POLITECNICO DI TORINO

Master's Degree in Communications and Computer Networks Engineering

Master's Degree Thesis

An in-depth study of Multipath TCP in a domestic environment

Supervisors

Prof. Paolo GIACCONE

Candidate

Maria Letizia AMORUSO

October 2021

Summary

Nowadays, telecommunication systems are extensively used every day in numerous applications: file sharing through cloud, instant messaging apps, conference calls, file transfer, videogames, telemedicine, etc... Each application needs a particular set of performance metrics in order to achieve a good user experience. The main ones are:

- bandwidth
- latency
- reliability

Network technologies are constantly evolving in order to meet the increasingly strict performance requirements, which is why exploiting the same resources with new approaches could be beneficial, for instance using multiple network connections at the same time when available.

Regular services delivered over Internet applications make use of a transport protocol, such as TCP (Transmission Control Protocol), which provides a reliable, connection-oriented service to the invoking application [1], providing a host-to-host connectivity at the transport layer [2]. Multipath TCP (MPTCP) [3] allows such application to use multiple paths, creating several sub-flows to a regular TCP session.

The aim of this thesis is to investigate the behaviour of MPTCP and to study the gain introduced in terms of throughput and robustness, whenever different access technologies are available.

Keywords: MPTCP, Throughput, FTP

Acknowledgements

Vorrei cominciare col ringraziare il mio Relatore, il Prof. Paolo Giaccone, il quale ha saputo sapientemente direzionarmi nello svolgimento della mia tesi, rendendosi sempre disponibile nel fornire un fondamentale supporto.

Ringrazio i colleghi della WindTre che mi hanno fornito il sostegno necessario per l'elaborazione della tesi e con cui ho il piacere e l'onore di iniziare la mia carriera lavorativa.

Vorrei ringraziare gli amici che in questi cinque anni sono sempre stati al mio fianco, a partire da Annalisa e Nadia le quali mi hanno guidata in una città di cui conoscevo molto poco e non mi hanno mai fatta mancare un caloroso affetto sul quale ho sempre contato molto.

Ringrazio le amiche di sempre, Giulia ed Irene che mi hanno vista affrontare tante difficoltà e, sebbene a distanza, mi hanno supportata in ogni mia scelta e sono sempre state pronte ad ascoltare ogni mia lamentela. La consapevolezza di avere nella mia vita due ancore come voi, mi aiuta ad affrontare con più serenità i momenti di maretta.

Un ringraziamento va a chi ho conosciuto nel mio percorso universitario, nello specifico a Francesca con la quale ho avuto il privilegio di crescere in questi anni a dir poco complessi e dalla quale ho imparato nuovamente il significato dell'amicizia; a Giovanni, con cui abbiamo convidiso infiniti momenti di spensieratezza e a Francesco, un perfetto compagno di studi senza il quale probabilmente starei ancora preparando Analisi II, nonchè un amico fidato su cui poter sempre contare.

Ringrazio gli amici del Rotaract Club di Crotone che mi hanno accolta in un momento storico difficile e con i quali è sempre un piacere fare service e condividere momenti di allegria e di crescita.

L'ultimo ringraziamento va alla mia famiglia che dal giorno in cui ho scelto di frequentare il Politecnico è sempre stata entusiasta e mi ha inondata di supporto che continuo a percepire intensamente.

Un ringraziamento in particolare va ai miei genitori e a mio fratello, che in questi anni mi hanno vista abbandonare il nido per percorrere la mia strada. So che non è stato facile e che non ci sono parole sufficienti per esprimere quanto io sia orgogliosa di essere vostra figlia e sorella, per cui vi ringrazio così, dicendovi

che ogni singolo giorno prendo più consapevolezza di quanto il vostro supporto sia per me fondamentale e di come senza di esso non sarei mai giunta a scrivere questi ringraziamenti.

Table of Contents

Chapter 1

Introduction

1.1 Motivation scenario

The motivation behind the choice of investigating the behaviour of Multipath TCP and deploying it in a network is to improve the performance provided by regular TCP in host-to-host reliable connections. The main idea of this protocol is to use multiple paths from host to host by aggregating several TCP connections, providing several improvements such as increased throughput, reliability, resilience and redundancy, maximizing the available resources in the network. Multipath TCP uses the different network interface cards available on the devices, providing a multi-homing [4] capability and thus an improved throughput. Speaking of reliability, thanks to Multipath TCP it is possible to achieve smooth hand-overs between networks, minimizing the impact on the user experience when a fault is detected in one of the connections.

Particularly, Wind Tre S.p.A. [5] expressed interest in this topic as they feel it could be useful to increase the overall transfer speed experienced by the user, without needing to add new resources into their network. The company has therefore interest in investigating the MPTCP operation in a real-world setting, examining first a small and domestic environment which this thesis will be considering. Once a first batch of experiments is carried out in this setting and signs of improvement are appreciable in the results, the Research Department will then make use of this framework to replicate such experiments on a larger scale in Wind Tre's network.

1.2 Problem Statement

The goal of these experiments is to study the functioning of Multipath TCP, evaluating its performances in terms of load balancing, throughput and reliability. Specifically, different scenarios will be considered and all of them are selected to be very close to a domestic and real-world setting: this way we will be able to determine how much performance benefits MPTCP can actually offer to the users.

1.3 Document structure

This document is divided in several chapters:

First, we will present an introduction of Multipath TCP, providing the reader with a background knowledge of the protocol and its main characteristic behaviours in Chapter 2. A brief explanation of the considered scenario is presented in Chapter 3 as well as all the tools and configurations necessary for the testbed which are explained in detail. Such tools are then used for the experiments carried out in Chapter 4 where the specific testing scenarios and evaluation tools are presented, followed by the results in Chapter 5. Finally, in Chapter 6 we draw our conclusions on the obtained results and provide the reader with a meaningful interpretation.

Chapter 2

Background

In this chapter we go through the main characteristics of Multipath TCP, providing the reader with all the notions necessary to master the analysis of this protocol. First, an overview of MPTCP is presented, explaining the main idea behind this extension of TCP, then we will go through its working procedure and main building blocks, such as schedulers and congestion window.

2.1 MPTCP overview

Multipath TCP was first introduced in 2013 by the IETF as an experimental standard in RFC 6824 [3]; it was designed as an extension to TCP which was capable to support mobility, load balancing, resilience and multi-homing [4]. The main addition of Multipath TCP is the ability to multiplex one-byte stream over separate paths, through which a higher throughput can be achieved. For instance it can be quite useful to aggregate the accesses of a mobile network and a Wi-Fi access, since they provide two very differentiated paths.

Focusing on the operation of MPTCP, it is safe to say that it very much relies on TCP, as it uses TCP options to advertise its control informations such as signalling multipath capability and advertising the available IP addresses, which we will see in further detail.

MPTCP connection begins similarly to a regular TCP connection with a three-way handshake; then, if extra paths are available, some additional TCP sessions called "sub flows" are created over such paths and combined with the existing session. This new set of parallel sessions continues to appear as a single connection to the applications at both Client and Server sides. The MPTCP-aware application initiates the data transmission by regularly opening a TCP socket, while the MPTCP signalling and operation are taken care of by the MPTCP implementation. The availability of multiple paths is linked to the presence of multiple addresses at host, i.e. if the Client advertises two IP addresses, then the Client can be reached through two different paths; such paths can be exploited by Multipath TCP which will then multiplex the traffic toward the Client across two different subflows. The so-called additional addresses are discovered and used to set up additional subflows through a path management method: a host can either initiate new sub flows by using its own additional addresses or it can signal its availability of an additional address to the other host.

One of the main aspects of multipath TCP is the diversification of the multiple subflows which are exploited to multiplex the traffic: a packet transmitted on a path may experience a different network delay than a similar packet transmitted in parallel on another path. In order to reassembly out-of-order packets at receiver side (Server), Multipath TCP uses its own sequence numbering system (as well as buffers where packets are temporarily stored before being reassembled, as we will see later) for which each segment sent at transmitter side is labelled with two sequence numbers: the regular sequence number that appears inside the TCP header and the Multipath-level Data Sequence Number (DSN) that is declared inside Multipath TCP options.[6] The DSN option is ignored by the MPTCP-unaware middle-boxes while it is used by the MPTCP-aware receiver in order to reorder the stream before passing it on to the running application.

2.1.1 Connection establishment

The connection establishment procedure is somewhat similar to a regular TCP one, as shown in Figure 2.1 where an MPTCP connection is established between Host 1 and Host 2.

Let us now focus on how a new multipath TCP connection is started and how the multiple subflows are established. The application establishing a new connection will first open a regular TCP socket, which will set up the initial TCP sub-flow. It is worth noticing that up until this point, the connection appears as a regular TCP one to both the network and application layers, thanks to the transparent nature of Multipath TCP; in fact, a Multipath TCP connection appears like a regular TCP connection to any application. Instead, to the network layer, each MPTCP subflow looks like a regular TCP flow whose segments carry a new TCP option type [3].

Once the initial connection is established, if both endpoints, Host A and Host B, support MPTCP capability, the MPTCP session can be started by either one of them.

We will now go through a simplified scenario in order to describe the basic procedures of the establishment of a new MPTCP connection together with the new sub-flows. In Figure 2.1 the two end points we consider are *Host A* with *Address 1* and *Address 2* and *Host B* with *Address 1.* In order to establish the so-called *main flow* from Host A to Host B, Host A receives the information that it can reach Host

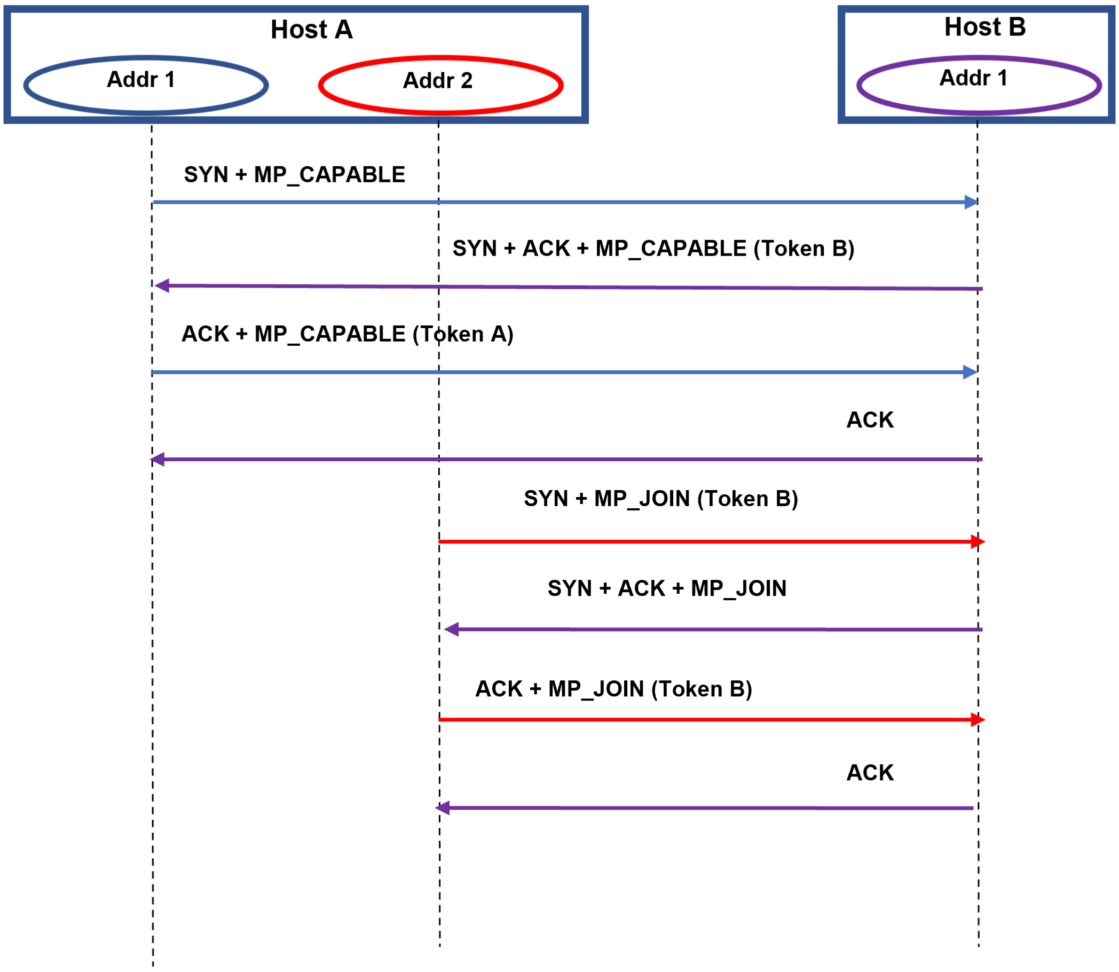


Figure 2.1: MPTCP Connection establishment

B at Address 1. Subsequently, a three-way handshake procedure is carried out, just like in the case of regular TCP in order to maintain the transparency to network and application layers. However, one modification to the regular procedure is added: the option field carries some MPTCP specific options which are understood only by the end-system stack.

One of the fundamental options in getting an MPTCP connection started is the *MP CAPABLE* one, which is used by both hosts to inform each other that they support MPTCP capability. The first handshake done to initiate the MPTCP session from the Host A, starts by sending a SYN segment with an MP CAPABLE option in order to advertise its capability of handling MPTCP connections. Host B then responds with a SYN+ACK and an MP CAPABLE option, announcing token B to Host A, which is a local identification of the connection. Host A now responds with an ACK and an MP CAPABLE option, announcing its token A. In the end a final ACK is sent from Host B to Host A to make sure that the final MPTCP security informations are received.

Now that the MPTCP session is complete, we focus on the additions of new sub-flows to the newly established session. Suppose that Host A wishes to create a sub-flow between its Address 2 and the Address 1 of Host B, then another TCP three-way handshake is carried out by the end points. This time, however the MPTCP option advertised is different since the new sub-flows need to be linked to the already existing MPTCP session; such new option is called *MP JOIN* and it is used to create a sub-flow, joining the master one. To be more specific, the hosts must exchange the identification tokens through the MP JOIN option: Host A must send token B to the peer to identify which MPTCP connection it is joining. Host B responds with a SYN+ACK with MP JOIN, then Host A sends an ACK with MP JOIN option. Finally, Host B sends an ACK and the new sub-flow is ready to be used.

Suppose a new address becomes available on one of the hosts during the session, then such address must be advertised through another MPTCP option called *ADD ADDRESS*. By doing so, the other host will be aware of the new path which can be used to establish a new sub-flow.[7]

If a previously announced address becomes invalid during the lifetime of a Multipath TCP connection, i.e., if the interface gets disconnected, then the affected host is required to announce such change in order to allow the peer to remove the sub-flows related to such address. The MPTCP option used in this case is called REMOVE ADDRESS. [3]

2.1.2 Data exchange

Once we have gone through the establishment scheme of an MPTCP connection, let us now explain the procedure used by Multipath TCP to exchange data across the multiple sub-flows.

As previously mentioned, due to the heterogeneity of the paths used to establish the sub-flows, MPTCP usually deals with out-of-order packet delivery caused by the different network conditions that packets go through when reaching the peer. In order to counteract such phenomenon, MPTCP provides two numbering schemes: a 64-bit Data Sequence Number (DSN), which numbers all data sent over the MPTCP connection, and a 32-bit sequence number space for each sub-flow, just like in regular TCP.

MPTCP then maps the sub-flow sequence space to the data sequence space through an option called *Data Sequence Mapping* which in carried by the Data Sequence Signal (DSS) and reassembles the data stream. The Data Sequence Mapping consists of the sub-flow sequence number, data sequence number, and length for which the mapping is valid. This option can also carry a connection-level acknowledgement (the "Data ACK") for the received DSN [3]. In fact, the data is scattered in segments over all the available sub-flows and it clearly needs to be acknowledged when received. MPTCP operates with a two layer acknowledgement: first, the data is acknowledged on each sub-flow, just like regular TCP flows, then the data is acknowledged by the data sequence numbering by using a cumulative Data Acknowledgement, which is sent on one of the sub-flows [7].

2.1.3 Connection release

Let us now review the closing procedure of an MPTCP connection. In the case of regular TCP, the FIN segment is announced by the host when the running application requires *close()* on a socket, meaning that it no longer has data to be sent to the peer. In MPTCP, instead when a FIN is sent from a host, the only connection to be closed is the one of the sub-flow over which the FIN is announced.

To successfully close an MPTCP connection, a mechanism equivalent to the FIN of regular TCP must be triggered through the so-called DATA FIN segment. The DATA FIN is sent by the host and it is acknowledged by the peer at the connection level with a DATA ACK, which is sent only once all the data on the MPTCP connection is successfully received. When the first DATA FIN is sent it triggers the return of both the DATA ACK and DATA FIN segments from the other host. The DATA FIN only needs to be sent on one sub-flow and from the moment that it is acknowledged, all remaining sub-flows close their connections with standard FIN exchanges. All of these FIN exchanges allows the middleboxes to clean up their state.

The MPTCP connection is considered to be completely closed once all the DATA FINs sent by both hosts are acknowledged by DATA ACKs. There is also an MPTCP option used to abruptly close the whole connection, called *MP FASTCLOSE*, however it is only used by specific implementations of Multipath TCP [7].

2.1.4 Congestion Control in MPTCP

One of the main components of TCP is the congestion controller, whose goal is to make sure that TCP can adapt its throughput dynamically in response to varying network conditions, such as packet loss, delay and congestion. In order to do this, each TCP sender maintains a congestion window, which manages the amount of packets that the sender can send without waiting for an acknowledgement.

The congestion window is updated dynamically and it grows linearly when there is no congestion, while it is halved when a packet loss occurs. If multiple streams are utilizing the same congested link, the TCP congestion controller will ensure fairness and the congestion window of each stream will converge to the same average value [8].

Let us define three different goals of the congestion controller to better understand how Multipath TCP handles congestion:

1. Fairness to TCP: if several sub-flows of an MPTCP session are running alongside a regular TCP connection over a bottleneck link, MPTCP should not be able to get more throughput than the TCP connection.

2. Incentive to deploy Multipath TCP: The performance of the sub-flows of an MPTCP session should be able to obtain at least the same throughput as a regular TCP connection.

3. Load balancing: the paths used to send data by the active sub-flows should be the ones which are experiencing less congestion than the others.

Multipath TCP is able to fulfil these goals by making simple adjustments to the congestion controller used by regular TCP connections. To be more specific, each sub-flow will have its own congestion controller which works exactly like a regular TCP one.[9]

2.1.5 Receiver Window

In TCP the receive window is a parameter used to implement flow control, which consists of throttling down fast senders when receivers can no longer keep up with the pace: the receiver advertises a receive window in each packet, telling the sender how much data the receiver can accept and store in its buffer. MPTCP uses a unique receive window, shared among all the sub-flows: each one of them can keep sending data as long as the receiver is willing to accept it. Using one receive window for each sub-flow, instead may lead to stalling some of them, while others would not use up their window

With MPTCP, all sub-flows share the same receive buffer and advertise the same receive window. A key design value is the dimension of the receive buffer. The lower bound to achieve full network utilization is the maximum bandwidth-delay product of any one of the paths, however this may not be sufficient when a packet is lost on a slower subflow and needs to be retransmitted. A tight upper bound could be the maximum RTT of any path multiplied by the total bandwidth available across all paths, allowing all subflows to continue transmitting at full speed while a packet is fast-retransmitted on the maximum RTT path. However, even this choice might be insufficient to achieve full performance in the case of a retransmit timeout on the maximum RTT path [3].

2.1.6 Schedulers

One of the main characteristics of Multipath TCP is the presence of multiple sub-flows for each connection, as stated previously, allowing the traffic to be multiplexed across the available paths, thus achieving a higher throughput. A question arises as to which path to use in order to divide the traffic; the element responsible for such choice is the scheduler, which is required to indicate the sub-flow to use for each segment transmission.

Usually the decision is done considering network properties of the sub-flows paths, however other policies can be used. In the standard version of MPTCP Linux Kernel the avaialble scheduling algorithms are Default, Round Robin and Redundant, which are here described.

Default Minimum RTT Scheduler The current default scheduler of the MPTCP Linux Kernel [10] is the Minimum RTT. It schedules the last packet of the sending queue to the sub-flow with the lowest Round Trip Time (RTT) and with a non-exhausted congestion window. Then, the next packet is sent on the sub-flow with the next higher RTT, and so on and so forth.

The minimum RTT scheduler is particularly effective in heterogeneous networks where scheduling data based on the lowest RTT is beneficial, improving the user experience. Basically, the default scheduler selects the 'fastest' sub-flow to send next packet [11]. Just like for the Round Robin scheduler, as soon as the congestion window is filled, the transmission over that sub-flow is blocked. When a packet is correctly received, the receiver sends the acknowledgement to make room in the congestion window and thus allow the scheduler to once again transmit data over the given path. As a negative aspect, the Minimum RTT scheduler allows an unfair distribution of the traffic load: the sub-flows with better performance metrics will have transmit most of the traffic [12].

Round Robin Scheduler Round Robin scheduler [13] is a simple scheduling mechanism. There is no priority among sub-flows: the scheduler selects them one after the other and skips the ones with an exhausted congestion window. This way, the capacity of each path is fully exploited and the distribution of the data packets among all the sub-flows is uniform. Its performance however is not optimal and it is only used for academic or testing purposes.[12]

Redundant Scheduler The redundant scheduler is designed to deal with heterogeneous networks that are subject to events such as failures and traffic load variation. Redundant packet scheduling aims to provide the best transmission in terms of throughput and latency for delay-sensitive applications, such as streaming services. Simply put, this scheduler tries to transmit the traffic on all available sub-flows in a redundant way, which is useful when one wishes to achieve the lowest possible latency by sacrificing the overall throughput, which is actually one of the main advantages of MPTCP transmission[14].

For what concerns this thesis, the default scheduler will be used for all the experiments, unless specified otherwise.

