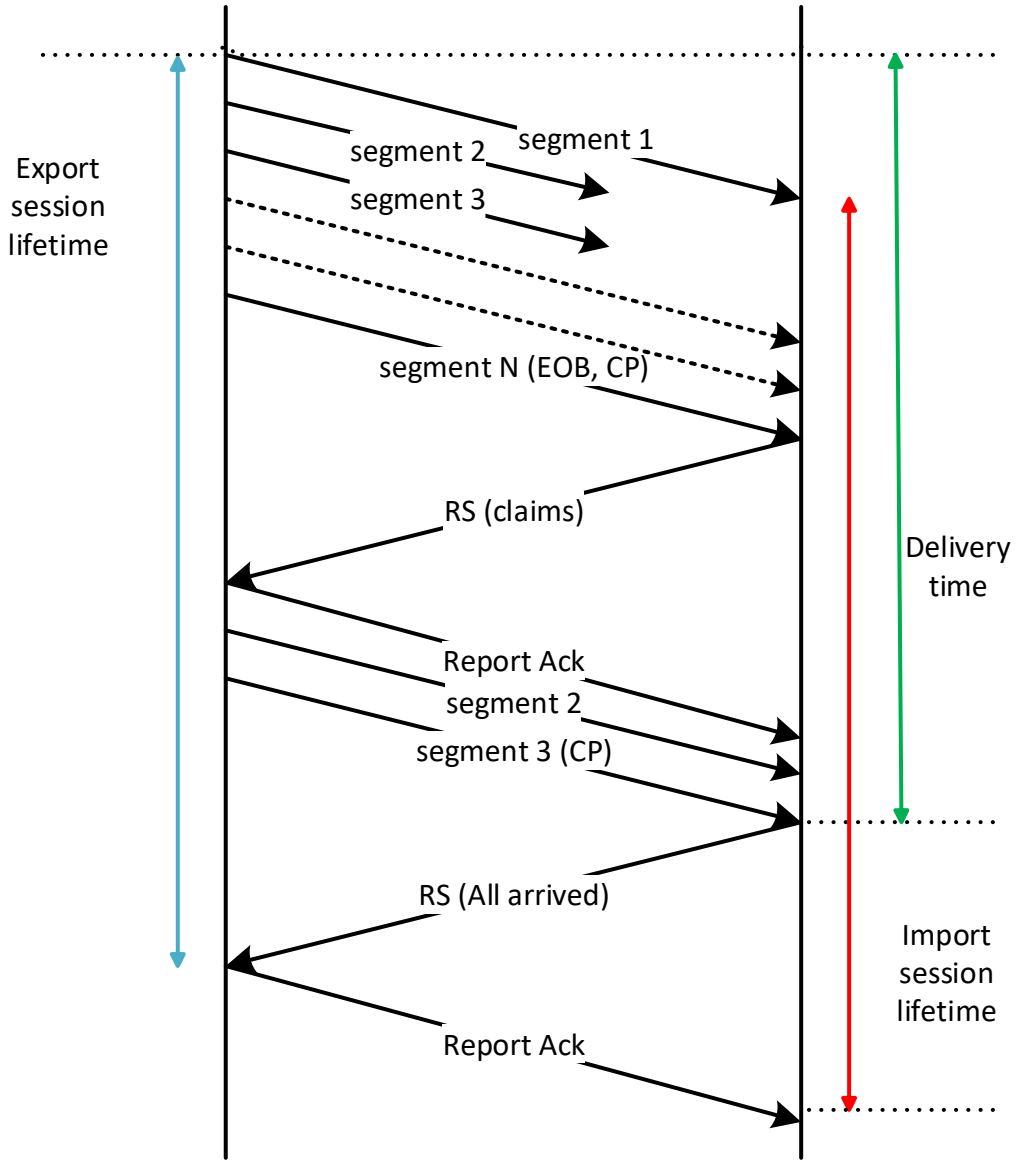

***Figure 4**:example of LTP transmission with losses on data segments*

Assuming that the transmission time of re-transmitted segments is much lower than the RTT (as in space), the retransmission penalization is one RTT, independently of the number of losses. Losses on retransmitted segments would have caused one RTT more. [25]

Case losses on data and signaling segments. As per the image below, all segments are sent and the last one is flagged as EOB and CP. Segments 2, 3 and N (last one) are lost due to channel impairments. All segments but 2, 3 and N arrive to the receiver node. Since the CP has been lost, the receiver node does not send any feedback at all. The Retransmission Time Out (RTO) fires and the sender node automatically retransmit the last segment (N), which is flagged again as EOB and CP. This time the segment N is received and, since it is flagged as CP, it triggers the shipment of a RS from the receiver side who claims data segments 2 and 3. The arrival of the RS triggers a RA and retransmissions of the claims (segments 2 and 3): the current last segment (3) is flagged as CP. RA, 2 and 3 are correctly received at the receiver side which replies to the CP with a RS confirming all data has been correctly delivered. The arrival of the RS triggers a RA from the sender node confirming that the RS has been correctly received and the session is closed.

The loss of one CP results in one RTO of penalization; the same holds true for the loss of all other signaling segments subjected to retransmission. As stated, losses on pure data segments (one or more it does not matter) result in one RTT of penalization for each retransmission cycle (one if there are not losses on retransmitted segments).

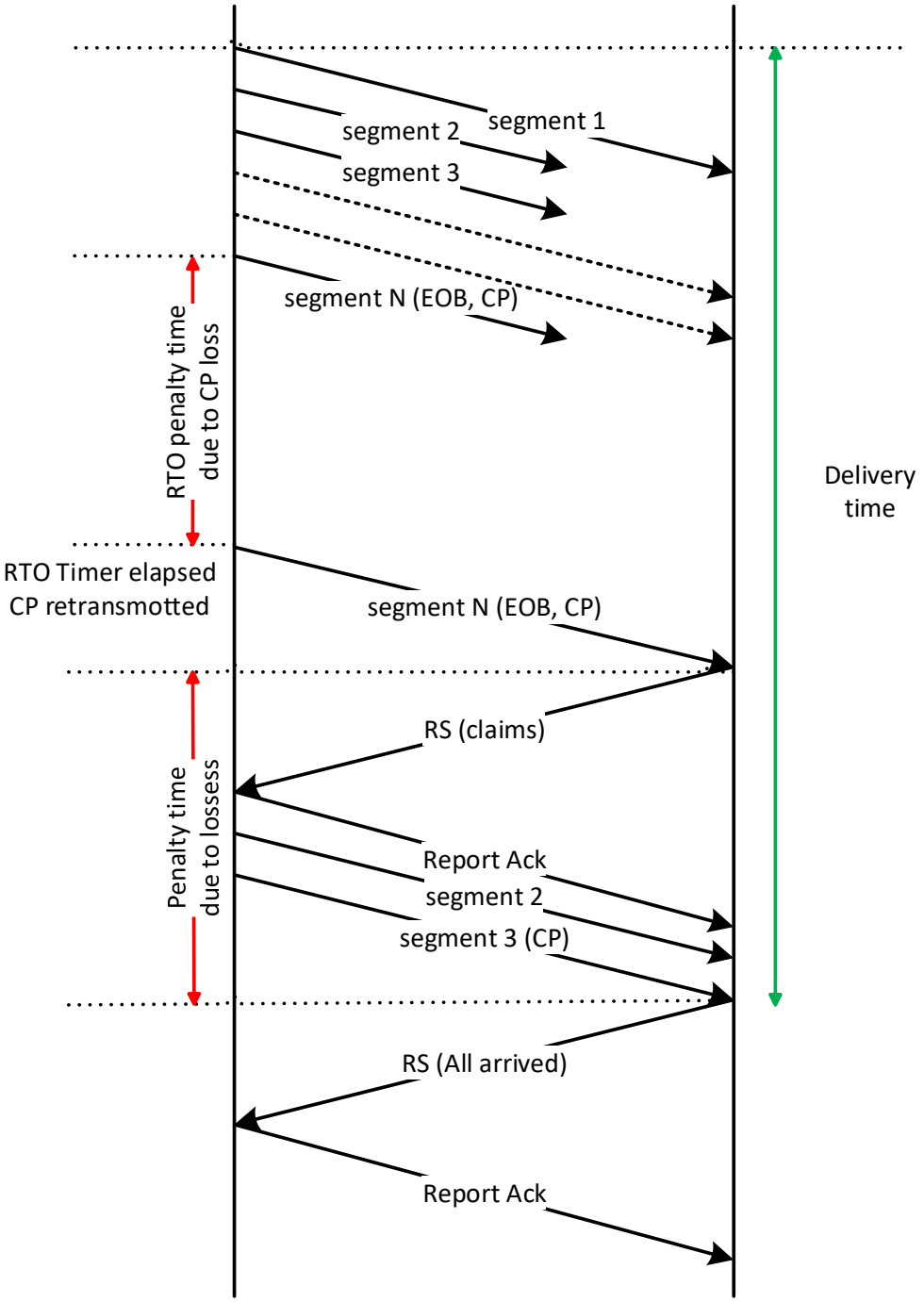

***Figure 5**:example of LTP transmission with losses on data and signaling segments*

The minimum penalization to recover from one or many data losses is always one RTT, as the first and further losses are tackled by the same retransmission cycle. However, if there are losses on retransmitted segments, a further retransmission cycle is needed, resulting in one more RTT of penalization. Further penalizations appear at each cycle with losses.

Losing signaling segments is even worse as there is 1 RTO of penalization for each loss (in contrast to TCP, RTO is set a priori, usually as the sum between the expected RTT and processing times). [25]

1.3.4 LTP implementations

There are several LTP implementations: Those considered in this work are:

ION-LTP. [26] ION-LTP is managed as a collection of "spans", which are transmission/reception relationships between the local LTP engine and each other LTP engine with which the local engine can exchange LTP segments. When adding or updating a span, the following parameters must be set:

- The remote LTP engine number. For ION it is the same as the remote BP node number
- The maximum number of export sessions that can be parallelly held open on this span. This results in the LTP flow control, as no new session can be started until the actual number of active sessions is equal to this parameter. This parameter will play a fundamental role in tests.
- The maximum number of import sessions that can be parallelly held open on this span. Of course, this parameter must coincide with the remote engine's own value for the maximum number of export sessions. In recent versions of ION this value actually applies to the maximum number of red sessions still waiting for retransmissions to complete the block (an import session which has received all data actually closes 1 RTT later, as it will be explained later).
- Maximum LTP segment size. Max size of the data portion of an LTP segment in bytes
- Aggregation size limit. To reduce the overhead (signaling segments) associated with transmitting many small bundles in independent LTP blocks, bundles are aggregated in one LTP block until either this number of data bytes is reached or the time limit (see next point) is reached, whichever the first.
- Aggregation time limit. Bundles are aggregated until this threshold is reached (seconds)
- Link Service Output command. This parameter (typically a string) declares the command that will be used to start the link service output task for this span. When the "udplso" link service output module is used for a given span (e.g. LTP segments are encapsulated in UDP datagrams), it is followed by the IPAddress:Port of the remote engine
- Estimated total number of export sessions, for all spans. This value is used to size the hash table that LTP uses for storing and retrieving export session information

To lighten the task of tuning the above configuration parameters, an LTP Configuration Worksheet is available in [26].

DTNME-LTP. The major DTNME-LTP parameters which can be set in the implementation of version 1.2.0 are:

- LTP engine ID of the remote peer

- Max number of outgoing sessions (default is 100)
- Aggregation size. To reduce the overhead (signaling segments) associated with transmitting many small bundles, it is possible to accumulate/aggregate bundles for an LTP session until this number of data bytes is reached (default is 1Mbyte)
- Aggregation time. Accumulate bundles for an LTP session for up to this number of milliseconds (default is 500 ms)
- Inactive interval. Cancels an incoming LTP session if no activities are carried for this number of seconds (default is 30 seconds)
- Retransmission interval. RTO in seconds (default is 7)
- Retransmission retries. Number of retransmissions before cancelling an LTP session (default is 7)
- Use files. Whether to use files (or memory) to hold data for incoming and outgoing LTP sessions (default is False)
- Segment size. Max size of the data portion of an LTP segment in bytes (default is 1400)
- UDP transmit rate throttle in bits per second (default is 0, no throttle)
- Token bucket type (0 = standard, 1 = leaky. Default is 0)
- Token bucket throttle depth in bits (default is 524280)
- UDP socket send buffer size to use (default is 0, which means operating system managed)

In the latest versions of DTNME, it is possible to configure even more parameters, which are fully described in [27].

Unibo-LTP.

Unibo-LTP is an implementation of Multicolor LTP, an enhanced LTP version proposed by the University of Bologna and DLR [25]. Multicolor LTP is essentially compliant with [22]and [24], but it supports only monochrome blocks, promote the use of a new colour, orange, lacks aggregation, include session timers and other minor enhancements. Unibo-LTP implementation started with Bisacchi's Master's thesis [28] which realized a first preliminary version as an ION plug-in. It was designed to overcome ION's data rate limitations (about 50-100 Mbit/s on ordinary PCs) and included a token bucket traffic shaper. The second version, just released (February 2023), is compatible also with Unibo-BP, the brand new (February 2023) implementation of BP of the University of Bologna [29].

With Unibo-LTP, the following parameters must be set for each span:

- Peer engine number. Node number of the destination engine
- Max export and import sessions. Maximum number of sending sessions
- Max import sessions. Maximum number of receiving sessions
- Max segment payload size. Max size of the data portion of an LTP segment in bytes
- Destination IP address of the remote LTP engine
- Destination port. Port number of the remote LTP engine
- Estimated remote processing delay, which is used in computing the RTO

- Token bucket burst size. Maximum size of the token bucket, used for regulating the shipment of LTP segments
- Default colour (QoS) to be used in sending sessions (Green, Red or Orange [25])
- Outer lower protocol to be used under LTP (UDP or ECLSA, an experimental Unibo-DLR protocol based on Packet Layer FEC [30] [31])
- Outer lower protocol string, to customize the out lower protocol (especially if ECLSA)

Within the code of Unibo-LTP (config.h) it is possible to set other parameters, including:

- Max retransmission number. After this number of failed attempts, the session is cancelled.
- Signal segment copies. Number of copies of the same signaling segment, in order to reduce its loss probability [32]
- Distributed copies. States whether signal segment copies have to be transmitted in a time-distributed way (at fraction of RTO) instead as in a burst [32]
- Token update time. Time period at which new tokens are inserted in the token bucket, which controls LTP segments transmission rate
- Max number of claims that can be in a single report segment. If this number is exceeded, multiple RSs are needed to signal all claims

Version 2 provides on request a "csv" log, with a line for each transmitted or received segment. This log can be directly processed by the Python program ltp_perfromance developed in this thesis, thus skipping the step of frame capture with tcpdump or similar programs. It also allows the user to insert the value of RTO to be used as an option, thus overriding the RTO calculated automatically from ranges and processing times. It is worth noting than in ION LTP RTO is always calculated automatically, while in DTNME always set by the user. In both cases, however, the time granularity is quite coarse, 1s, which may result excessive in near earth application of LTP, where RTT<<1s, as those considered in this thesis. For this reason, in Unibo-LTP the optional RTO value has the much finer granularity of 1 ms.

1.4 SATELLITE COMMUNICATIONS

Satellites are widely used for communication, Earth observation, broadcasting, navigation, weather forecasting and many other different applications: as of 2022, there are 4852 active satellites orbiting Earth [33], and of course this number is constantly changing due to the launch of new objects and the retirement of older ones.

Satellites are mainly classified in relation to their distance from the Earth's surface.

Low Earth Orbit (LEO). Most of the satellites orbiting Earth do so at an altitude between 300 and 2000 kilometres. This orbital regime is known as Low Earth Orbit (LEO) and it is particularly suited for Earth observation: the short distance allows for high resolution images of Earth and grants short propagation delay communications between satellites and ground stations. One key example of LEO satellite is the International Space Station (ISS), orbiting at an average altitude of about 400 kilometres.

Medium Earth Orbit (MEO). The region of space between Low Earth Orbit and Geosynchronous Earth Orbit (GEO) is known as Medium Earth Orbit, which altitude ranges from 2000 kilometres and 35786 kilometres. One key example of MEO application is the Global Positioning System (GPS), being a 24 satellites constellation, with each satellite located at about 20000 kilometres from Earth's surface: the constellation is oriented such that at any given moment, every point on Earth has access to four GPS satellites.

Geosynchronous Earth Orbit (GEO). It is a circular geosynchronous orbit 35786 kilometres in altitude above Earth's equator and following the direction of Earth's rotation. A GEO satellite features an orbital period which is equal to the Earth's rotational period: one sidereal day. Therefore, a GEO satellite appears motionless to ground observers. Communications satellites and weather satellites are often given geostationary orbits, so that the satellite antennas that communicate with them do not have to move to track them, but can be pointed permanently at a fixed position in the sky.

The main focus of this thesis regards LEO satellites, featuring short propagation delays but intermittent connectivity with a ground station on Earth; indeed, the typical LOS (Line of Sight) window lasts for few minutes (e.g. 11). These brief contact durations with a ground station highlights the strong need of a fast and reliable communication channel which is capable of achieving massive data rates at least in the downlink (from satellite to ground station). As an example, a LEO satellite equipped with a high-resolution camera can produce data at a rate of 6.7 Gbit/s, which is equivalent to 26411 Terabytes per year at 100% usage [34]. In such a scenario, the conventional RF downlink became the bottleneck for Earth Observation systems, not being able to achieve transmission data rates higher than some hundred Mbit/s. For these reasons, large interest and funding have been recently given to optical communication systems, being able to achieve transmission data rates spanning from 1 Gbps to potentially 1 Tbps.

1.5 FREE-SPACE OPTICAL (FSO) COMMUNICATION

Optical Free-Space transmission is a very effective way to transmit information from point to point with lowest transmit power and at highest data rates. While systems operating at radio frequencies (RF) require comparatively high-power consumption and large antenna structures, are subject to frequency regulation, and are limited in data rate, the optical technology does not have these drawbacks and can already today transmit data up to several gigabits per second. For example, some DLR reference scenarios are the transmissions with OSIRIS4CubeSat, OSIRISv1, CubeLCT and OSIRISv3, showing FSO downlink data rate up to, respectively, 100 Mbit/s, 200 Mbit/s, 1 Gbit/s and 10 Gbit/s. Theoretically, optical satellite communication links enable data rates in the order of Terabits per second [35]. These features will lead to an extreme size and power reduction while enabling new applications which are not feasible with existing RF-technology.

FSO principles. Basically, any space lasercom system will encode some information on a laser beam, collimate and transmit it by means of a telescope, and, after been propagated through free space, it will be collected in other distant telescope and focused on a small spot

in the focal plane, where a photodetector will transform the optical signal into an electrical one, which will be decoded to extract the original information. [36]

FSO challenges. Despite its tremendous advantages, Free-Space Optical (FSO) communication is challenging because of different atmospheric effects (e.g. fogs, clouds, rain) that deteriorate the incoming signal. In addition, FSO being a line-of-sight communication, inaccurate pointing and tracking also causes intermediate signal reduction. Moreover, fading occurs when the received power is less than certain threshold, below which the signal cannot be correctly demodulated and decoded.

In presence of clouds, optical links are usually not possible and, since LEO contacts are very short, the overall link availability is seriously limited when there are clouds at the time of the transmission. To face this issue, site diversity is required: a network of optical ground stations with uncorrelated atmospheric conditions is needed to guarantee that at least one of them is available with a certain probability, increasing the overall network availability. The correlation of cloud coverage between two locations usually decreases for distances larger than 80 kilometres. [36]

From the communications perspective, the atmospheric turbulence impact can be described with the following parameters:

- Scintillation: normalized variance of the signal fluctuations
- Intensity correlation time: describes the time characteristics of the signal fading
- Fading distribution: statistical distribution of the intensity fluctuations

To this thesis' ends it is was necessary, as discussed later on, to build upon a FSO channel emulator to allow for observations and consequent analysis of LTP and, in particular, its ARQ mechanism, over typical FSO impairments given by atmospheric conditions and pointing errors. Of course, the level of these impairments may be stronger or weaker depending on the intensity of the disruptions. Therefore, there was the need to unbind this general tool from the tracks resembling the state of the channel at a particular time instant. Thus, the channel track is the input for this channel emulation software.

Tracks resembling the state of FSO channels were given by the Optical Link System Department within the Institute of Communication and Navigation at DLR (DLR-KN) and describe the link Bit Error Rate (BER) with a sampling frequency of 10KHz. It must be specified that these tracks, also called Power Vectors (PV), do not come from real physical measurements but from simulations with the Power Vector Generation Tool (PVGeT) [37], developed according to real satellite link measurements collected in the course of the last decade. In more detail, the provided traces are based on numerical simulations with received power vectors over time – depending on several parameters like aperture diameter, beam divergence, pointing jitter, strength of atmospheric scintillations, orthogonal velocity of wind and platform movements –. [37] [38]

In particular, PVs used in this work describe the state of the channel during a contact between a LEO satellite and an optical ground station. Accordingly, another parameter having

influence on the BER is the elevation angle, being the angle of direction of the satellite, with respect to the horizontal plane at the ground station. A higher elevation angle means that the satellite is closer to the zenit and the signal must travel less of the Earth's atmosphere, resulting in higher possibility error-free transmission. A lower elevation angle means that the satellite is closer to the horizon and the signal has a larger distance to travel through the atmosphere; consequently, the signal is weaker and possibly more disturbed.

1.6 DTN OVER NEAR-EARTH OPTICAL COMMUNICATION

Optical communication is a bedrock for future space missions, as it provides faster opportunities for the download of large amounts of data (e.g. from Earth Observation) and the Delay-/Disruption Tolerant Networking architecture can be extremely useful in overcoming typical challenges of FSO communication scenarios. DTN could best harness the links capacity, which could not be far accomplished by classical TCP/IP protocols.

Apart from interplanetary networks, where all the advantages of the DTN architecture have already been discussed, also a near-Earth optical communication scenario can largely benefit of DTN features: this thesis focuses on the scheduled communication between a LEO satellite and an optical ground station where applying DTN protocols eases the impairments typically involved by atmospheric turbulences [32]. Although, for this scenario, the RTT is in the order of milliseconds, the use of DTN can be beneficial as outage by clouds can by many orders of magnitude higher and cause meaningfully high BERs.

2. FREE SPACE OPTICAL CHANNEL EMULATION TRACKS

2.1 ERASURE VECTORS

2.1.1 Fading vectors

Free Space Optical Communication (FSO) links suffer atmospheric turbulences and pointing errors of the transmitter telescope. The combination of both effects results in intensity scintillation and fading at the receiver telescope.

With the purpose of evaluating how different communication technologies and network protocols behave over typical FSO conditions, the Optical Communication department at DLR Oberpfaffenhofen – Institute of Communication and Navigation, developed a MATLAB-based simulation program called PVGeT (Power Vector Generation Tool). This instrument produces tracks which describe the state of the FSO channel in a point-to-point transmission, accounting the impacts of atmospheric turbulence, pointing jitter, and several other parameters [37].

Typical tracks that were used for the aims of this thesis describe the link Bit Error Rate (BER) with a sampling frequency of 10KHz of a LEO satellite to ground station optical link. These tracks are provided in a .mat file, being a unidimensional array of real numbers. Each entry of these tracks describes the BER of all the bits that are transmitted in the corresponding time interval of 0.1 ms.

2.1.2 From fading to erasure vectors

To emulate the behaviour of the channel during a transmission there is the need to understand how ethernet frames are affected from the varying BER of the fading vectors, accounting the correction ability of the chosen code, being Reed Solomon (255, 223) [39].

