

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Connectivity Gateway for Heterogeneous Environments

Raquel Maria Pereira Lopes



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Ana Cristina Costa Aguiar

October 29, 2024

Resumo

A quinta geração de comunicações móveis (5G) promete revolucionar a sociedade com taxas de dados melhoradas, latência reduzida e conectividade ampliada. Inovações como a multi-conectividade e o Access Traffic Steering, Switching, and Splitting (ATSSS) são essenciais para alcançar estas capacidades, permitindo que dispositivos utilizem simultaneamente várias tecnologias de acesso rádio (RATs). Apesar do potencial dos protocolos de transporte multipath na otimização da taxa de transferência e resiliência da rede, o seu desempenho em ambientes reais de alta mobilidade permanece pouco explorado.

Esta tese tem como objetivo avaliar o desempenho do Protocolo de Controlo de Transmissão Multipath (MPTCP) em redes operacionais 5G e WiFi em condições dinâmicas que envolvem mobilidade. Realizada como parte do projeto Route 25, apoiado pelo Instituto de Telecomunicações (IT), a investigação envolve o desenvolvimento e teste de um gateway de conectividade multi-tecnológica que integra redes 5G e WiFi para fornecer um serviço contínuo durante as mudanças de ligação. Focando-se em métricas-chave de desempenho, como taxa de transferência, fiabilidade e latência, o estudo compara o MPTCP com o previamente estudado Protocolo de Agregação Multipath em Tempo Real (MA-RTP), analisando o impacto de diferentes algoritmos de escalonamento no desempenho em ambientes dinâmicos.

Os resultados fornecem insights sobre o desempenho prático dos protocolos de transporte multipath em cenários de alta mobilidade, abordando uma lacuna significativa na literatura. As conclusões oferecem recomendações práticas para a implementação de protocolos de transporte multipath em redes 5G e WiFi, contribuindo para o desenvolvimento de soluções de transporte mais eficientes e escaláveis. Este trabalho aprofunda a compreensão das forças e fraquezas relativas destes protocolos, auxiliando na otimização do desempenho da rede para aplicações em veículos conectados, cidades inteligentes e muito mais.

Abstract

The fifth generation of mobile communications (5G) promises to revolutionize society with enhanced data rates, reduced latency, and expanded connectivity. Innovations like multi-connectivity and Access Traffic Steering, Switching, and Splitting (ATSSS) are integral to achieving these capabilities, enabling devices to simultaneously utilize multiple radio access technologies (RATs). Despite the potential of multipath transport protocols in optimizing throughput and network resilience, their performance in real-world, high-mobility environments remains underexplored.

This thesis aims to evaluate the performance of the Multipath Transmission Control Protocol (MPTCP) in operational 5G and WiFi networks under dynamic conditions involving mobility. Conducted as part of the Route 25 project supported by Instituto de Telecomunicações (IT), the research involves the development and testing of a multi-technology connectivity gateway that integrates 5G and WiFi networks to provide seamless service during handovers. By focusing on key performance metrics such as throughput, reliability, and latency, the study compares MPTCP with the previously studied Multipath Aggregated Real-Time Protocol (MA-RTP), analyzing the impact of different scheduling algorithms on performance in dynamic environments.

The results provide insights into the practical performance of multipath transport protocols in mobility scenarios, addressing a significant gap in the literature. The findings offer practical recommendations for deploying multipath transport protocols in 5G and WiFi networks, contributing to the development of more efficient and scalable transport solutions. This work enhances the understanding of the relative strengths and weaknesses of these protocols, aiding in optimizing network performance for applications in connected vehicles, smart cities, and beyond.

Acknowledgements

I would like to express my deepest gratitude to my mother, Cláudia Pereira, whose unwavering support and tireless efforts have been my greatest inspiration. Without her guidance and sacrifices, achieving this goal would not have been possible.

I also wish to express my appreciation to my younger sister, Susana Lopes. She has always been a role model and an inspiration to me. During the most challenging times in my life, she motivated me to persevere and gave me the strength to strive for excellence. I could always count on her to listen and offer advice.

I am deeply grateful to my boyfriend, Filipe Peixoto, whose support and companionship were indispensable throughout this journey. His presence, his willingness to listen and his advice all contributed to the completion of this work. More than a companion, he is an exceptional friend and an inspiration of resilience and ambition.