2.2 State of the art

One of the very first application scenarios of MPTCP is reported in [15] where in 2010 it was used to improve data center networking by performing very short time-scale distributed load balancing, making use of parallel paths. Since the release of the experimental standard in 2013, a lot of research work has been done about Multipath TCP implementation, performances and applications, focusing on different key characteristics of the protocol. Here we report the state-of-the-art studies in multipath TCP and its applications.

One of the main strengths as well as weaknesses of Multipath TCP is its rigidity: it makes its operation very clear and easy to investigate, however it does not provide much flexibility which may be needed in some working scenarios. A proxy-based solution for multipath, called *MPFlex* [16] is designed and implemented in order to employ a flexible software architecture of mobile multipath. The key characteristic of MPFlex is the usage of multiplexing to consolidate multiple (potentially short-lived) connections into two long-lived connections; this way, an application connection only needs one handshake to create the corresponding connection, instead of exchanging handshakes for every subflow in Multipath TCP, allowing all subflows to be usable sooner and improving the overall bandwidth utilization. Another peculiar aspect of the this design is that, unlike MPTCP, MPFlex is an OS service and can transparently provide multipath support for non-TCP protocols, such as reliable UDP, and SCTP [17], opening the door to new transport-layer innovation.

One of the limiting factors in the usage of Multipath TCP is the existence of numerous middle-boxes which do not support this protocol. Such topic is investigated in "Designing a Deployable Multipath TCP" [10]: this paper is about the design of the Multipath TCP protocol, taking into account different characteristics such as making it robust to path failures also in the presence of middleboxes that attempt to optimize single-path TCP flows, examining the practical limitations the OS poses on MPTCP design and how MPTCP metadata should be encoded. The research team encoded a patch to the Linux kernel which can deal with middleboxes really efficiently, operating safely through all the ones that were identified in the experiment. A new set of algorithms is proposed to solve the problem of sharing a limited receive buffer between multiple flows, proposing an adaptive solution: as more receive buffer becomes available, MPTCP should use up more of the capacity it has available. This way, if the OS is prepared to spend the memory it will achieve higher throughput; if not, it will receive the same as TCP. Regarding the MPTCP metadata, it is shown that some middleboxes pass TCP options that they don't understand. As a consequence, whenever a TCP extension changes the semantics of parts of the packet header, it must then include mechanisms to cope with middleboxes that do not understand the new semantics.

In this paper [18] they implement a multipath congestion control algorithm, which is crucial for the efficiency of MPTCP. In this study, the main focus is on adapting the sub flow windows of a multipath TCP in order to get the maximum performance possible, making sure that the existing TCP traffic can co-exist gracefully. In order to do that, two main constraints for the multipath congestion control algorithm are highlighted:

- Incentive on using multipath: a sub flow should give a connection at least as much throughput as it would get with single-path TCP on the best of its paths
- Fairness: a sub flow should take no more capacity on any path or collection of paths than if it was a single path TCP flow using the best of those paths

This algorithm is tested on various environments and is proven to be safe to use: it seamlessly balances traffic over 3G and WiFi radio links, as signal strength fades in and out, making sure that no harm is done to other traffic (fairness constraint) and that there is always an incentive to aggregate the different paths (incentive constraint).

One of the main issues of multipath TCP is to reorder the packets at receiver side due to the out-of-order data paired with bounded receive buffers. In the article "Tolerating path heterogeneity in multipath TCP with bounded receive buffers" [19], a new multipath TCP protocol called SC-MPTCP is proposed. Such protocol makes use of a linear systematic coding (SC), using coded packets as redundancy: instead of encoding all sent-out packets, some redundancy is transmitted first, followed by the original packets. The main contributions of such article are the following:

- mitigating buffer delay and jitter through coding
- pre-blocking warning mechanism to allow the receiver to retrieve missing packets without waiting for retransmissions
- aggregating all available subflows bandwidth, even if small

The results show that in a heterogeneous network context where the out-of-order data issue is more critical, SC-MPTCP requires much smaller receive buffers to approach the maximum aggregate throughput compared to regular MPTCP, i.e., the path heterogeneity has limited impact on the minimum required receive buffers.

Chapter 3

Implementation

This chapter describes the goal scenario as well as the steps needed to create the MPTCP testbed implementation used for the testing part. The testbed is realized to investigate the MPTCP performances and to compare them to the behaviour of TCP in similar scenarios. Just like regular TCP, Multipath TCP is only run by the end users, thus we only need to modify the endpoints of our network. In this case, the MPTCP capability is provided by an open-source Linux Kernel implementation.

In this chapter we will then go through both the hardware and software components required by the testbed for all tested scenarios, focusing in the end on the tools used to evaluate the performance of the network.

3.1 Scenario description

In this section we provide the reader with an extended overview of considered the scenario.

As previously stated in Chapter 1, the motivation for using Multipath TCP is due to the possibility of improving the user experience of Wind Tre's customers without having to add new resources into their network. Specifically Wind Tre is a telecom operator which offers both mobile and fixed Internet services, thus it is very interesting to investigate whether exploiting these two types of networks simultaneously could lead to an improved user experience.

As we know MPTCP is an extension of the layer 3 transmission protocol TCP, which is why we will first focus on the aggregated layer 3 throughput obtained using multiple paths to multiplex the data stream; then, we will move on to a selected application which is a the file transfer one, namely using FTP where we will investigate the transfer time of different files, verifying whether by aggregating more than one path a faster file transmission can be obtained.

As anticipated above, the final goal is to test both the fixed and mobile networks in cooperation, but of course we need to study the functioning of MPTCP in detail before we can move on to a more complex scenario. Because of that we have identified some scenarios with an increasing level of complexity, starting from a Local Area Network setup and ending in a multiple Wide Area Networks one, where eventually both fixed and mobile accesses will be used for transmission. The detailed presentation of such scenarios is presented in Section 4.1, while here we just anticipate the main concepts.

The final goal is to speed up the file transfer procedure for a Customer with several offices located in different areas. In Figure 3.1 we report a typical scenario with two Customers' offices which continuously transfer files throughout the day to exchange work related data over the Internet. This setting is the one we aim to collect some data on, in order to prove the effectiveness of Multipath TCP in the Customer's experience.

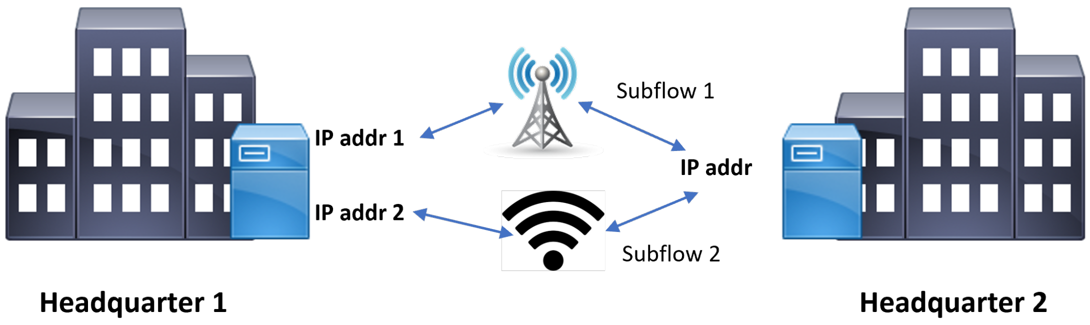


Figure 3.1: Multi-homing goal scenario

3.2 Testbed setup

The testbed is composed of the following hardware elements.

Client:

- Ethernet NIC: Realtek RTL810xE Fast Ethernet
- WiFi NIC: Intel dual band wireless AC 3160
- LTE: Xiaomi Redmi Note 7 connected via USB
- CPU: Intel Core i3-4030u
- OS: Ubuntu 20.04

Local Server:

- WiFi NIC:TP LINK Archer T4E connected via PCI Express
- CPU: Intel Core i5-8400
- OS: Ubuntu 20.04

Azure Server:

- vCPUs: 2
- RAM: 8GiB
- OS: Ubuntu 20.04

Router: Fastgate DGA4131FWB.

It is important to point out that, since we are using an Azure server, we do not know how the physical hardware is shared with other users and also the background traffic impact is unknown to us, and it may vary between tests.

3.2.1 Jupyter Notebook on Visual Code Studio

Some tests carried out in this thesis were at times repetitive and quite simple, but also time consuming for the tester. In order to minimize the time invested in such cumbersome tasks, a Jupyter Notebook was deployed: it lets the user combine different executable Python source codes in independent code cells. This way, we created a cell for basically each test scenario in order to switch on and off the necessary interfaces, modify the traffic shaper and basically carry out the experiments in a smarter way.

We will report in Appendix A portions of the Notebook to provide the reader with an idea of the basic functioning of this very helpful tool.

3.3 MPTCP configuration

The computers involved in the tests are using MPTCP enabled kernels using the off-the-shelf Linux kernel implementation [14]. Let us now explain how to configure the MPTCP options in such kernel. To modify the kernel parameters to configure the system variables at runtime, the *sysctl* is used.

The available MPTCP options are:

- *net.mptcp.enabled*: This option enables the use of MPTCP when it is set with the value 1, otherwise it can be disabled by setting the value to 0;
- *net.mptcp.mptcp_syn_retries*: This option configures the number of retransmitted SYN with the MP CAPABLE option after which the MP CAPABLE option will no longer be sent on the SYN. The default value is 3. This option exists to handle middleboxes that drop SYNs with unknown TCP options.
- *net.mptcp.mptcp_checksum*: This option enables/disables the use of MPTCP checksum when set with the value 1 or 0.
- *net.mptcp.mptcp_path_manager*: Path manager is a structure that allows to choose between the two compiled path-managers.
  This structure is necessary for the creation of new sub-flows and also to advertise alternative IP addresses through the ADD ADDR option. This option has three possible values:
  - default: the path manager will accept passive creation of new subflows. The host will not create new subflows nor it will announce the available IP addresses.
  - fullmesh: the path manager will create a full-mesh between the available subflows and all addresses.

3.3.1 Scheduler configuration

The schedulers are introduced in section 2.1.6, while here we present the choices in the MPTCP kernel implementation. At run-time you can select one of the compiled schedulers through the sysctl *net.mptcp.mptcp_scheduler*, namely:

- default;
- roundrobin;
- redundant.

3.4 Evaluation tools

In order to analyse and test the behaviour of Multipath TCP, several well-known tools were used, such as **Wireshark**, **iPerf**, **Ping** and **Sockperf**. Wireshark [20] is a network protocol analyser which is widely used in the experiments. The captured network data allows to measure the performance of the protocol by examining the packets overhead and evaluating the throughput of the transferred data, the load balancing among the sub-flows, the connection establishment time and the handover among connections.

iPerf is a tool used for network performance measurement and tuning. It has client and server functionality, and it can create data streams to measure the throughput between the two end points [21].

PING is a Linux command used to check the network connectivity between host and server by sending a data packet to the specified address with the message “PING”; once the host receives a response from the server/host, the time elapsed is recorded and is a measure of the latency [22]. Ping uses ICMP(Internet Control Message Protocol) to send an ICMP echo message to the specified host, which is why the latency had to be measured differently when involving the Azure server: in fact, the Azure firewalls do not allow ICMP messages, which made the use of the PING command not feasible.

SockPerf is a network testing tool used to measure network latency by creating UDP/TCP data streams between two end points. This tool provides a client and server functionality, and can measure the throughput and latency between the two ends [23]. In our experiments it is used to measure the latency when the Azure server is involved.

By using the performance tools described above we can obtain all the experimental data necessary to validate the tested solution. Once the results of Multipath TCP are collected, we can compare them to the regular TCP solution in order to prove that MPTCP performs either as good or better than TCP in some scenarios.

Chapter 4

Methodology and evaluation

The main goal of using our testbed is to compare the performances of TCP and Multipath TCP protocols. In order to provide a meaningful comparison, several tests are carried out in various network scenarios where different types of access technologies are used. This chapter therefore describes the methodology and the evaluation of the implemented testbed in order to investigate the behaviour of Multipath TCP.

In Section 4.1 we go through the considered scenarios and the way the experiments were conducted, using the components presented in detail in Chapter 3. In Section 4.4 the metrics used to evaluate the performance of MPTCP are presented.

4.1 Tests description

As anticipated in Chapter 3.1 we identified some test scenarios with an increasing level of complexity on which all tests are performed. They are of course all real environment ones and are here described in detail, ranging from Scenario A, which considers TCP in a LAN, to Scenario E, in which we will investigate MPTCP performances with two WANs.

4.1.1 Scenario A - TCP in LAN with WiFi and Ethernet connections

First of all, we start by consolidating our setup in a more controlled environment: a Local Area Network where simple TCP is initially used, where every behaviour can be somewhat explained and understood fully. In this current scenario we will investigate the TCP throughput behaviour which will later on be used when studying the behaviour of MPTCP sub-flows.

Scenario A.1 - WiFi only TCP

In Figure 4.1 we report the configuration of scenario A.1 in which the Client is connected to the Server in a LAN entirely over WiFi.

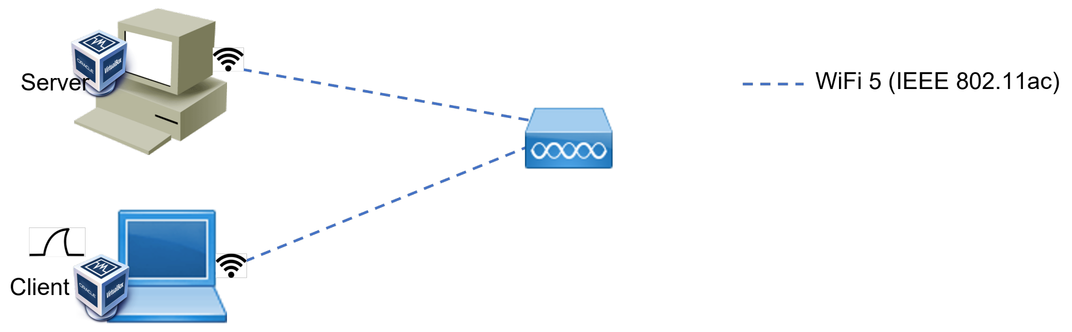


Figure 4.1: Scenario A.1: WiFi only

Scenario A.2 - Ethernet only TCP

In Figure 4.2 we report the configuration of scenario A.2 in which the Client is connected to the Access Point over Ethernet, while the Server over WiFi.

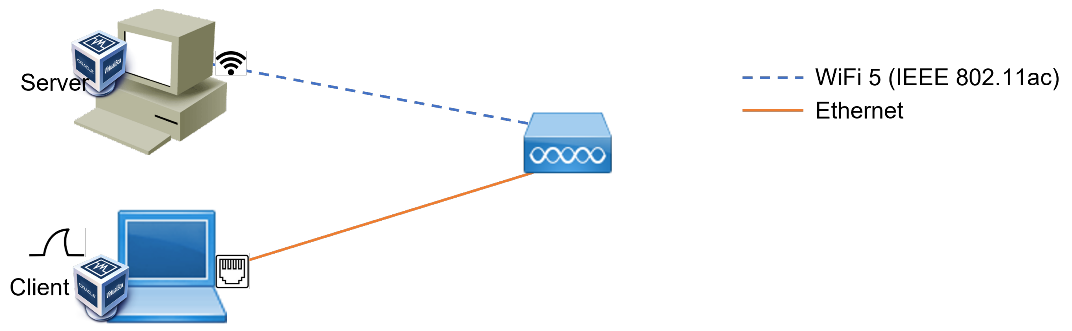


Figure 4.2: Scenario A.2: Ethernet only

4.1.2 Scenario B - MPTCP in LAN with WiFi and Ethernet connections

In Figure 4.3 we report the first configuration where MPTCP is deployed, which we will refer to as Scenario B. In this setting, the Client is connected to the Server in LAN over both WiFi and Ethernet simultaneously.

In this scheme we will study the basic functioning of Multipath TCP, investigating the connection establishment procedure for the sub-flows and the throughput aggregation advantage.

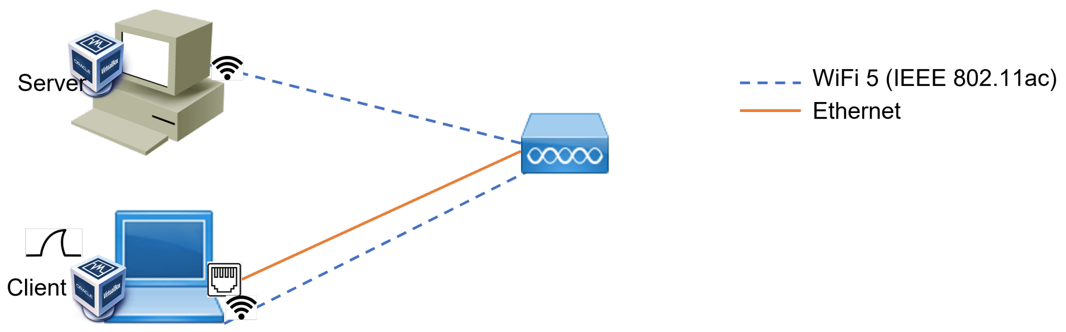


Figure 4.3: Scenario B: WiFi and Ethernet in LAN

4.1.3 Scenario C - MPTCP in WAN with WiFi and Ethernet connections

Scenario C is the first one considering a Wide Area Network where the Client needs to reach the Server over the Internet. This detail certainly adds a level of complexity and reality to the experiment, making it closer to the goal scenario described in Chapter 3.1. In this setting, the Client is connected through WiFi and Ethernet to the Server.

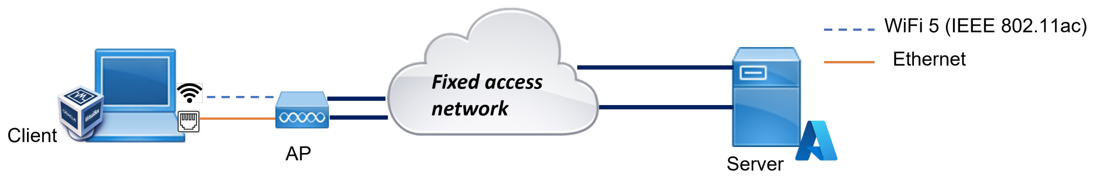


Figure 4.4: Scenario C: WiFi and Ethernet in WAN

4.1.4 Scenario D - Multihoming MPTCP in WAN with Ethernet and LTE connections

Scenario D is very similar to the previous one as it considers a Wide Area Network, as well, but it actually considers two of them: the fixed access network and the mobile one. Specifically, the first network is reached through Ethernet, while the second over LTE.

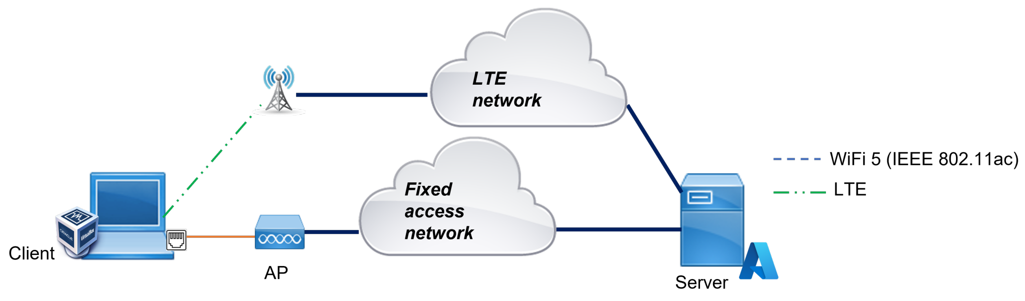


Figure 4.5: Scenario D: WiFi and LTE in WAN

4.1.5 Scenario E - Multihoming MPTCP in WAN with WiFi and LTE connections

Similarly to Scenario D, this scenario involves two WANs: the fixed access network is reached through a WiFi access, while the mobile one is reached through LTE.

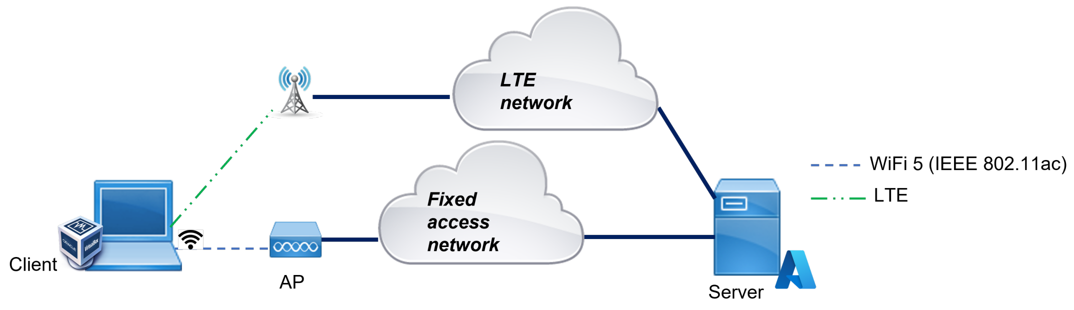


Figure 4.6: Scenario E: LTE and Ethernet in WAN

4.2 TCP throughput analysis

Before investigating the behaviour of Multipath TCP, it is better to test the performance of regular TCP in Scenario A. Specifically, we will investigate the behaviour of the throughput of wireless and wired connections in LAN in order to draw some conclusions from them in more complex scenarios.

Focusing first on the downlink connection, we can compare the behaviour of the two access technologies in Figures 4.7 and 4.8, where the results for the WiFi and Ethernet connections, respectively are reported.