From the fading vectors, it is possible to compute a vector of symbol error rates where each element Vs is given by:

$$Vs = 1 - (1 - Vb)^J \tag{1}$$

Where J = 8 and Vb is the generic BER entry of the starting fading vector. From this last array, another vector is computed in which the generic entry describes the Codeword Error Rate of the 0.1 ms time interval. Each element of this vector is given by:

$$\sum_{j=E+1}^{n} \binom{n}{j} Vs^j (1 - Vs)^{n-j} \tag{2}$$

Where E = 16 (correction ability of the chosen Reed Solomon code) and n = 255. Considering Ethernet frames of 1500 bytes and having this last computed vector of CWERs it is now possible to compute the packet error rate (PER) of each 0.1ms time interval as:

$$p_l = 1 - (1 - CWER)^N \quad (3)$$

Where N is the number of codewords included in a packet. In general, N is variable: for example, the first packet of 1500 bytes will be spread among 7 codewords of 223 informative bytes; the last codeword has still 61 free bytes that will be used by the second packet that, consequently, will be spread across 8 codewords – the first 61 bytes on the previous codeword, and the remaining bytes on 7 more codewords; on the last of these 7 codewords there will be 122 free bytes that are going to be used by the next packet, and so on. Together with this aspect, the transmission data rate must be considered. Depending on it, the number of packets involved in a 0.1ms time interval, is:

$$num\_packets\_per\_sample = data\_rate * time\_interval \,/\, 8 \,/\, 1500$$

For example, harnessing 500Mbps in 0.1ms results in transmitting 50000 bits, which are 6250 bytes: about 4 ethernet packets of 1500 bytes. Therefore, each entry of the CWERs vector is replicated *num_packets_per_sample* times and, only after this replication process, formula (3) is applied to each entry of the enlarged CWERs vector. All these steps are executed via Python scripts which are available at [40].

At the end of the process, there are *num_packets_per_sample* PERs for each time interval of 0.1ms. To always have tracks featuring the same sampling frequency, the average of these *num_packets_per_sample* PERs for each time interval is kept, concluding with an erasure vector having one average PER for each time interval of 0,1 milli-seconds.

2.1.3 Binarization of erasure vectors

Coding and interleaving techniques are completely ineffective when long outages (e.g. given by cloud blockages), occur, resulting in highly correlated losses. Given the correction ability of Reed Solomon (255, 223), fully described in [39], the performances of this code drop when the BER is about 10^{-4}, which results (on equal Signal to Noise ratio) in a CWER of about 1e-2 (Figure 6). Assuming that, on average, a packet is spread among 7 RS (255, 223) codewords, it could be said, under the hypothesis of FSO channel usage, that all the packets in a time interval of 10^{-4}seconds are lost if the corresponding erasure vector PER is higher than:

$$1 - (1 - 110^{-2})^7 \simeq 6.8\, x\, 10^{-2}$$

For this reason, the last step applied to erasure vectors is a binarization of the entries:

$$bin_{PER}(t) = \begin{cases} 0 & if\ PER(t) < 6.8\, x\, 10^{-2} \\ 1 & if\ PER(t) \geq 6.8\, x\, 10^{-2} \end{cases}$$

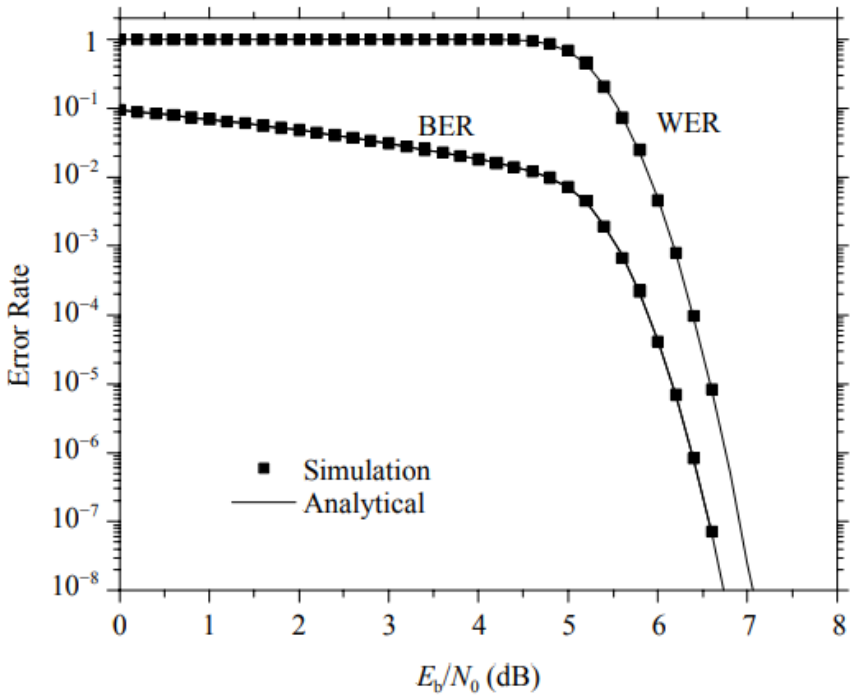

Figure 6: RS 255, 223 correction performances [39]

2.1.4 LEO to optical ground station scenarios

Four different scenarios have been considered to produce a benchmark of the LTP performances on FSO channels: scenarios A, F, H of [41] for validating tools, methods and procedures, having durations of 100 seconds, and a scenario resembling the full pass of an OSIRIS4CubeSat (04C) over an optical ground station, having duration of 692 seconds.

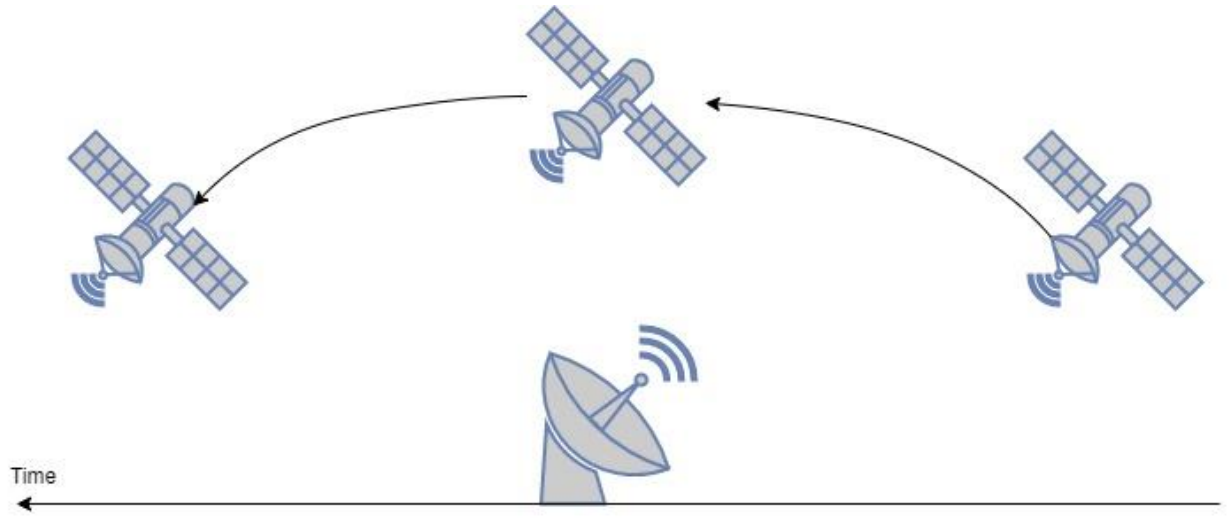

Figure 7: satellite over OGSOP

Given diverse initial conditions at the time of experiments, erasure vectors of scenario A, F and H have been binarized with a PER threshold of 10^{-4}, but since it is more conservative (it

feature more losses) than 6.8e-2, the following paragraphs and the results exposed in chapter (5) keep their validation purposes. For instance, vectors A, F, H describe punctually typical-to-worst situations at 5° and 15° elevation assuming a data rate transmission of 1Gbps. On the other hand, the full pass scenario of an O4C assumes a data rate of 100Mbps. [42]

From now on, '1's and '0's sequences in the erasure vectors will be also referred to as 'bad state's and 'good state's, implying that all packets transmitted in the time interval covered by these entries will be respectively lost or successfully received at the receiver.

2.1.5 Erasure vector scenario A

The erasure vector relative to scenario A provides 4.9% of entries at '1', meaning that all packets in the corresponding time interval are lost. The average fading duration is 0.953 ms, std. deviation of 0.657 ms. On the other hand, the average duration of the good state is 18.175 ms, std. deviation of 17.467 ms.

In the following, a column graph representation of the durations of bad states and good states.

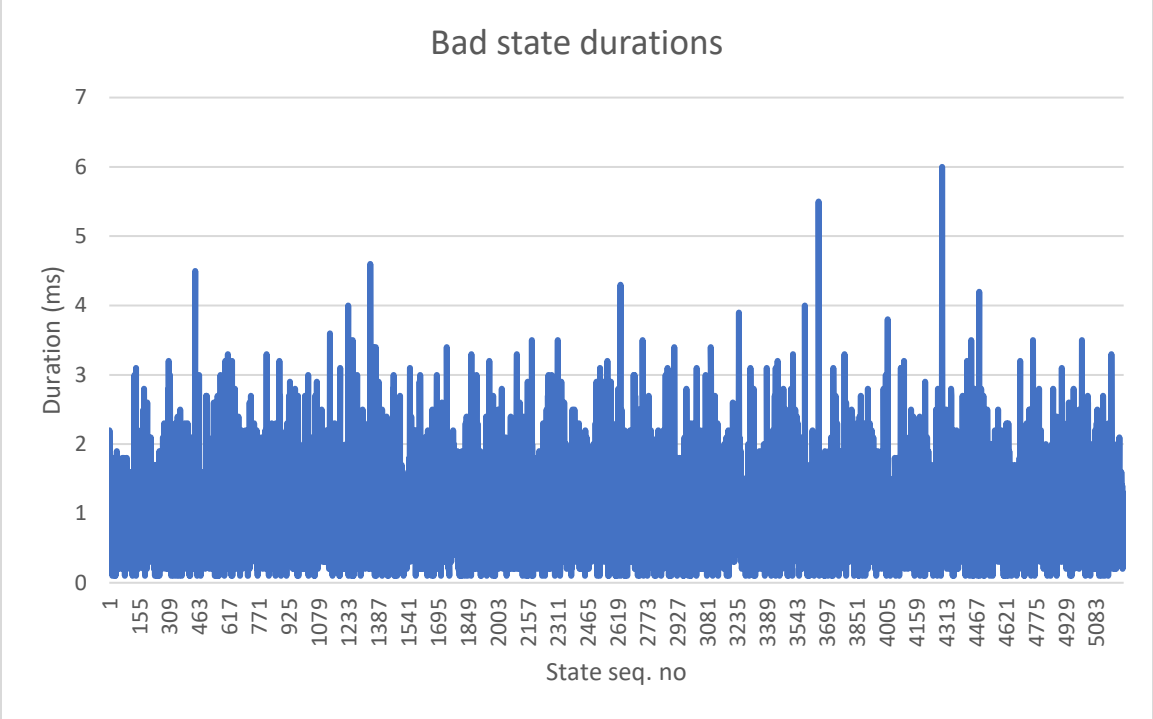

Figure 8: bad state durations of scenario A

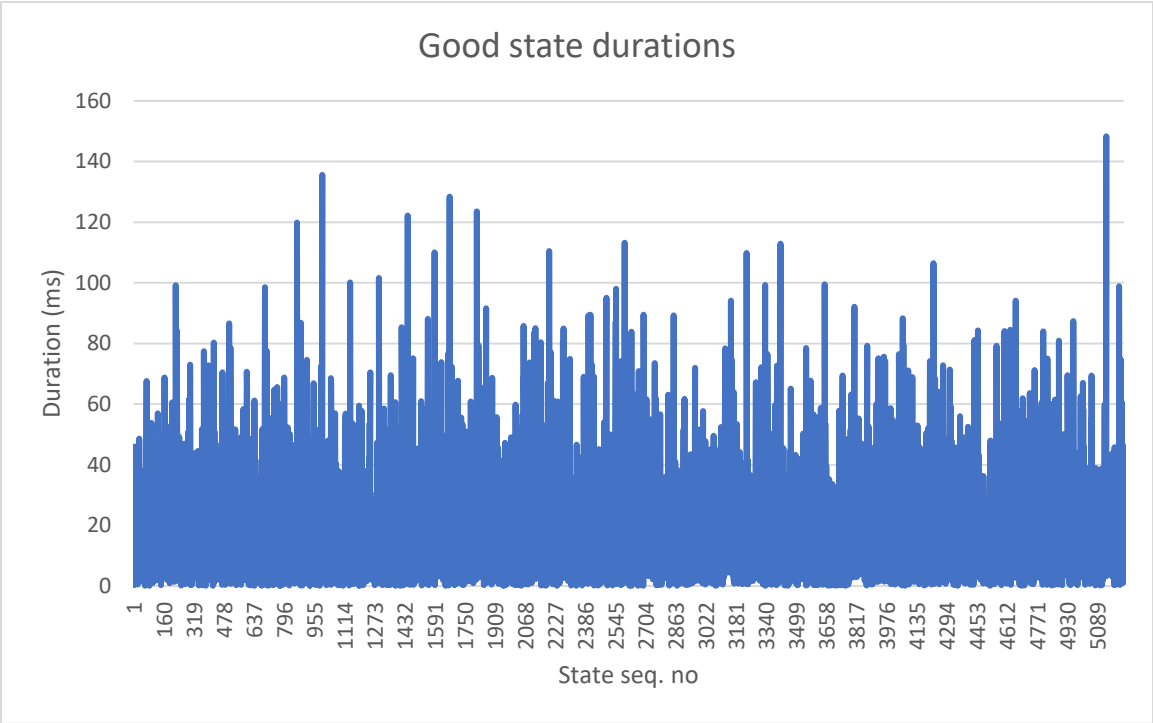

Figure 9: good state durations of scenario A

And the cumulative density functions:

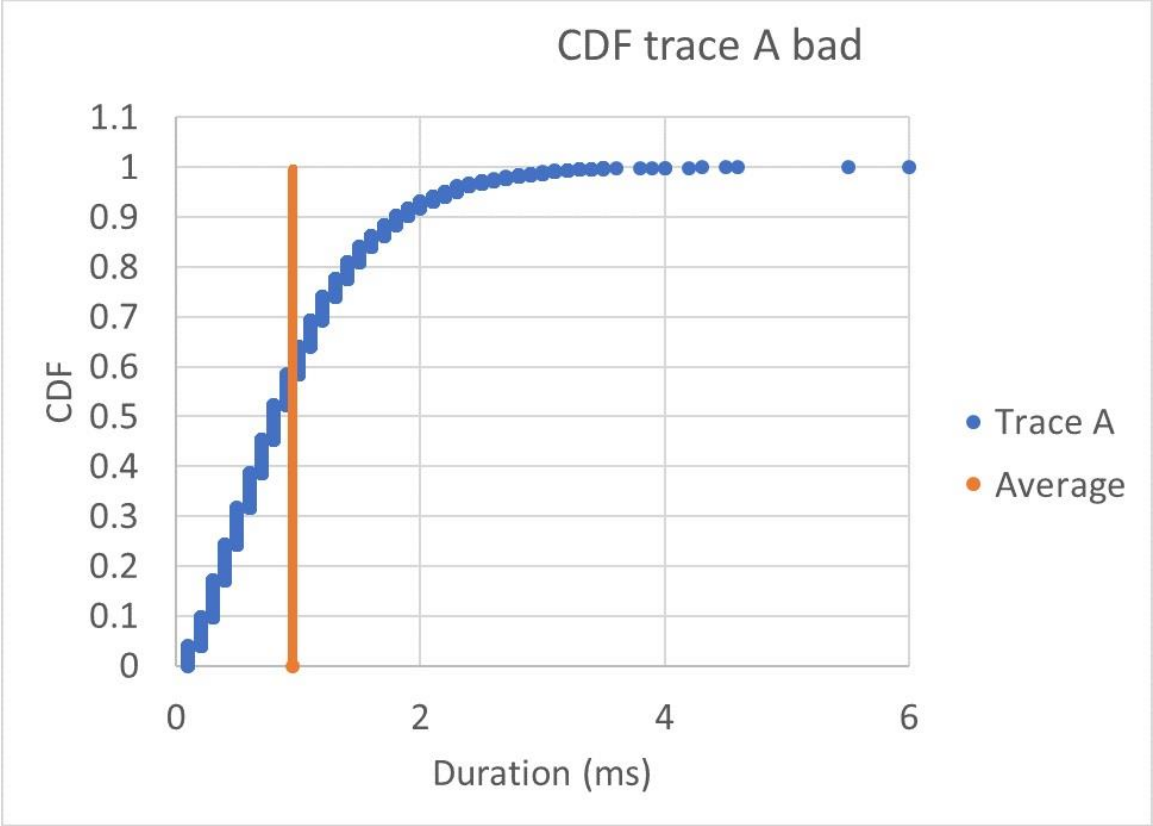

Figure 10:CDF of bad state durations in trace A

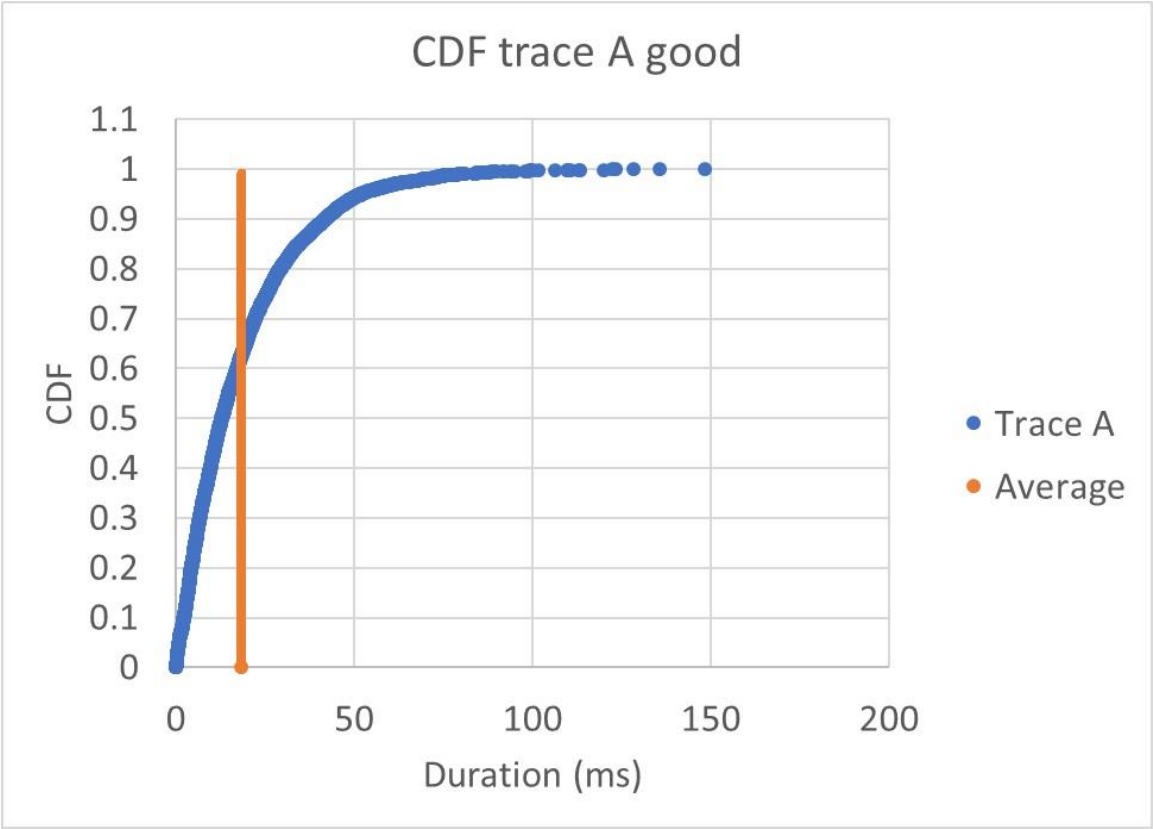

Figure 11: CDF of good state durations in trace A

2.1.6 Erasure vector scenario F

The erasure vector relative to scenario F provides 13.7% of entries at '1', meaning that all packets in the according time interval are lost. The average fading duration is 2.145 ms, std. deviation of 3.15 ms. On the other hand, the average duration of the good state is 13.46 ms, std. deviation of 18.19 ms.

In the following, a column graph representation of the durations of bad states and good states.