I would like to thank my father, Marcos Lopes, for his advice and for being a supportive presence, even when we were apart.

To the rest of my family and friends, thank you for always being there to lift my spirits when I needed it most.

I sincerely appreciate Professor Ana Aguiar for her constant guidance, excellent advice, and encouraging meetings. Her support has been crucial in the completion of this work.

Raquel Maria Pereira Lopes

This work was supported by the European Union / Next Generation EU through the *Programa de Recuperação e Resiliência* (Recovery and Resilience Plan) [Project Route 25, Number C645463824-00000063], and by the *Fundação para a Ciência e a Tecnologia* (Foundation for Science and Technology), Public Institute, under project reference UIDB/50008/2020 (DOI: <https://doi.org/10.54499/UIDB/50008/2020>).

Sustainable Development Goals (SDGs)

This dissertation is part of the Route 25 initiative, specifically focusing on Work Package 2 (WP2): Digital Experience and Adaptive Connectivity for Cooperative Mobility. The aim of this initiative is to enhance safety for vulnerable road users (VRUs) through the development of Cooperative Perception systems. These systems facilitate real-time data exchange between vehicles, sensors, and mobile devices, supporting more accurate environmental mapping and object detection. A key aspect of this research is optimizing the transmission of high-volume video data streams within 5G and WiFi networks, a crucial element in improving mobility services and road safety.

The work directly aligns with SDG 9: Industry, Innovation, and Infrastructure, SDG 11: Sustainable Cities and Communities, and SDG 4: Quality Education. A detailed breakdown of the contributions to these goals can be found in Table 1.

Table 1: Contribution to the Sustainable Development Goals (SDGs)

SDG	Target	Contribution	Performance indicators and metrics
4	4.3 Quality technical education	Contributes to the advancement of knowledge in communication networks, supporting education and technical skills development in engineering.	- Increased access to technical resources for education - Enhanced knowledge in communication technologies
9	9.c Expand access to ICT	Optimizes data transmission across multiple network paths (5G, WiFi), enhancing digital infrastructure and expanding access to reliable communication services.	- Increased throughput - Reduced latency - Lower packet loss
11	11.2 Sustainable transport systems	Supports the deployment of reliable communication networks in connected vehicles and smart cities, facilitating continuous connectivity and improving transport systems.	- Reliable data transfer rates in high-mobility scenarios - Seamless network handovers

“A problem is a chance for you to do your best”

Duke Ellington

Be kind to yourself, allow room for failure, but never for thoughts of giving up.

Raquel Maria Pereira Lopes

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives and Problem Definition	2
1.4	Dissertation Structure	3
2	Background and State-of-the-Art	4
2.1	Introduction to Multipath Transport	4
2.1.1	Key Advantages	5
2.1.2	Implementation Challenges	6
2.2	Fundamentals of Transmission Control Protocol (TCP)	7
2.2.1	TCP Congestion Control	8
2.2.2	Impact of Round-Trip Time (RTT) on TCP Performance	9
2.2.3	Effects of Packet Loss on TCP Efficiency in Wireless Networks	10
2.3	Multipath Transmission Control Protocol (MPTCP)	11
2.3.1	MPTCP Operation	12
2.3.2	Path Management	18
2.3.3	Scheduling Algorithms	19
2.3.4	Congestion Control Mechanisms	21
2.3.5	Reliable Data Transfer	23
2.4	Multipath Adaptive Real-time Transport Protocol (MA-RTP) compared to MPTCP	24
2.4.1	Overview of Real-time Transport Protocol (RTP) and MPRTCP	24
2.4.2	Introduction to MA-RTP	25
2.5	OpenMPTCProuter (OMR)	26
2.5.1	Main Software Tools	26
2.5.2	Key Features	28
2.6	State-of-the-Art	29
3	Design and Implementation of an MPTCP Evaluation System	32
3.1	Problem Definition and Methodology	32
3.1.1	Context and Use Case	32
3.1.2	Previous work	34
3.1.3	Benchmark Metrics	34
3.2	Experimental Setup	35
3.2.1	Architecture	35
3.2.2	Wi-Fi Path	37
3.2.3	5G Path	39
3.2.4	Specification of Components	39