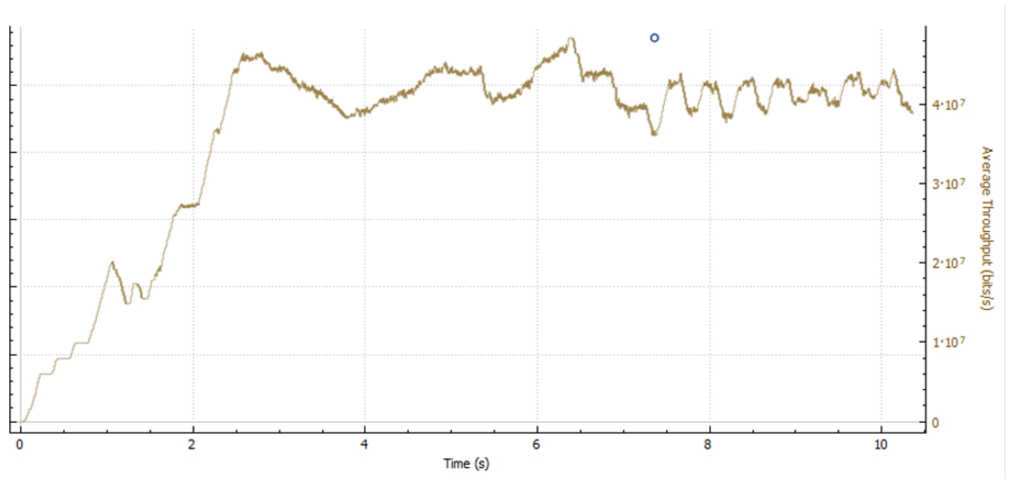


Figure 4.7: Download speed of wireless connection over TCP

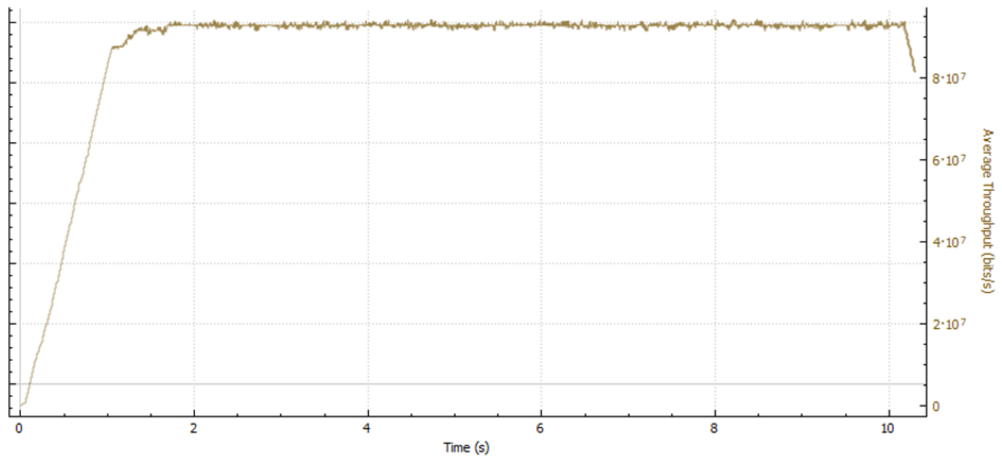


Figure 4.8: Download speed of wired connection over TCP

Due to the nature of the shared channel of WiFi, it is expected to obtain some time-varying behaviour of the throughput, which is indeed evident in Figure 4.7, while Ethernet appears to be more stable and less prone to channel interference, as reported in Figure 4.8.

The exact same observations can be done on the uplink connection, where a more stable throughput value is achieved by the wired connection in Figure 4.10 while in Figure 4.9 a more unsteady behaviour is experienced by the wireless connection.

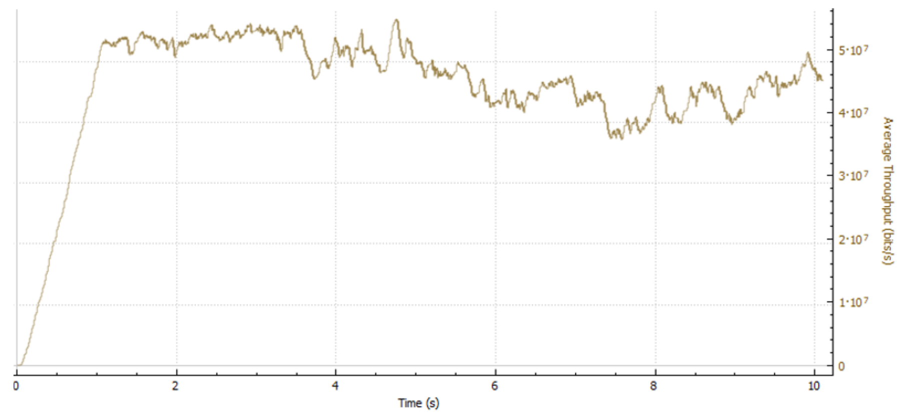


Figure 4.9: Upload speed of wireless connection over TCP

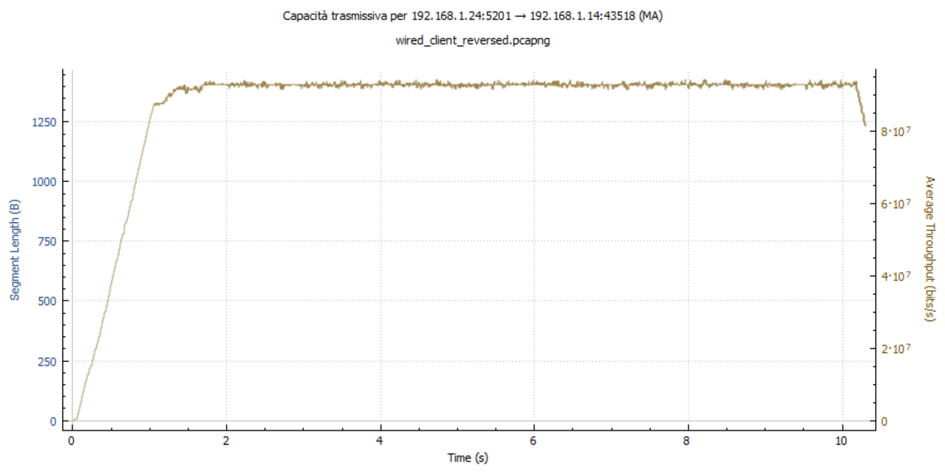
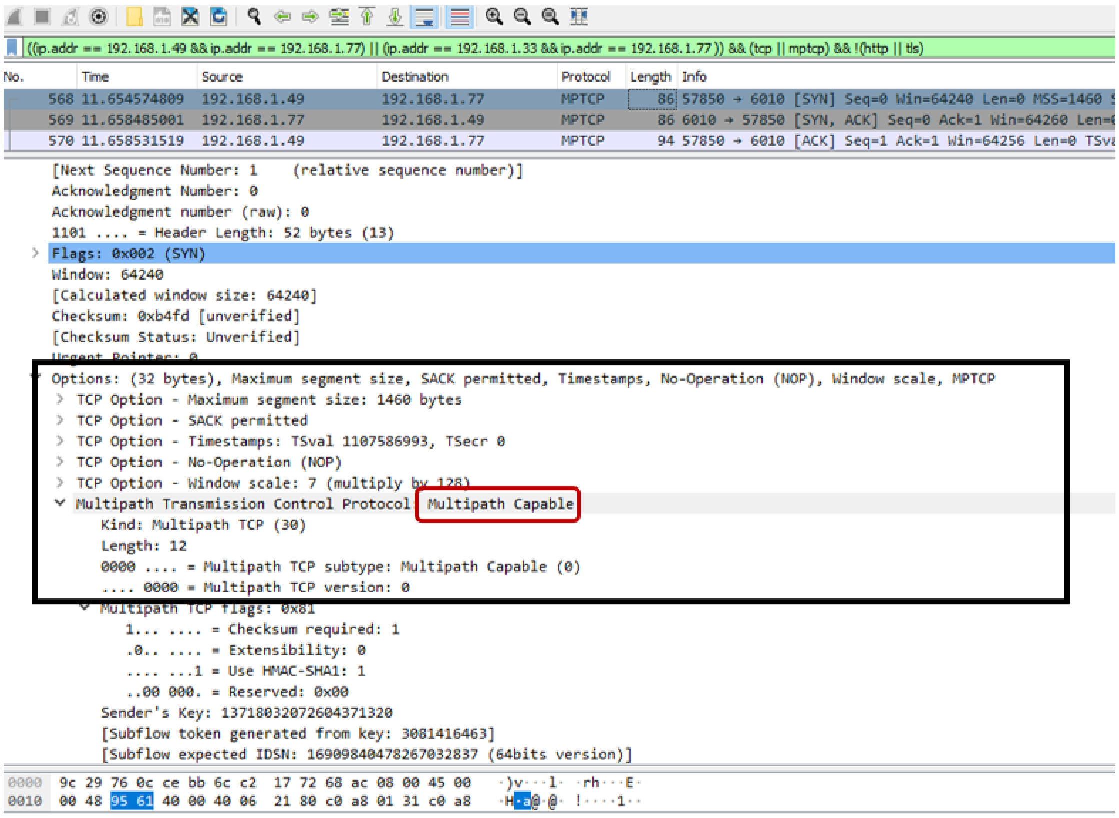


Figure 4.10: Upload speed of wired connection over TCP

This first and very basic observation will be very helpful when explaining the results obtained with Multipath TCP, as it highlights the low predictability of the available wireless throughput when aggregating the sub-flows, while the contribution of the Ethernet connection will be more stable and clear in the results.

4.3 MPTCP connection establishment

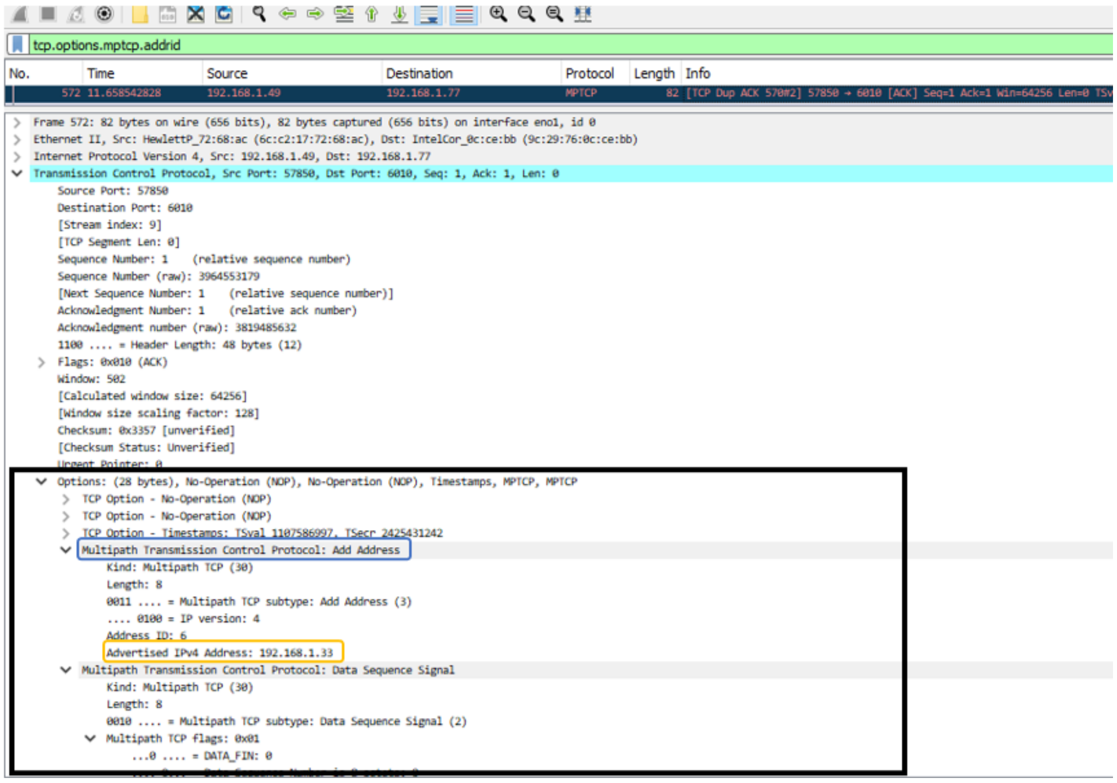
In this section we will go through the experimental data collected through Wireshark to validate the theoretical behaviour of MPTCP described in Section 2.1.1. The results here presented are obtained in the so-called Scenario B.



```
((ip.addr == 192.168.1.49 && ip.addr == 192.168.1.77) || (ip.addr == 192.168.1.33 && ip.addr == 192.168.1.77)) && (tcp || mptcp) && !(http || tls)

No.  Time          Source        Destination   Protocol  Length Info
 568 11.654574809  192.168.1.49  192.168.1.77  MPTCP         86 57850 → 6010 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
 569 11.658485001  192.168.1.77  192.168.1.49  MPTCP         86 6010 → 57850 [SYN, ACK] Seq=0 Ack=1 Win=64260 Len=
 570 11.658531519  192.168.1.49  192.168.1.77  MPTCP         94 57850 → 6010 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSva

    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1101 .... = Header Length: 52 bytes (13)
  > Flags: 0x002 (SYN)
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0xb4fd [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (32 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale, MPTCP
      > TCP Option - Maximum segment size: 1460 bytes
      > TCP Option - SACK permitted
      > TCP Option - Timestamps: TSval 1107586993, TSecr 0
      > TCP Option - No-Operation (NOP)
      > TCP Option - Window scale: 7 (multiply by 128)
      v Multipath Transmission Control Protocol: Multipath Capable
          Kind: Multipath TCP (30)
          Length: 12
          0000 .... = Multipath TCP subtype: Multipath Capable (0)
          .... 0000 = Multipath TCP version: 0
        v Multipath TCP flags: 0x81
            1... .... = Checksum required: 1
            .0.. .... = Extensibility: 0
            .... ...1 = Use HMAC-SHA1: 1
            ..00 000. = Reserved: 0x00
          Sender's Key: 13718032072604371320
          [Subflow token generated from key: 3081416463]
          [Subflow expected IDSN: 16909840478267032837 (64bits version)]

0000  9c 29 76 0c ce bb 6c c2  17 72 68 ac 08 00 45 00   ·)v···l· ·rh···E·
0010  00 48 95 61 40 00 40 06  21 80 c0 a8 01 31 c0 a8   ·H·a@·@· !····1··
```

Figure 4.11: MP capable

In Figure 4.11 the Client informs the Server of its capability to support multipath through the Multipath Capable option. The first flow to be established is the one from the Ethernet interface of the Client towards the Server: this is defined as the *master flow* while the second subflow to be joining the master one is announced later on in the connection.

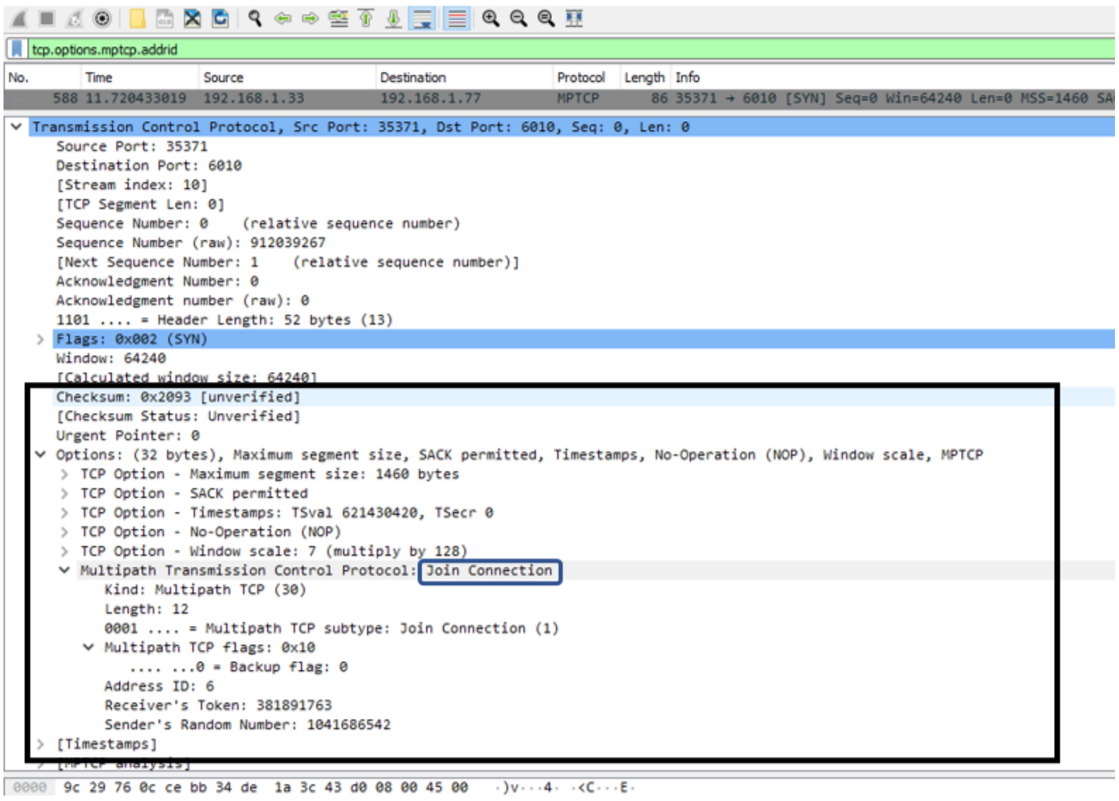


```
tcp.options.mptcp.addrid

No.  Time          Source        Destination   Protocol  Length  Info
572  11.658542828  192.168.1.49  192.168.1.77  MPTCP     82      [TCP Dup ACK 570#2] 57850 → 6010 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv

> Frame 572: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface eno1, id 0
> Ethernet II, Src: HewlettP_72:68:ac (6c:c2:17:72:68:ac), Dst: IntelCor_0c:ce:bb (9c:29:76:0c:ce:bb)
> Internet Protocol Version 4, Src: 192.168.1.49, Dst: 192.168.1.77
v Transmission Control Protocol, Src Port: 57850, Dst Port: 6010, Seq: 1, Ack: 1, Len: 0
    Source Port: 57850
    Destination Port: 6010
    [Stream index: 9]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3964553179
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3819485632
    1100 .... = Header Length: 48 bytes (12)
  > Flags: 0x010 (ACK)
    Window: 502
    [Calculated window size: 64256]
    [Window size scaling factor: 128]
    Checksum: 0x3357 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  v Options: (28 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps, MPTCP, MPTCP
    > TCP Option - No-Operation (NOP)
    > TCP Option - No-Operation (NOP)
    > TCP Option - Timestamps: TSval 1107586997, TSecr 2425431242
    v Multipath Transmission Control Protocol: Add Address
        Kind: Multipath TCP (30)
        Length: 8
        0011 .... = Multipath TCP subtype: Add Address (3)
        .... 0100 = IP version: 4
        Address ID: 6
        Advertised IPv4 Address: 192.168.1.33
    v Multipath Transmission Control Protocol: Data Sequence Signal
        Kind: Multipath TCP (30)
        Length: 8
        0010 .... = Multipath TCP subtype: Data Sequence Signal (2)
      v Multipath TCP flags: 0x01
          ...0 .... = DATA_FIN: 0
```

Figure 4.12: Add address flag

Once the master sub-flow is established on the Ethernet interface, the Client advertises the available address of its WiFi interface through the Add Address MultiPath TCP option, as shown in Figure 4.12.

The information on the available IP address and address ID are eventually used when joining the subflow to the master one. In this case, referring to scenario B, the Client is advertising the address available on the wireless interface.