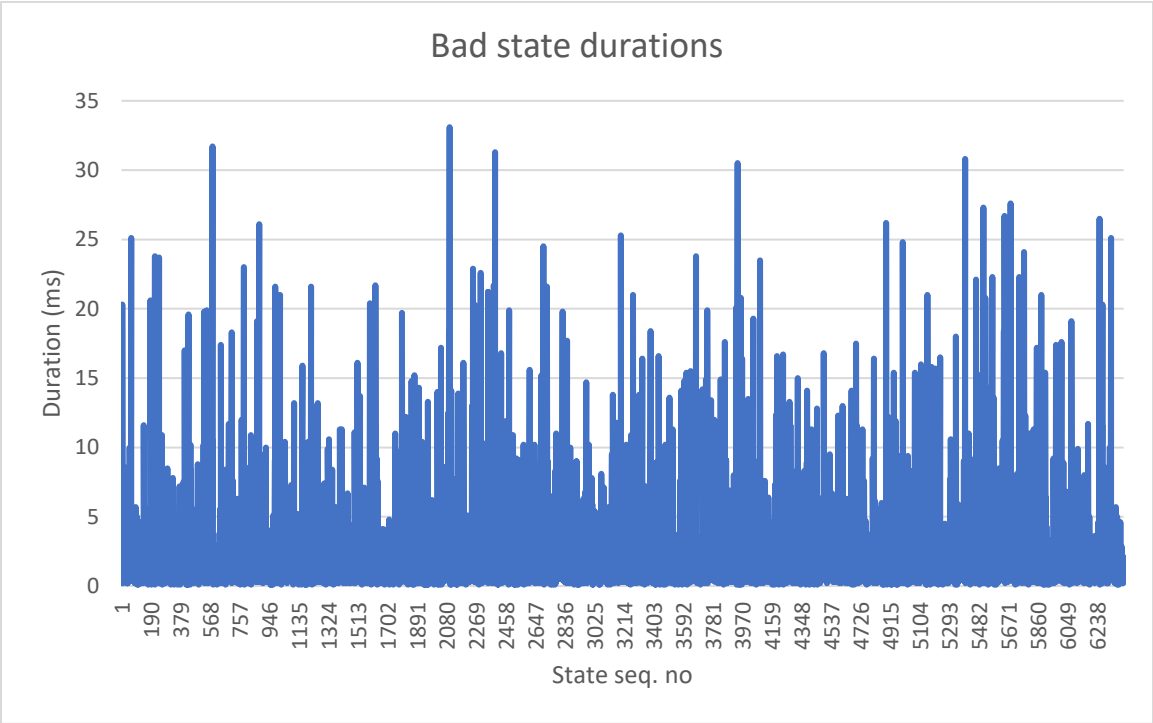

Figure 12: bad state durations of trace F

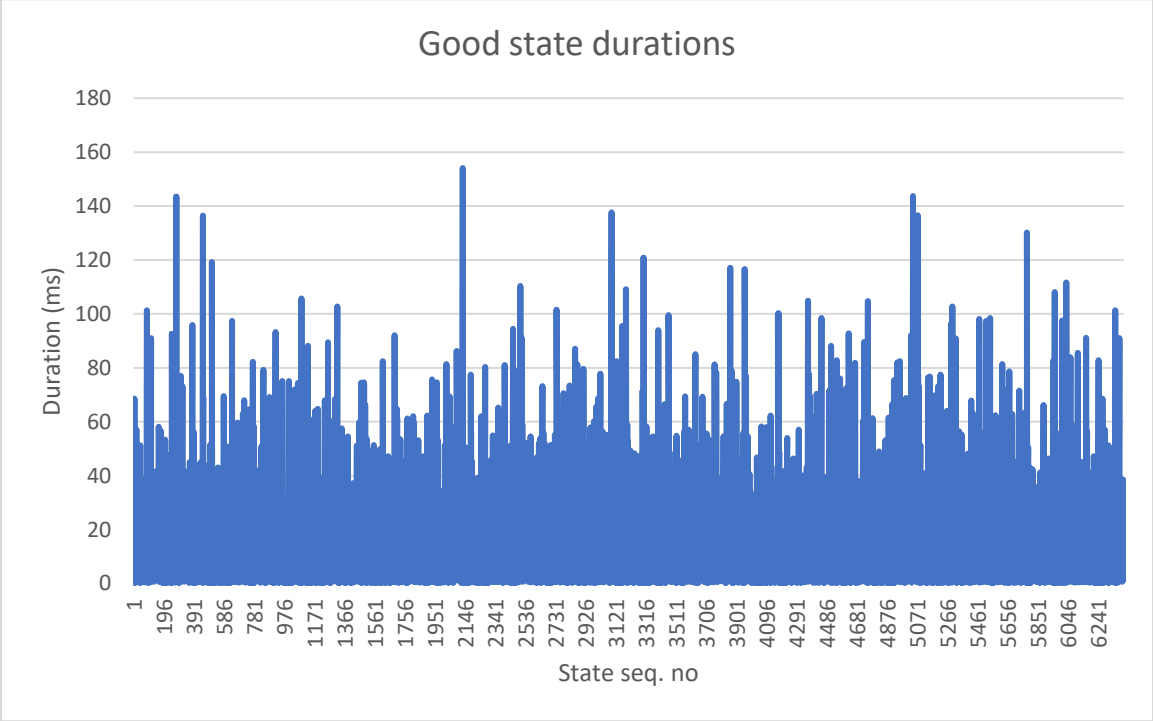

Figure 13: good state durations of trace F

And the cumulative density functions:

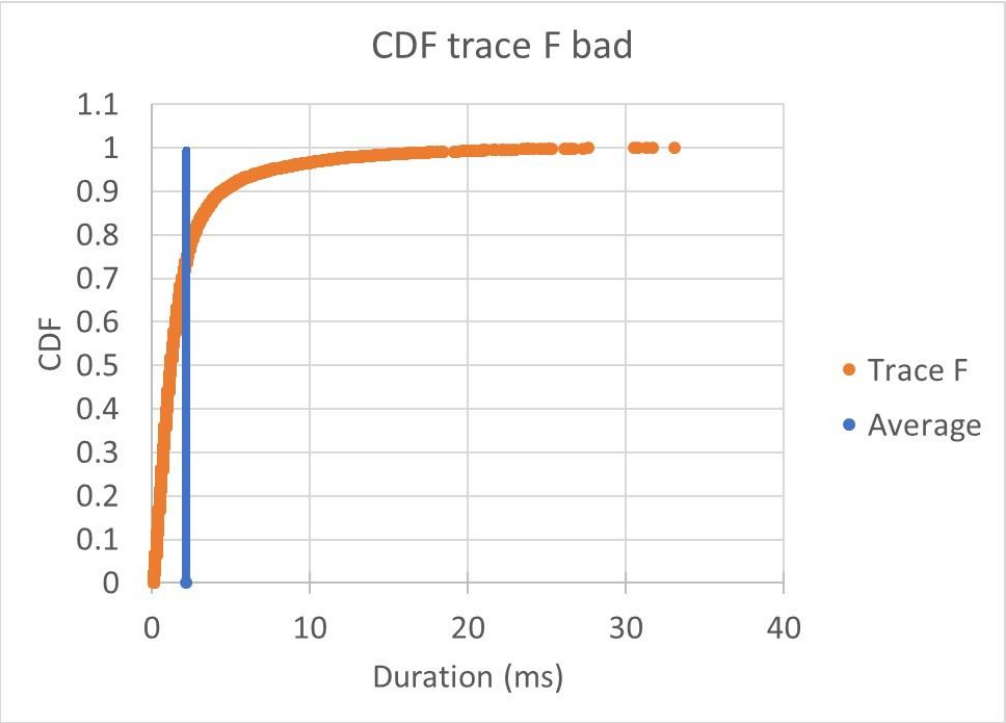

Figure 14: CDF of bad state durations in trace F

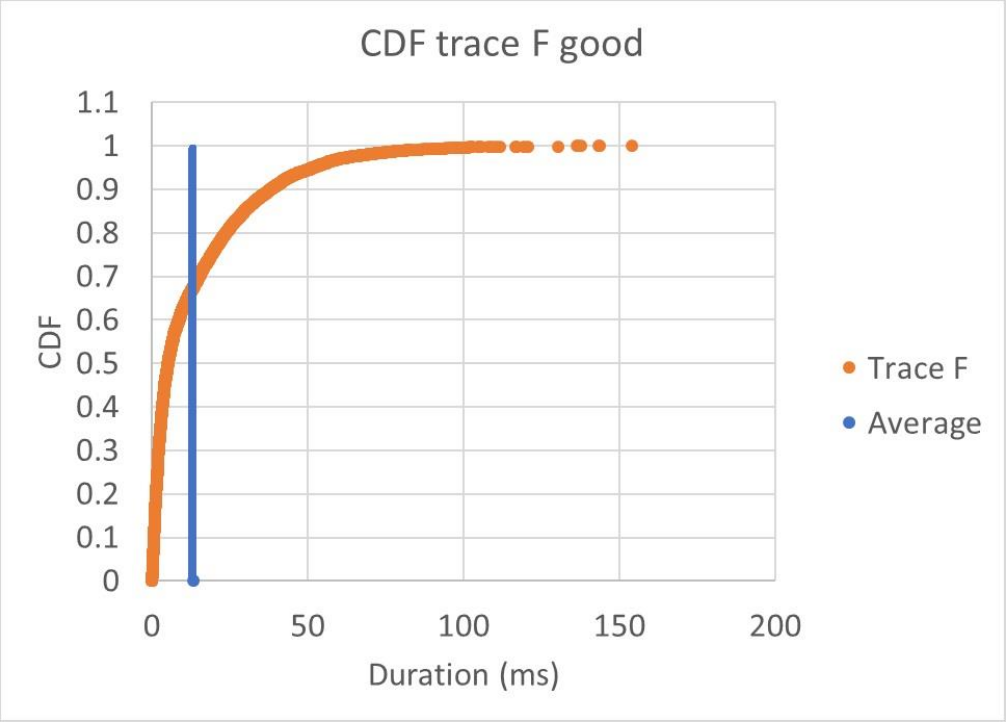

Figure 15: CDF of good state durations in trace F

2.1.7 Erasure vector scenario H

The erasure vector relative to scenario H provides 28.8% of entries at '1', meaning that all packets in the according time interval are lost. The average fading duration is 5.46 ms, std. deviation of 5.0 ms. On the other hand, the average duration of the good state is 13.48 ms, std. deviation of 12.85 ms.

In the following, a column graph representation of the durations of bad states and good states.

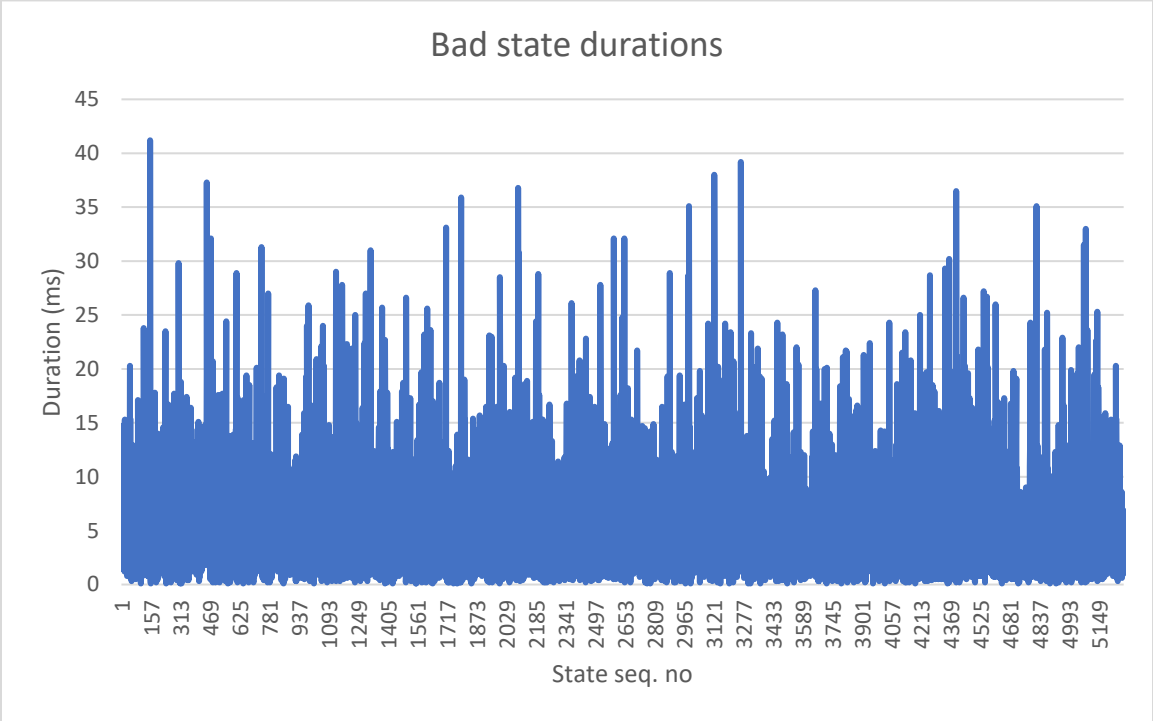

Figure 16: bad state durations of trace H

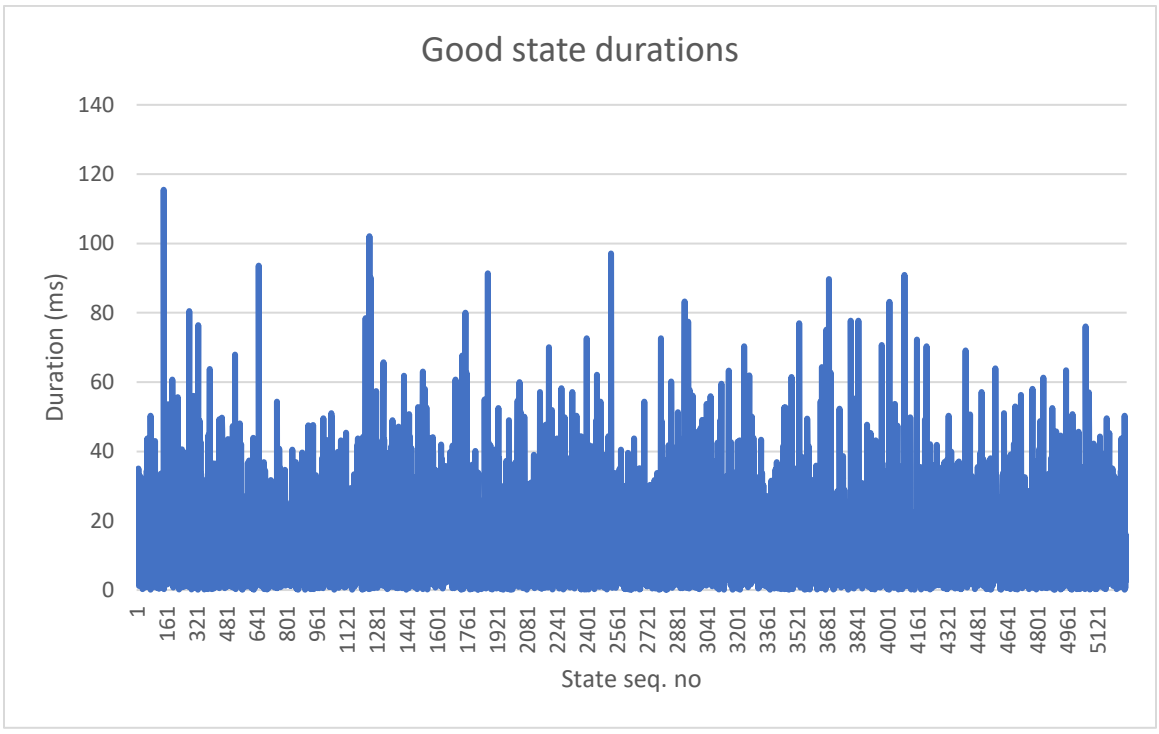

Figure 17: good state durations of trace H

And the cumulative density functions:

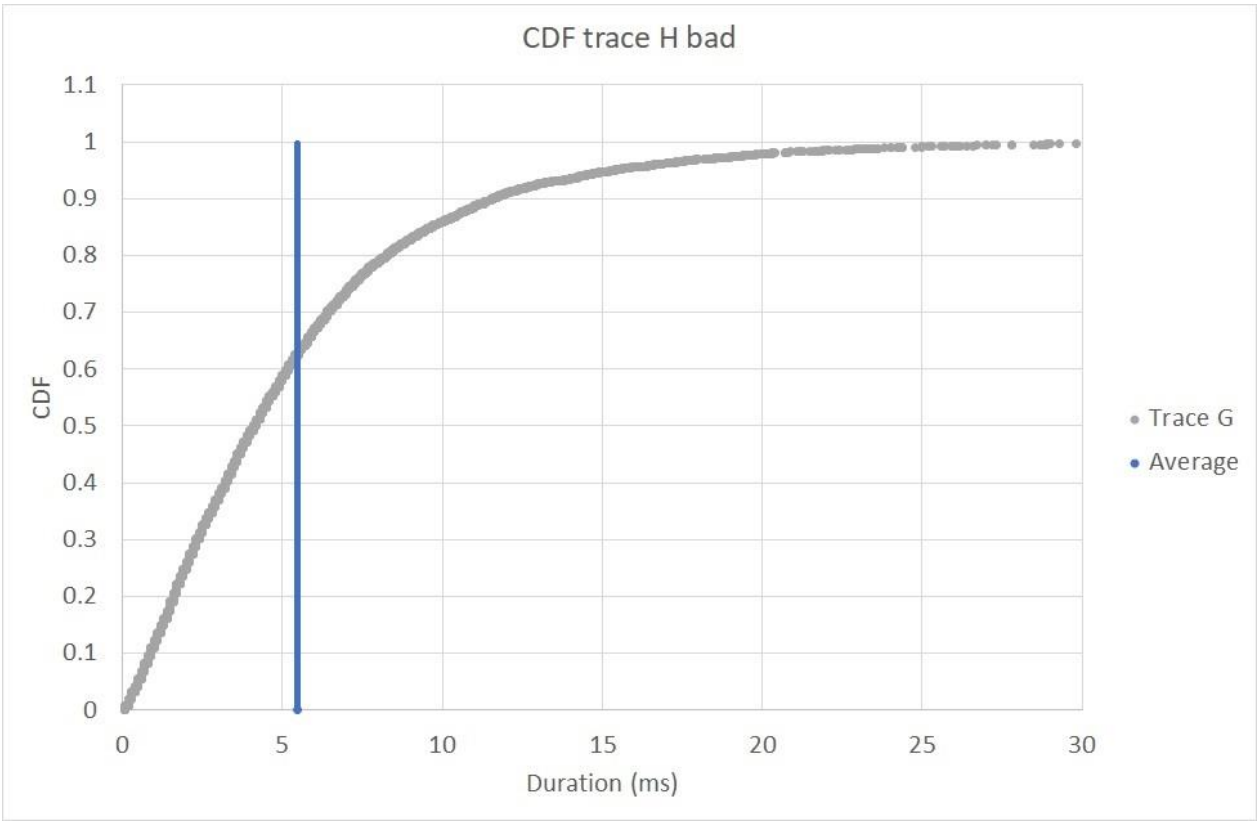

Figure 18: CDF of bad state durations in trace H

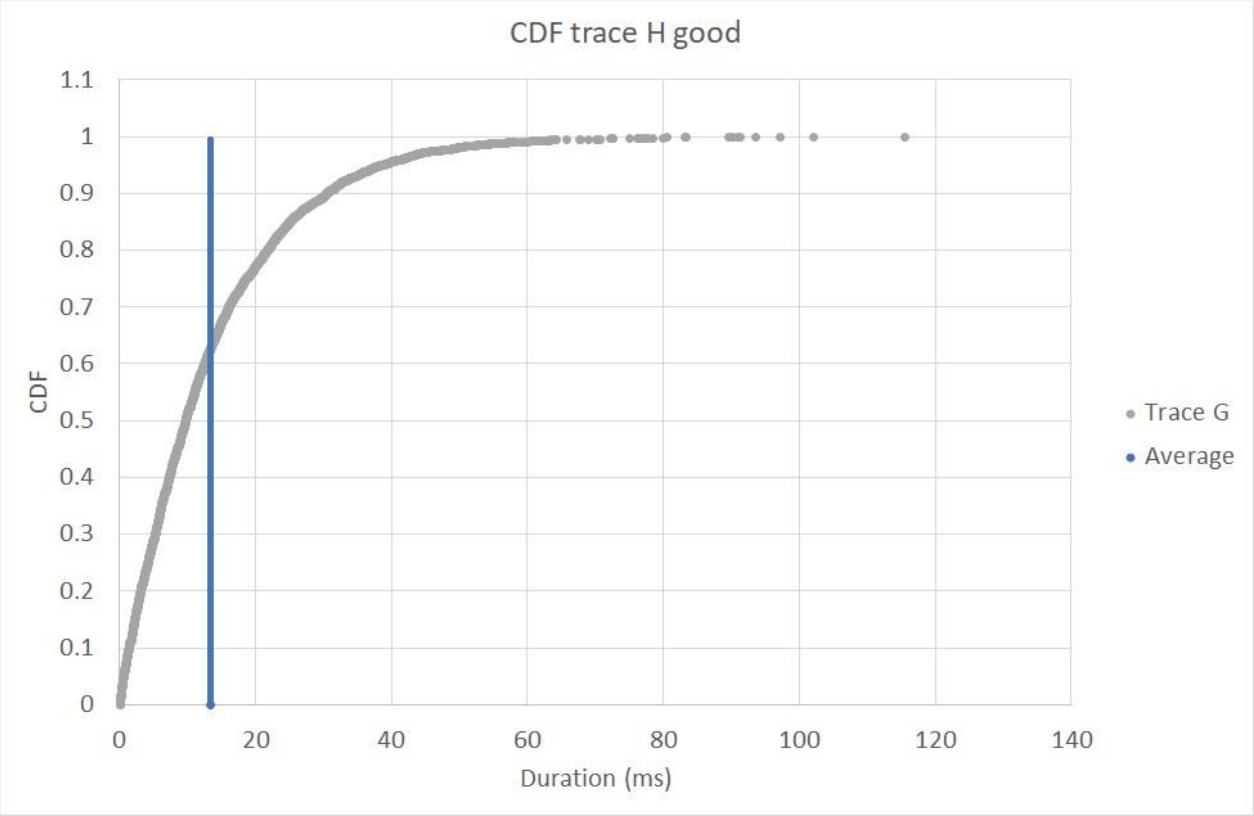

Figure 19: CDF of good state durations in trace H

2.1.8 Erasure vector full pass scenario

The erasure vector relative to the full pass scenario provides 8.4% of entries at '1', meaning that all packets in the according time interval are lost. The average fading duration is 3 ms, std. deviation of 4.32 ms. On the other hand, the average duration of the good state is 32.56 ms, std. deviation of 977.26 ms.

In the following, a column graph representation of the durations of bad states and good states.

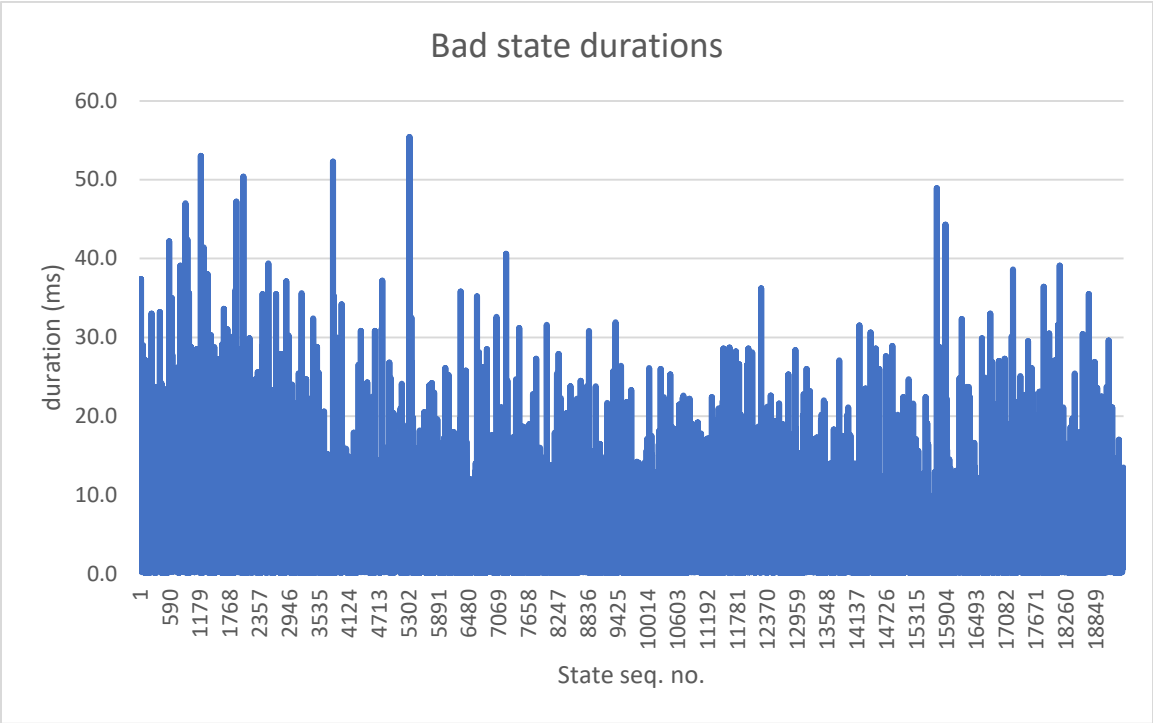

Figure 20: bad state durations of trace Full pass

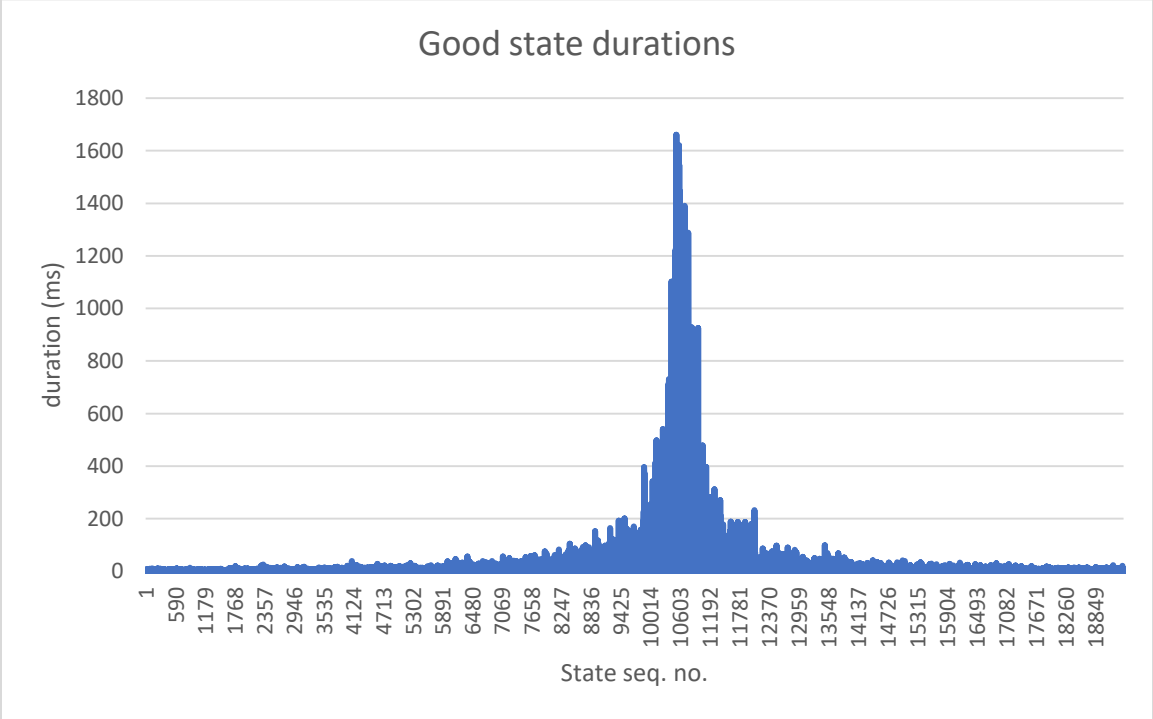

Figure 21: good state durations of trace Full pass

And the cumulative density functions:

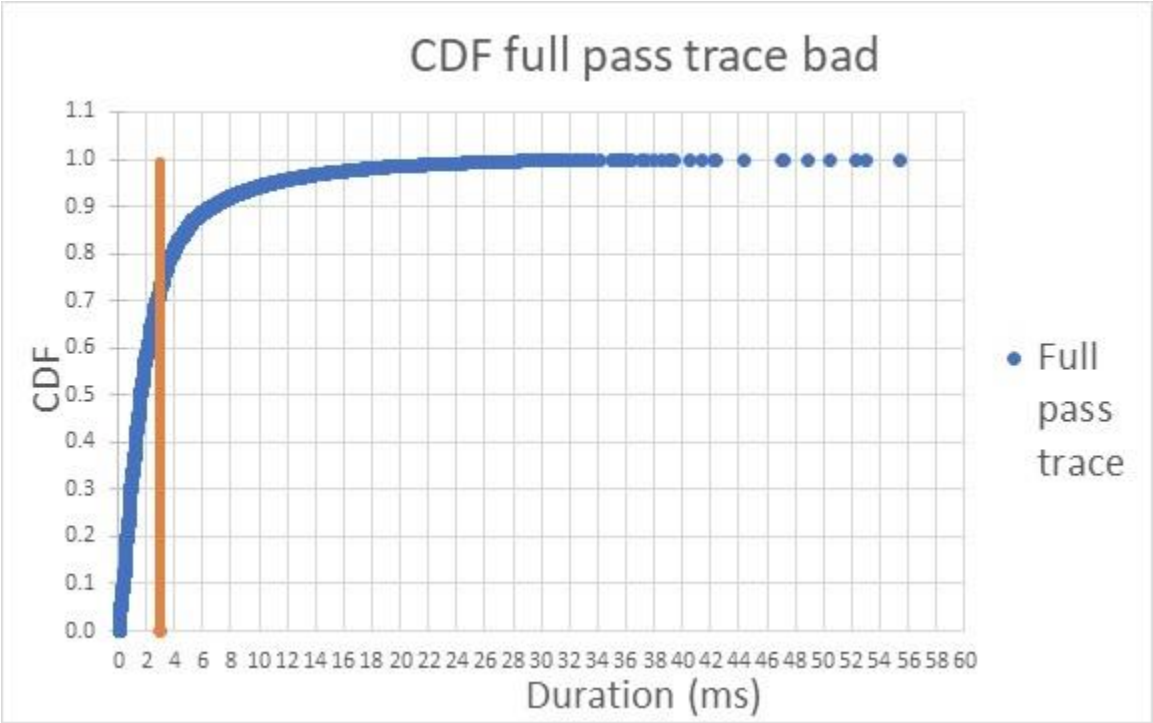

Figure 22: CDF of bad state durations in Full pass trace

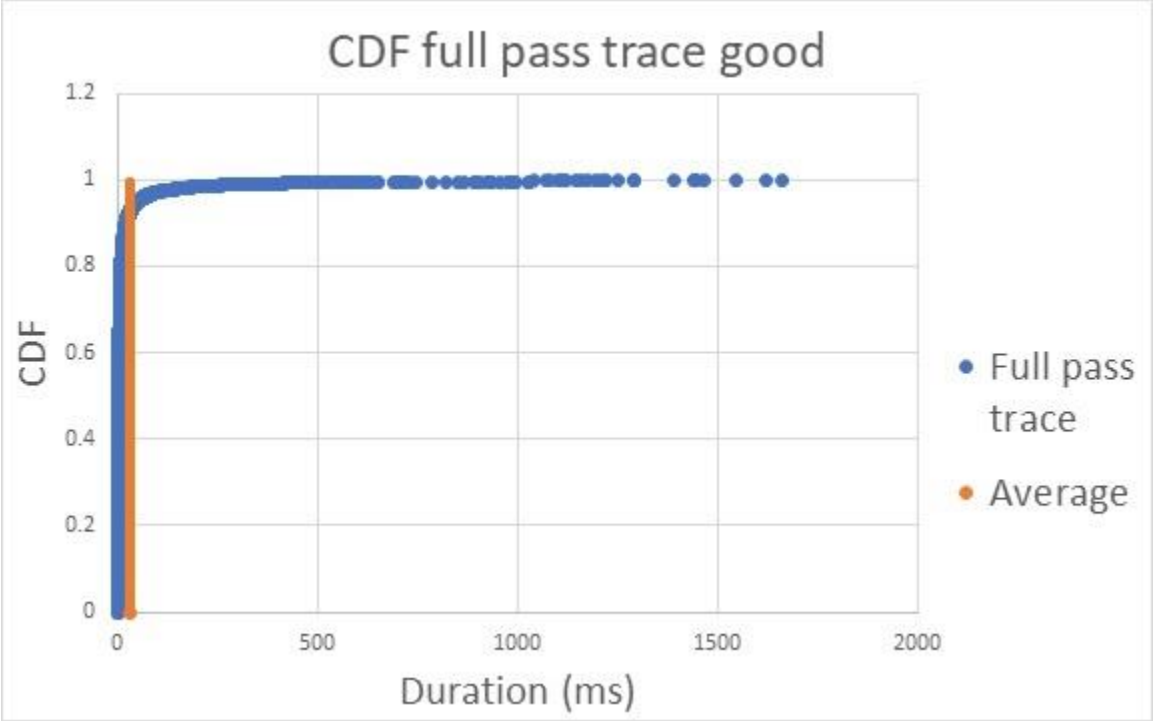

Figure 23: CDF of good state durations in Full pass trace

It has to be pointed out that in both the CDF and the histogram of good state the highest 43 duration values were not plotted for representation reasons as they outlie between 1.662 and 127.916 s of duration.

3. DETEMU

3.1 A NEW DETERMINISTIC CHANNEL EMULATOR: DETEMU

Once derived the erasure tracks, there is the need of a tool able to deterministically emulate this channel. Several existing network tools, like Netem [43] or Dummynet [44], have been evaluated, but none of these feature the opportunity of deterministically reproduce the desired channel as a function of time; most of the times, these tools are probabilistic and packet-based. An alternative solution, using scripts (e.g. in bash) for turning on and off the network interfaces during the transmission between a sender a receiver on the basis of input tracks, resulted not appropriate: the granularity of the tracks is of tenth of milliseconds, while measurements, carried out even on high-performant machines, highlighted that at least half a millisecond is needed for switching on/off the network interface. Therefore, it was necessary to design form scratch a software which could achieve the following two requirements:

- To be able to intercept frames and possibly drop them in accordance with the current state of the track (bad or good)
- To add a negligible delay

As the main purpose of this software is to emulate a two-state deterministic channel, it was called detemu [45]. It will be described later, after the description of the testbed.

3.1.1 Testbed design

Given the interest in achieving high data rates (hundreds of megabit/s), we opted for a physical testbed rather than a virtual one. Consequently, a physical testbed following the underlying scheme was set-up in the laboratory of the Institute of Communication and Navigation at DLR Oberpfaffenhofen. An almost identical copy was set up at ARCES (University of Bologna), for redundancy.

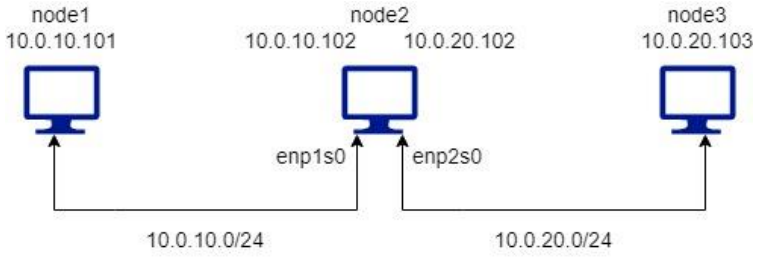

***Figure 24**: testbed schema*

The testbed is conceived such that there are three active hosts: node1, node2 and node3. Only node1 and node3 are DTN nodes (i.e. we have just one DTN hop in our testbed) and act as LTP sender and receiver. By contrast, node2 is an ordinary IP node, inserted just to have a machine entirely devoted to the channel emulation, i.e. to detemu.

A further requirement which has emerged after the first tests is that the central machine must rely on two real network cards. Indeed, a preliminary version of the testbed, based on a notebook instead of a desktop as central machine, was affected by packet disordering due to the USB NIC output card of the central machine. This packet disorder resulted in an unwanted behaviour of LTP. In particular, we observed that if an LTP transmission is affected by packet disorder, the checkpoint segment at the end of a block may arrive a little earlier than the last data segments of the same block; in this case, in the report segment triggered by the checkpoint the LTP receiver would erroneously consider as lost the data segments which are going to arrive, causing spurious retransmissions.

The definitive version of the testbed is based on a desktop as central machine, featuring two physical network interfaces. Therefore, node1 and node2, as well as node2 and node3, are directly connected through ethernet cables, and each network interface rely upon Gigabit network cards. Two networks are set up:

- 10.0.10.0/24 between node1 and node2 such that enp1s0 on node1 features the static IP address 10.0.10.101 and enp1s0 on node2 is 10.0.10.102
- 10.0.20.0/24 between node2 and node3 such that enp2s0 on node2 features the static IP address 10.0.20.102 and enp1s0 on node3 is 10.0.20.103

Therefore, all the traffic from node1 to node3 must pass through node2, on which the ipv4 forwarding must be enabled:

```
sudo sysctl -w net.ipv4.ip_forward=1
```

Resembling FSO satellite-to-ground station scenario, the downlink between satellite and ground station is simulated by the link from node1 to node3, while the uplink is conceived to be the connection from node3 to node1. Usually, in FSO scenarios, optical systems are engaged in the downlink only, which actually carries data, while on the reverse channel RF can be employed, being conceived for signaling traffic only. Therefore, the need is to characterize the downlink only, being of optical nature, which can be impaired as discussed in the introduction and described by the input tracks. Consequently, the testbed is set up such that the operating system of node2 will take care of forwarding the traffic from node3 to node1, but the traffic from node1 to node3, which is on behalf of detemu. To this end, the following iptables rule must be inserted on node2:

```
sudo iptables -I FORWARD -s 10.0.10.101 -j DROP
```

It basically means that all the traffic coming from node1 (10.0.10.101) conceived to be forwarded by node2 (because node2 is not the receiver of the traffic) will not actually be forwarded: it will simply stop at this hop. This step is essential because now a network traffic sniffer can capture packets on enp1s0 of node2, even if these packets are not intended for node2, and at the same time they will not reach node3 automatically; but having captured them, it is eventually possible to modify them and decide whether to send or drop them. This is exactly what detemu does.

3.1.2 Detemu overview

Given the testbed, the main purpose of detemu is to capture packets flowing on one interface, figure out if it is a packet to drop because it felt in a Bad state of the channel, or forward it to the other interface if it belongs to a Good state of the channel. Once the packet is captured, the dropping operation is straightforward as detemu must only discard it. On the other hand, if the packet must be forwarded to the receiver end of the transmission, some processing is required. When a packet travels from a machine A to a machine C passing through a node B, a packet arriving to the central host features the following:

- Source IP address: A
- Destination IP address: C
- Source MAC address: A
- Destination MAC address: B

The forwarding operation comprises:

- Changing the source MAC address to B
- Changing the destination MAC address to C
- Send the packet over the network according to the routing table

All of these operations must be consequently carried out by detemu when a packet falls in a Good state.

Therefore, the primary actions this software must provide are: capturing packets, discard them or eventually craft and send them over the network through a network interface. The full advantage of emulating the channel like this is that no interaction with the hardware is needed, and high-speed evaluation of packets is provided: according to some measurements, the forwarding operation requires at most few nanoseconds, therefore it is fully compatible with the erasure vector's deadline of each time interval (0.1ms).

3.1.3 Detemu implementation

The implementation phase of detemu started by searching for the existence of libraries for managing network packets as designed. This process led to PcapPlusPlus: a C++ library enabling fast packet processing with minimum overhead. It is designed to be efficient and allow a lightweight modality for capturing, reading and writing network packets [46].

Many benchmarks are available on the web regarding which library for parsing and crafting packets is the most efficient. As expected, libpcap-dev [47] is the best one in terms of performances (parsed packets per second), but it is very challenging to use its functionalities: it provides low-level API only, and anyone should need to write ad-hoc code to modify the contents of captured packets; this process can be complex as it would require full memory management in editing packet fields, resulting in longer coding times and less code readability and management. On the other hand, PcapPlusPlus provides simple and easy-to-use wrappers for libpcap. The main PcapPlusPlus wrappers/classed used for implementing detemu are:

- PcapLiveDevice: wraps libpcap functionality of capturing and sending packets and also provide information and statistics on the network interface
- RawPacket: holds the packet as raw data. This data is held as byte array. In addition to the data itself every instance also holds a timestamp representing the time the packet was received by the NIC. RawPacket instance is not read only: the user can change, add and remove packet data
- Packet: this class represents a parsed packet. It contains the RawPacket instance, and a linked list of layers; each layer is a parsed protocol that this packet contains. The layers linked list is ordered where the first layer is the lowest in the packet (currently it's always Ethernet protocol as PcapPlusPlus supports only Ethernet packets). An example of this linked list of layers could be: EthLayer -> IPv4Layer -> TcpLayer -> HttpRequestLayer. Packet instance is not read only: the user can add or remove layers, update current layer, etc.

The really small overhead in terms of performance introduced by PcapPlusPlus with respect to libpcap-dev dramatically simplifies the process of capturing and editing packets. Moreover, experimental results showed that forwarding packets from node1 to node3 with PcapPlusPlus instead of the native operating system forwarding fully keeps the desired data rate between the two nodes.

It is always possible to link a captured packet to its corresponding Bad or Good state in the tracks because the RawPacket class provide a timestamp representing the time the packet was received by the NIC.

Other benchmarked C/C++ libraries performed slightly worse than PcapPlusPlus and, as expected, projects written in higher level languages (such as Python or Java) cannot compete with these libraries [48]. Consequently, the C++ language was chosen to implement detemu.

The information this program needs by input from the user is:

- The name of the network interface where to capture incoming packets (here enp1s0)
- The name of the network interface where to ship outgoing packets (here enp2s0)
- Source IP address of the incoming packets to be forwarded (here 10.0.10.101)
- The new destination MAC address (the MAC address of the node2 can be retrieved by the name of the capturing interface thanks to the PcapLiveDevice class)
- Relative or absolute path of the erasure vector file (having a time granularity of 0.1ms) to follow

When the input parameters do not match these amount and types of parameters, a usage help is be printed out:

```
Usage: detemu <dev_capture> <dev_send> <src_ip_capture> <new_dst_mac_address> <ev_file>
```

A representation of the execution flow of detemu is provided in **Figure 25** and the corresponding steps are implemented as follows:

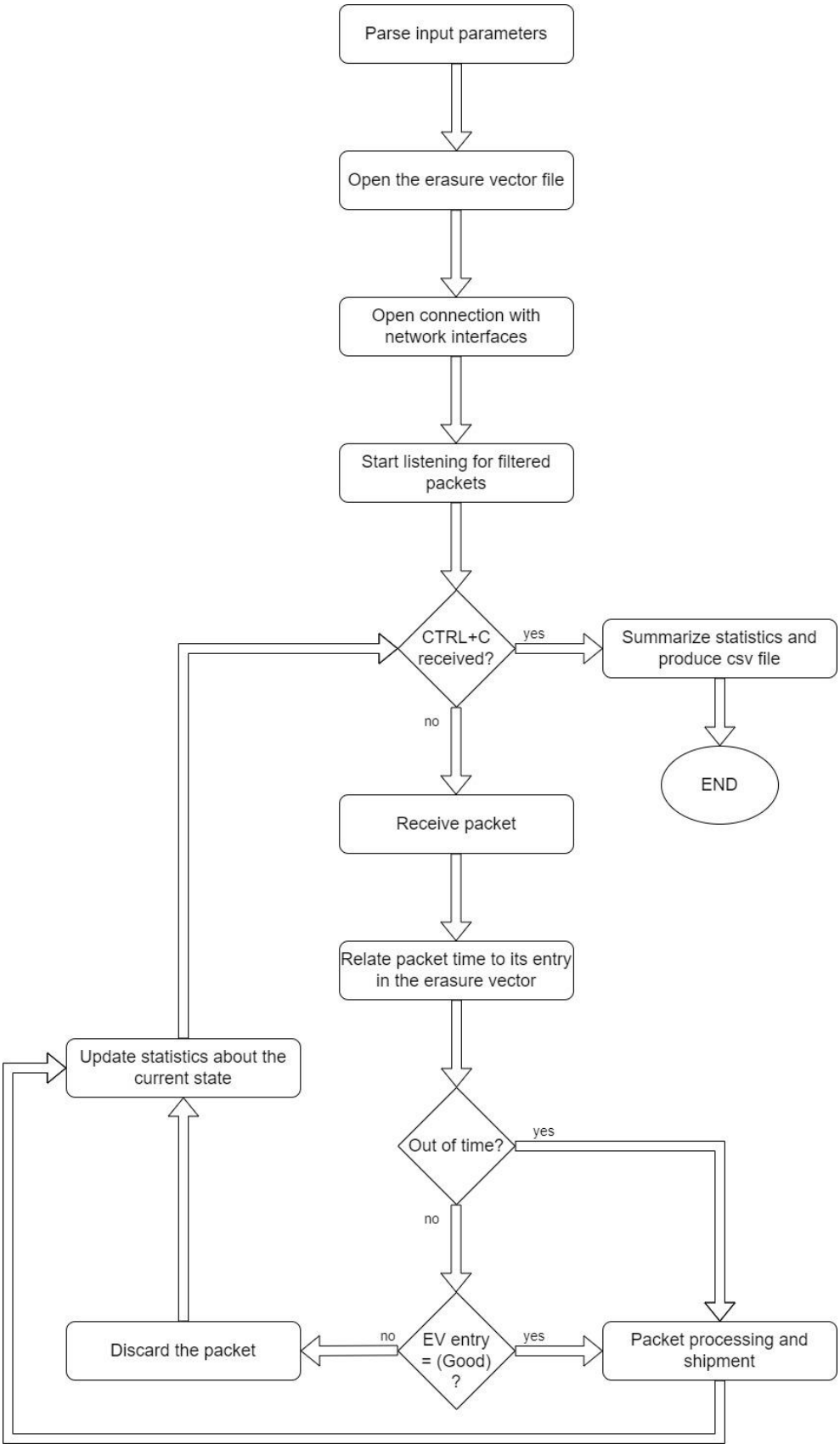

***Figure 25**: execution flow of detemu*

- Parse input parameters: each input to the program is catch in the exact same order reported on the help. If the format of one input is not correct, an error message is printed and the execution is stopped
- Open the erasure vector file: the *fopen* function is used to start reading at the erasure vector file and to open in append mode the csv file upon which reporting statics for the off-line evaluation of the channel behaviour
- Open connection with network interfaces: from the input strings describing the name of the interfaces where to capture and send packets, two PcapLiveDevice instances are created. Thanks to the sending PcapLiveDevice instance, the new source MAC address for the packet is automatically retrieved and no user parameter is needed
- Start listening for filtering packets: in this scenario, the software is interested in capturing UDP packets from 10.0.10.101. To this end, two new classes of PcapPlusPlus are used: IPFilter and ProtoFilter; all other packets are discarded and not considered in the statistics
- CTRL+C received condition: the program keeps executing until CTRL+C is pressed. When this happens, it prints out the following statistics:
    - Total number of lines of input file
    - Total duration of Good state (PER = 0) in ms
    - Total duration of Bad state (PER = 1) in ms
    - Bad state (PER = 1) duration ratio
    - Average duration of state Good in ms
    - Average duration of state Bad in ms

  A full report of how the channel behaved during the transmission is provided in a csv file called detemu_stats.csv where each record is related to one state and details:
    - PER of this state (0 or 1)
    - Sequence number of this state (e.g. B140 or G140)
    - Duration in ms of this state
    - Number of ethernet packets involved in this state
- Receive packet and relate its arrival time at the NIC to the erasure vector: when a packet is received, PcapPlusPlus allows to retrieve its timestamp with the getPacketTimeStamp() method of the RawPacket class. This timestamp is expressed in seconds plus a part in nanoseconds. Therefore, for each packet, its "relative arrival time" is computed: the timestamp of the first arrived packet is stated as start_time, and for the sequent frames, the relative time is computed as timestamp – start_time. This relative arrival time is therefore used to index the correct entry of the erasure vector file (having a time granularity of 0.1ms) as:

$$index = relative\_time\_sec * 10000 + relative\_time\_nsec / 100000$$

- Out of time condition: when the index computed from the relative arrival time is higher than the number of lines of the erasure vector, this packet is forwarded and marked as "out of time"; otherwise the next condition is evaluated
- EV entry = Good state condition: each time a packet arrives, its relative arrival time indexes the erasure vector entry to retrieve the state of the channel in which it falls. If

this entry is 0, it means that the PER is lower enough to let the packet be forwarded; if the entry is 1, it means that the PER is too much higher and the packet is discarded
- Packet processing and shipment: when a packet has to be forwarded to node3, the wrapper class EthLayer of PcapPlusPlus is used to retrieve the Ethernet layer of the packet and to change its source MAC address to the one of node2, and the destination MAC address to the one of node3. Consequently, the sending PcapLiveDevice instance is used to ship the packet
- Update statistics about the current state: for every state of the erasure vector, the number of involved packets is recorded. Moreover, when a packet indexing causes a state change, the duration of the just past state is computed as the number of its number of entries times 0.1 ms (granularity of the erasure vector)

3.1.4 Building detemu

The channel emulation software must be installed on the central machine of the testbed (node2) and makes usage of PcapPlusPlus in addition to several C++ development libraries. Thus, the following packages have to be installed as a prerequisite for building detemu:

- build-essential
- libpcap-dev
- latest version of PcapPlusPlus, which can be found at [46]

Once every prerequisite is met, it is possible to locally clone the git repository of detemu [45] and move into it to install the software:

- cd detemu
- make
- sudo make install

3.1.5 Detemu usage and demo

Using detemu, make sure the ipv4 forwarding is enabled just in one direction on node2:

```
sudo sysctl -w net.ipv4.ip_forward=1

sudo iptables -I FORWARD -s 10.0.10.101 -j DROP
```

Start detemu providing all the input parameters:

```
detemu enp1s0 enp2s0 10.0.10.101 aa:bb:cc:dd:ee:ff erasure_vector_file.txt
```

And with detemu running on node2, a simple transmission test can be achieved with iperf. On node3:

```
iperf -s -u -i1
```

On node1:

```
iperf -c 10.0.20.103 -u -t 10 -b100M
```

During the 10 seconds transmission test at 100Mbps, detemu on node2 prints real time statistics like follows:

```
PER = 1 (B236) - duration: 0.100000 ms - n. frame proc. = 2
PER = 0 (G237) - duration: 48.300000 ms - n. frame proc. = 430
PER = 1 (B237) - duration: 1.700000 ms - n. frame proc. = 15
PER = 0 (G238) - duration: 25.700000 ms - n. frame proc. = 229
PER = 1 (B238) - duration: 1.700000 ms - n. frame proc. = 16
PER = 0 (G239) - duration: 23.100000 ms - n. frame proc. = 206
PER = 1 (B239) - duration: 1.000000 ms - n. frame proc. = 8
PER = 0 (G240) - duration: 3.400000 ms - n. frame proc. = 30
```

Allowing for a real time monitoring of how packets are managed, e.g. in the 237th Good state, which had a duration of 48.3ms, 430 packets were processed (forwarded), while in the 237th Bad state, having a duration of 1.7ms, 15 packets were dropped.