3.3	Setup Components Configuration	41
3.3.1	Multipath Tunnel: VPS and Router	41
3.3.2	Wi-Fi Subsystem	43
3.4	Measurement Subsystem	45
3.4.1	Data Collection	45
3.4.2	Time Synchronization	47
3.4.3	Global Positioning System (GPS)	51
3.4.4	Data Processing Methodology	53
4	Experiments, Results and Discussion	56
4.1	Experiment Overview	56
4.1.1	Experimental Route	57
4.1.2	Planned Testing Scenarios	58
4.1.3	Experimental Setup and Location Constraints	59
4.1.4	Blueprint	60
4.2	Differences in Evaluation Conditions for MPTCP and MA-RTP	60
4.2.1	Traffic Generator: Video Sender Constraint	61
4.2.2	Adjustments to the Experimental Setup	64
4.3	Comparative Analysis of TCP Performance Over Wi-Fi and 5G Networks	66
4.3.1	Throughput	67
4.3.2	Latency	68
4.3.3	Packet Loss	68
4.4	Evaluation of MPTCP Performance Metrics with Different Schedulers	69
4.4.1	Round-Robin	70
4.4.2	Redundant	72
4.4.3	ECF	75
4.4.4	BLEST	77
4.5	MPTCP Schedulers Comparison	80
4.6	Comparative Analysis of MPTCP and MA-RTP	81
4.6.1	Round-Robin	81
4.6.2	Redundant	83
4.6.3	ECF and Context-based Average Throughput	85
4.6.4	BLEST and Timely Arrival	85
4.7	Take-Aways	86
5	Final Remarks	88
5.1	Conclusion	88
5.2	Contributions	89
5.3	Future Work	89
A	Commands for Setup Configuration	94
A.1	OpenMPTCProuter Configuration	94
A.1.1	Virtual Private Server (VPS) Installation and keys extraction	94
A.2	Wi-Fi Path Configuration	94
A.2.1	Setting Up the 5G Hat: Configuring the SIM820X on the Raspberry Pi	94
A.2.2	Setting Up the AP subsystem	97
A.3	GPS Configuration	99
A.4	Time Synchronization	100
A.4.1	NTP Configuration on Raspberry Pi	100

A.4.2	NTP Configuration on Google Cloud VM	101
B	Codes	103
B.1	GPS Data Logging Script	103
B.2	Throughput	104
B.3	Latency	107
B.4	Packet Loss	109
B.5	Packet Delay Calculation	112
B.6	Packet Transmission	113

List of Figures

2.1	Single-path and multi-path transport protocol stack architectures. Adapted from source: [1].	4
2.2	TCP header, that has a constant size of 160 bits, or 20 bytes. Adapted from source: [2].	7
2.3	Effect of a short disconnection in TCP transmission. Source: [3].	11
2.4	Comparison of Standard TCP and MPTCP Protocol Stacks. Source: [4].	12
2.5	Main MPTCP operations to maintain a connection. Adapted from source: [5]. . .	14
2.6	Start of a new connection, initial subflow setup, and the process to add a new subflow to an existing MPTCP connection.	15
2.7	Adding a new IP from Host A, interface 2 and removing the same IP. Source: [4].	16
2.8	Example of mapping - DSN can be mapped with multiple subflows, such as in cases where retransmission is needed. Source: [6].	16
2.9	The encapsulation of the MPTCP DSS within the TCP option field. Source: [7]. .	17
2.10	Example of MPTCP Connection: Full mesh paths. Adapted from source: [1] . .	18
2.11	Applications, schedulers, and their overall goals. Source: [8].	19
2.12	Default MPTCP scheduler and examples of its blinder scheduling decisions in scenarios where two paths have heterogeneous characteristics. (a) Lowest Latency—LL (default MPTCP scheduler). (b) r_{best} with less delay, but less capacity. (c) r_{best} with less delay, but higher loss. Source: [8].	20
2.13	TCP and MPTCP CC examples. Source: [9].	22
2.14	General views of the MPTCP Linux kernel implementation regarding a multi-homed end system equipped with two network interface cards (NICs). (a) TCP/IP protocol stack. (b) MPTCP architecture. (c) Modular framework of packet scheduling. Source: [8].	23
2.15	Standard RTP and MP-RTP stack. Source: [10].	24
2.16	Stack of MA-RTP. Source: [10].	25
2.17	Illustration of the OMR operating mechanism. Representation of two points of the multipath tunnel, the router and the VPS. Two paths are used. The Internet connection is achieved after reaching the VPS. Source: [11].	26
2.18	OMR software architecture.	27
3.1	Use Case Scenario. This work is responsible for implementing multipath transport (highlighted in yellow) using Wi-Fi and 5G for data transmission.	33
3.2	Software components used for the MPTCP evaluation experimental setup.	35
3.3	Hardware components used for the MPTCP evaluation experimental setup.	36
3.4	Integration of GPS antenna into the experimental setup and representation of data files collected during experiments.	37
3.5	Photograph of mobile components on left and fixed components of setup on right.	38