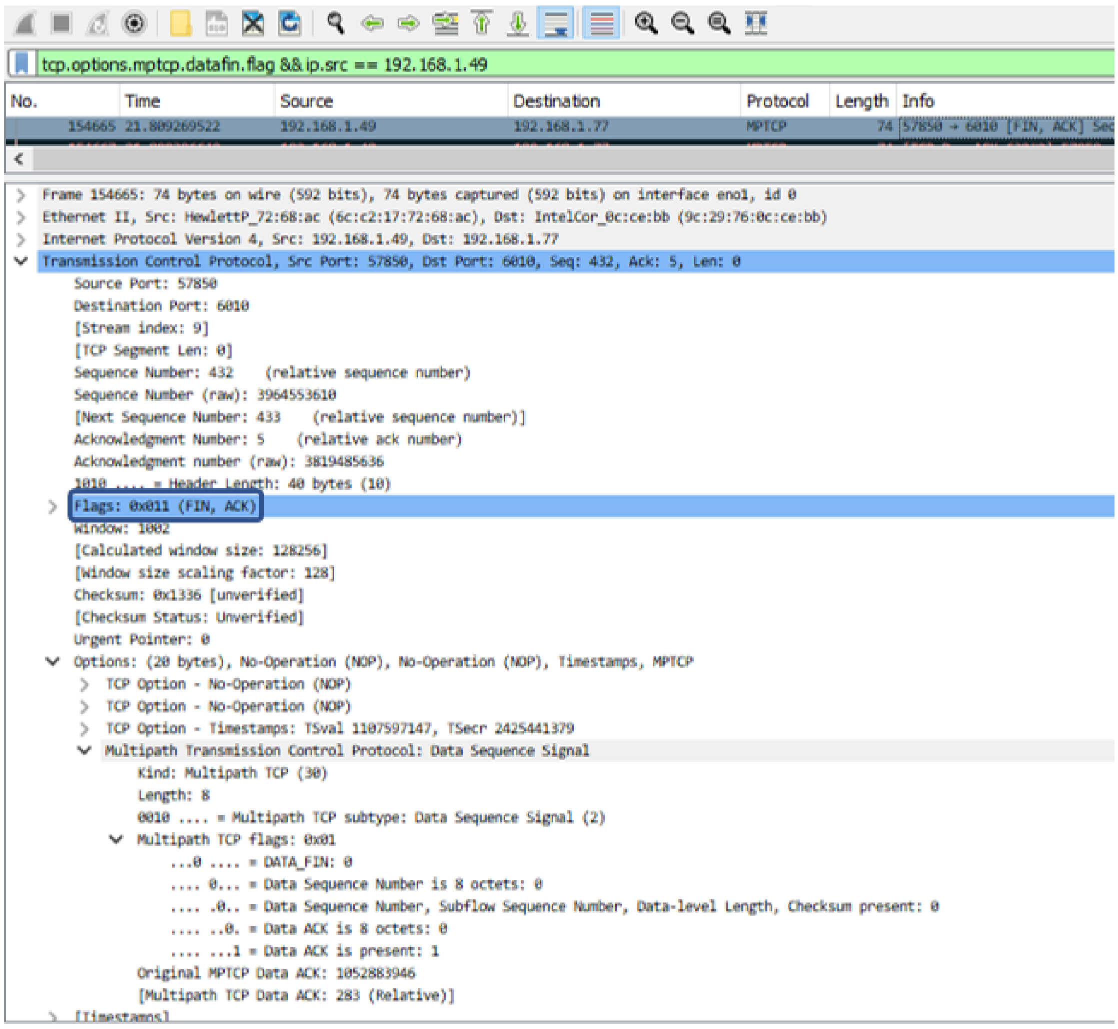


```
tcp.options.mptcp.addrid

No.     Time           Source          Destination     Protocol  Length Info
    588 11.720433019   192.168.1.33    192.168.1.77    MPTCP         86 35371 → 6010 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SA

v Transmission Control Protocol, Src Port: 35371, Dst Port: 6010, Seq: 0, Len: 0
    Source Port: 35371
    Destination Port: 6010
    [Stream index: 10]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 912039267
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1101 .... = Header Length: 52 bytes (13)
  > Flags: 0x002 (SYN)
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0x2093 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  v Options: (32 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale, MPTCP
    > TCP Option - Maximum segment size: 1460 bytes
    > TCP Option - SACK permitted
    > TCP Option - Timestamps: TSval 621430420, TSecr 0
    > TCP Option - No-Operation (NOP)
    > TCP Option - Window scale: 7 (multiply by 128)
    v Multipath Transmission Control Protocol: Join Connection
        Kind: Multipath TCP (30)
        Length: 12
        0001 .... = Multipath TCP subtype: Join Connection (1)
      v Multipath TCP flags: 0x10
          .... ...0 = Backup flag: 0
        Address ID: 6
        Receiver's Token: 381891763
        Sender's Random Number: 1041686542
  > [Timestamps]
    [MPTCP analysis]
0000  9c 29 76 0c ce bb 34 de  1a 3c 43 d0 08 00 45 00   ·)v···4· ·<C···E·
```

Figure 4.13: MP Join flag

Once the additional address and the address ID are advertised by the Client, a new sub-flow with the Server is created through the «Join Connection» option. The Client announces the will to join the MPTCP session, as reported in Figure 4.13.

Finally, once the data exchange happened, in the capture of Figure 4.14 we report the Client sending a «DATA,FIN» packet to close all the sub-flows exploited in the MPTCP session.



```
tcp.options.mptcp.datafin.flag && ip.src == 192.168.1.49

No.     Time          Source         Destination    Protocol  Length  Info
154665  21.809269522  192.168.1.49   192.168.1.77   MPTCP     74      57850 → 6010 [FIN, ACK] Seq

> Frame 154665: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eno1, id 0
> Ethernet II, Src: HewlettP_72:68:ac (6c:c2:17:72:68:ac), Dst: IntelCor_0c:ce:bb (9c:29:76:0c:ce:bb)
> Internet Protocol Version 4, Src: 192.168.1.49, Dst: 192.168.1.77
v Transmission Control Protocol, Src Port: 57850, Dst Port: 6010, Seq: 432, Ack: 5, Len: 0
    Source Port: 57850
    Destination Port: 6010
    [Stream index: 9]
    [TCP Segment Len: 0]
    Sequence Number: 432    (relative sequence number)
    Sequence Number (raw): 3964553610
    [Next Sequence Number: 433    (relative sequence number)]
    Acknowledgment Number: 5    (relative ack number)
    Acknowledgment number (raw): 3819485636
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x011 (FIN, ACK)
    Window: 1002
    [Calculated window size: 128256]
    [Window size scaling factor: 128]
    Checksum: 0x1336 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  v Options: (20 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps, MPTCP
    > TCP Option - No-Operation (NOP)
    > TCP Option - No-Operation (NOP)
    > TCP Option - Timestamps: TSval 1107597147, TSecr 2425441379
    v Multipath Transmission Control Protocol: Data Sequence Signal
        Kind: Multipath TCP (30)
        Length: 8
        0010 .... = Multipath TCP subtype: Data Sequence Signal (2)
      v Multipath TCP flags: 0x01
          ...0 .... = DATA_FIN: 0
          .... 0... = Data Sequence Number is 8 octets: 0
          .... .0.. = Data Sequence Number, Subflow Sequence Number, Data-level Length, Checksum present: 0
          .... ..0. = Data ACK is 8 octets: 0
          .... ...1 = Data ACK is present: 1
        Original MPTCP Data ACK: 1052883946
        [Multipath TCP Data ACK: 283 (Relative)]
  > [Timestamps]
```

Figure 4.14: Data Fin flag

4.4 MPTCP analysis

What we look at to study these experiments is

- subflow analysis
- experimental data for throughput aggregation
- experimental data for path utilization
- behaviour of MPTCP for a file transfer application

We will eventually figure out that some sub-optimal performances are caused by the scheduling algorithm, the flow duration and the unpredictability of the available resources.

Chapter 5

Results

In this section we will present all the obtained results. In each subsection we will investigate a given metric for each considered scenario in order to test the main characteristics of Multipath TCP.

First, we will consider the aggregated throughput measure, analysing the throughput advantage of this protocol in both LAN (Section 5.1) and WAN (Section 5.2); we will then go through the path utilization, testing the load balancing property again in both LAN (Section 5.3) and WAN (Section 5.4).

All the above described tests are carried out selecting the default scheduler, which is why Section 5.5 is reported, where we investigate the impact of the Round Robin scheduler in the path utilization of some of the available scenarios.

In the end we will test MPTCP with a file transfer application in Section 5.6.

5.1 MPTCP Throughput in LAN

Let us consider the so-called Scenario B, where the Client is connected to the Server through a wired and a wireless connection in a LAN environment. The throughput here presented is evaluated with several iPerf tests initiated by the Client, both in default and reverse mode. The MPTCP scheduler is the default one.

5.1.1 Sub-flow analysis

In Figures 5.1 and 5.2 we report the throughput in DL of the isolated sub-flows for the wireless and the wired paths, respectively.

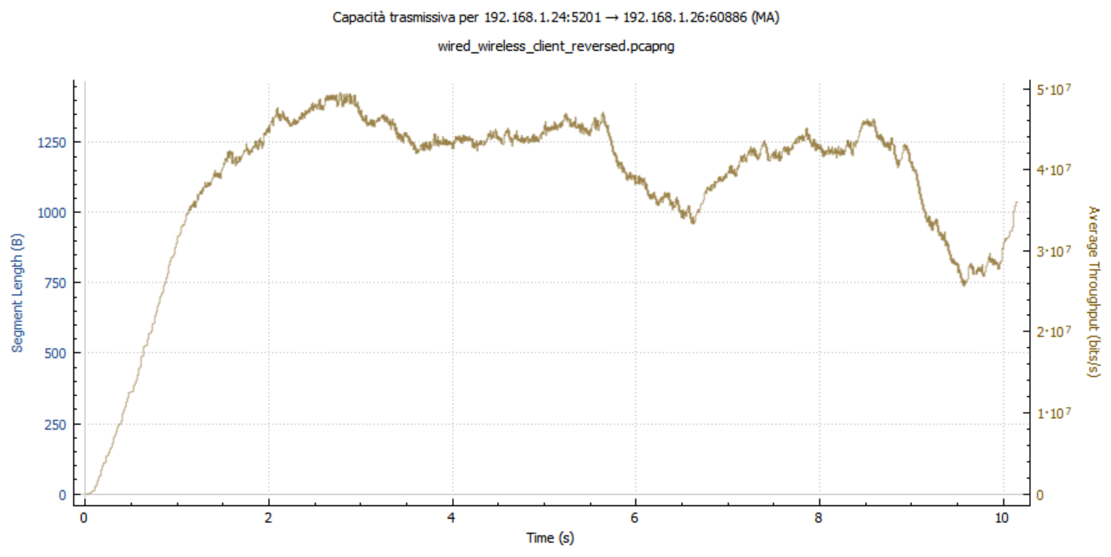


Figure 5.1: Download speed of wireless connection over MPTCP

As expected, the Ethernet path is more robust to interference, thus the average throughput has a constant behaviour over the connection time, unlike the WiFi path for which the bit rate oscillates quite a bit.

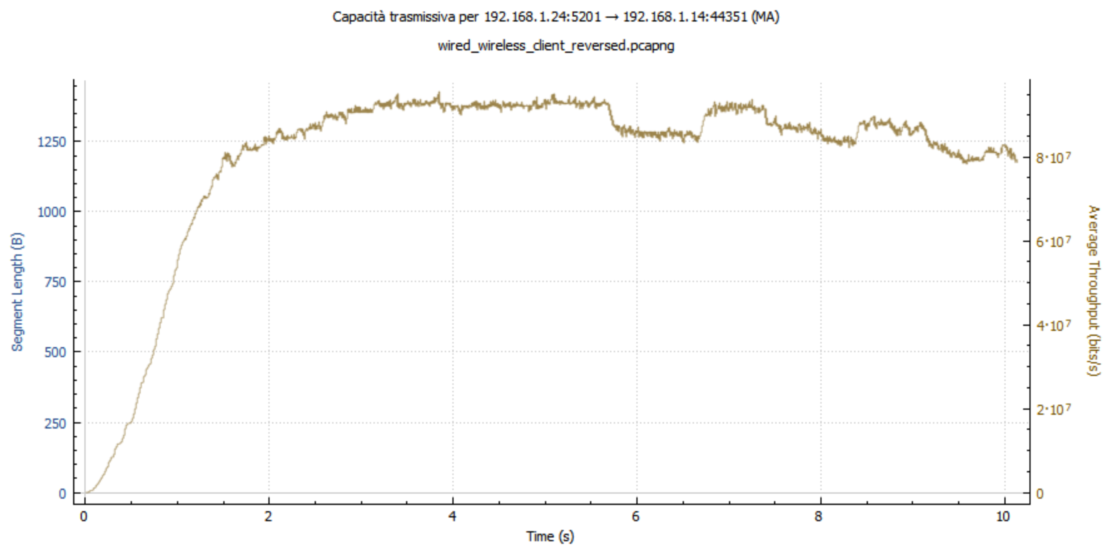


Figure 5.2: Download speed of wired connection over MPTCP

Similarly, in Figures 5.3 and 5.4 we report the throughput in UL of the isolated sub-flows for the wireless and the wired paths, respectively.

Once again, the Ethernet path appears to be more robust than the WiFi one.

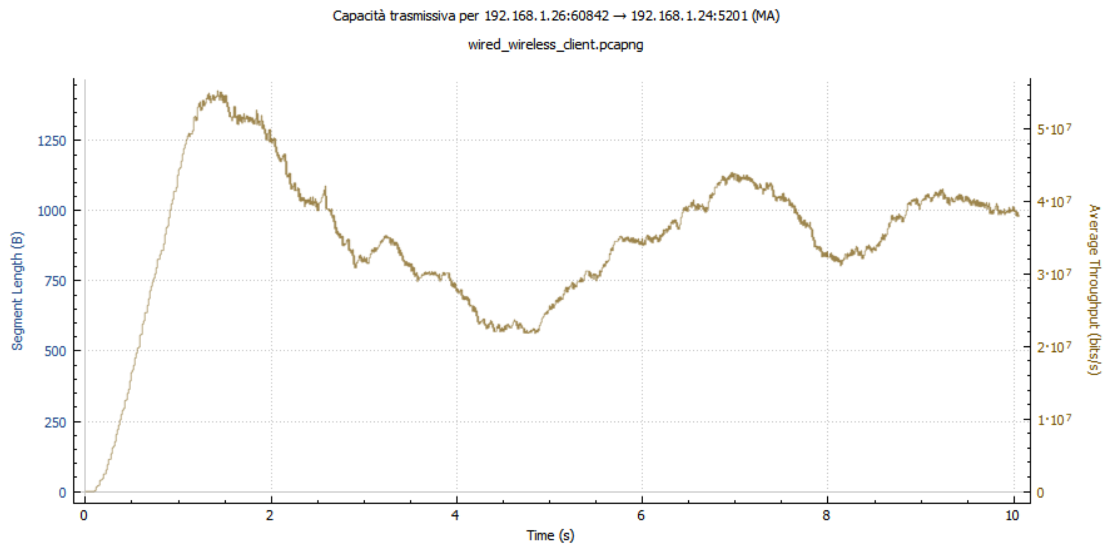


Figure 5.3: Upload speed of wireless connection over MPTCP

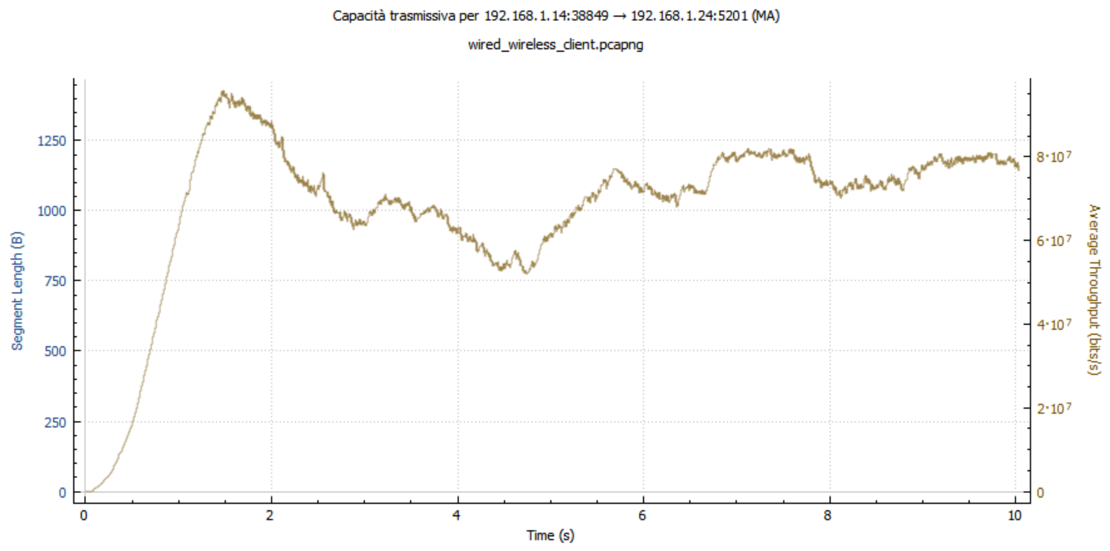


Figure 5.4: Upload speed of wired connection over MPTCP

Let us point out how the throughput of the isolated wired sub-flow at Client side in Figure 5.4 appears to be less stable than the one of the wired TCP connection. This instability is most likely to be caused by the Server side, which is connected to the access point over the wireless interface: when interference impairs the connection from the AP to the Server, the throughput experienced by the wired Client will degrade, as well.

5.1.2 Throughput aggregation scenario B

In Figures 5.5 and 5.6 we can appreciate the gain introduced by MPTCP in terms of throughput aggregation.

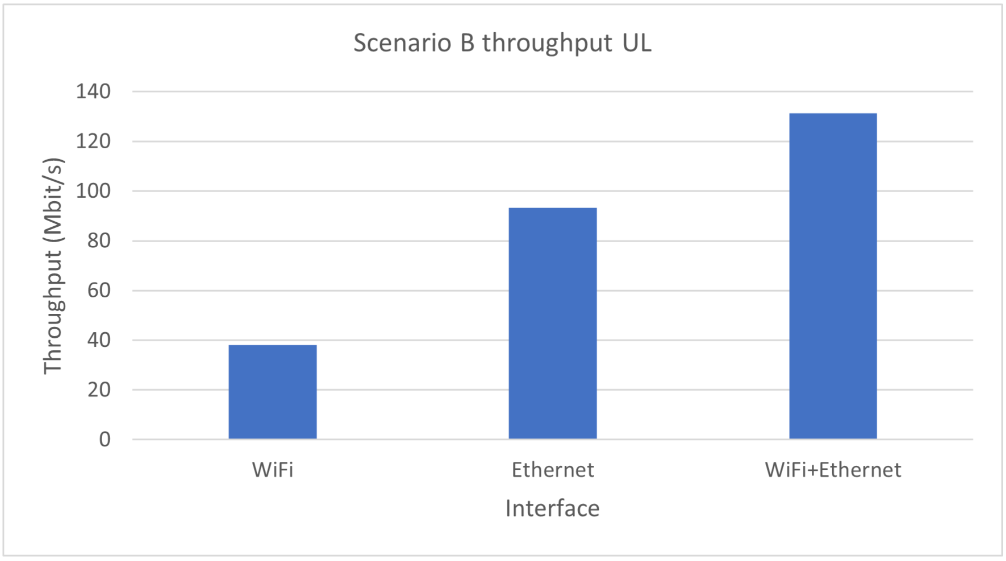


Figure 5.5: Aggregated upload throughput in Scenario B

From these first results we can notice the impact of Multipath TCP: in the uplink measurement the aggregated throughput is roughly the sum of the Ethernet and WiFi throughputs alone, which is the ideal case limit; in the downlink measurement the aggregation is also very impactful, achieving $115Mbit/s$ versus the $93Mbit/s$ of Ethernet only.

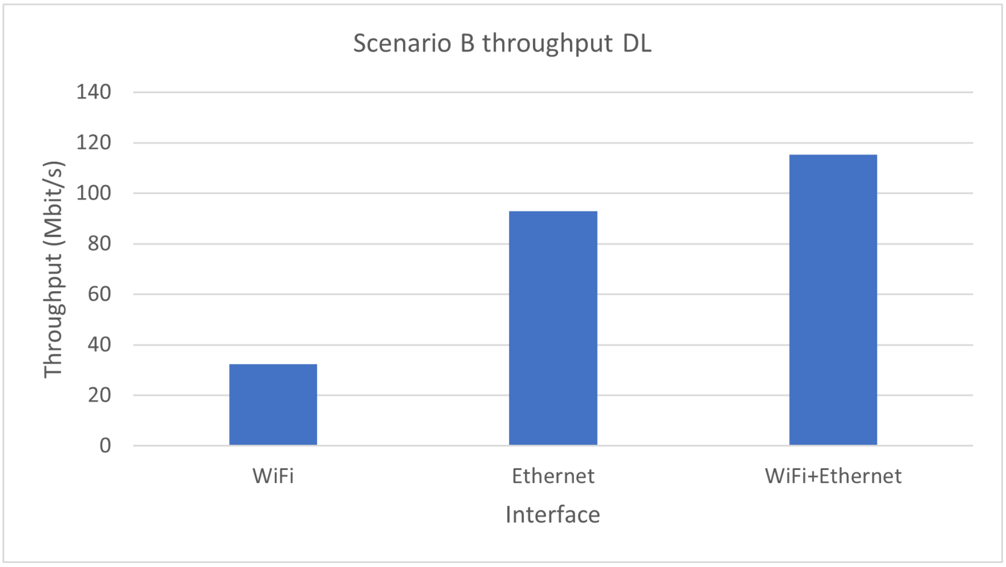


Figure 5.6: Aggregated download throughput in Scenario B

We can conclude that in a controlled environment like a Local Area Network, Multipath TCP's aggregating property is really meaningful in both directions.

5.2 MPTCP Throughput in WAN

Let us consider the remaining scenarios where the throughput is once again evaluated with several iPerf tests initiated by the Client, both in default and reverse mode and the MPTCP scheduler is the default one.

5.2.1 Throughput aggregation scenario C

Here we report the resulting aggregated throughput of Scenario C (Figure 5.7 and 5.8) in both uplink and downlink direction, when the MPTCP scheduler is set to default.

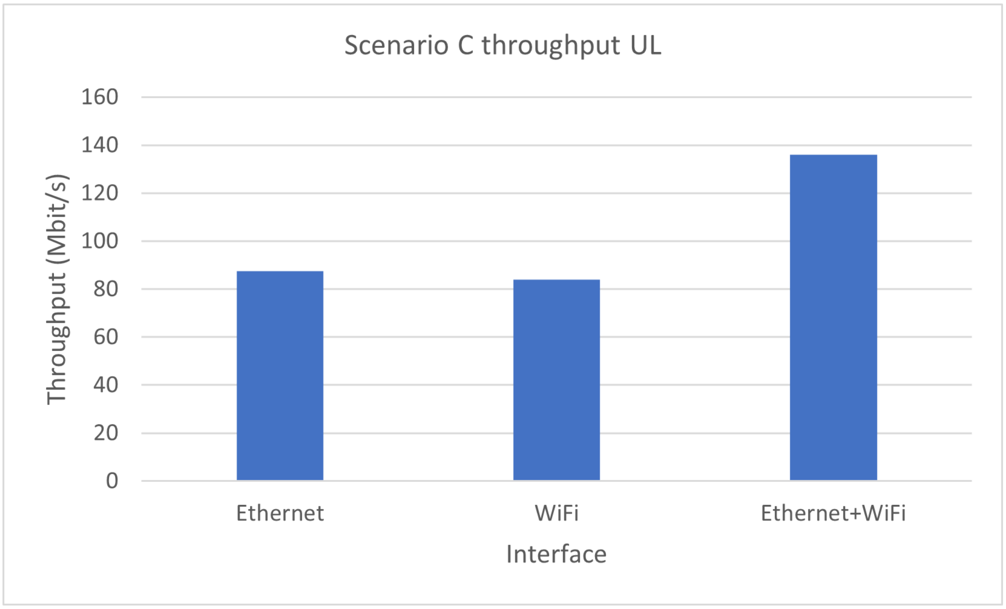


Figure 5.7: Aggregated upload throughput in Scenario C

Similarly to the results obtained in the Local Area Network, we can appreciate the impact of Multipath TCP in both directions: in the uplink measurement the WiFi throughput is pretty much comparable to the Ethernet one, thus the aggregated one reaches up to $137Mbit/s$; in the downlink measurement the WiFi contribution is smaller but still comparable to the Ethernet one, thus the aggregated throughput increases up to $87Mbit/s$.