3.1.6 Further functionalities

Other than these aspects, detemu has been provided with two additional features:

- PER setting: instead of using an erasure vector as track to follow during the transmission, a static PER can be given as an input. In this case, each received packet will be dropped with probability equal to PER, otherwise it will be forwarded. This feature may be useful to compare the differences between correlated and uncorrelated losses
- Packet mode interpretation: instead of using each packet arrival time to index the erasure vector file, the track can be interpreted "per packet". This means that the actual entry of the erasure vector is the description of how to treat the actual packet: 0, if it has to be forwarded, 1 if it has to be discarded. This feature can be activated using the "-p" option after the erasure vector filename in the inputs and may be useful to test corner cases of a protocol: in this thesis work it was used to test how different implementations of LTP behave when the final RA of the transmission is lost

Being designed to be extremely readable and modular, the code of detemu can always be enlarged with new functionalities. Moreover, it can be coupled with different network tools to achieve the desired characteristics in the testbed: for example, in this thesis work, detemu is sided by netem to insert a fixed propagation delay between node1 and node3 and vice versa.

3.1.7 Why detemu?

Atmospheric scintillations in optical communication scenarios are usually modelled by a 2-state Markov chain, a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. The 2-state Markov chain is also called Gilbert-Elliot model and consists, in this context, in describing the channel through 2 states and 4 probabilities: bad (B) and good (G) states, the probabilities of moving from B to G, from G to B, from B to B (remain in B) and from G to G

(remain in G). This model is also implemented in netem but it did not perform as expected. For example, considering the scenario A of [41], the average duration of state G is Tg = 18.175 ms, while the average duration of state B is Tb = 0.953 ms. The according 2-state Markov chain probabilities are:

$$p_{gg} = \frac{1}{1 + \frac{Tb}{Tg}} = 0.95$$

$$p_{gb} = 1 - p_{gg} = 0.05$$

$$p_{bb} = \frac{1}{1 + \frac{Tg}{Tb}} = 0.05$$

$$p_{bg} = 1 - p_{bb} = 0.95$$

This 2-state Markov chain model has been inserted within netem on the outgoing interface of node1 [49] :

```
sudo tc qdisc add dev eno1 root netem loss gemodel 0.05 0.95 0.05 0.95
```

While node2 was deprived of the iptables rule to let everything coming from node1 being passed to node3. Therefore, an iperf transmission was run at 500Mbps, having the following results on node3, being the iperf server (for concision only intervals 0-6 s, 47-53 s and 93 - 99 s are shown):

```
[  1] 0.0000-1.0000 sec  60.8 MBytes   510 Mbits/sec   0.029 ms 394/43752 (0.9%)
[  1] 1.0000-2.0000 sec  60.3 MBytes   506 Mbits/sec   0.043 ms 384/43426 (0.88%)
[  1] 2.0000-3.0000 sec  61.0 MBytes   512 Mbits/sec   0.033 ms 370/43899 (0.84%)
[  1] 3.0000-4.0000 sec  60.6 MBytes   509 Mbits/sec   0.039 ms 373/43628 (0.85%)
[  1] 4.0000-5.0000 sec  60.8 MBytes   510 Mbits/sec   0.035 ms 417/43761 (0.95%)
[  1] 5.0000-6.0000 sec  60.7 MBytes   509 Mbits/sec   0.031 ms 373/43669 (0.85%)
…
 [  1] 47.0000-48.0000 sec  60.6 MBytes   509 Mbits/sec   0.035 ms 413/43658 (0.95%)
[  1] 48.0000-49.0000 sec  60.7 MBytes   509 Mbits/sec   0.034 ms 384/43692 (0.88%)
[  1] 49.0000-50.0000 sec  60.7 MBytes   509 Mbits/sec   0.039 ms 431/43697 (0.99%)
[  1] 50.0000-51.0000 sec  60.7 MBytes   509 Mbits/sec   0.033 ms 390/43677 (0.89%)
[  1] 51.0000-52.0000 sec  60.8 MBytes   510 Mbits/sec   0.026 ms 412/43749 (0.94%)
[  1] 52.0000-53.0000 sec  60.6 MBytes   509 Mbits/sec   0.038 ms 385/43633 (0.88%)
…
 [  1] 92.0000-93.0000 sec  60.7 MBytes   509 Mbits/sec   0.036 ms 397/43708 (0.91%)
[  1] 93.0000-94.0000 sec  60.6 MBytes   509 Mbits/sec   0.077 ms 382/43640 (0.88%)
```

```
[  1] 94.0000-95.0000 sec  60.8 MBytes   510 Mbits/sec   0.031 ms 412/43747 (0.94%)
[  1] 95.0000-96.0000 sec  60.7 MBytes   509 Mbits/sec   0.019 ms 385/43665 (0.88%)
[  1] 96.0000-97.0000 sec  60.7 MBytes   509 Mbits/sec   0.027 ms 419/43699 (0.96%)
[  1] 97.0000-98.0000 sec  60.7 MBytes   509 Mbits/sec   0.036 ms 390/43698 (0.89%)
[  1] 98.0000-99.0000 sec  60.7 MBytes   509 Mbits/sec   0.031 ms 350/43666 (0.8%)
```

Removing the netem rule regarding the Gilbert-Elliot model, inserting back the iptables rule to give detemu full control of forwarding and starting it on node2 to emulate scenario A, the same iperf transmission was performed, giving the following results node3, being the iperf server (for concision only intervals 0-6 s, 47-53 s and 93 -99 s are shown):

```
[  1] 0.0000-1.0000 sec  59.0 MBytes   495 Mbits/sec   0.031 ms 1691/43761 (3.9%)
[  1] 1.0000-2.0000 sec  58.1 MBytes   488 Mbits/sec   0.028 ms 2223/43685 (5.1%)
[  1] 2.0000-3.0000 sec  58.5 MBytes   491 Mbits/sec   0.023 ms 1948/43712 (4.5%)
[  1] 3.0000-4.0000 sec  58.2 MBytes   488 Mbits/sec   0.029 ms 2244/43741 (5.1%)
[  1] 4.0000-5.0000 sec  58.9 MBytes   494 Mbits/sec   0.030 ms 1686/43684 (3.9%)
[  1] 5.0000-6.0000 sec  58.5 MBytes   491 Mbits/sec   0.032 ms 1931/43663 (4.4%)
…
[  1] 47.0000-48.0000 sec  58.3 MBytes   489 Mbits/sec   0.033 ms 1992/43601 (4.6%)
[  1] 48.0000-49.0000 sec  58.1 MBytes   487 Mbits/sec   0.106 ms 2348/43780 (5.4%)
[  1] 49.0000-50.0000 sec  57.9 MBytes   486 Mbits/sec   0.054 ms 2374/43693 (5.4%)
[  1] 50.0000-51.0000 sec  58.5 MBytes   490 Mbits/sec   0.048 ms 1995/43693 (4.6%)
[  1] 51.0000-52.0000 sec  58.3 MBytes   489 Mbits/sec   0.041 ms 2230/43789 (5.1%)
[  1] 52.0000-53.0000 sec  58.5 MBytes   491 Mbits/sec   0.046 ms 1869/43590 (4.3%)
…
[  1] 93.0000-94.0000 sec  58.2 MBytes   488 Mbits/sec   0.025 ms 2173/43675 (5%)
[  1] 94.0000-95.0000 sec  58.3 MBytes   489 Mbits/sec   0.055 ms 2139/43690 (4.9%)
[  1] 95.0000-96.0000 sec  58.2 MBytes   488 Mbits/sec   0.024 ms 2190/43687 (5%)
[  1] 96.0000-97.0000 sec  58.0 MBytes   487 Mbits/sec   0.039 ms 2292/43684 (5.2%)
[  1] 97.0000-98.0000 sec  57.6 MBytes   483 Mbits/sec   0.038 ms 2624/43684 (6%)
[  1] 98.0000-99.0000 sec  58.8 MBytes   493 Mbits/sec   0.033 ms 1730/43684 (4%)
```

Error rates are severely different in all phases of the transmission emulation. The 2-state Markov chain with netem approach gives an overall PER of 0.9%, in contrast to the 5% of detemu deterministically emulating scenario A. Reasons of these differences, out of implementation considerations, have to be mostly found in the fact that netem works in a packet-based probabilistic way, while detemu is deterministically time-based. Moreover, a 2-state Markov chain is not able to capture the dynamics of optical link scintillation, which is the result of a superposition of different stochastic process, whereby Markovian properties can be hardly observed.

4. LTP PERFORMANCE ANALYSER

Being able to emulate the typical conditions of FSO links, there is the need of analysing how the Licklider Transmission Protocol behave upon these. The purposes of this thesis are producing general benchmarks of the protocol and trying to understand which are crucial LTP parameters to best harness the channel.

To this end two main processes have been employed: capturing all the segments sent from the transmission side with wireshark [50], tshark or tcpdump, and using a Python script to analyse afterwards these captures, producing several metrics and statistics. For this reason, the name of this program is LTP performance analyser (available at [51]).

Just before starting the LTP transmission, tshark is launched on node1 to sniff all traffic on the outgoing interface to node2. At the end of the capture, the according pcap file is saved locally and used as input for LTP performance analyser.

4.1 A NEW PYTHON SCRIPT FOR LTP ANALYSIS

A Python program has been realized to synthetize performance metrics of the LTP transmission from its pcap capture file. Usually, these pcap files have huge sizes, especially when dealing with typical transmissions considered in this work, having durations of 100 or 692 seconds. Therefore, the first step of the Python script is to filter from the input pcap file the only data it needs and print it on a .txt file.

4.1.1 Filtering pcap captures

For each LTP packet, LTP performance analyser needs:

- The timestamp of the packet
- The LTP type, having a number which describes whether the LTP segment is:
    - 0: Red data, NOT {Checkpoint, EORP or EOB}
    - 1: Red data, Checkpoint, NOT {EORP or EOB}
    - 2: Red data, Checkpoint, EORP, NOT EOB
    - 3: Red data, Checkpoint, EORP, EOB
    - 4: Green data, NOT EOB
    - 5: Green data, undefined
    - 6: Greed data, undefined
    - 7: Green data, EOB
    - 8: Report segment
    - 9: Report-acknowledgment segment
    - 10: Control segment, undefined
    - 11: Control segment, undefined
    - 12: Cancel segment from block sender
    - 13: Cancel-acknowledgment segment to block sender
    - 14: Cancel segment from block receiver

    - 15: Cancel-acknowledgment segment to block receiver
- The session numbers
- The checkpoint serial number, if it is a segment of type 1, 2, or 3
- The claim count, if the segment is a report segment (type 8)
- The offset of the segment in the session, used as segment unique identification number
- The originator LTP engine number

In order to retrieve these data from the input pcap file, the following python-os command, which makes usage of tshark, is launched:

```
tshark -r capture_file.pcap -Y ltp -Tfields -e frame.time.relative -e ltp.type -e
ltp.session.orig -e ltp.session.number -e ltp.data.offset -e ltp.data.chkp -e
ltp.rpt.clm.cnt > filterd.txt
```

Basically, the tshark functionality of filtering only ltp packets (-Y ltp) and the desired fields (-Tfields -e <name_of_field>) is used from Python. To this end, the Python "os" library is used for launching a command as if it is executed from the bash command line.

4.1.2 Metrics computation

After the filtering step, the Python script has been programmed to parse the txt file including the filtered information and computing the following LTP metrics:

- Total number of sessions: total number of sessions successfully carried out (not cancelled) during the captured transmission
- Average session duration: average of the duration of all sessions (but the cancelled). Each duration is computed as the time interval between the first segment and the first report segment confirming all the data of the session is arrived
- Average number of rtx cycles per session: the number of retransmission cycle for each session is computed as the number of occurrences of the most frequent data segment in the session. Each data segment is identified by its offset in the block
- Average number of data segments per rtx cycle: for each session, the occurrences of replicated data segments are counted. Each time a segment is replicated, it coincides with a rtx cycle. The result of this count is a Python dictionary where, for each session number, a list is stored like the following → session_number: [15, 10, 10, 5, 2] which means that for this particular session (identified by session_number) the first rtx cycle corresponds to 15 retransmitted data segments, the second rtx cycle corresponds to 10 data segments, and so on. The average number of data segments per rtx cycle is therefore computed as the average among all the numbers in the list of all the sessions
- Total number of lost CPs: for each session, the serial number of the encountered checkpoints are recorded. A lost CP is stated if more than 1 occurrence of the same CP serial number is present in this list
- Average number of lost CPs per session: the numbers of lost CPs for each session are stored, and so the mean is computed among all sessions

- Average number of RAs per session: the numbers of RAs for each session are stored, and so the mean is computed among all sessions
- Average penalization time per session: the penalization time of a session is computed as the time between the last and the first CPs encountered for this session. So, the average is computed among all sessions
- Total number of cancelled sessions: number of sessions in which a Cancel Segment has been sent from the sender

These data are printed out from the Python program when its computation ends and saved in a csv file called "*brief_summary.csv*" such that it is easy to copy and aggregate data of different transmission tests.

	A	B
1	Tot_num_sessions	10048
2	avg_session_dur	0.165442331
3	tot_num_rtx_cycles	4638
4	avg_num_rtx_cycles	1.109303994
5	avg_data_seg_rtx_cycle	41.74471755
6	tot_num_lost_cp	765
7	avg_lost_cp_per_session	0.076134554
8	avg_ra_per_session	1.52647293
9	avg_penaliz_per_session	0.090196561
10	avg_penaliz_per_session_only_losses	0.215835923
11	tot_num_canc_sessions	4
12	avg_num_cs_per_canc_session	2.5

Figure 26: exemplar metrics in csv of LTP performance analyser

In addition, a summarizing csv file with a line for each session is produced from the Python dictionaries containing all data upon which the above metrics are computed. Then name of this csv file is decided by the user and given as input to LTP performance analyser. An exemplar portion of the described csv file is showed in Figure 27 (opened with Microsoft Excel).

It worth to be pointed that using this script is possible only after having installed the latest version of Python3, pip and numpy, like follows:

```
sudo apt-get install –upgrade python3

sudo apt-get install python3-pip

pip3 install numpy
```

And the script (ltp_performance_analyser.py) can be executed with:

```
python3 ltp_performance_analyser.py <ltp_originator_number>
<plain_text_capture_file.txt> <output_filename.csv>
```

	A	B	C	D	E	F	G	H
1	Session	Duration(sec)	Losses	Rtx_cycles	Segments_per_cycle	Lost_CP	Num_RA	Penalization(sec)
2	1	0.017263153	No	0	0	0	1	0
3	2	0.018809672	No	0	0	0	1	0
4	3	0.017551862	No	0	0	0	1	0
5	4	0.032822545	Yes	1	28	0	2	0.014280226
6	5	0.018593899	No	0	0	0	1	0
7	6	0.259768534	Yes	3	111-20-15	2	4	0.240732699
8	7	0.034594153	Yes	1	30	0	2	0.015066037
9	8	0.018448677	No	0	0	0	1	0
10	9	0.020162389	No	0	0	0	1	0
11	10	0.016960289	No	0	0	0	1	0
12	11	0.029994367	Yes	1	38	0	2	0.01429676
13	12	0.018227278	No	0	0	0	1	0
14	13	0.01887229	No	0	0	0	1	0
15	14	1.117312207	No	0	0	0	2	0
16	15	0.01712438	No	0	0	0	1	0
17	16	0.030725116	Yes	1	40	0	2	0.014212759
18	17	0.017440828	No	0	0	0	1	0
19	18	0.01730063	No	0	0	0	1	0
20	19	0.032749894	Yes	1	15	0	2	0.013076304
21	20	0.016741043	No	0	0	0	1	0
22	21	0.018448645	No	0	0	0	1	0
23	22	0.01817743	No	0	0	0	1	0
24	23	0.018495618	No	0	0	0	1	0
25	24	0.130340132	Yes	1	24	1	2	0.111909948
26	25	0.029996659	Yes	1	6	0	2	0.01254056
27	26	0.01699297	No	0	0	0	1	0
28	27	0.029144529	Yes	1	36	0	2	0.012069391
29	28	0.01831729	No	0	0	0	1	0
30	29	0.018417134	No	0	0	0	1	0
31	30	0.017369982	No	0	0	0	1	0
32	31	0.030652025	Yes	1	27	0	2	0.013290832
33	32	1.116341458	No	0	0	0	2	0
34	33	0.018648125	No	0	0	0	1	0
35	34	0.030648312	Yes	1	15	0	2	0.01305558
36	35	0.017317675	No	0	0	0	1	0
37	36	0.018788317	No	0	0	0	1	0
38	37	0.016233203	No	0	0	0	1	0
39	38	0.029030538	Yes	1	42	0	2	0.011957866
40	39	1.117689601	No	0	0	0	2	0
41	40	0.018460747	No	0	0	0	1	0
42	41	0.0306672	Yes	1	60	0	2	0.014026197
43	42	0.016947801	No	0	0	0	1	0

Figure 27: exemplar csv output of LTP performance analyser

As said, pcap files of long transmissions can be extremely high sized. For each captured packet, indeed, it features all information about every used protocol and the payload, like showed here (Wireshark screenshot; it is also possible to enlarge the dissection of a protocol to read all its data):

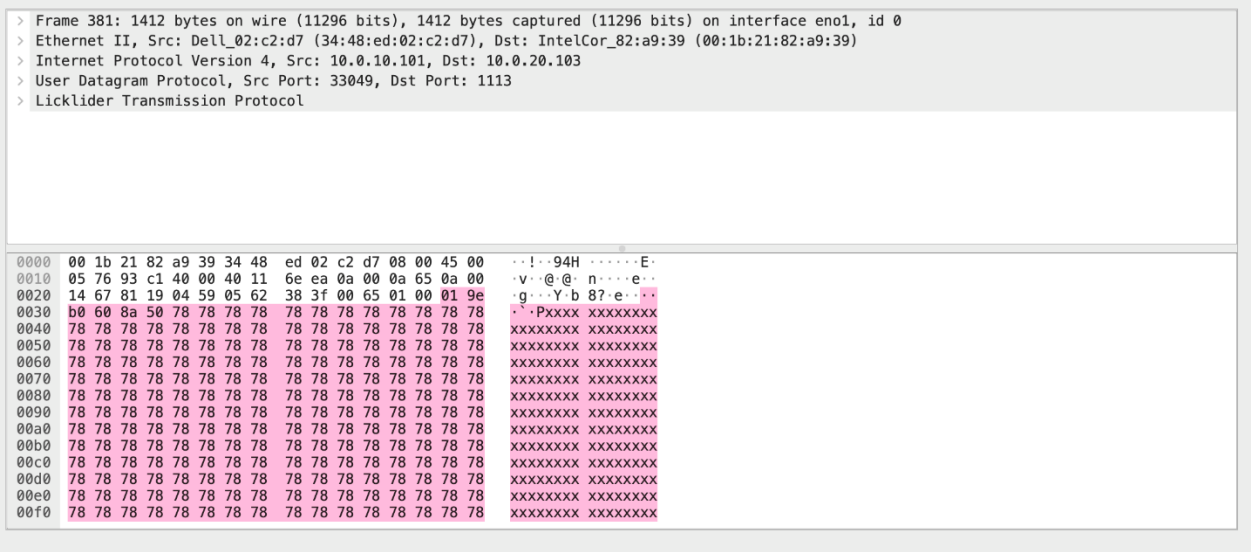

Figure 28: pcap capture of a generic LTP segment opened in Wireshark

Consequently, a typical pcap file of a long transmission (e.g. 10 minutes) has a size of more than 10 Gbyte. This can significantly slow down the time needed by LTP performance analyser, especially in its filtering step.

4.2 UNIBO-LTP VARIANT

To work with files which are easier to manage and reduce the time needed by LTP performance analyser, a variant of this program has been produced to work with the typical Unibo-LTP log of an LTP transmission. The only difference with the base version is only the way of parsing the information required to compute the metrics.

An exemplar transmission log file of Unibo-LTP is showed:

```
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 0; Length: 1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 1024; Length:
1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 2048; Length:
1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 3072; Length:
1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 4096; Length:
1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 5120; Length:
1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 6144; Length:
1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 7168; Length:
1024
```

```
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 0; RedData; Offset: 8192; Length:
1024
Time: 1675245629463; SessOrig: 108; Sess#: 1; Type: 3; Checkpoint; Offset: 9216;
Length:872; CP: 1; RS: 0
Time: 1675245632727; SessOrig: 108; Sess#: 1; Type: 3; Checkpoint; Offset: 9216;
Length:872; CP: 1; RS: 0
Time: 1675245635728; SessOrig: 108; Sess#: 1; Type: 3; Checkpoint; Offset: 9216;
Length:872; CP: 1; RS: 0
Time: 1675245638728; SessOrig: 108; Sess#: 1; Type: 3; Checkpoint; Offset: 9216;
Length:872; CP: 1; RS: 0
Time: 1675245641729; SessOrig: 108; Sess#: 1; Type: 3; Checkpoint; Offset: 9216;
Length:872; CP: 1; RS: 0
Time: 1675245644729; SessOrig: 108; Sess#: 1; Type: 12; CS
Time: 1675245647730; SessOrig: 108; Sess#: 1; Type: 12; CS
Time: 1675245650730; SessOrig: 108; Sess#: 1; Type: 12; CS
Time: 1675245653730; SessOrig: 108; Sess#: 1; Type: 12; CS
Time: 1675245656731; SessOrig: 108; Sess#: 1; Type: 12; CS
```

For a transmission of about 10 minutes, Unibo-LTP produces a log file of 3 Gbyte, more than three times lighter than the corresponding pcap file. The reason behind this save of size is that Unibo-LTP, for each LTP segment, just produces a line with the essential information, while the pcap file report data about all protocols included in a packet, even for non-LTP ones.

5. HARDWARE AND SOFTWARE SETUP

5.1 HARDWARE CHARACTERISTICS

All experiments are performed on two almost identical testbeds, as mentioned in chapter 2. The one built in the DLR laboratory, at the Institute of Communication and Navigation, has the following hardware characteristics:

- Node1: notebook Dell Precision 7540, 64 GB of RAM, Intel i7 @2.60 GHz
- Node2: desktop computer Intel, 4 GB of RAM, Intel i5 @3.50GHz
- Node3: notebook Dell Precision 7540, 16 GB of RAM, Intel i7 @2.60 GHz

Node1 and node3 feature 1 Gigabit Ethernet network interfaces, while node3 features 2 network interfaces, each being 1 Gigabit Ethernet.

The one at University of Bologna, mainly used as backup, consists of three identical machines:

- HP Desktop 6300 Pro Small Form Factor, I5, 16MB of RAM, @3GHz, Gigabit Ethernet NIC cards

5.2 SOFTWARE SETUP

All machines run the GNU/Linux distribution Ubuntu 22.04 plus other software as detailed below.