3.6	5G router and its constituent parts. Adapted from source: [12].	39
3.7	Keys generated by the script, required for router configuration.	42
3.8	General system status when everything is functioning correctly. The Air-de-Raquel is the desktop used, OMR is the RPi that is running the multipath gateway, the Wi-fi is the wifi path, USB is the 5G path, and the instance is the VPS running on the Google Cloud VM.	43
3.9	Synchronization Subnet in an NTP Configuration. Arrows represent the flow of synchronization control, with numbers indicating different strata levels. Source: [13].	48
3.10	Location output of the Google Cloud VM using the <i>curl ipinfo.io</i> command, showing that the VM is in London, England. This information helps in selecting an optimal NTP server for time synchronization based on proximity.	49
3.11	The multipath gateway's network location, identified through the <i>curl ipinfo.io</i> command, indicates it operates from London, England, despite its physical presence in Porto.	50
3.12	Ultimate GPS Breakout with the adapter cable. Source: [14].	51
3.13	The output of the <i>cgps -s</i> command when the GPS module is receiving a satellite signal, with coordinate data presented on the left and satellite information on the right.	52
4.1	Experimental Route. Adapted from source: [10].	57
4.2	Throughput performance over time for the packet transmission code, to simulate the video sender code characteristics, used in MPTCP evaluation under a Loopback interface, averaging 1.52 Mbits/s. Total Time Taken: 390.22 seconds.	63
4.3	Throughput performance over time for the video sender code used in MA-RTP evaluation under a Loopback interface, averaging 1.66 Mbits/s. Total Time Taken: 343.700418 seconds.	63
4.4	Hardware experimental setup from MA-RTP evaluation. Source: [10].	64
4.5	Hardware experimental setup used for the MPTCP evaluation experimental setup.	64
4.6	Schematic representation of the TCP packet transmission experiment using Wi-Fi network.	66
4.7	Schematic representation of the TCP packet transmission experiment using 5G network.	66
4.8	TCP over Wi-Fi and TCP over 5G connection. Throughput between computer and AWS VM. Average Throughput for Wi-Fi: 1.56 Mbits/s. Total Transmission Time for Wi-Fi: 399.89 s. Average Throughput for 5G: 1.55 Mbits/s. Total Transmission Time for 5G: 402.03 s.	67
4.9	TCP over Wi-Fi and TCP over 5G connection. Latency between computer and AWS VM. Average Latency for Wi-Fi Path: 84.36 ms. Average Latency for 5G Path: 109.80 ms.	68
4.10	TCP over Wi-Fi and TCP over 5G connection. Packet loss percentages based on the last 10 packets between computer and AWS VM. Lost packets in Wi-Fi: 6. Lost packets in 5G: 15.	69
4.11	MPTCP connection throughput using RR scheduler. Average throughput for Wi-Fi Path: 0.81 Mbits/s. Average throughput for 5G Path: 0.82 Mbits/s. Combined Average Throughput: 1.62 Mbits/s. Total Transmission Time: 402.34s.	70
4.12	MPTCP connection latency using RR scheduler. Average latency for Wi-Fi interface: 41.48 ms. Average latency for 5G interface: 42.74 ms.	71