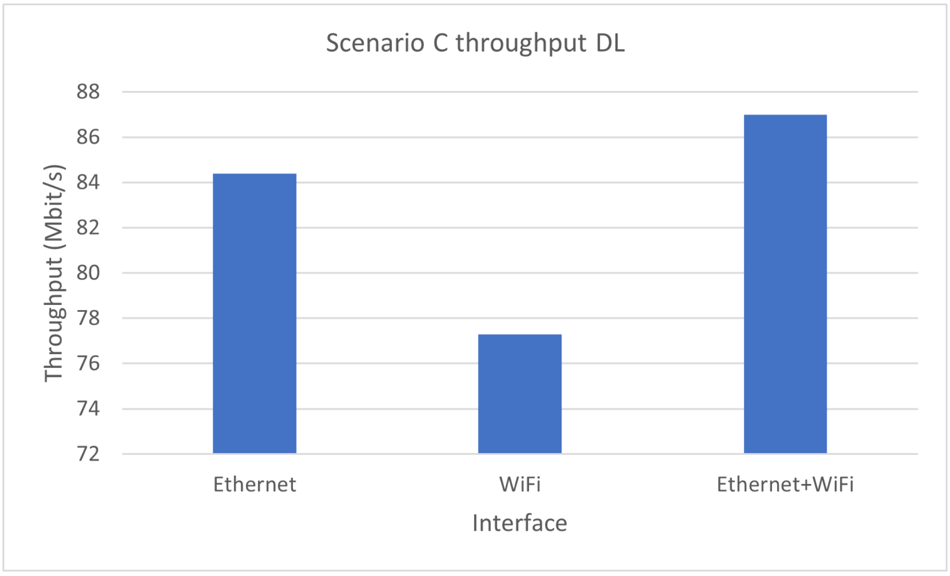


Figure 5.8: Aggregated download throughput in Scenario C

We can conclude that even in an environment where we do not have control over all the involved parties, Multipath TCP's aggregating property is impactful in both directions.

5.2.2 Throughput aggregation scenario D

Here we report the resulting aggregated throughput of Scenario D (Figure 5.9 and 5.10) in both uplink and downlink direction, when the MPTCP scheduler is set to default. In this case two completely different networks are used and aggregated: the mobile one and the fixed one, adding another degree of complexity to the experiment.

In Figure 5.9 we can notice the low available throughput of the LTE connection in uplink which is lower than the one in downlink reported in Figure 5.10: this difference is due to the number of available resource blocks of LTE in the two directions which is larger in DL and smaller in UL. As a consequence, when the default scheduler selects the LTE sub-flow, the performance of the overall connection will degrade in UL. Instead, since in DL the LTE throughput is more comparable to the Ethernet one, it is possible to successfully aggregate them achieving a larger capability.

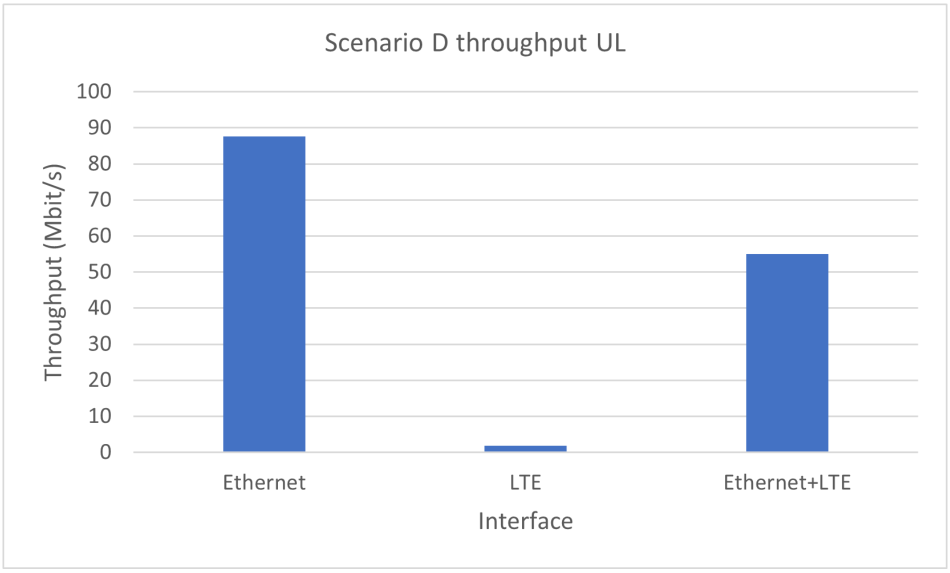


Figure 5.9: Aggregated upload throughput in Scenario D

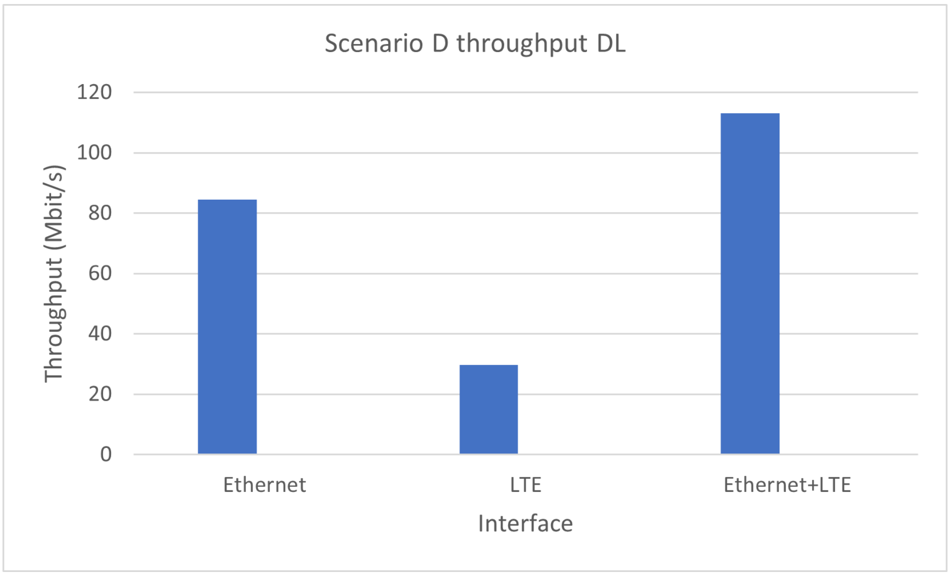


Figure 5.10: Aggregated download throughput in Scenario D

A legitimate question is, given the default MPTCP scheduler selection of the min-RTT path first, why does MPTCP not perform as well as regular TCP? Such path selection only takes place when both paths have room in their congestion windows (cwnd). Suppose the Ethernet cwnd is full and LTE has some space left, then the data will still be transferred over LTE even though its latency is higher.

5.2.3 Throughput aggregation scenario E

Here we report the resulting aggregated throughput of Scenario E (Figure 5.11 and 5.12) in both uplink and downlink direction, when the MPTCP scheduler is set to default.

In this setting we are once again considering both the fixed and mobile networks, but in this case the fixed one is reached through a WiFi access which we expect to be more unstable compared to the Ethernet one in terms of available throughput.

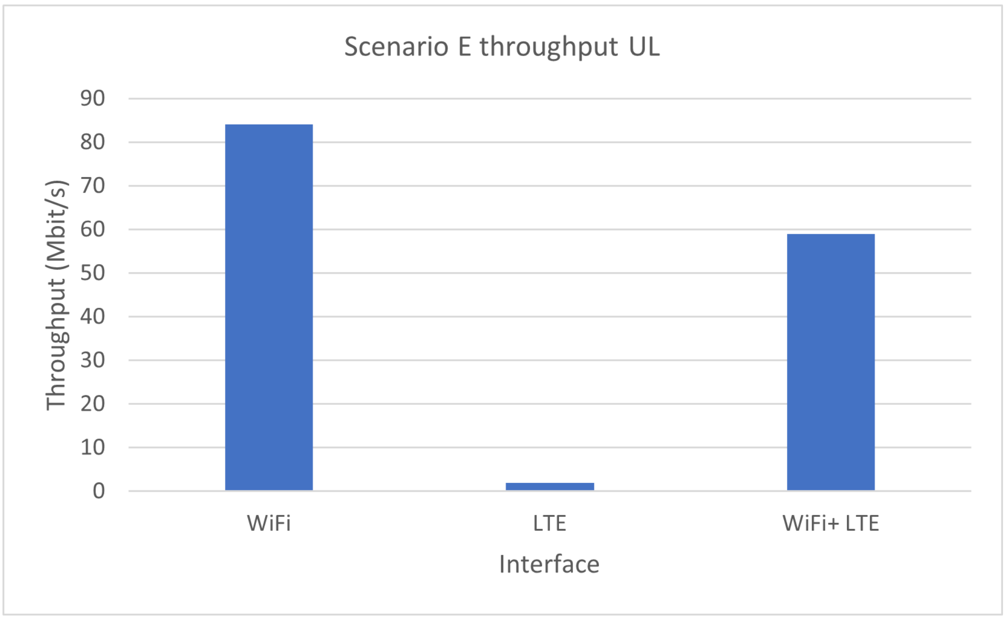


Figure 5.11: Aggregated upload throughput in Scenario E

As expected, the uplink connection is completely compromised by the low available throughput of the LTE access, as reported in Figure 5.11.

Instead, in this case also the downlink connection is degraded by LTE unlike in scenario D, although in a less impactful way: this is due to the low stability of the wireless access which, unlike Ethernet, cannot provide a constant and stable contribution to the overall throughput.

As a consequence, the downlink aggregated throughput is slightly smaller than the WiFi one, as reported in Figure 5.12.

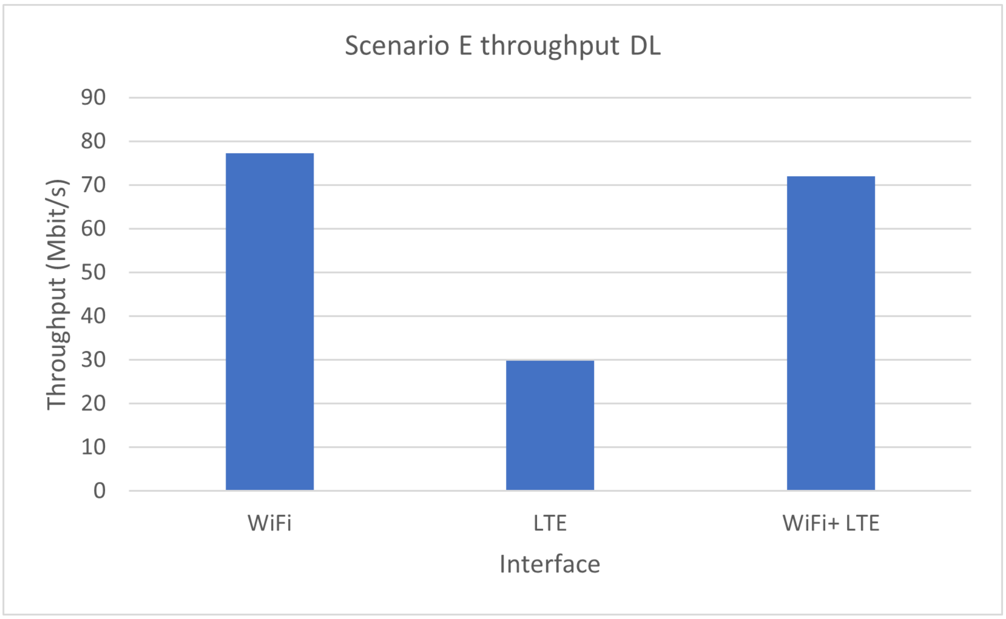


Figure 5.12: Aggregated download throughput in Scenario E

5.3 Path utilization in LAN

Once we have successfully shown the improvement introduced by Multipath TCP in terms of aggregated throughput, we will now focus on the load balancing property of this protocol; in fact, what we expect is for MPTCP to be able to distribute the traffic over the different available paths. The metric that we will keep an eye on is the percentage of data sent out on each interface, also referred to as *path utilization*. Here we investigate the path utilization in a Local Area Network and precisely in Scenario B.

The percentage of data sent out on an interface depends mainly on the scheduler choice: for instance, the default one first selects the min-RTT path and then, once such path is full, it moves on to the next one. In a LAN environment, the behaviour of the latencies measured by the Ethernet and the WiFi interfaces is quite intuitive: the first one has a smaller RTT compared to the latter. This is mainly due to the nature of WiFi, which has a shared channel over air, while Ethernet makes use of a dedicated cable and is much faster in that sense. We thus expect the default scheduler to be preferring the Ethernet connection over the WiFi one, although trying to maintain a fair sharing between the two.

Instead, the Roundrobin scheduler in RTT-agnostic, i.e., it selects a first sub-flow and then it alternates it with the second one, trying to achieve a more fair share between the two sub-flows. We will examine the effect of the RR scheduler in Section 5.5.

In order to examine the distribution of the sent data, we force the interfaces to throttle their available bandwidth: by forcing its value for each interface, we can get a good idea of how the traffic is distributed over the interfaces by the scheduler.

We carried out these experiments through the *WonderShaper* utility first, however we noticed that it has an issue: in DL it forces the interfaces to half the desired bandwidth. As a consequence, we decided to shape the traffic through the Traffic Control Linux command *tc*.

5.3.1 Scenario B

We first report the average latency of the WiFi and Ethernet interfaces in Figure 5.13 where, as expected, the latency of the wireless link is larger than that of the wired one.

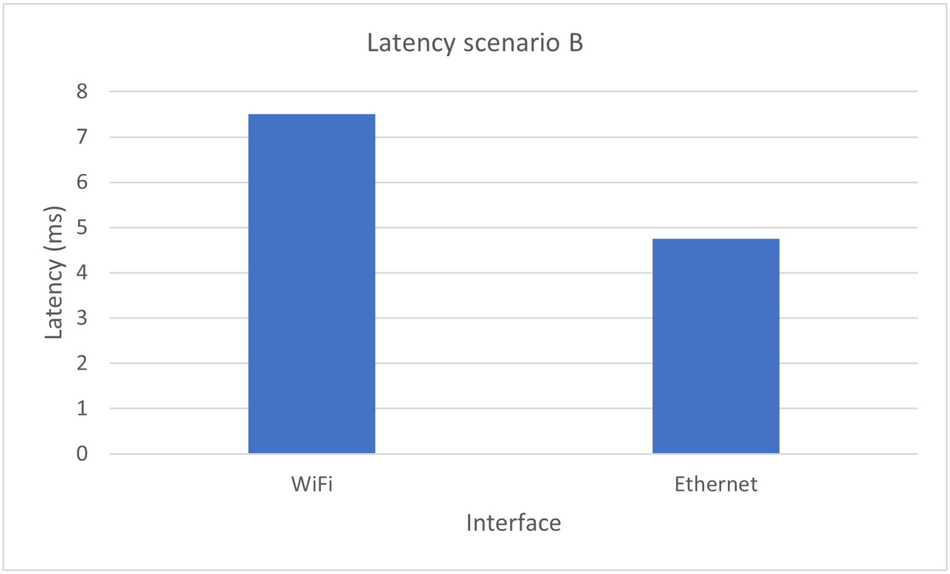


Figure 5.13: Latency in Scenario B

As a consequence, the default scheduler gives priority to the Ethernet interface, as shown in Figure 5.14 for the uplink and Figure 5.15 for the downlink.

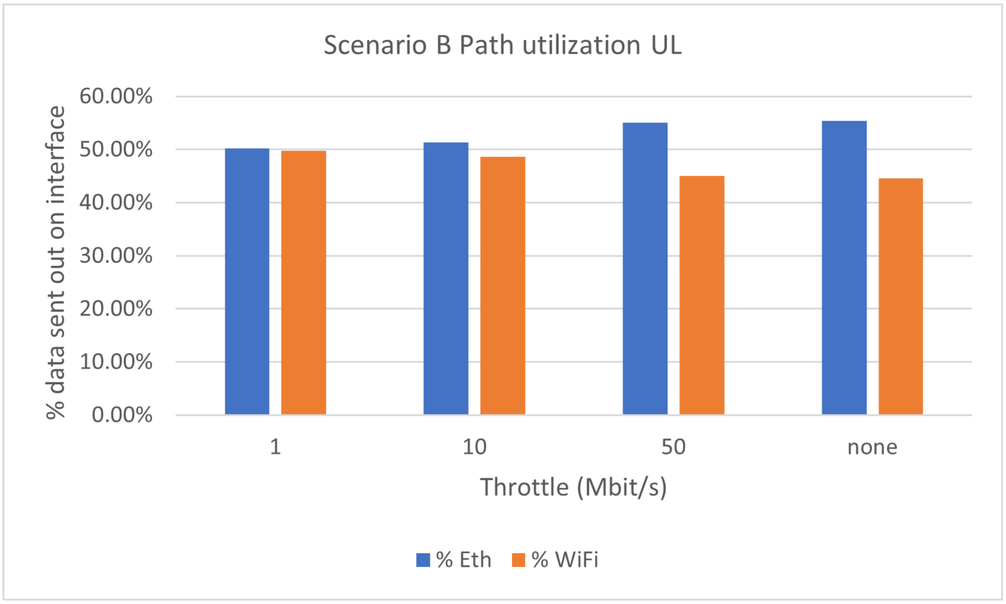


Figure 5.14: Path utilization UL using throttling in Scenario B

As expected, since the latency of the interfaces differs by very little amount, the min-RTT scheduler gives a small priority to the Ethernet sub-flow, achieving however a very balanced usage of the the resources.

The downlink results introduce one of the main sources of uncertainty of multipath TCP, i.e., flows whose duration is small. In those cases, the protocol has barely enough time to initiate the flow aggregation procedure, thus the benefits of having multiple available paths fades, as performance outcome is dictated by the choice of the scheduler which is quite random; in fact, when the throttling is set to be equal to $1Mbit/s$, the path utilization is not really fair in downlink, as reported in Figure 5.15. In other words, the large flows' longer duration lets both paths be established and be able to transfer an amount of data which is non-trivial.

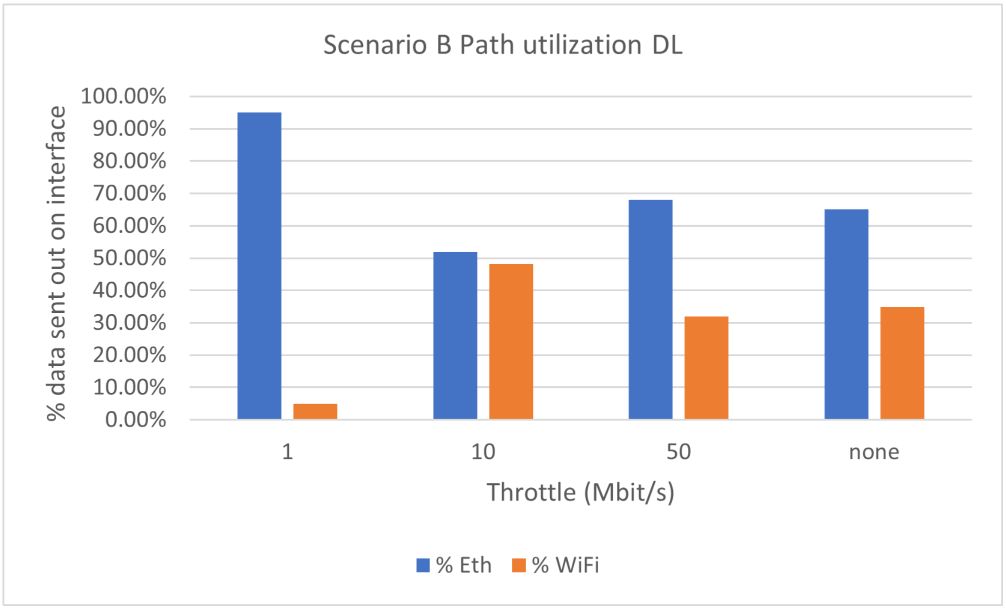


Figure 5.15: Path utilization DL using throttling in Scenario B

5.4 Path utilization in WAN

Moving now into the Wide Area Network tests, we will be able to appreciate more in detail the different scheduler implementations of multipath TCP, thanks to the heterogeneity of the tested scenario.

5.4.1 Scenario C

We first report the average latency of the WiFi and Ethernet interfaces in Figure 5.16 where, as expected, the latency of the two interfaces is comparable. The measured latency is mainly introduced by the network placed beyond the Access Point, which is shared by both paths: as a consequence the latency of the WiFi interface is only slightly larger than that of the Ethernet interface.