1.6.1 LTP implementations used

Tests were mainly carried out by using the LTP implementation included in DTNME v1.2.0, released in November 2022. We also experimentally used the new version of Unibo-LTP and the brand-new Unibo-BP, both released at the beginning of February, on the Unibo testbed, as a benchmark. ION-LTP was tested in a preliminary phase, but later abandoned in favour of DTNME LTP, because it could not reach data rates higher than 70 Mbps on an ideal channel, which was too little to match the speed for which channel traces had been obtained (100 and 500 Mbit/s).

1.6.2 Chanel emulator

Detemu, developed in this thesis, was used on node2, to emulate the alternance of good and bad states typical of the FSO channel. It used in input the erasure vectors mentioned in previous chapters. The 10ms delay was enforced by means of Linux “tc-netem” command.

1.6.3 DTNperf and other software

DTNperf version 4 was used as a client on node1, i.e. to generate the desired bundle traffic, and as a server on node3.

Wireshark and Tshark were used on node1 to capture all the traffic between node 1 an node 2. These traces were later analysed by means of the Python scripts developed in this thesis and described in previous chapters.

5.3 DTNME CONFIGURATIONS

Tests of 100 seconds of transmission are carried out to evaluate how LTP behave over typical FSO scenarios A, F, H introduced earlier. On the other hand, transmission tests of 692 seconds regard the Full pass scenario.

These scenarios are designed by the Optical Communication Department at DLR – KN accounting 1 Gbit/s of data rate between the sender (satellite) and the receiver (optical ground station) for the first three scenario and 100 Mbit/s for the latest one.

Several experiments showed that performing LTP transmissions at data rate higher that 550 Mbps over an ideal channel on our testbed would result in spurious losses either on the network interface of node1 or the one of node3. Therefore, the DTNME daemon was configured to achieve a maximum UDP data rate of 440 Mbps in case of scenarios A, F, H. This reduction, however, does not affect the validity of the results as there is just a factor 2 between the real and the simulated data rate.

No data rate reduction, of course, is needed with Full pass scenario.

Overall, the following DTNME parameters (introduced in section 1.3.4) are common to every scenario:

```
Inact_intvl=30, retran_retries=5, agg_size=1000 agg_time=1
seg_size=1360 use_files=false bucket_depth=524280 bucket_type=0
buffer_size=0
```

In each test the metrics produced by the Python program LTP performance analyser have been evaluated varying the retransmission time out (RTO) and the maximum number of parallel session (from now on called "P").

6. NUMERICAL RESULTS

Statistics collected by the Python program LTP performance analyser are aimed at evaluating changes of the outcome metrics, as well as of the goodput, with the variation of the RTO and the P (maximum number of parallel sessions). Since we are using DTNperf in rate-based mode, the goodput is computed as follows:

$$goodput = \frac{num\_confirmed\_sessions \ * bundle\_payload\_size}{transmission\_duration}$$

Where:

- num_confirmed_sessions is the number of sessions which were successfully carried out, i.e. for which no Cancel Segment has been found
- bundle_payload_size is the amount of payload data included in each shipped bundle
- transmission_duration is the time for which the transmission lasts, which is given by the number of lines of the input erasure vector to detemu (number of lines * 0.1ms)

In particular, after the nominal transmission duration the DTNME daemon is shut down on the receiver side, such that no more sessions are completed after this time. For all scenarios, RTOs of 0.1 seconds and 1.1 seconds have been considered, and, for each RTO, four configurations of P are studied: 1, 5, 20, 40.

6.1 SCENARIOS A, F, H

6.1.1 Goodput

In scenarios A, F and H the DTNperf client performs transmissions in rate-based modality, sending data at a nominal speed of 410Mbps. The goodput results are showed in Figure 29,Figure 30 and Figure 31.

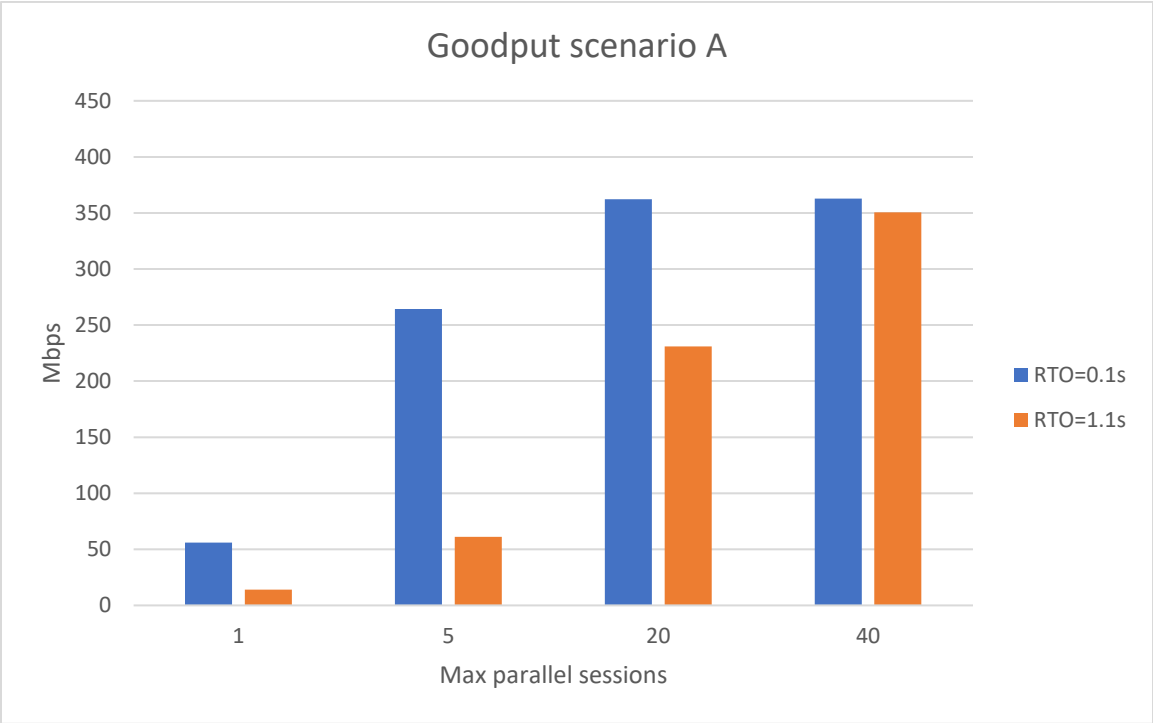

Figure 29: goodput variation on scenario A

The shorter RTO provides positive results among all cases: the goodput with RTO=0.1s is always higher or equal than the one with RTO=1.1s on every scenario and with every number of P. On the other hand, increasing the parallelism counteracts the high number of losses in every scenario and this is particularly noticeable when the RTO is high. Both these results find their explanation with the fact that the average fading duration of all scenarios are in the order of few milliseconds (0.95ms for A, 2.14ms for F, 5.45ms for H); therefore, when a segment is lost it is not worth to wait for a long period of time before retransmitting or transmitting something else.

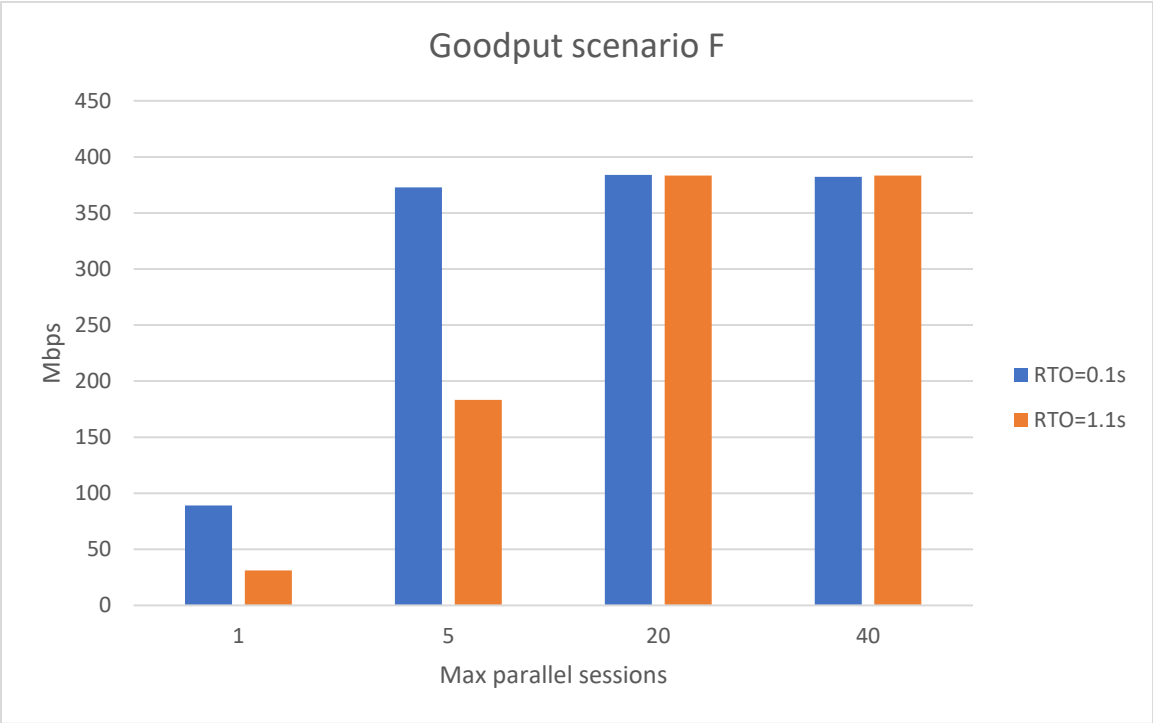

Figure 30: goodput variations on scenario F

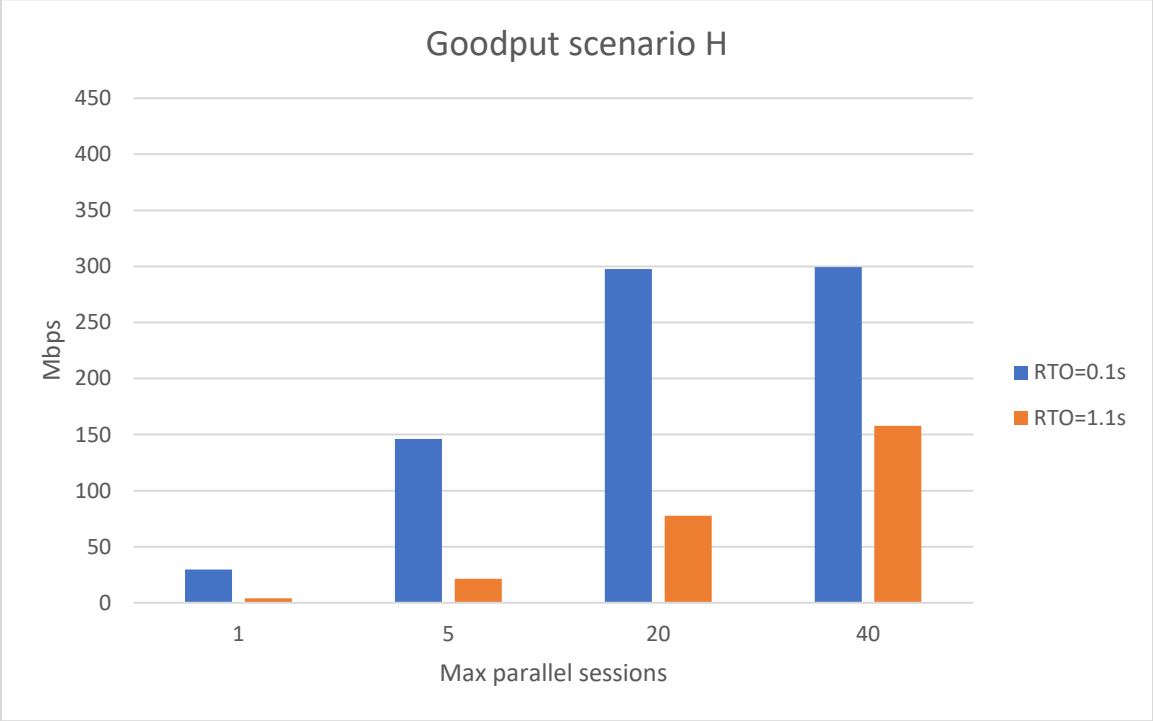

Figure 31: goodput variations on scenario H

6.1.2 Penalization time

For all scenarios, varying the maximum number of parallel sessions produces no significant changes on the penalization time (defined in 4.1.2). Therefore, in Figure 32, the percentage of

average penalization time over the average session duration is showed for each scenario, considering P = 20.

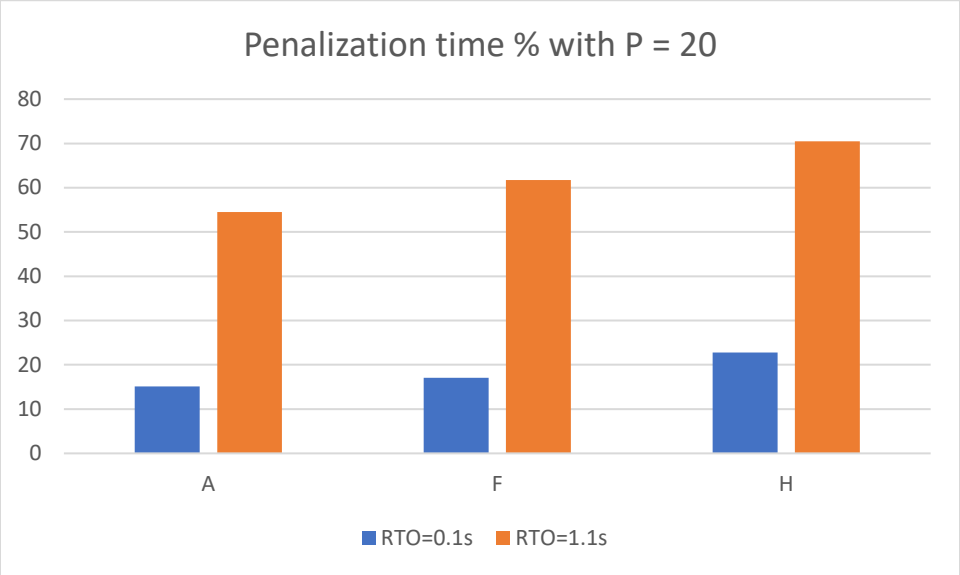

Figure 32: penalization time % of scenarios A, F and H with P=20

In all three scenarios:

- The average penalization time when RTO=1.1s is always more than three times higher the average penalization time when RTO=0.1s
- The average penalization time does not significantly vary when the parallelism does so, as already introduced

The reason behind the first point is clearly related to the losses of the CP segments and the extra time the sender waits when the RTO is higher.

The reason behind the second point instead is more complex and depends on the bundle dimension, the transmission data rate and the RTT. Defining the *irradiation time* as the time needed by the sender to inject all the segments of a block in the network, in scenarios A, F and H the irradiation time coincides with the RTT: the bundle dimension is 500 Kbyte, i.e. 4 Mbit, that requires 10 ms to be irradiated at 400 Mbps. To move further, let us consider the cases of one session and two parallel sessions which suffered 4 retransmission cycles each, as showed in Figure 33 and Figure 34, respectively.

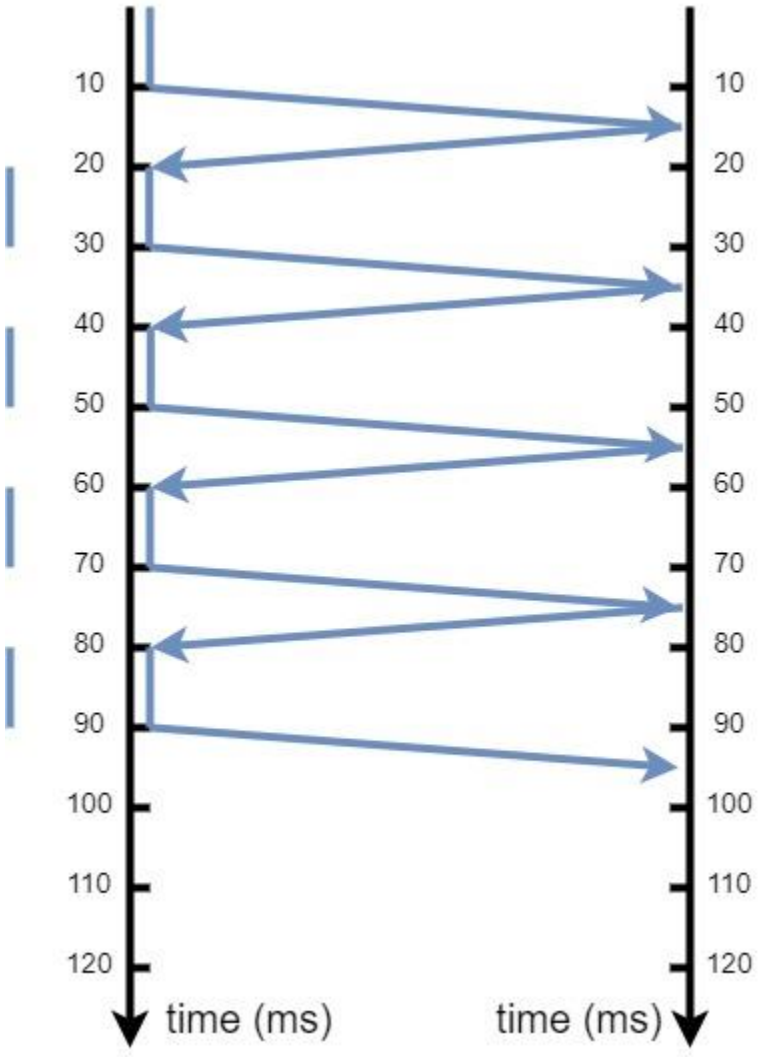

Figure 33: example of a single session which suffered of 4 retransmission cycles. Block dimension = 500Kbyte, data rate = 400Mbps

As stated in chapter 3, the penalization time of one session is computed as the time between the last and the first CPs encountered for this session. Considering the penalization time of one retransmission cycle, it can be looked as the sum between the time needed for receiving the claiming RS, the time needed by the sender to irradiate the lost segments and the one-way propagation delay. The left side segments out of the temporal lines of Figure 33 and Figure 34 represent the central components of this sum, which in these scenarios coincides with the irradiation times comprised in the retransmission cycles. As it can be observed, these time intervals are not subject to variations when passing from one single session to two parallel sessions. For instance, the blue segments of Figure 33 have the same length of the blue and yellow segments of Figure 34. As anticipated, this is due to the fact that, in scenarios A, F and H the irradiation time coincides with the RTT and when a RS which claims lost data segments arrives to the sender side of the communication, it has just finished to transmit (or retransmit) all the segments related to the parallel session. As exposed in the related

paragraph of the Full pass scenario, this is not true when the irradiation time is different from the RTT, causing a variation in the penalization times when the parallelism does so.

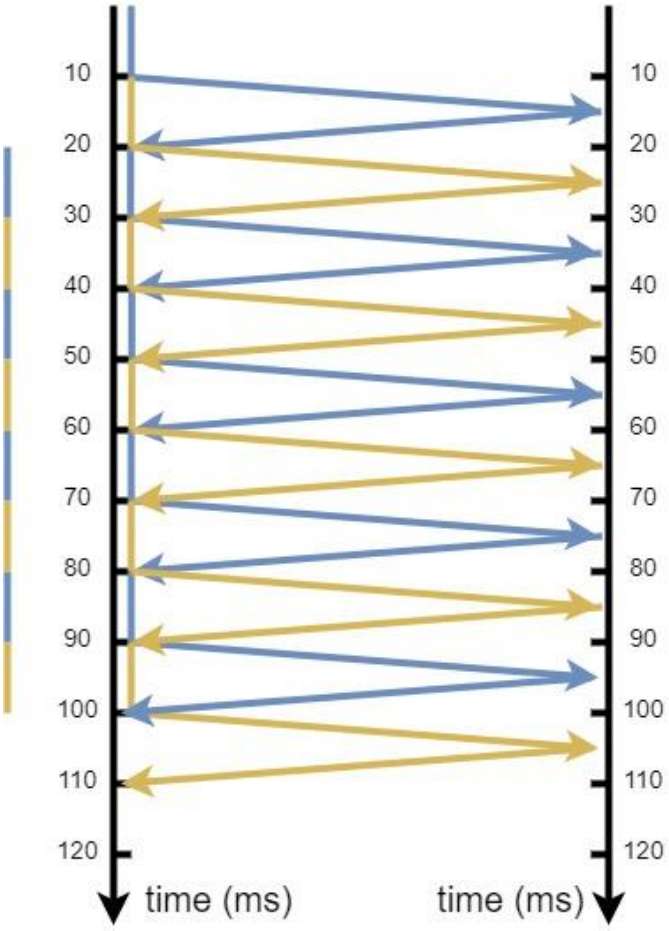

Figure 34: example of 2 parallel sessions which suffered of 4 retransmission cycles each. Block dimension = 500Kbyte, data rate = 400Mbps

6.1.3 Other metrics

All the experiments show that the **average session duration** naturally depends on the average penalization time per session. In absence of losses all the sessions would have almost the same duration. Losses on data or signaling segments result if penalizations. Since in these scenarios the average penalization time does not change when the parallelism does so, the same behaviour is present on the average session duration, which is the same with different levels of parallelism, and change only if the RTO varies. An illustration of the different average session durations with respect to the diverse RTO in the three scenarios is showed in Figure 35.

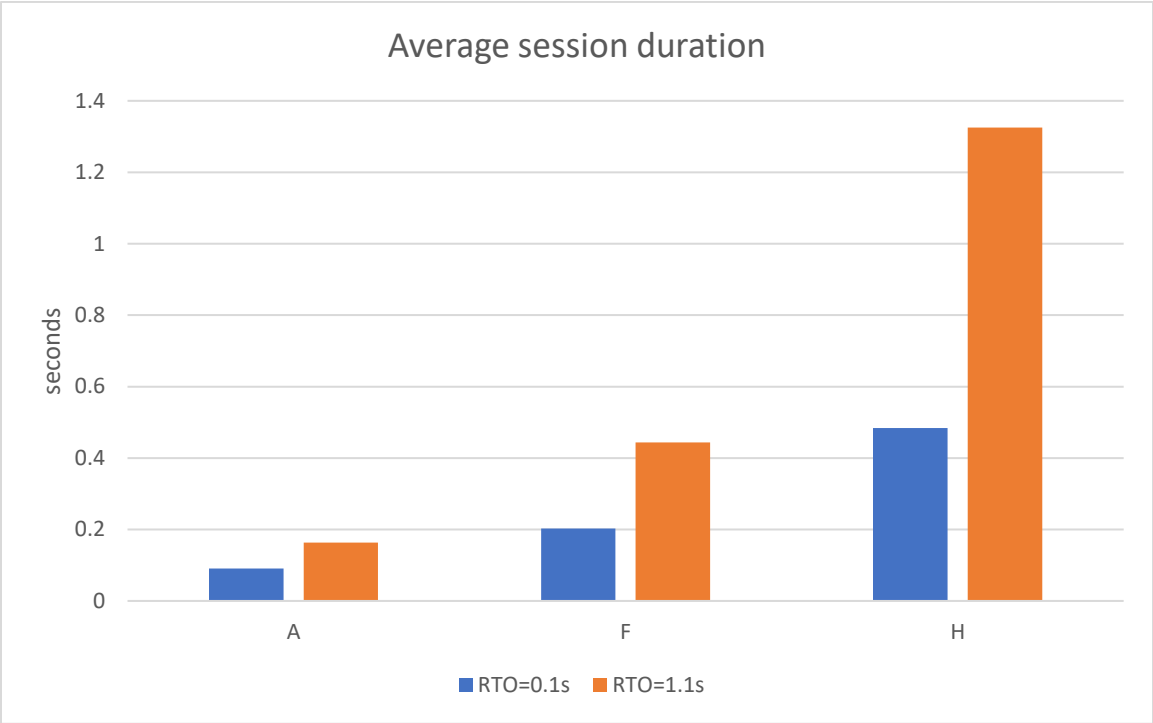

Figure 35: average session duration of scenarios A, F, H

Regarding all the other computed metrics, no significant variations are present when changing the RTO and/or the maximum number of parallel sessions:

- The **number of retransmission cycles per session** keeps steady regardless variations of RTO and number of parallel sessions: scenario A gives 1.10 retransmission cycles per session, on average; scenario F gives 1.29 retransmission cycles per session, on average; scenario H gives 1.68 retransmission cycles per session, on average
- The **average number of data segments per retransmission cycle** keeps almost steady across all the considered variations. It is 44.17 for scenario A, 93.51 for scenario F and 139.872 for scenario H, on average
- The **average number of lost CP per session** is 0.07, 0.28 and 0.82, on average, for scenarios A, F and H, respectively
- The **average number of RAs per session** is 1.50 for scenario A, 1.77 for scenario F and 2.58 for scenario H, on average

This data is coherent with the average fading (Bad state) durations of the considered scenarios: 0.95 ms for scenario A, 2.14 ms for scenario F, 5.45 ms for scenario H.