4.13	MPTCP connection packet loss using RR scheduler. Total lost packets: 46 (43 packets lost in Wi-Fi and 3 in 5G).	72
4.14	MPTCP connection throughput using Redundant scheduler. Average throughput for Wi-Fi Path: 1.63 Mbits/s. Average throughput for 5G Path: 1.63 Mbits/s. Combined Average Throughput: 3.25 Mbits/s.	73
4.15	MPTCP connection latency using Redundant scheduler. Average latency for Wi-Fi interface: 47.95 ms. Average latency for 5G interface: 49.80 ms.	74
4.16	MPTCP connection packet loss using Redundant scheduler. Total lost packets for Wi-Fi.pcap: 140. Total lost packets for 5G: 2.	74
4.17	MPTCP connection throughput using ECF scheduler. Average Throughput for Wi-Fi Path: 1.58 Mbits/s. Average Throughput for 5G Path: 0.05 Mbits/s. Combined Average Throughput: 1.63 Mbits/s.	75
4.18	MPTCP connection latency using ECF scheduler. Average latency for Wi-Fi interface: 42.04 ms. Average latency for 5G interface: 50.82 ms.	76
4.19	MPTCP connection packet loss using ECF scheduler. Total lost packets for Wi-Fi: 12. Total lost packets for 5G: 0.	77
4.20	MPTCP connection throughput using BLEST scheduler. Average Throughput for Wi-Fi Path: 1.21 Mbits/s. Average Throughput for 5G Path: 0.41 Mbits/s. Combined Average Throughput: 1.61 Mbits/s.	78
4.21	MPTCP connection latency using BLEST scheduler. Average latency for Wi-Fi interface: 36.18 ms. Average latency for 5G interface: 51.40 ms.	79
4.22	MPTCP connection packet loss using BLEST scheduler. Total lost packets for Wi-Fi: 15. Total lost packets for 5G: 2.	80
4.23	MA-RTP connection using RR scheduler. Source: [10].	82
4.24	MA-RTP connection using Redundant scheduler. Source: [10].	83
4.25	MA-RTP connection using Timely Arrival scheduler. Source: [10].	85

List of Tables

1	Contribution to the Sustainable Development Goals (SDGs)	iv
2.1	MPTCP options subtypes. Source: [4].	13
3.1	Summary of <i>minicom</i> commands and their functions.	44
3.2	Average Round-Trip Time (RTT) measurements between the RPi and Google Cloud VM to the Milton Keynes NTP Server over five tests showing the performance of using the same NTP server for both devices.	50
3.3	Average Round-Trip Time (RTT) measurements between the RPi and Google Cloud VM to the Cambridge NTP Server over five tests.	50
3.4	Average Round-Trip Time (RTT) measurements between the RPi and Google Cloud VM to the Técnico Lisbon NTP Server over five tests	51
4.1	Planned experimental scenarios.	59
4.2	Average maximum throughput values for Wi-Fi backhaul and 5G, based on five 1-minute <i>iperf</i> tests conducted in the FEUP garden, using the TCP protocol without mobility or MPTCP.	65
4.3	Mapping correspondences between MPTCP schedulers and similar MA-RTP schedulers.	81
4.4	Comparative Analysis of Average Throughput, Latency, and Packet Loss for TCP and MPTCP Using Different Schedulers Across Wi-Fi and 5G Networks.	86
4.5	Performance evaluation results summary of MA-RTP and MPTCP protocols using Round Robin, Redundant, and BLEST/Timely Arrival schedulers, detailing average throughput, maximum and average latency, and packet loss in Wi-Fi and 5G environments.	87

Abbreviations

3G	Third Generation
3GPP	Third Generation Partnership Project
4G	Fourth Generation
5G	Fifth Generation
ABC	Always Best Connected
AP	Access Point
ATSSS	Access Traffic Steering, Switching, and Splitting
BLEST	Blocking Estimation-Based MPTCP Scheduler
CWND	Congestion Window
CMT-SCTP	Concurrent Multipath Transfer using Stream Control Transmission Protocol
DCCP	Datagram Congestion Control Protocol
DSN	Data Sequence Number
DMZ	Demilitarized Zone
eMBB	Enhanced Mobile Broadband
ECF	Earliest Completion First
FEC	Forward Error Correction
FEUP	Faculty of Engineering, University of Porto
GPS	Global Positioning System
HD	High Definition
HetNets	Heterogeneous Networks
HMAC	Hash-based Message Authentication Code
IETF	Internet Engineering Task Force
ISP	Internet Service Provider
ITU	International Telecommunication Union
KVM	Kernel-based Virtual Machine
MA-RTP	Multipath Aggregated Real-Time Transport Protocol
MC	Multi-Connectivity
MP_CAPABLE	Multipath Capable
MP_JOIN	Multipath Join
MPTCP	Multipath Transmission Control Protocol
MPDCCP	Multipath Datagram Congestion Control Protocol
MP_FASTCLOSE	Multipath Fast Close
MPQUIC	Multipath Quick UDP Internet Connections
NAT	Network Address Translation
NC	Network Coding
NIC	Network Interface Card
NTP	Network Time Protocol
OMR	OpenMPTCProuter
QoS	Quality of Service