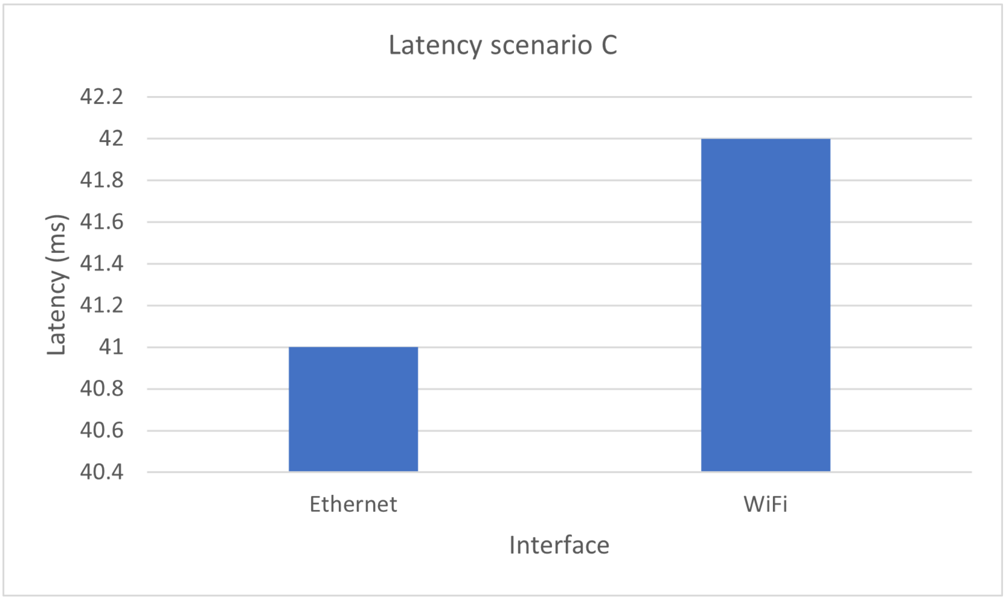


Figure 5.16: Latency in Scenario C

As expected, the default scheduler usually gives priority to the Ethernet interface, as shown in Figure 5.14 for the uplink and Figure 5.15 for the downlink.

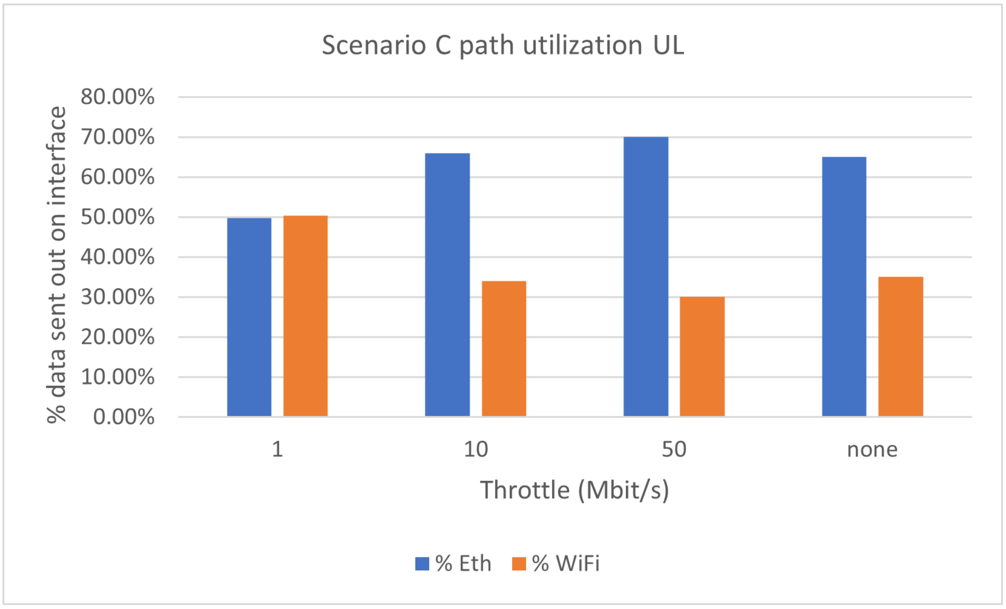


Figure 5.17: Path utilization UL using throttling in Scenario C

The path utilization looks quite balanced for both the uplink and downlink connections, albeit more fair for the latter where the utilization is almost split in a

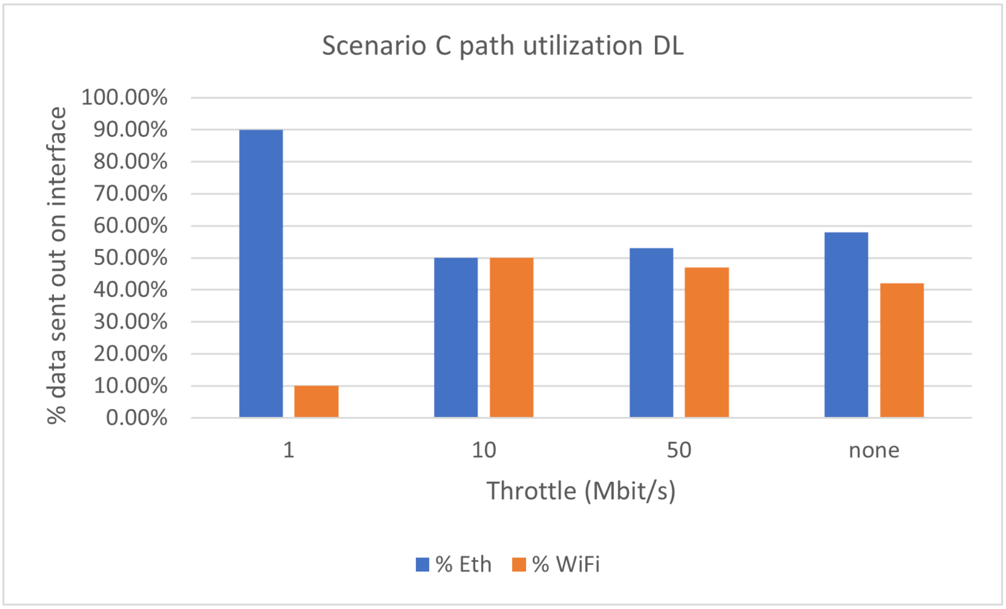


Figure 5.18: Path utilization DL using throttling in Scenario C

50/50 fashion.

Once again in Figure 5.18 we can notice the sub-optimal behaviour of MPTCP for short flow duration, where one path is selected and all the traffic is sent out across such interface.

5.4.2 Scenario D

Let us now consider Scenario D, where both mobile and fixed networks are involved.

We first report the average latency of the Ethernet and LTE interfaces in Figure 5.19 where we can now notice how different the two measures are.

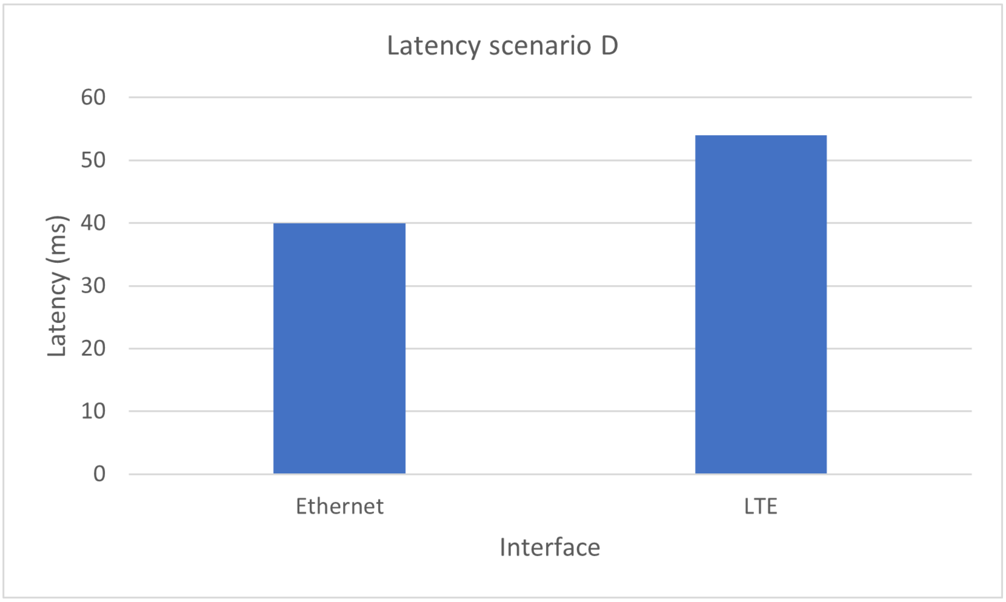


Figure 5.19: Latency in Scenario D

Due to such big difference compared to the previous scenarios, we expect the default MPTCP scheduler to prioritize the Ethernet access more significantly as the traffic shaper effect fades.

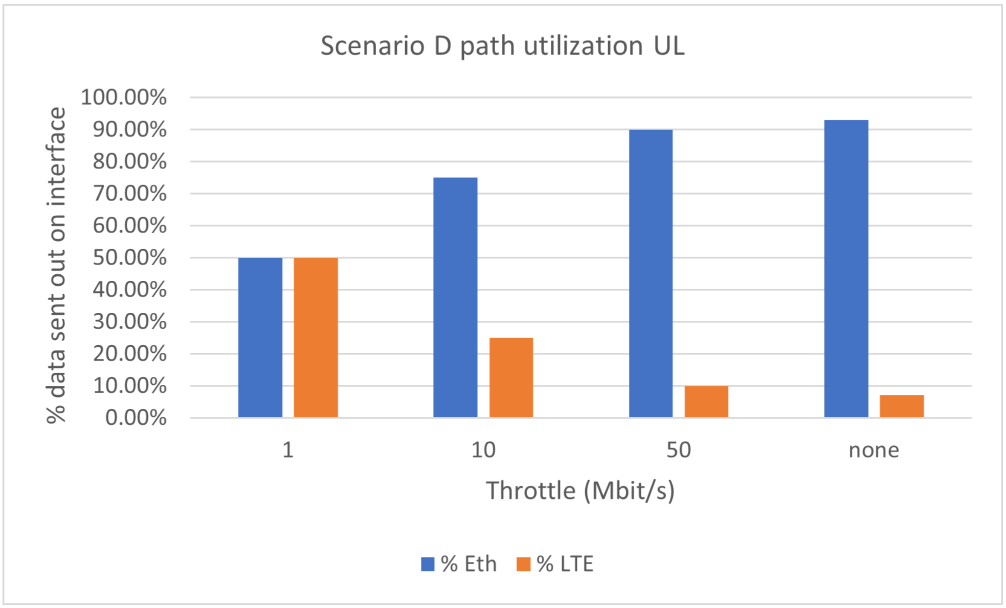


Figure 5.20: Path utilization UL using throttling in Scenario D

This behaviour can be appreciated in Figure 5.20 and Figure 5.21 where we report the uplink and downlink path utilizations, respectively.

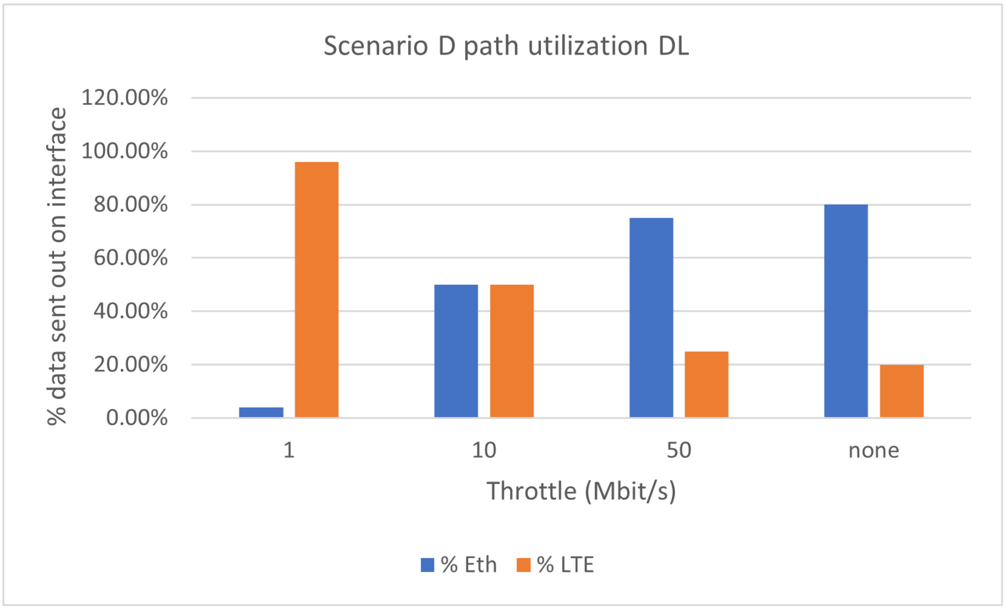


Figure 5.21: Path utilization DL using throttling in Scenario D

Clearly, most traffic is sent out on the Ethernet interface with increasing percentage as the throttle effect decreases.

The default scheduler only seems to prefer the LTE path in DL for $Throttle = 1Mbit/s$: as we have previously explained, for short flows MPTCP behaves inefficiently; in this specific case, the LTE connection had a smaller latency at connection establishment instant, thus it was selected by the scheduler and all the traffic is sent out across such interface before its congestion window is full and the Ethernet connection can be exploited.

5.4.3 Scenario E

Let us now consider Scenario E where both mobile and fixed networks are involved through a WiFi and an LTE accesses.

We first report the average latency of the Ethernet and LTE interfaces in Figure 5.22 where we can now once again appreciate how different the two measured values are.

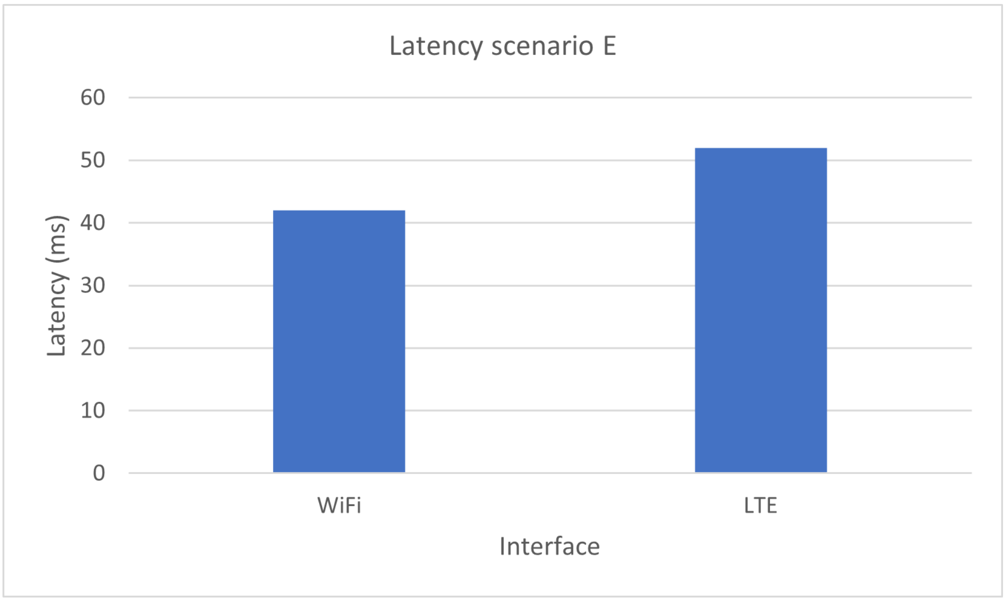


Figure 5.22: Latency in Scenario E

Similarly to the scenario D case, we expect the default MPTCP scheduler to prioritize the WiFi access over the LTE one as the traffic shaper effect fades. This behaviour can be appreciated in Figure 5.23 and Figure 5.24 where we report the uplink and downlink path utilizations, respectively.

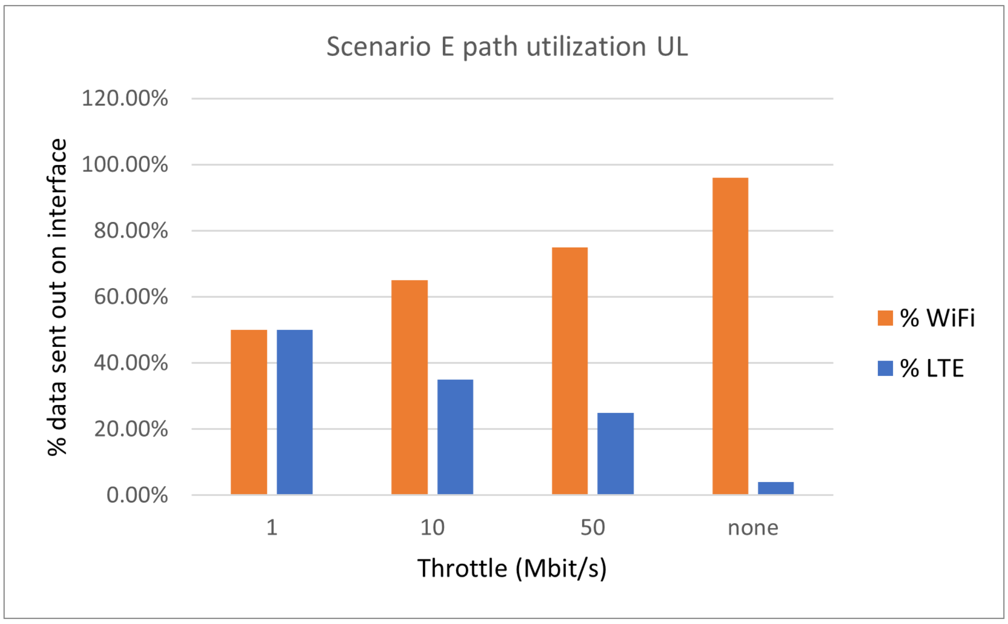


Figure 5.23: Path utilization UL using throttling in Scenario E

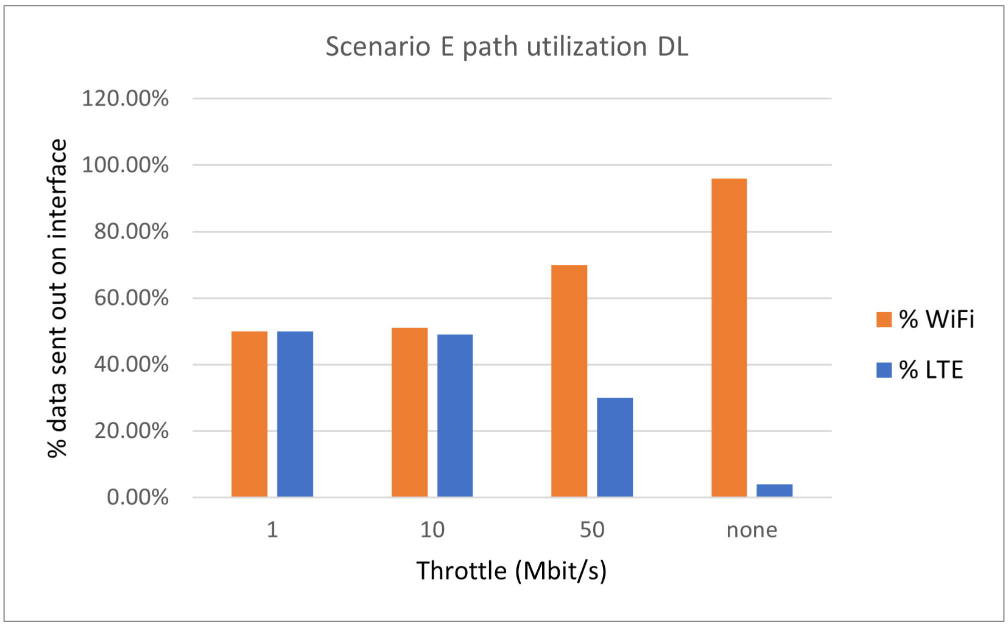


Figure 5.24: Path utilization DL using throttling in Scenario E

We can finally notice that most traffic is sent out on the WiFi interface with increasing percentage as the throttle effect decreases.

5.5 Path utilization Round Robin MPTCP scheduler

In this section we will go through the effect of Round Robin scheduling in some of the considered scenarios. In particular, we will focus on a LAN environment (Scenario B) a WAN environment where only the fixed network is considered (Scenario C) and another one where both fixed and mobile networks are used (Scenario D).

As we have seen in sections 5.3 and 5.4 where we investigated the path utilization in LAN and WAN respectively, the default MPTCP scheduler does not balance the traffic across the available interfaces, as it gives priority to the path with minimum RTT, first and then moves to the second path only when the congestion window of the first path is full. This way the default scheduler gives priority to the "good" flow, where the goodness is measured in RTT, trying to provide a better data transmission quality instead of ensuring fairness between the available paths.

On the other hand, the Round Robin scheduler does not consider the RTT when selecting the path to be used for the transmission; in fact, it simply selects a first sub-flow and then it alternates it continuously with the second one, achieving a more balanced traffic distribution between the two sub-flows.
We thus expect a very fair usage of the interfaces when RR is selected with the negative effect of degraded performances when one of the two sub-flows has large RTT and little available throughput.

In Figure 5.25 and Figure 5.26 we report the comparison of the aggregated throughput of Scenarios B, C and D in uplink and downlink, respectively when Round Robin and default scheduler are selected.

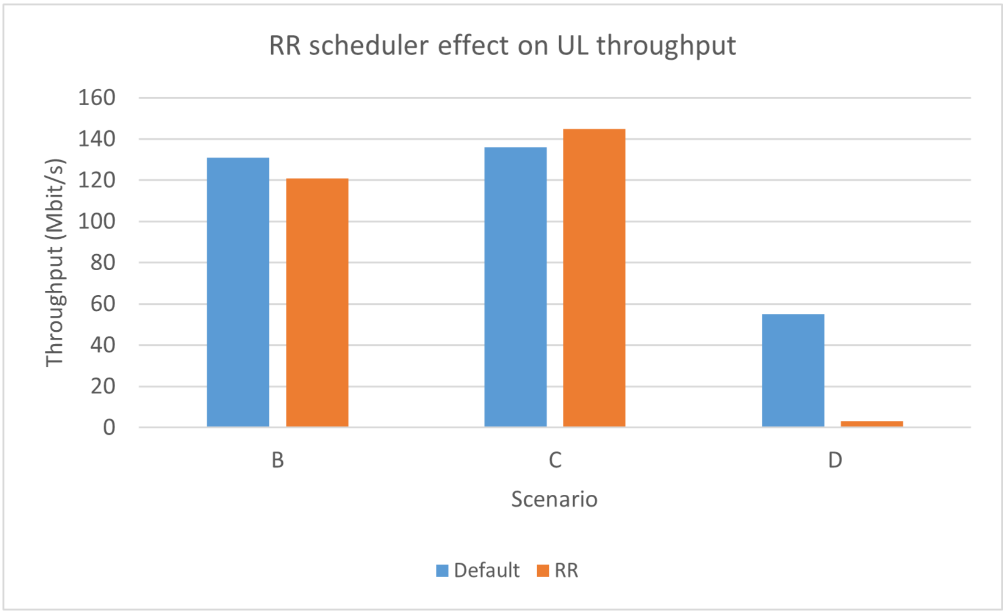


Figure 5.25: Upload throughput for selected scenarios with RR scheduler

What we can notice is that the effect of the MPTCP scheduler choice is not very impactful when the two sub-flows share similar bandwidth availability, i.e., in scenario B and C; in fact, even if the Round Robin scheduler does not prioritize the better path like the default scheduler does, going back and forth between the available interfaces does not really degrade performances. Actually, in some cases it may slightly improve them, like in scenario C in uplink or scenario B in downlink.