6.2 FULL PASS SCENARIO

6.2.1 Goodput

In the Full pass scenario, the DTNperf client performs transmissions in rate-based modality, sending data at a nominal speed of 110Mbps. The goodput results are showed in Figure 36.

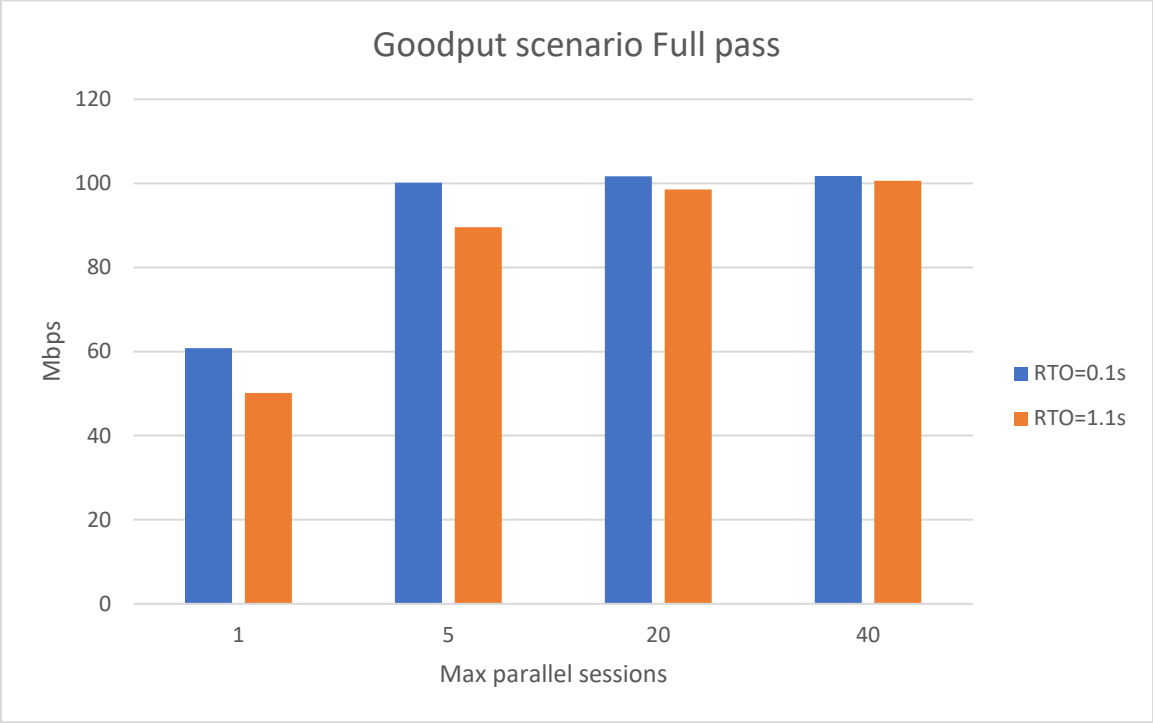

Figure 36:goodput variations on Full pass scenario

Here the drawbacks of losses are more limited: the whole bandwidth is almost already saturated with 5 parallel sessions, for both the cases with RTO=0.1s and RTO=1.1s. Passing from 20 to 40 parallel sessions shows no advantages in both the considered RTO values.

6.2.2 Penalization time

About the first and the last 400 sessions of the Full pass scenario are affected by a high number of retransmission cycles (≥ 3). All the following considerations are therefore valid when considered in these periods of time with an extremely high number of losses. Most of the penalization, indeed, comes from these stages of the trace.

In contrast to the scenarios A, F and H, here the irradiation time does not coincide with the RTT. The transmission data rate is 100 Mbit/s; therefore, the irradiation of 500 Kbyte requires 40 ms.

With transmission parallelism, in case of cross retransmissions, the penalization time (defined in 4.1.2) included in a retransmission cycle may be longer than the one that could exist without parallelism. To observe such phenomena let us consider the example Figure 37 and Figure 38: here we consider the transfer of a 500Kbyte LTP block at 200 Mbit/s (a simpler

parallel transmission with respect to the case of 100 Mbit/s, but the consideration keeps the same). In Figure 37 a single session is achieved after 4 retransmission cycles; as noticeable, the penalization time is the same with each retransmission cycle: time needed to receive the RS ($\frac{1}{2}$ RTT = 5 ms) + irradiation time (20 ms) + one-way propagation delay ($\frac{1}{2}$ RTT = 5 ms). Overall, the total penalization time with this session is 4 times 30 ms, i.e. 120 ms. Indeed, in absence of losses it should have been closed at t=30ms, while it lasts until 150ms. For even more simplicity of representation, the shipment of the RA is not drawn.

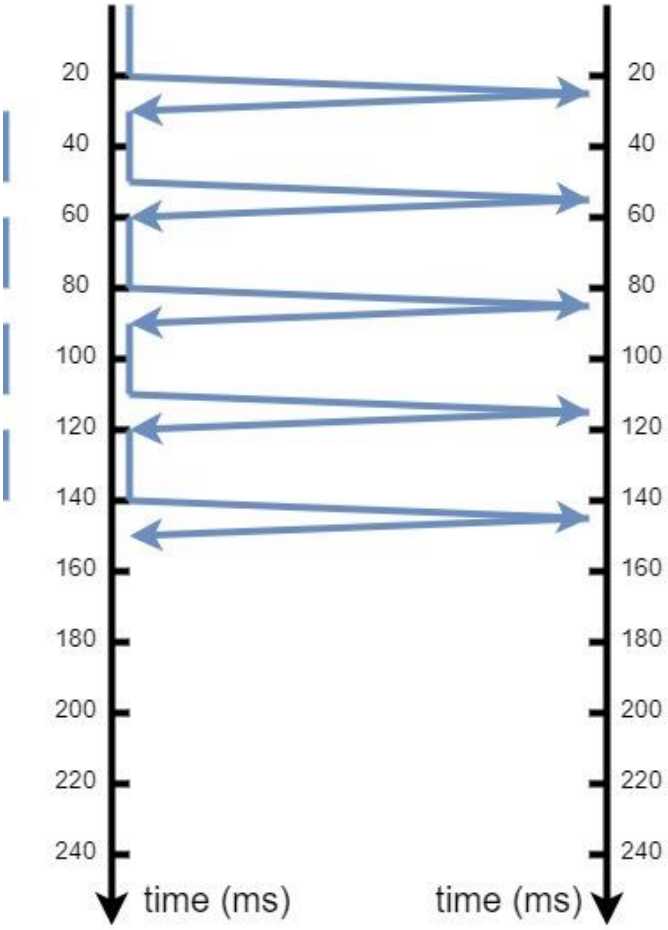

Figure 37: example of a single session which suffered of 4 retransmission cycles. Block dimension = 500Kbyte, data rate = 200Mbps

In Figure 38 we consider the case of the parallel transmissions of two 500Kbyte LTP blocks at 200 Mbit/s. The blue session and the yellow session suffer of four and three retransmission cycles, respectively. Before analysing the evolution of the two sessions in time, we state that the LTP implementation of DTNME gives higher priority to segments that must be retransmitted rather than segments to be transmitted for the first time; i.e. if the sender receive a RS claiming for lost segments of the blue session during the first transmission attempt of segments related to the yellow session, it stops the irradiation of segments related to the yellow session and start retransmitting claimed segments of the blue session. In Figure

38 the first session to be transmitted is the blue one, and the corresponding CP segment is shipped at time 20ms; when this happens, the transmission of the parallel (yellow) block starts. The blue session is subject to losses (we keep assuming that every time almost everything but the CP is lost) and at time t = 30ms the corresponding RS is received.

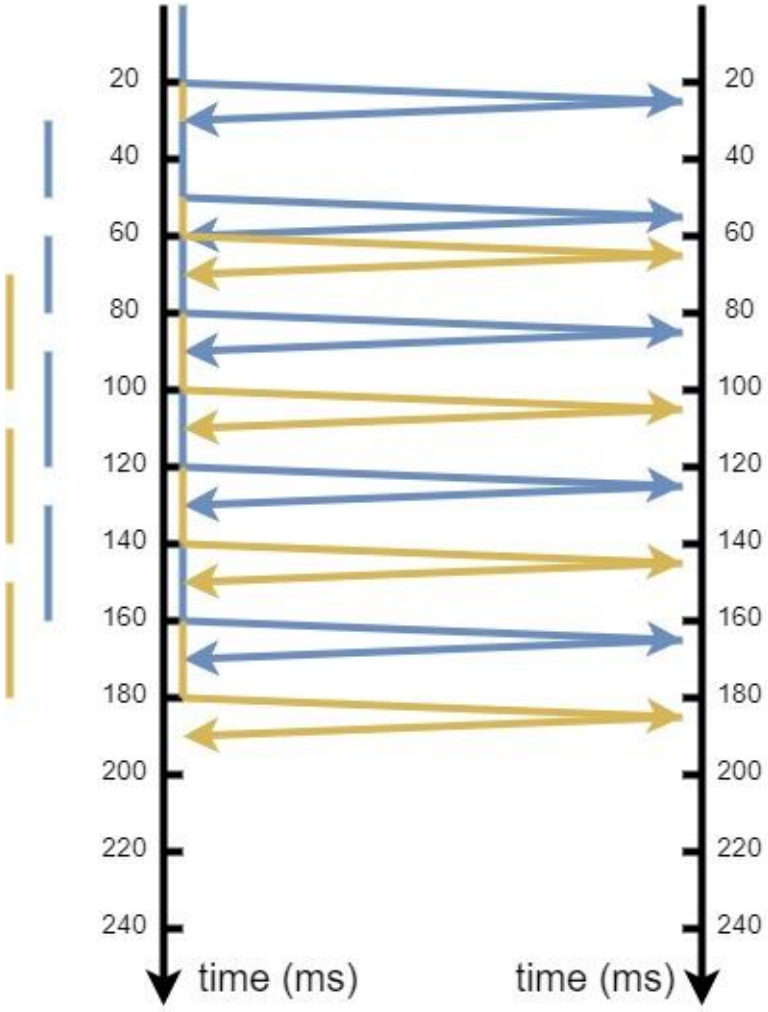

Figure 38:example of 2 parallel sessions in which the blue session suffered of 4 retransmission cycles and the yellow session suffered of 3 retransmission cycles. Block dimension = 500Kbyte, data rate = 200Mbps

The reception of the RS at time t = 30ms triggers the retransmission of the blue segments, stopping half-way the shipment of the yellow-block segments. The irradiation time of retransmitted blue segments lasts until t=50ms, a new blue CP is shipped and the yellow-block shipment is resumed from where it was interrupted. After 10ms the transmission of the yellow-block is finished and the corresponding CP is sent. At the same time a new blue RS arrives to the sender, who immediately starts retransmitting the blue segments as almost everything has been lost. Half-way the blue retransmission, the yellow RS claiming for almost every segment of this session is received. This time the just received RS does not pause the irradiation of the parallel blue session segments, because they are retransmitted segments and not “new” ones. For this reason, the claimed retransmission of the yellow segments is

delayed by the time needed to inject all the remaining blue segments to be retransmitted. Only after this time, here 10 ms, the irradiation of the lost yellow segments can happen. The penalization time of the yellow session is consequently enlarged because the portion of penalization introduced by the sender is increased by the just discussed **waiting time** to start retransmitting. From now on, if further retransmission cycles happen in both the parallel sessions, the penalization time will be always affected by this waiting time. In this exemplar case, for simplicity, every time almost everything but the CP was lost in a session; i.e. we analysed the worst-case, in which the waiting time was half the irradiation time.

Back to the 100 Mbit/s Full pass scenario, the waiting time in the worst case is 30 ms: as showed in Figure 39, the irradiation time of 500Kbyte at 100Mbit/s is 40 ms and every time a CP is sent there is a time interval equal to the RTT (10 ms) in which the parallel session can advance in being transmitted. Assuming that every time almost everything of the first session (blue) is lost, the second session (yellow) is interrupted always after one RTT and has to wait 40ms (irradiation time of a full block) to be resumed. At time t = 200ms the CP of the yellow session is sent and, immediately, the segments of the blue session start to be retransmitted as requested by the last received blue RS. After one RTT the RS of the yellow session claiming for almost every segment of the yellow block is received but the retransmission of this segments is queued after the end of the current retransmission: 30 more ms of penalization are therefore introduced for the yellow session, being the worst (largest) additional waiting time to be paused before starting the retransmission.

Moving further the assumption of losing almost everything of a block but the CP, it can be said that when cross retransmission cycles happen, the additional waiting time introduced in the penalization is:

$$waiting\_time = parallel\_irradiation\_time - RTT$$

Where parallel_irradiation_time is the time needed to irradiate the remaining segments of the parallel session when the RS of the queued session arrives. For instance, since in scenarios A, F and H the maximum parallel_irradiation_time is less or equal than the RTT, no waiting_time is added to the penalization of a retransmission cycle.

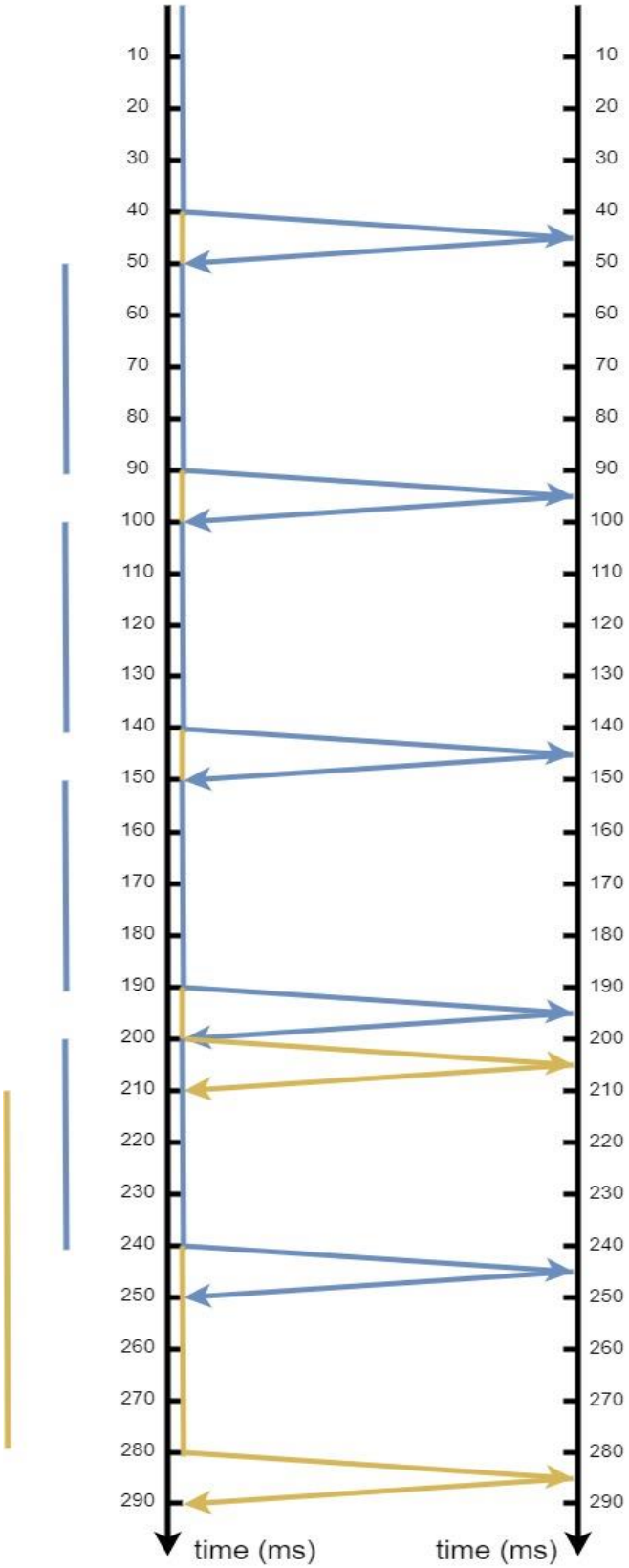

Figure 39:example of 2 parallel sessions in which the blue session suffered of 4 retransmission cycles and the yellow session suffered of 3 retransmission cycles. Block dimension = 500Kbyte, data rate = 100Mbps

If cross retransmission cycles would be the only reason for the penalization time to be larger when passing from 1 to 2 sessions in parallel, this penalization time should not increase anymore when passing from 2 to N nominal sessions in parallel, where $N \geq 2$. Indeed, looking at Figure 39, there is no time between all the transmissions and retransmissions to start sending segments of a third session. However, analysing the capture files of this scenario with our Python program, the following statistics are retrieved:

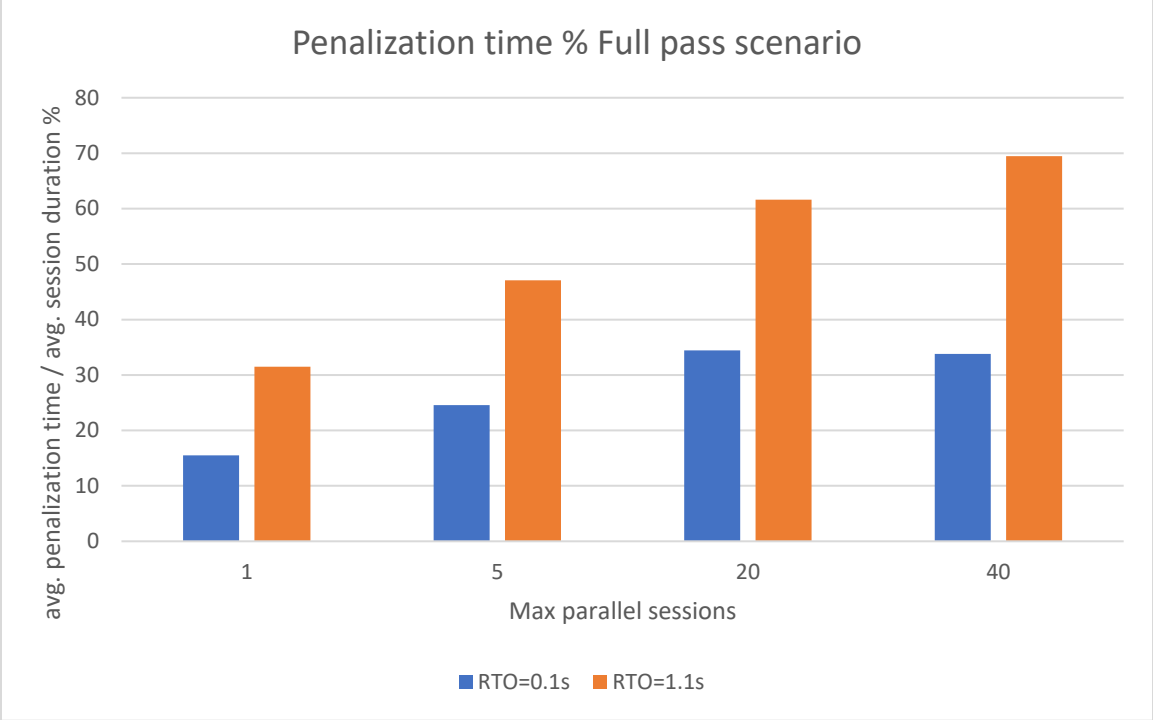

Figure 40: penalization time percentage in Full pass scenario

The penalization time increases as the nominal maximum number of parallel sessions rises. The reason of this trend can be found by looking at the behaviour of another metric: the average number of lost CP per session, represented in Figure 41. Considering now that also the CP is lost with the high number of losses affecting a block, a retransmission of this CP happens when RTO seconds have passed. In the meantime, new parallel sessions can be started. Since the minimum settable RTO in DTNME is 100ms, and the maximum number of consecutive CP retransmissions is set to 5 (after which the session is cancelled) there are 500ms in which, if no session is triggering retransmission cycles (because the CP of all sessions is lost), about 12 sessions can be parallelly started (500 ms divided by 40ms of irradiation time per each block). For these sessions the penalization times increment by the corresponding RTOs, other than the waiting times of cross retransmissions, if happen. Accordingly, looking at Figure 40, with RTO=0.1s the penalization time increases when passing from W=1 to W=5 and, again, from W=5 to W=20. However, passing from W=20 to W=40 does not affect penalization times: the reason is that in these high losses' periods we can always consider $W \leq 12$. For example, assuming the first session is cancelled because at time t=500ms its fifth consecutive CP has been declared lost, if no retransmission cycles of

other sessions are active, a new session would be started replacing the just cancelled one (no parallelism increment). As always in this scenario, the irradiation time of this new block is 40ms and at t = 540ms two main situations may appear because of the high losses:

- This second session is cancelled and no retransmission cycles of other sessions are active; in this case the second session is replaced by a new session, but still, the level of parallelism keeps steady
- There are active retransmission cycles related to other blocks not allowing new sessions to be started

All presented statistics are about confirmed sessions only, therefore all sessions in the starting or ending period of this scenario will, sooner or later fall in the second pointed case.

Considering RTO=1.1s, instead, the penalization time keeps increasing even passing from 20 to 40 maximum parallel sessions because a session could not be cancelled before 5.5 seconds, and within this period of time up to 137 sessions can be started parallelly. Of course, the maximum number of parallel sessions, in this case, is dictated by the configuration parameter of DTNME.

Again, these considerations regard the starting and the ending period of the Full pass trace, which is full of losses, but we expected that most of the penalizations come exactly from these periods.