QUIC	Quick UDP Internet Connections
RFC	Request for Comments
RPi	Raspberry Pi
RR	Round-Robin
RTT	Round-Trip Time
SA	Stand-Alone
SHLO	Server Hello
TCP	Transmission Control Protocol
TS	Technical Specification
UDP	User Datagram Protocol
UE	User Equipment
UniFi	Ubiquiti UniFi
URLLC	Ultra-Reliable Low-Latency Communications
VHO	Vertical Handover
VPS	Virtual Private Server
DATA_FIN	Data Finish

Chapter 1

Introduction

1.1 Context

The fifth generation of mobile communications (5G) is designed to transform society through enhanced data rates, reduced latency, and expanded connectivity. According to the International Telecommunication Union (ITU), 5G services are divided into three primary categories: Enhanced Mobile Broadband (eMBB), which supports high-bandwidth needs like HD video streaming; Ultra-Reliable Low-Latency Communications (URLLC), designed for critical services such as autonomous driving; and Massive Machine Type Communications (mMTC), which enables high-density connections for applications like smart cities [1].

To achieve these capabilities, advancements in both radio access and core networks are required. A key innovation is multi-connectivity, which allows devices to simultaneously use multiple radio access technologies (RATs), increasing bandwidth, reliability, and overall network efficiency [15]. This also enhances Quality of Service (QoS) and facilitates better load balancing. One important architecture, Access Traffic Steering, Switching, and Splitting (ATSSS), improves 5G performance by using multiple data paths to ensure low-latency, high-reliability services for applications like autonomous vehicles [1].

In this context, the 3rd Generation Partnership Project (3GPP) introduced Technical Specification (TS) 23.501 (Release 16), which incorporates ATSSS to manage traffic between 3GPP and non-3GPP networks using multipath transport protocols [1]. These protocols optimize both throughput and network resilience, allowing seamless data transfer during mobility events such as switching between WiFi and cellular networks.

This dissertation is part of the Route 25: Agenda for Autonomous, Intelligent, Interoperable, and Inclusive Mobility project¹, supported by Instituto de Telecomunicações (IT). It focuses on the development of a multi-technology connectivity gateway that integrates 5G and WiFi networks to provide seamless service during handovers. By testing in real-world scenarios, this thesis

¹<https://www.it.pt/Projects/Index/4824>

provides insights into the performance of multipath transport protocols in high-mobility environments, offering practical solutions for enhancing connectivity in smart cities and connected vehicle applications.

As 5G networks continue to evolve, the need to understand multi-connectivity technologies under real-world conditions becomes critical. This leads to the central goal of this thesis: to evaluate the performance of multipath transport protocols under dynamic conditions in real networks.

1.2 Motivation

In densely populated areas, wireless connectivity relies on a variety of networks—WiFi, 3G, 4G, and 5G—with users demanding the best possible connection, a concept known as Always Best Connected (ABC). Devices equipped with multiple network interfaces further highlight the need for seamless handovers between heterogeneous networks (HetNets), particularly for applications with strict timing requirements such as VoIP, videoconferencing, and interactive gaming.

Several methods have been proposed to ensure seamless vertical handover (VHO), including network-layer solutions like Mobile IP (MIP) and transport-layer alternatives such as multipath transport protocols. These transport-layer solutions are particularly attractive due to their deployment flexibility, ease of implementation, and lack of dependence on application-level modifications [16].

The motivation for this thesis arises from the need to evaluate multipath transport protocols in real-world environments, involving both 5G and WiFi networks. Unlike many studies that rely on simulations, this research addresses a significant gap in the literature by focusing on real-life testing, where mobility and network interference play a substantial role in performance. By conducting tests in operational conditions, this study aims to assess how well these protocols perform under the complexities of real-world environments.

1.3 Objectives and Problem Definition

The primary objective of this thesis is to evaluate the performance of MPTCP in real-world scenarios, particularly those involving mobility. Experiments will be conducted in operational 5G and WiFi networks, focusing on key performance metrics such as throughput, reliability, and latency.