In other cases, like in scenario B uplink the overall aggregated throughput is somewhat smaller than the one achieved by the default scheduler, but still very close to such value.
Finally in scenario C downlink the RR scheduler behaves exactly like the default one. This effect is once again due to how similar both the RTT and available bandwidth of both interfaces are.

On the contrary, in scenario D the performances are impacted negatively by the RR scheduler, especially in the uplink connection where the LTE bandwidth is much smaller than the Ethernet one. In DL the aggregated throughput is also quite smaller compared to the one obtained with the previous scheduler choice, but it is less critical than the UL case; in fact the LTE bandwidth in DL is closer to the Ethernet one.

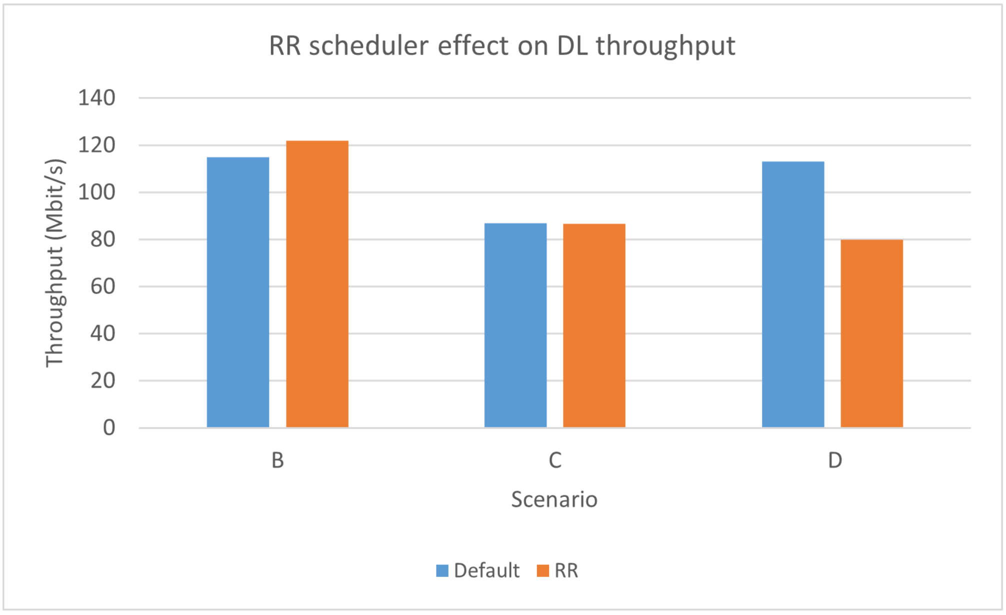


Figure 5.26: Download throughput for selected scenarios with RR scheduler

What happens with Round Robin is that it simply selects the LTE path as often as it selects the Ethernet one and since the LTE bandwidth is both unpredictable and small, for long periods of time the scheduler will transmit over LTE instead of Ethernet, degrading the overall performances.

5.6 FTP application

Once we have examined the aggregation and load balancing properties of MPTCP, in this last section we focus on an application in order to test the gain introduced by this protocol.

Specifically, we will deal with a file transfer experiment obtained through the File Transfer Protocol (FTP), where the gain introduced by Multipath TCP is measured in transfer time decrease. For each of the previously considered scenarios, we compare the transfer time of the aggregated sub-flows to that of the individual flows for different file sizes, ranging from $16kB$ to $50MB$.

5.6.1 Scenario B

We start by considering the simplest scenario, i.e. the LAN one.

In Figure 5.27 and Figure 5.28 we report the upload times for smaller and larger files, respectively, while in Figure 5.29 we report download time.

In the uplink connection the WiFi transfer duration for larger files is much larger than that of the smaller files, making it hard to represent them together; this is why we provide the reader with two separate graphs for small and large file sizes.

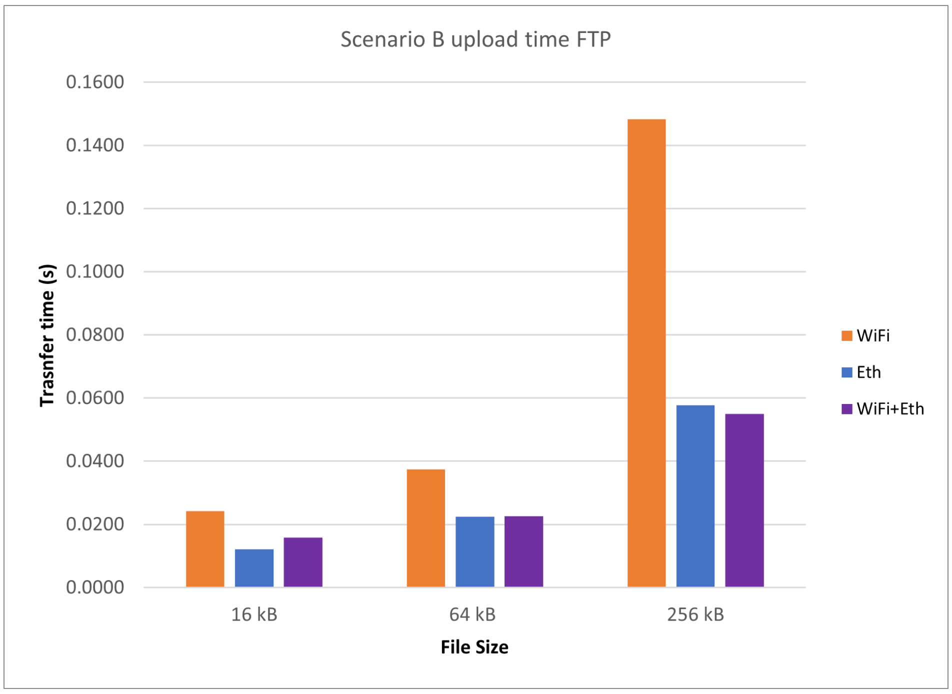


Figure 5.27: FTP upload time Scenario B small files

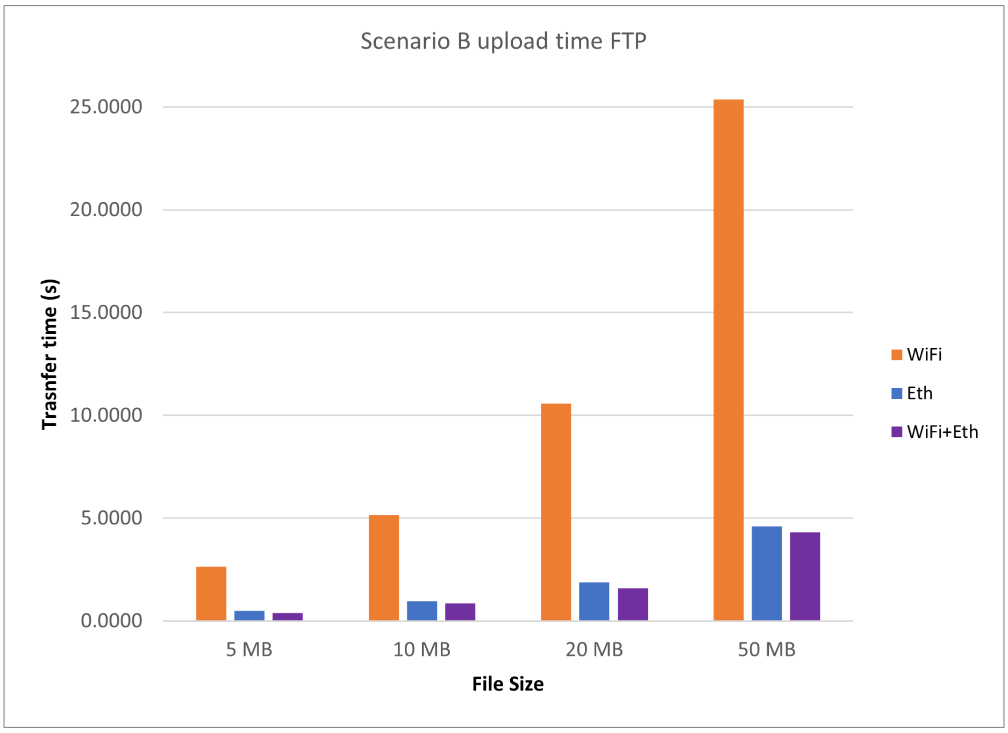


Figure 5.28: FTP upload time Scenario B large files

We can notice one of the main trends of the FTP results: for small transfers (Figure 5.27), Multipath TCP does not provide much improvement; in fact, for such small values, the files download time is largely determined by latency, while the aggregation cannot bring much advantage. Instead, for larger file downloads (Figure 5.28), generally both paths are saturated, thus the aggregated bandwidth is more important than the latency.

We can conclude that in both UL and DL directions Multipath TCP introduces a gain, reducing the overall transfer time, consistently with the results obtained for the throughput measurements. Specifically, for the download of a $50MB$ file, Multipath TCP allows the client to reduce the transfer time by 27% and 31% with respect to Ethernet and WiFi only connections.

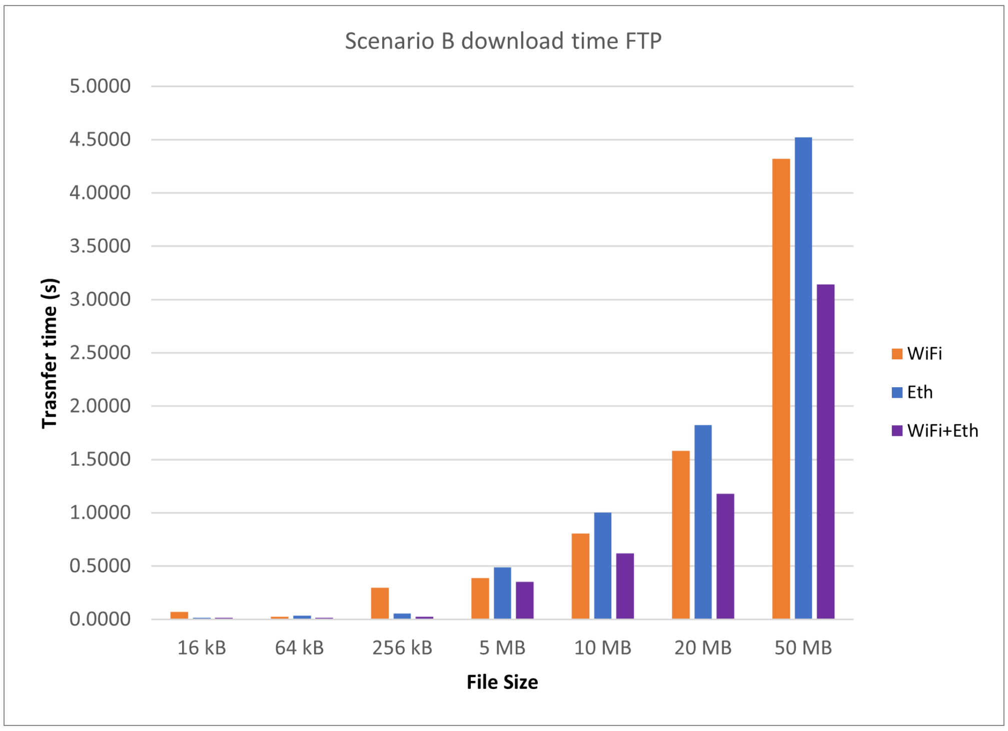


Figure 5.29: FTP download time Scenario B

5.6.2 Scenario C

We now consider the first of the WAN scenarios where only the fixed network in involved through WiFi and Ethernet connections.

In Figure 5.30 and Figure 5.31 we report the upload and download times for different file sizes, respectively.

Similarly to the scenario B case, the file transfer time using MPTCP brings an advantage in both directions, especially for large file sizes where the aggregated bandwidth is more relevant, while for small file sizes no particular gain can be appreciated.

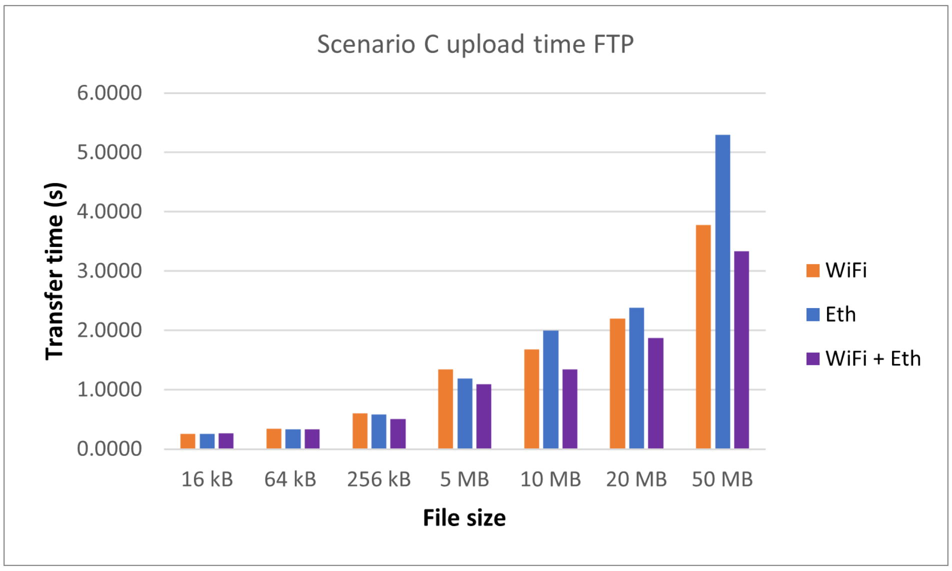


Figure 5.30: FTP upload time Scenario C

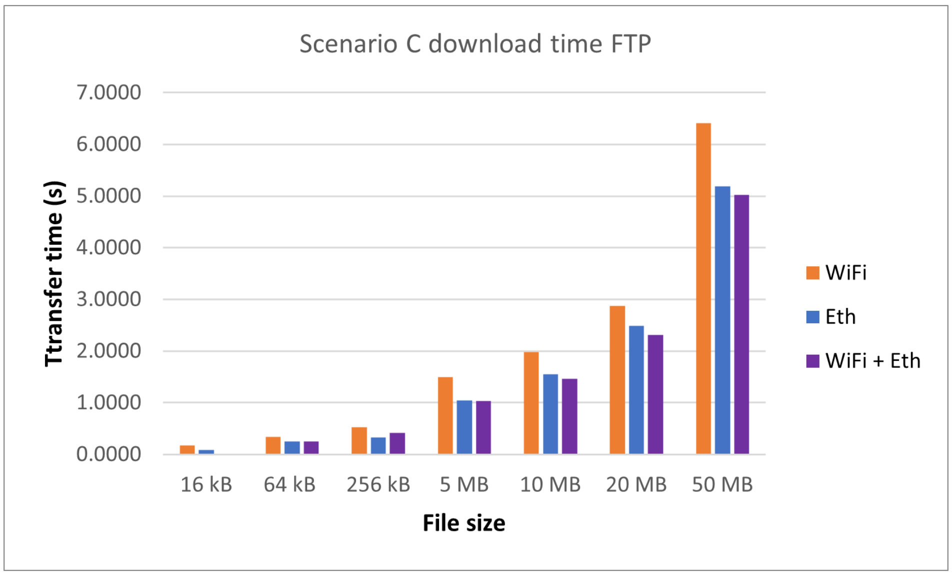


Figure 5.31: FTP download time Scenario C

We can conclude that in a WAN environment where both accesses share similar RTTs and available bandwidth, FTP performs better when aggregating the accesses through Multipath TCP.

5.6.3 Scenario D

We now consider the first of the WAN scenarios where both fixed and mobile networks are involved through Ethernet and LTE connections.

Once again, the transfer time over LTE is much larger than that of Ethernet, thus a graphical representation may not allow the reader to fully appreciate the small files transfer behaviour, which is why we report the upload times for smaller and larger files, respectively in Figure 5.32 and Figure 5.33, and the download times for smaller and larger files, respectively in Figure 5.34 and Figure 5.35.

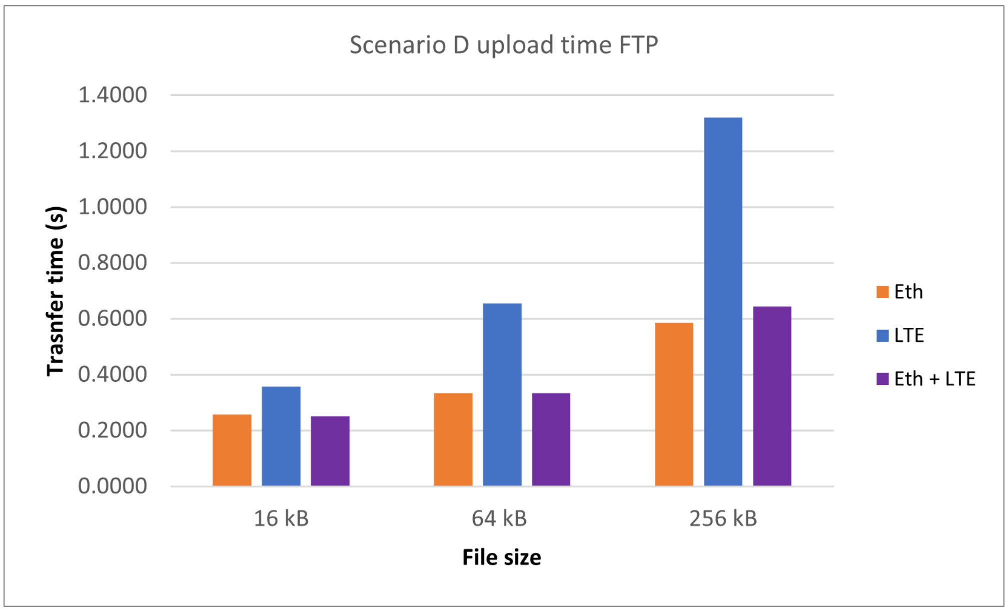


Figure 5.32: FTP upload time Scenario D small files

Going through the previous throughput results of Figure 5.9, we may expect some degrading performances due to the low available bandwidth of LTE especially in UL. This finding is confirmed in this set of results, where we cannot appreciate any gain introduced by MPTCP; instead, in some cases the transfer time of the files is even larger when aggregating.

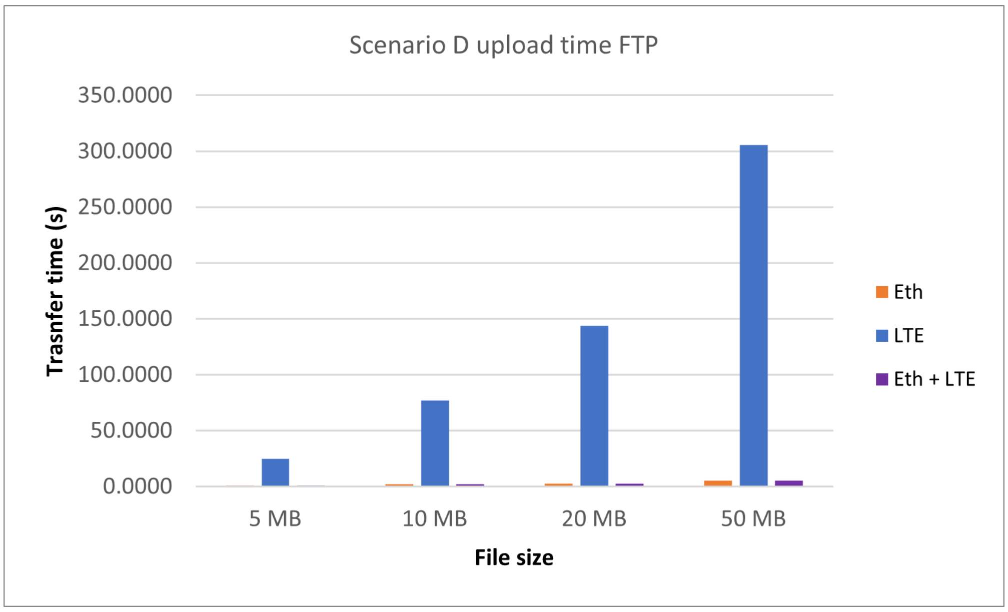


Figure 5.33: FTP upload time Scenario D large files

When looking at the DL throughput results of this scenario reported in Figure 5.10, we can expect some slight improvements, which can be appreciated for instance for the download of a $20MB$ file, where the aggregated transfer takes $5.065s$ while the Ethernet one takes $5.192s$ (Figure 5.35).