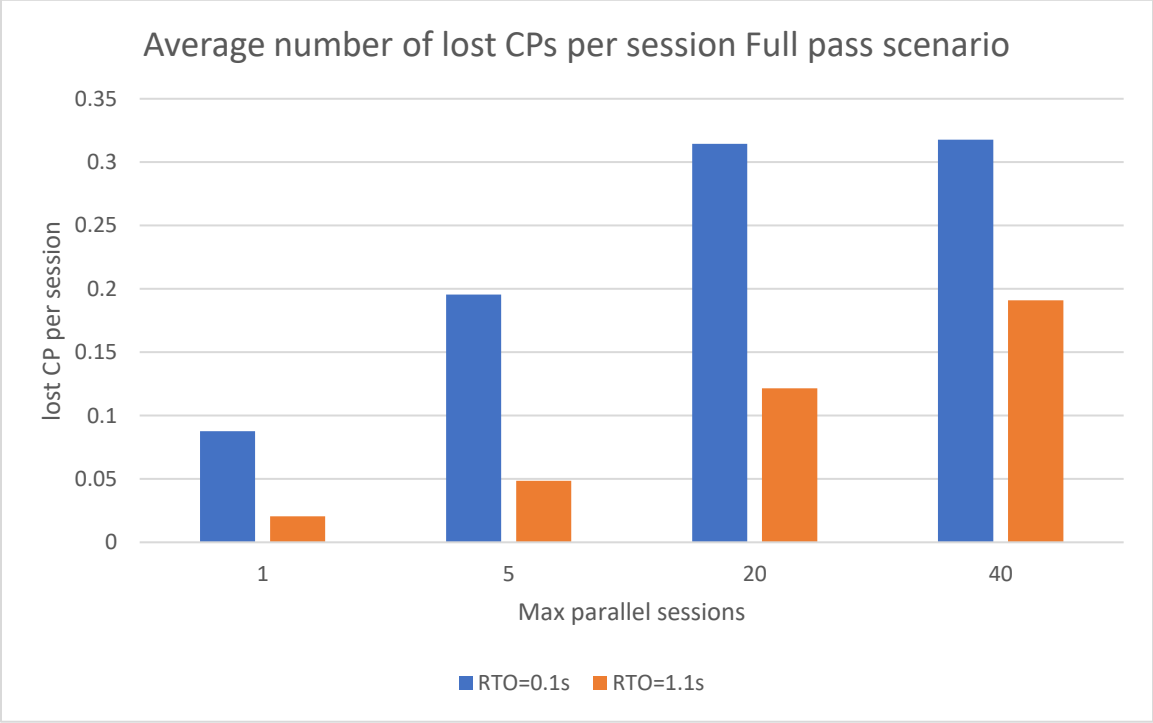

Figure 41: average number of lost CPs per session Full pass scenario

6.2.3 Other metrics

All the experiments show that the **average session duration** naturally depends on the average penalization time per session. In absence of losses all the sessions would have almost the same duration. Losses on data or signaling segments result if penalizations. Since in this scenario the average penalization time changes when the parallelism does so, the same trend is present with the average session duration. An illustration of the different average session durations with respect to the diverse RTO and number of parallel sessions is showed in Figure 42.

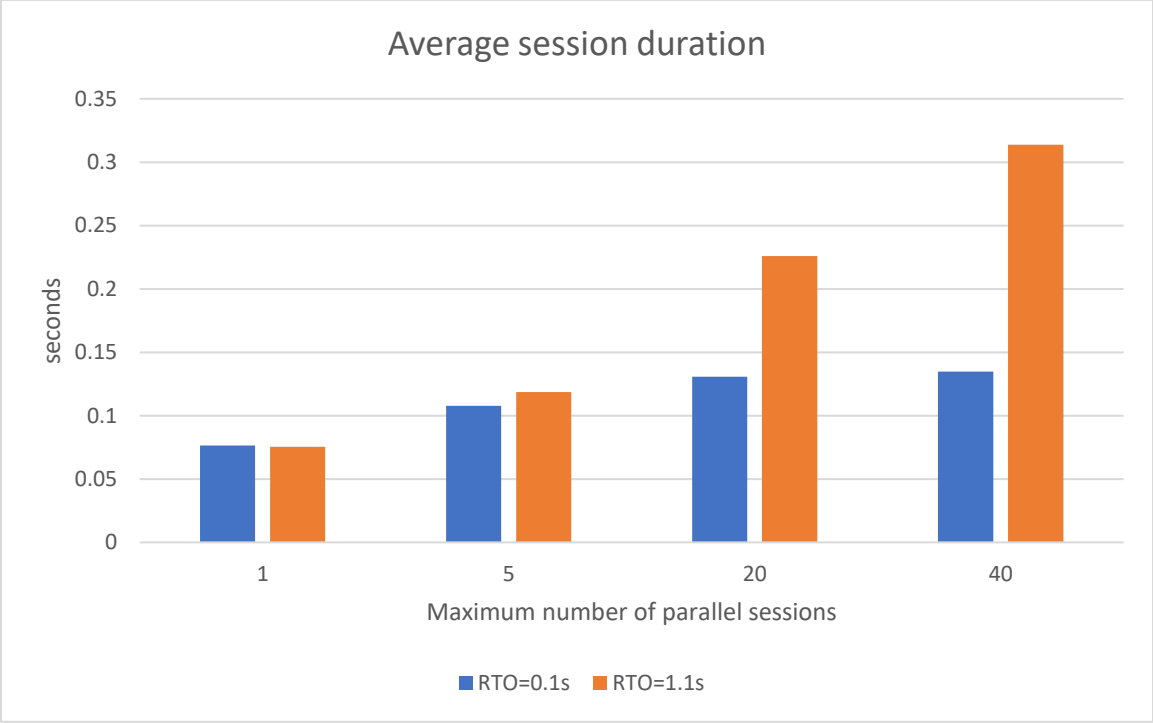

Figure 42: average session duration of Full pass scenario

Coherently, also the average number of retransmission cycles and the average number of RA follow a similar trend, as showed in Figure 43 and Figure 44.

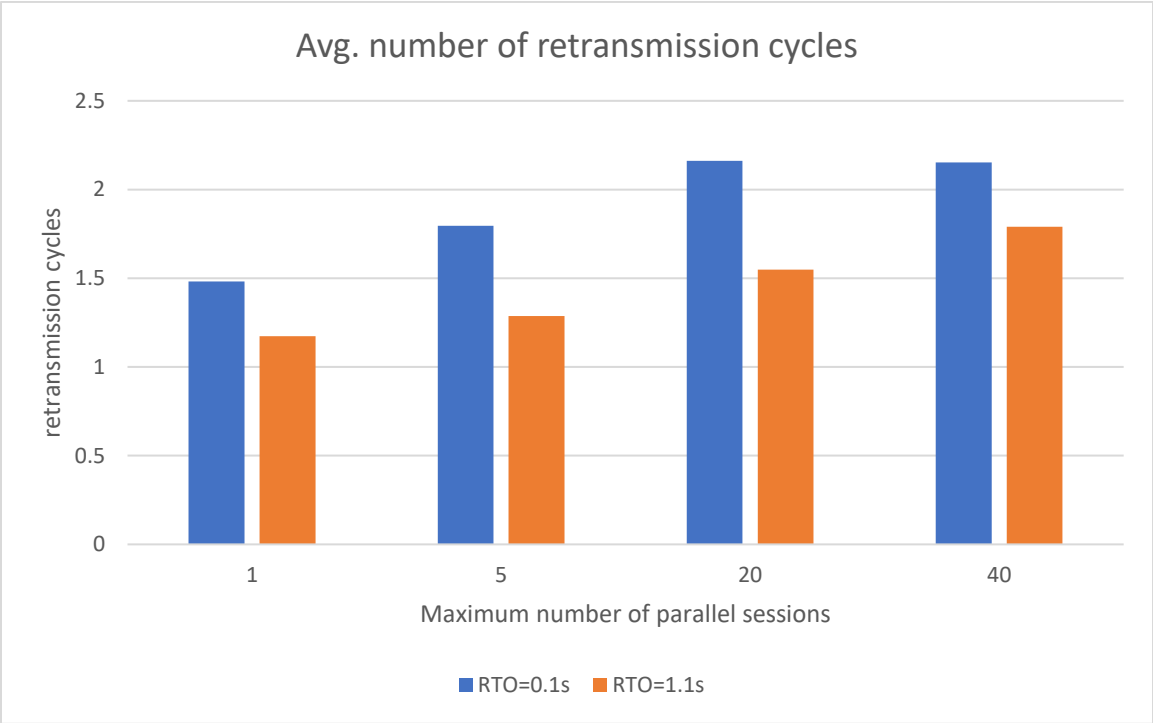

Figure 43: average number of retransmission cycles Full pass scenario

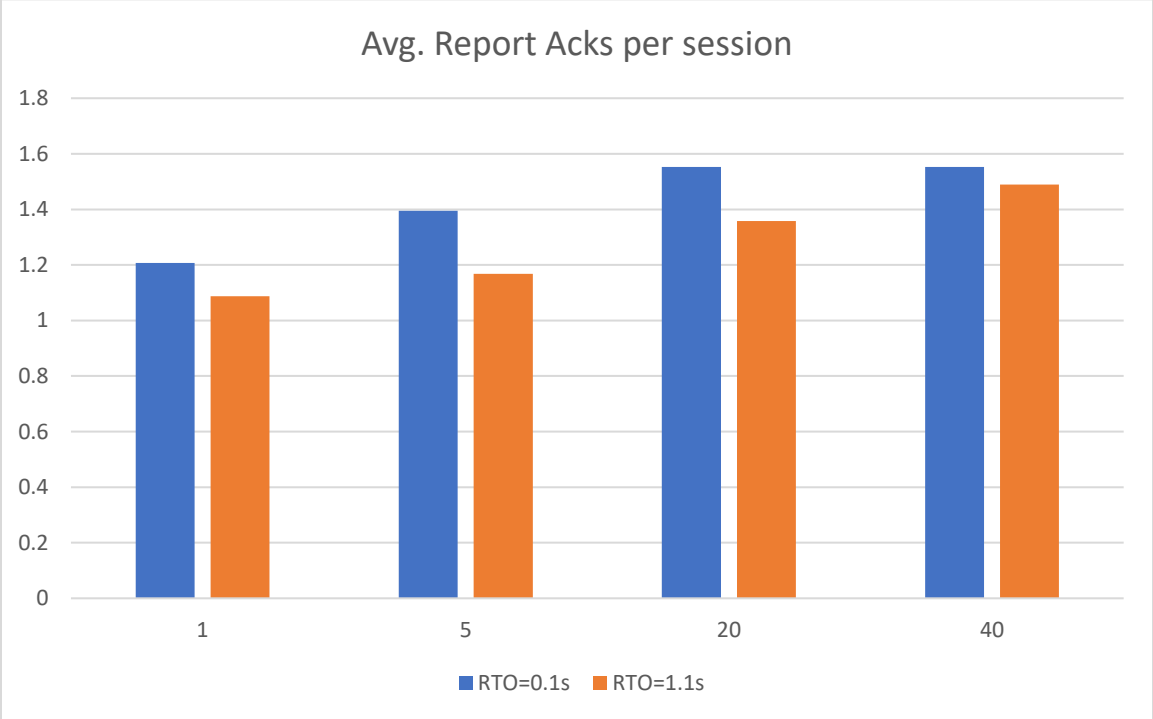

Figure 44: average number of Report Ack segments per session Full pass scenario

The average number of data segments per retransmission cycles, instead, showed a steady trend among different RTOs and W parameters, with an average value of 64.55 data segments per retransmission cycle.

7. CONCLUSIONS

The German Aerospace Center (DLR) is interested in assessing Licklider Transmission Protocol (LTP) performances over typical Free Space Optical (FSO) communication channels. For this reason, the Optical Communication Department of DLR Oberpfaffenhofen, Institute of Communication and Navigation (DLR-KN), provided us with four synthetic tracks representing the dynamic state of an FSO link between an optical ground station and a LEO satellite. More precisely, these tracks are sequences of Bit Error Rate (BER) samples obtained at a sampling frequency of 10kHz; three of them have duration of 100 seconds, while the fourth lasts 692 seconds, corresponding to the full passage of an OSIRIS4CubeSat over an optical ground station. Starting from these tracks, we applied a Reed Solomon (255, 223) error correcting code and other processing to obtain four "erasure vectors" representing a binary on/off channel. Each track consists of a sequence of 0s and 1s, encoding the Good state (Packet Error Rate 0) or the Bad state (Packet Error Rate 1) of the channel. Each sample represents the channel state for the next 0.1ms, corresponding to the original sampling frequency of 10 kHz.

Once obtained the erasure vectors of four different scenarios, it was necessary to develop a channel emulator, called "detemu", to enforce losses on our testbeds. Detemu is written in C++ language and makes use of the PcapPlusPlus library: a wrapper library of libpcap (upon which Wireshark and Tcpdump are built). The name is justified by its ability at emulating a a deterministic link: each packet flowing from the sender to the receiver must pass through a central machine having a running instance of detemu, which decides whether this packet should be forwarded to the destination or dropped on the basis of the erasure vector provided in input. At the end of the track, i.e. after 100s or 692s, detemu provides the user with a few statistics about the durations of bad and good states in a brief summary and more detailed information in a .csv file.

Two similar testbeds have been setup at DLR-KN and at ARCES (University of Bologna), both accessible by remote, to carry out the LTP experiments.

Bundles are generated by means of DTNperf, which can send bundles of a specified size for a given amount of time and at a fixed data rate. Using Tshark, it was possible to capture all the LTP traffic at the Sender side of the communication, producing (very large) pcap files.

A new Python program, called LTP performance analyser, has been developed to extract a large number of metrics and statistics from pcap files of LTP transmissions. In particular, this program allowed to deeply study LTP session durations, losses, retransmission cycles and penalization times.

Preliminary results immediately showed the impact of the RTO on LTP performances. In particular, it emerged that the RTO granularity, 1s, was too coarse for the very short RTT of the channel (10ms), resulting in severe time penalization when LTP checkpoints were lost. In other words, the average LTP session duration was much longer than necessary because of the excessive RTO, even when this was set to 1. After inspecting the code of the chosen LTP

implementation (DTNME), we discovered that to the RTO value set by the user, it added 100ms, which allowed us to set a nominal null RTO to actually obtain 100ms. The new setting resulted in an evident performance improvement, as shown in the thesis.

A second kind of analysis was carried out by varying the maximum number of parallel sessions. As studied, FSO channels may suffer high losses when the pointing error between the satellite and the optical ground station is high or the link is under adverse atmospheric conditions. In such cases the LTP session penalization time increases with the parallelism. The entity of this phenomenon depends on the LTP block size, the RTT value and the transmission data rate, and appears when:

- Segments to be retransmitted cannot be sent immediately but must wait for the conclusion of concurrent session retransmissions
- Checkpoint segments belonging to parallel sessions are lost during the same bad state

On the other hand, by increasing the level of session parallelism the goodput generally improves, in accordance with theory: however, this enhancement stops before the channel utilization is reached, likely due to the interference between parallel sessions in the presence of long/frequent bad states. For this reason, a future work could be focused on a punctual analysis of the waiting times given by cross retransmissions of multiple sessions.

Summarizing, LTP proved to be extremely robust to high loss ratios and an excellent match to FSO links in near-Earth scenarios.

References

[1] Oberlo, “Internet statistics,” 2022. [Online]. Available: https://www.oberlo.com/blog/internet-statistics.

[2] ESA, “The Earth-Observation Programme,” [Online]. Available: https://www.esa.int/esapub/br/br114/br114ear.htm.

[3] “Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.,” [Online].

[4] C. Caini, H. Cruickshank, S. Farrell and M. Marchese, “Delay- and Disruption-Tolerant Networking (DTN): An Alternative Solution for Future Satellite Networking Applications,” Proceedings of IEEE, Vol. 99, N. 11, pp.1980-1997, Nov. 2011, Invited paper. DOI 10.1109/JPROC.2011.2158378.

[5] “Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, DOI 10.17487/RFC5050, November 2007, <https://www.rfc-editor.org/info/rfc5050>.,” [Online].

[6] “Burleigh, S.; Fall, K.; Edward, B., "Bundle Protocol Version 7", RFC 9171, January 2020, <https://www.rfc-editor.org/rfc/rfc9171.txt>[ONLINE],” 2022.

[7] Consultative Committee for Space Data Systems, “Consultative Committee for Space Data Systems, "CCSDS bundle protocol specification", CCSDS Reccomended Standard 734.2-B1, September 2015, <https://public.ccsds.org/Pubs/734x2b1.pdf>”.

[8] “Demmer, M.; Ott, J; Perreault, S., "Delay-Tolerant Networking TCP Convergence-Layer Protocol", RFC 7242, DOI 10.17487/RFC7242, June 2014, <https://www.rfc-editor.org/rfc/rfc7242.txt>,” [Online].

[9] S. Burleigh, C. Caini and J. R. M. Messina, “Toward a Unified Routing Framework for Delay-Tolerant Networking,” 2016 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE), Mar. 2017, DOI: 10.1109/WiSEE.2016.7877309.

[10] G. Araniti, N. Bezirgiannidis, E. Birrane, I. Bisio, S. Burleigh, C. Caini, M. Feldmann, M. Marchese, J. Segui and K. Suzuki, “Contact Graph Routing in DTN Space Networks: Overview, Enhancements and Performance,” IEEE Commun. Mag., Vol.53, No.3, pp.38-46, March 2015. DOI: 10.1109/MCOM.2015.7060480.

[11] C. Caini, G. M. De Cola and L. Persampieri, “Schedule-Aware Bundle Routing: Analysis and Enhancements, International Journal of Satellite Communications and Networking, vol. 39, no.3, pp. 237-243, May/June 2021. DOI: 10.1002/sat.1384”.

[12] S. Burleigh, “ION-DTN,” Nasa JPL, [Online]. Available: https://sourceforge.net/projects/ion-dtn/.

[13] W.-B. Pöttner, S. Schildt, J. Morgenroth and L. Wolf, “Performance comparison of DTN Bundle Protocol implementations,” Proceedings of the 6th ACM workshop on Challenged networks, ACM, 2011, pp. 61–64.

[14] Marshal Space Flight Center, “Nasa DTNME,” [Online]. Available: https://github.com/nasa/DTNME.

[15] A. Bisacchi, C. Caini and S. Lanzoni, “Design and Implementation of a Bundle Protocol Unified API,” in Proc. of ASMS 2022, Graz, Austria, Sept. 2022, pp. 1-6. doi: 10.1109/ASMS/SPSC55670.2022.9914734.

[16] A. Bisacchi and S. C. C. Lanzoni, “Unified API,” [Online]. Available: https://gitlab.com/dtnsuite/unified_api.

[17] University of Bologna, “DTNsuite,” [Online]. Available: https://gitlab.com/dtnsuite.

[18] C. Caini, A. D'Amico and M. Rodolfi, “DTNperf_3: A further enhanced tool for Delay-/Disruption- Tolerant Networking Performance evaluation,” in Proc. of IEEE Globecom 2013, Atlanta, USA, December 2013, pp. 3009 - 3015. DOI: 10.1109/GLOCOM.2013.6831533.

[19] A. Zappacosta, A. Bisacchi and C. Caini, “Reingegnerizzazione del client di DTNperf,” 2020.

[20] University of Bologna, “DTNperf,” [Online]. Available: https://gitlab.com/dtnsuite/dtnperf.

[21] S. Burleigh, M. Ramadas and S. Farrell, “Licklider Transmission Protocol - Motivation,” RFC 5325, DOI 10.17487/RFC5325 , September 2008, <https://www.rfc-editor.org/rfc/rfc5325>.

[22] M. Ramadas, S. Burleigh and S. Farrell, “Licklider Transmission Protocol - Specification,” RFC 5326, DOI 10.17487/RFC5326 , September 2008, <https://www.rfc-editor.org/rfc/rfc5326>.

[23] S. Farrell, M. Ramadas and S. Burleigh, “Licklider Transmission Protocol - Security Extensions,” RFC 5327, DOI 10.17487/RFC5327 , September 2008, <https://www.rfc-editor.org/rfc/rfc5327>.

[24] Consultative Committee for Space Data Systems, “Licklider Transmission Protocol (LTP) for CCSDS,” CCSDS Recomended Standard 734.1-B-1, May 2015, <https://public.ccsds.org/pubs/734x1b1.pdf>.

[25] A. Bisacchi, C. Caini and T. De Cola, “Multicolor Licklider Transmission Protocol: An LTP Version for Future Interplanetary Links,” in IEEE Transactions on Aerospace and Electronic Systems, vol. 58, no. 5, pp. 3859-3869, Oct. 2022, doi: 10.1109/TAES.2022.3176847. (Open Source).

[26] S. Burleigh and N. Ansell, “A Guide to Configuring LTP for ION, v.5,” 2016.

[27] D. Zoller, “DZ notes on the DTNME 1.0.1-BETA,” 2021.

[28] A. Bisacchi, “Progetto di una implementazione veloce del Licklider Transmission Protocol per link ottici,” Bologna, 2020.

[29] L. Persampieri and C. Caini, “Unibo-BP: an innovative free software implementation of Bundle Protocol Version 7 (RFC 9171),” 3 February 2023. [Online]. Available: https://amslaurea.unibo.it/27740/.

[30] N. Alessi, C. Caini, T. De Cola and M. Raminella, “Packet Layer Erasure Coding in Interplanetary Links: The LTP Erasure Coding Link Service Adapter,” in IEEE Transactions on Aerospace and Electronic Systems, vol. 56, no. 1, pp. 403-414, Feb. 2020, doi: 10.1109/TAES.2019.2916271.

[31] N. Alessi, C. Caini, A. Ciliberti and T. De Cola, “HSLTP—An LTP Variant for High-Speed Links and Memory Constrained Nodes,” in IEEE Transactions on Aerospace and Electronic Systems, vol. 56, no. 4, pp. 2922-2933, Aug. 2020, DOI: 10.1109/TAES.2019.2958190..

[32] N. Alessi, S. Burleigh, C. Caini and T. De Cola, “Design and performance evaluation of LTP enhancements for lossy space channels,” Wiley, International Journal of Satellite Communications and Networking, Vol.37, No.1, pp.3-14, Jan-Feb 2019, DOI: 10.1002/sat.1238..

[33] Statista, “Number of satellites in orbit by major country as of January 1, 2022,” 2022. [Online]. Available: https://www.statista.com/statistics/264472/number-of-satellites-in-orbit-by-operating-country/.

[34] D. Giggenbach, B. Epple, J. Morwath and F. Moll, “Optical Satellite Downlinks to Optical Ground Stations and High-Altitude Platforms,” 2008.

[35] C. Fuchs, “Satellite Laser Communications for Feeder Links and Mega Constellations,” 2022.

[36] A. Carrasco-Casado and R. Mata-Calvo, “Free-space optical link for space communication networks,” 2020.

[37] D. Giggenbach, S. Parthasarathy, A. Shrestha, F. Moll and R. Mata-Calvo, “Power Vector Generation Tool for Free-Space Optical Links - PVGeT,” 2017 IEEE International Conference on Space Optical Systems and Applications (ICSOS), Naha, Japan, 2017, pp. 160-165, doi: 10.1109/ICSOS.2017.8357228..

[38] A. Shrestha, D. Giggenbach, A. P.-L. J. Mustafa, J. Ramirez and F. Rein, “Fading testbed for free-space optical communications,” 2016.

[39] Consultative Committee for Space Data Systems, “TM SYNCHRONIZATION AND CHANNEL CODING - SUMMARY OF CONCEPT AND RATIONALE,” CCSDS Informational Report 130.1-G-3, June 2020, <https://public.ccsds.org/Pubs/130x1g3.pdf>.

[40] A. Zappacosta, “Thesis scripts,” [Online]. Available: https://gitlab.dlr.de/zapp_an/thesis_scripts.

[41] D. Giggenbach and A. Shrestha, “CCSDS-O3K: Explanation of Reference Optical Downlink Power Vectors,” 2022.

[42] Deutsches Zentrum für Luft- und Raumfahrt, “OSIRIS4CubeSat,” [Online]. Available: https://www.dlr.de/kn/en/desktopdefault.aspx/tabid-17840/#gallery/36173.

[43] F. Ludovici and H. Pfeifer, “tc-netem - Linux manual page,” 2011. [Online].

[44] “luigirizzo/dummynet,” [Online]. Available: https://github.com/luigirizzo/dummynet.

[45] A. Zappacosta, “detemu,” 2022. [Online]. Available: https://gitlab.com/antony_lion/detemu.

[46] “PcapPlusPlus,” [Online]. Available: https://pcapplusplus.github.io/.

[47] “tcpdump,” [Online]. Available: https://www.tcpdump.org/.

[48] PcapPlusPlus, “Benchmarks,” [Online]. Available: https://pcapplusplus.github.io/docs/benchmark.

[49] “tc-netem,” [Online]. Available: http://man.he.net/man8/tc-netem.

[50] “Wireshark,” [Online]. Available: https://www.wireshark.org/.

[51] A. Zappacosta, “LTP performance analyser,” 2022. [Online]. Available: https://gitlab.com/antony_lion/ltp_performance.