This thesis explores the potential of multipath transport protocols to improve both eMBB and URLLC services. These protocols could serve as a bridge between existing technologies, facilitating seamless mobility and enabling smooth transitions between heterogeneous networks, particularly important for next-generation applications that demand both high bandwidth and low latency. A practical example within this project involves uploading video streams or other sensor data from a mobile node to an edge server for real-time processing. This use case highlights the need for efficient, low-latency data transmission to ensure timely decision-making for applications like Cooperative Perception, where road safety for vulnerable users relies on real-time video analysis.

Additionally, this research compares the performance of the Multipath Transmission Control Protocol (MPTCP) with the previously studied Multipath Aggregated Real-Time Protocol (MA-RTP) [10] [17]. By analyzing these two protocols under the same real-world conditions, this thesis establishes a performance benchmark, enhancing the understanding of the relative strengths and weaknesses of these multipath transport solutions.

The results of this study provide practical recommendations for the deployment of multipath transport protocols in 5G and WiFi networks. By comparing MPTCP with MA-RTP, this research contributes to the development of more efficient and scalable transport solutions, helping network performance for applications in connected vehicles, smart cities, and beyond.

1.4 Dissertation Structure

In Chapter 2, multipath transport protocols are examined with a focus on MPTCP. This chapter explores essential components such as path management, scheduling, congestion control, and reliable handover. Additionally, it delves into the technical aspects of MPTCP, enhancing the understanding of its functionality across diverse network settings.

The experimental framework is detailed in Chapter 3, where the implementation of MPTCP using OpenMPTCProuter is described. This section covers the configuration of both WiFi and 5G network paths and outlines the performance metrics utilized for evaluation. It also specifies the setup components, including the VPS, WiFi subsystem, and 5G pathways.

Chapter 4 presents the experimental procedures and results. The study compares MPTCP with MA-RTP, evaluating performance metrics such as throughput, latency, and packet loss across different schedulers, including Round Robin, Redundant, ECF, and BLEST. The outcomes demonstrate MPTCP's performance characteristics and its behavior in dynamic network environments.

Finally, Chapter 5 synthesizes the research findings and contributions. It outlines potential directions for future research, such as exploring alternative congestion control algorithms and further assessing MPTCP's performance in WiFi and 5G networks to deepen the understanding of the protocol under varying network conditions.

Chapter 2

Background and State-of-the-Art

This chapter provides an in-depth study of multipath transport protocols, with a particular emphasis on the Multipath Transmission Control Protocol (MPTCP). It covers essential components such as path management, scheduling algorithms, congestion control, and reliable data transfer mechanisms. The discussion includes an examination of practical tools like OpenMPTCProuter (OMR) to highlight implementation aspects. The chapter concludes with the state-of-the-art research, about the performance of multipath transport protocols in advanced network environments like 5G and vehicular networks, establishing a foundation for the forthcoming evaluations.

2.1 Introduction to Multipath Transport

Multipath transport protocols aim to enhance communication throughput and resilience by leveraging multiple network paths simultaneously and seamlessly integrating failover capabilities [1]. Figure 2.1 illustrates a high-level representation comparing a single path (on the left-hand side) with a multipath approach (on the right-hand side) in the protocol stacks.

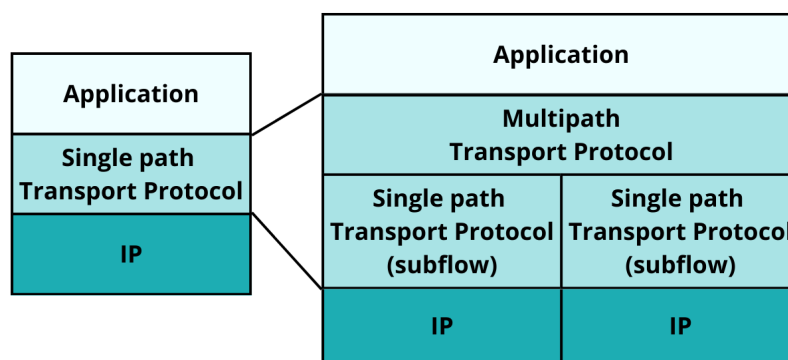


Figure 2.1: Single-path and multi-path transport protocol stack architectures. Adapted from source: [1].