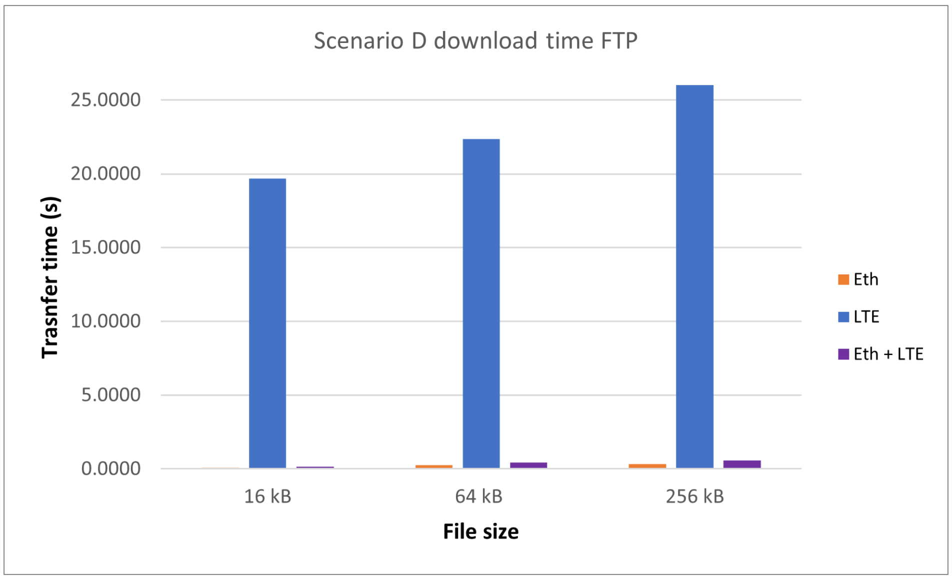


Figure 5.34: FTP download time Scenario D small files

This benefit is present, albeit rather small due to the unpredictability of the LTE throughput and how much it can vary over time.

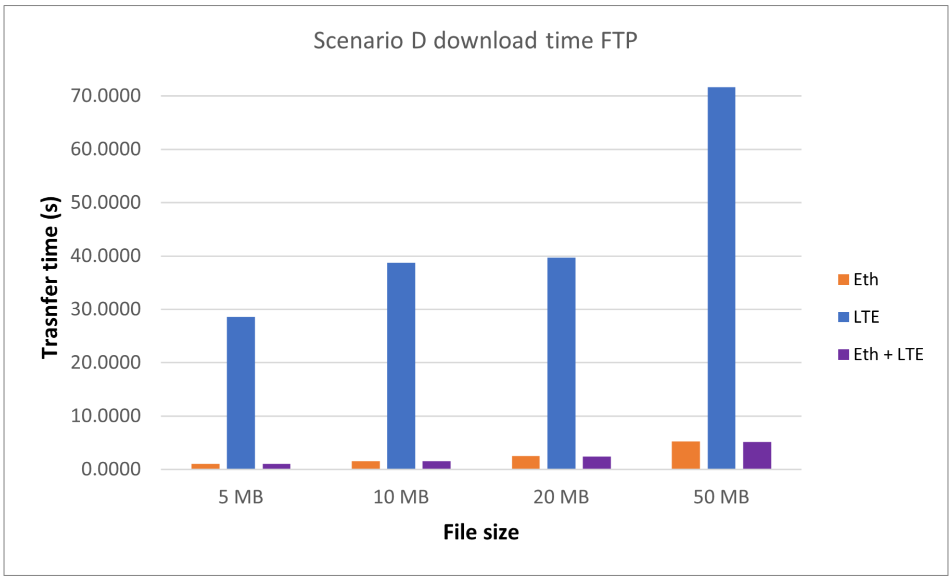


Figure 5.35: FTP download time Scenario D large files

We can conclude that, consistently with the throughput behaviour of scenario D, the lack of a guaranteed bandwidth of the LTE connection influences massively the outcome of the experiment: in UL the LTE path has very little available resource blocks, thus the available bandwidth is much smaller than that of the Ethernet connection, degrading the performances resulting in a slower file transfer; in DL the LTE sub-flow performs better and is able to provide a slight improvement of the transfer time, although rather small.

5.6.4 Scenario E

Finally, we report the results obtained in the last scenario where fixed and mobile networks are reached through WiFi and LTE connections.

Also in this scenario, the transfer time over LTE is much larger than that of WiFi, thus we report the upload times for smaller and larger files, respectively in Figure 5.36 and Figure 5.37, and the download times for smaller and larger files, respectively in Figure 5.38 and Figure 5.39.

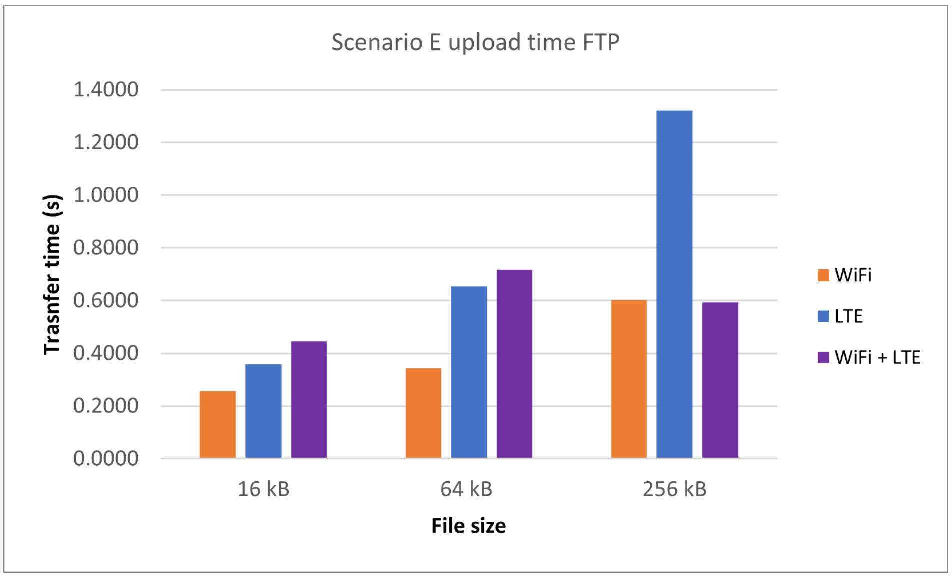


Figure 5.36: FTP upload time Scenario E small files

Basically, this setting is very similar to scenario D, with the added element of the wireless connection, brining a new level of uncertainty to the measure. If we go back to the throughput measurements of Figure 5.11 for UL and Figure 5.12 for DL, we can notice how by aggregating WiFi and LTE accesses, the performances tend to degrade due to the poor performances of LTE and also the channel interference intrinsic in the WiFi connection.

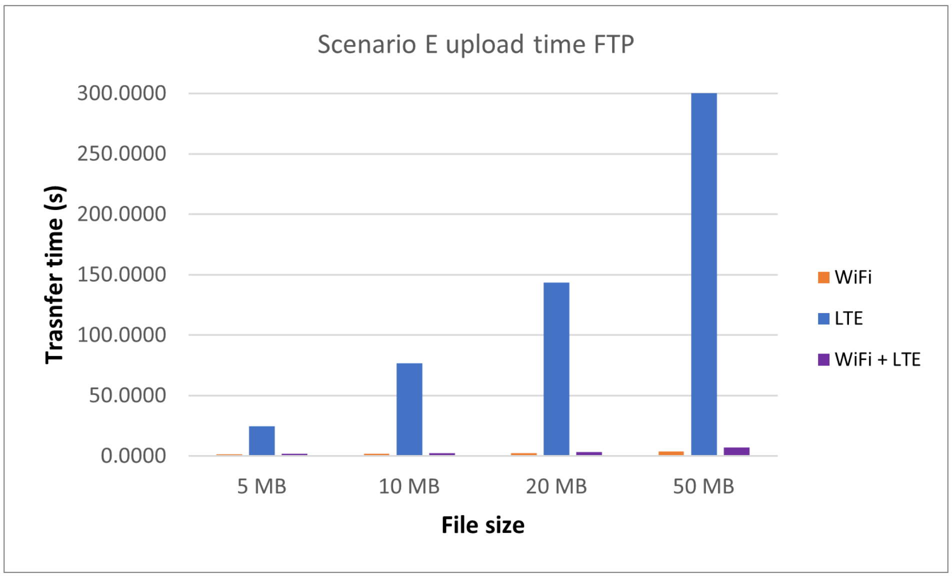


Figure 5.37: FTP upload time Scenario E large files

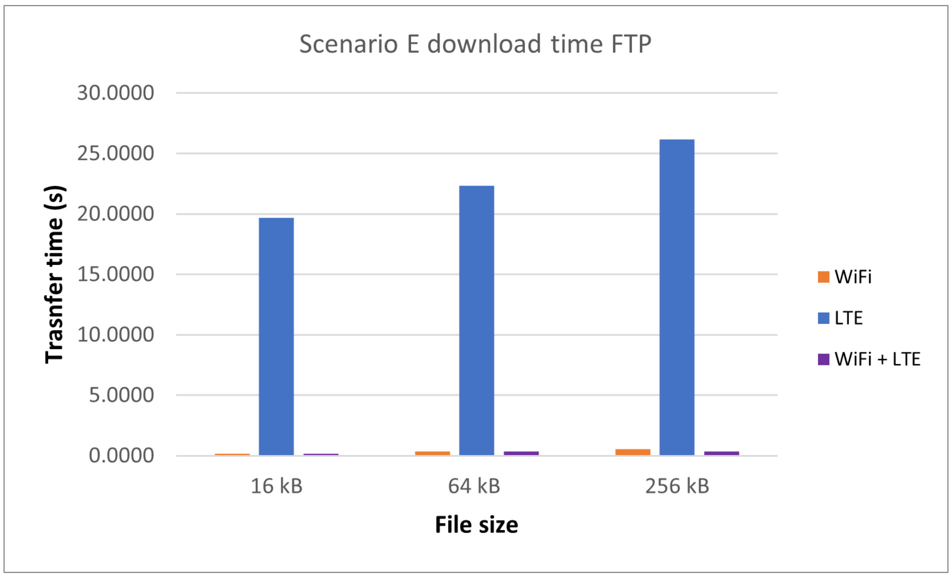


Figure 5.38: FTP download time Scenario E small files

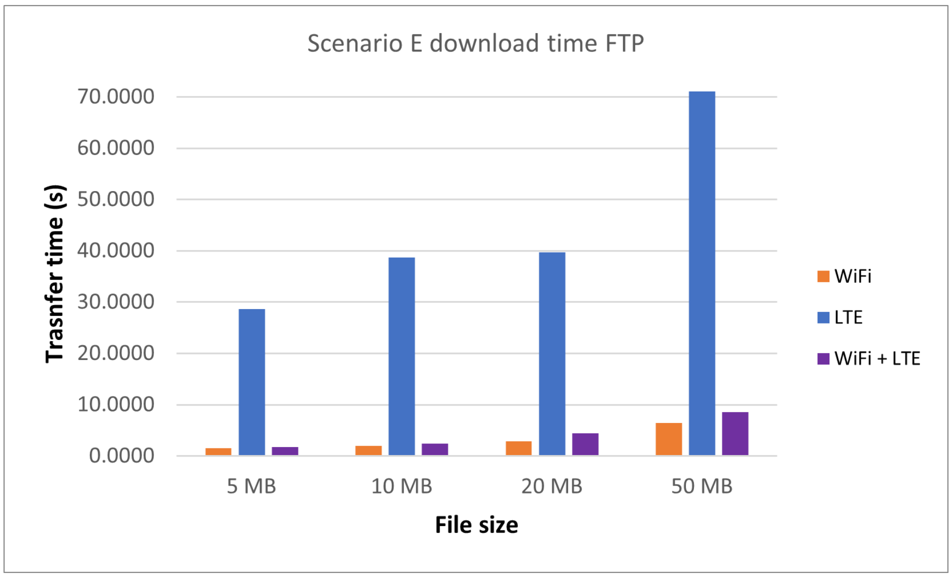


Figure 5.39: FTP download time Scenario E large files

We can conclude that, consistently with the throughput behaviour, the aggregation of LTE and WiFi connections in this setting does not bring benefits in either connection direction. In some cases Multipath TCP behaves very similarly to regular TCP, while in some other cases it degrades the performances due to both the lack of guaranteed bandwidth of LTE and the instability of the wireless one.

Chapter 6

Conclusions

In this final chapter we report the main findings of this thesis, showing the advantages as well as the limitations of the object of our study, Multipath TCP, in the considered setting.

6.1 Synthesis

In this thesis we have created a Multipath TCP testbed to analyse the potential as well as the limitations of this protocol in a real-world environment.

In the first part, we addressed the multipath implementation and its basic functioning, such as the sub-flow connection establishment, explaining its structure and how it works, from the start to the closing of a session, providing an in-depth understanding of how the protocol exploits the available set of paths.

We then decided to examine some key properties of MPTCP, namely the bandwidth aggregation and the load balancing, by developing and testing several scenarios with an increasing level of complexity. First, we focused on a simple Local Area Network where we tested the basic building blocks of our testbed, before moving to a more diverse setting with a WAN accessed using both LTE and home broadband.

Furthermore, we made an extensive evaluation of MPTCP and we verified that it can in fact balance traffic load across the available interfaces and it can offer higher throughput values whenever the two paths have similar RTT and available bandwidth. For instance, MPTCP introduced a big gain when pairing Ethernet and WiFi connections with an aggregated throughput roughly 53% larger than the Ethernet one.

We realized that whenever the LTE mobile network is involved, the aggregation introduces little to no advantage, degrading the performance at times. This is due to the choices made by the default MPTCP scheduling algorithm: once the other connection's congestion window is full, it selects the LTE path even though its latency is higher and it continues to send traffic across the mobile interface until that congestion window is full. Then, since the measured available bandwidth of LTE is quite smaller than the Ethernet/WiFi one and not at all guaranteed over time, the scheduler spends a lot of time in this slow connection, ending up in a reduced measured throughput. This limit of the MPTCP protocol is especially present for the uplink connection, where the available bandwidth of LTE is rather small, while in the downlink connection it can introduce some advantage in terms of aggregated throughput.

Afterwards, the load balancing property of MPTCP was considered, by examining the distribution of the data sent out on each interface. In order to allow a fair usage of the available interfaces, we carried out some tests shaping the traffic and setting the maximum available link capacity, comparing them to the results obtained with no shaping. Multipath TCP successfully managed to use all the available resources, more so when the maximum available bandwidth was equal for both interfaces; when no traffic shaping was applied, MPTCP still used both interfaces quite fairly, giving some priority to the one with the larger available link capacity.

We investigated the behaviour of another scheduler, the Round Robin one, which performed as well as the default one whenever the two connections shared similar RTT and bandwidth, while it performed worse when the mobile network was involved. This is due to the nature of the RR algorithm which balances equally the traffic across the two interfaces, using the LTE one as much as the Ethernet/WiFi one, degrading the overall performances.

In the end we consolidated our analysis by studying a file transfer application through our testbed, which proved that the transfer time can be reduced when using MPTCP, especially for well balanced accesses, coherently with the results obtained for the aggregated throughput.

Summarily, we believe that this testbed shows that an implementation of Multipath TCP over Wind Tre's network could surely improve the load balancing, exploiting all the available resources while providing also a network redundancy, and it could also improve the overall performances of the goal scenario, i.e. two remote offices exchanging data over FTP, given a set of similar available paths between the two. Specifically, we believe that with two accesses with guaranteed bandwidth, the performances could significantly improve in such scenario, providing a faster file transfer experience for the customer.

6.2 Future works

Most of the analysis done so far regarding Multipath TCP allowed the research community to investigate different implementations and topologies where MPTCP can be exploited.

We have seen that one source of sub-optimality in the mobile LTE connection is the large RTT as well as the unstable available bandwidth, thus future work could focus on testing the same scenarios using a 5G connection, instead: this way the latency will be limited and predictable, making it easier for the default scheduler to provide a good traffic distribution.

Another point of interest is the MPTCP scheduler which we have just mentioned: it could be interesting to develop an algorithm that would allow Multipath protocol to schedule the traffic differently when a mobile network is involved, reducing the involvement of the slower connection, for instance moving back to the wired/wireless connection whenever its congestion window has some space left, rather than switching to it only when the LTE cwnd is full, as well.

Appendix A

Extract of Jupyter notebook

In this appendix we report a portion of the VS Code framework used in this thesis to automate the tests, as mentioned in Chapter 3.
Specifically, we show the code cell used to test both Ethernet and WiFi connections of Scenario C:

```
 # Run this cell to perform an MPTCP test using both wired and wireless connection
 activate_wired = subprocess.Popen(['nmcli', 'c', 'up',
          wired_name])
 stdout, stderr = activate_wired.communicate()
 print(str(stdout))
 activate_wireless = subprocess.Popen(['nmcli', 'c', 'up',
          wireless_name])
 stdout, stderr = activate_wireless.communicate()
 print(str(stdout))
 process = subprocess.Popen(['iperf3', '-p', iperf_port ,'-J','-c', ip_lan], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
 stdout, stderr = process.communicate()
 result_wired_wireless = json.loads(stdout)
 throughput_lan[\"Ethernet and Wireless\"] = result_wired_wireless['end']['streams'][0]['receiver']['bits_per_second']/1e6 # Mbit/s]
```

Bibliography

[1] Keith W. Ross James F. Kurose. *Computer networking: a top-down approach.* Pearson College Div, 2012 (cit. on p. ii).

[2] Wikipedia contributors. *Transmission Control Protocol — Wikipedia, The Free Encyclopedia.* [Online; accessed 05-September-2021]. 2021. URL: `https://en.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&oldid=1042823004` (cit. on p. ii).

[3] Alan Ford, Costin Raiciu, Mark J. Handley, and Olivier Bonaventure. *TCP Extensions for Multipath Operation with Multiple Addresses.* RFC 6824. Jan. 2013. DOI: `10.17487/RFC6824`. URL: `https://rfc-editor.org/rfc/rfc6824.txt` (cit. on pp. ii, 3, 4, 6–8).

[4] Wikipedia contributors. *Multihoming — Wikipedia, The Free Encyclopedia.* [Online; accessed 6-September-2021]. 2021 (cit. on pp. 1, 3).

[5] *Wind Tre Group.* `https://www.windtregroup.it/en/Home.aspx`. Accessed: 2021-09-13 (cit. on p. 1).

[6] *Computing MPTCP's initial Data Sequence Number (IDSN).* `http://blog.multipath-tcp.org/blog/html/2014/02/04/idsn.html`. Accessed: 2021-09-07 (cit. on p. 4).

[7] Lima Correia Almeida Barreia. «Multipath TCP Protocols». MA thesis. Lisboa: Tècnico Lisboa, 2014 (cit. on pp. 6, 7).

[8] Veera Venkata Siva Ramakrishna Bonam. «Multipath TCP and Measuring end-to-end TCP Throughput». MA thesis. Faculty of Computing: Blekinge Institute of Technology, 2018 (cit. on p. 8).

[9] Kim Ellegaard Jacobsen. *Testing the performance of Multipath TCP for connections with limited bandwidth.* Tech. rep. Aalborg University, 2016 (cit. on p. 8).

[10] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. «How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP». In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX Association, Apr. 2012, pp. 399–412. ISBN: 978-931971-92-8. URL: `https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/raiciu` (cit. on pp. 9, 10).

[11] Fan Yang, Paul Amer, and Nasif Ekiz. «A Scheduler for Multipath TCP». In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. 2013, pp. 1–7. DOI: `10.1109/ICCCN.2013.6614091` (cit. on p. 9).

[12] Andrea Nota. «Implementation and Evaluation of Multipath TCP (MPTCP) Schedulers For Reliable and Low-latency Car-to-Cloud Communication». MA thesis. Corso di laurea magistrale in Ingegneria Informatica (Computer Engineering): Politecnico di Torino, 2020 (cit. on p. 9).

[13] N. D. Adesh and A. Renuka. «Impact of traffic burst on the behavior of TCP variants for different LTE downlink schedulers». In: *2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)* (2017), pp. 18–23 (cit. on p. 9).

[14] C. Paasch and S. Barre. *Multipath TCP in the Linux Kernel*. [Online]. 2021. URL: `http://www.multipath-tcp.org` (cit. on pp. 10, 16).

[15] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. «Improving Datacenter Performance and Robustness with Multipath TCP». In: *SIGCOMM Comput. Commun. Rev.* 41.4 (Aug. 2011), pp. 266–277. ISSN: 0146-4833. DOI: `10.1145/2043164.2018467`. URL: `https://doi.org/10.1145/2043164.2018467` (cit. on p. 10).

[16] Ashkan Nikravesh, Y. Guo, Feng Qian, Z. Morley Mao, and S. Sen. «An in-depth understanding of multipath TCP on mobile devices: measurement and system design». In: *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking* (2016) (cit. on p. 10).

[17] Randall R. Stewart. *Stream Control Transmission Protocol*. RFC 4960. Sept. 2007. DOI: `10.17487/RFC4960`. URL: `https://rfc-editor.org/rfc/rfc4960.txt` (cit. on p. 10).

[18] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. «Design, Implementation and Evaluation of Congestion Control for Multipath TCP». In: *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. Boston, MA: USENIX Association, Mar. 2011. URL: `https://www.usenix.org/conference/nsdi11/design-implementation-and-evaluation-congestion-control-multipath-tcp` (cit. on p. 11).

[19] Ming Li, Andrey Lukyanenko, Sasu Tarkoma, Yong Cui, and Antti Ylä-Jääski. «Tolerating Path Heterogeneity in Multipath TCP with Bounded Receive Buffers». In: *SIGMETRICS Perform. Eval. Rev.* 41.1 (June 2013), pp. 375–376. ISSN: 0163-5999. DOI: `10.1145/2494232.2465750`. URL: `https://doi.org/10.1145/2494232.2465750` (cit. on p. 11).

[20] *Wireshark.* `https://www.wireshark.org/`. Accessed: 2021-09-09 (cit. on p. 17).

[21] Wikipedia contributors. *Iperf — Wikipedia, The Free Encyclopedia.* [Online; accessed 11-September-2021]. 2021 (cit. on p. 17).

[22] *PING Command in Linux with examples.* `https://www.geeksforgeeks.org/ping-command-in-linux-with-examples/`. Accessed: 2021-09-09 (cit. on p. 17).

[23] *sockperf (3) - Linux Man Pages.* `https://www.systutorials.com/docs/linux/man/3-sockperf/`. Accessed: 2021-09-09 (cit. on p. 17).