Several Internet transport protocols have been extended to take advantage of multiple paths available on multihomed hosts, i.e., hosts with various network interfaces, typically connected through different datalink networks and accessible via different IP addresses. This also applies to

dual-stack hosts with a single network interface, allowing connectivity over IPv4 and IPv6. Examples of such protocols include MPTCP, Multipath Transfer Stream Control Transmission Protocol (CMT-SCTP), Multipath Datagram Congestion Control Protocol (MP-DCCP), and Multipath Quick UDP Internet Connections (MPQUIC). These protocols are designed to improve network performance and resilience by efficiently managing data transmission across multiple paths [18].

Despite the different designs and implementations of transport protocols, which differ in the way they are organized and exchange control information and user data, they share four common features relevant to ATSSS that contribute to the functionality and performance of multipath transport systems [1]:

- **Path management:** responsible for initiating and managing connections;
- **Scheduling:** in charge of distributing packets over different paths following a certain policy, e.g., aggregated throughput (utilize all available capacity), reduce latency (prefer low latency paths) or improve reliability (duplicate packets);
- **Congestion control:** aims to recognize network congestion, adjust the sending speed accordingly (similar to a single path), and addresses other aspects of using multiple paths, like being fair to single path traffic;
- **Reliable transfer:** manages the detection and recovery of lost data (similar to a single path) by having a system at the sender that identifies packet losses and another system responsible for recovering these packets through retransmissions.

2.1.1 Key Advantages

Multipathing in network environments offers enhancements such as load balancing, resource pooling, diversification, and significant roles in emerging Internet architectures. Unlike traditional network layer management, where load balancing attempts to redirect traffic to avoid congestion, this approach can cause instability and traffic oscillations. For example, when a source-destination pair operates over a congested path and then shifts to a less congested one, the network layer might hastily reroute the majority of the traffic to the new path, inadvertently causing new congestion points. This often results in the cyclic shifting of traffic burdens between paths [19].

In contrast, transport-layer multipathing adjusts the congestion window over multiple round-trip times (RTTs), facilitating a smoother and more stable traffic distribution. This approach shifts traffic from more congested to less congested paths, promoting a balance in loss rates throughout the network. The efficacy of transport-layer multipathing derives from its utilization of detailed network data, such as bandwidth and RTT, to manage traffic more adeptly.

Resource pooling in multipathing combines several network paths into a united resource, improving the overall characteristics of the path, such as bandwidth, delay and RTT. This method dynamically allocates resources to accommodate traffic swings, thus increasing the reliability and robustness of the network. Unlike scenarios involving a single path, where a failure leads to

service interruptions, multipathing redirects traffic from non-functional paths to functional paths, maintaining uninterrupted application performance.

Diversification in multipathing plays a critical role in performance optimization, especially in environments like data centers and wireless networks. Studies show that utilizing alternative paths can improve bandwidth aggregation and reliability in up to 80% of cases. By assessing path-specific characteristics, such as congestion window size, RTT, and bandwidth, multipathing enables more precise traffic engineering, which is particularly beneficial for applications like multimedia streaming. This approach enhances throughput while mitigating common issues like data loss and latency.

Furthermore, multipathing plays a key role in the architecture of future Internet technologies such as 5G and cloud networks. It facilitates seamless transitions across heterogeneous access technologies, such as between 3G and Wi-Fi, and supports high-demand applications by managing extensive bandwidth requirements and enhancing connectivity reliability. As these networks evolve, the strategic integration of multipathing is expected to address throughput demands and improve connection resilience significantly.

While multipathing introduces numerous advantages, it also presents various technical challenges that require careful consideration to ensure effective implementation. These challenges highlight the need for ongoing research and development in the field of network engineering to optimize the benefits of multipathing in contemporary and future network architectures.

2.1.2 Implementation Challenges

Implementing multipath transport in diverse network environments introduces several challenges that impact performance. One significant issue is the reordering of packets at the receiver. Without proper management, out-of-order packets may cause unnecessary retransmissions due to delays, leading to inefficient bandwidth use and increased network congestion. This undermines the benefits multipath transport aims to provide [19].

Another concern is the accurate assembly of data streams when transmitting over multiple paths simultaneously. Proper management of sequence numbers is crucial for detecting losses and addressing packet reordering, particularly when intermediate devices, or middleboxes, are involved. These devices can modify packet headers, complicating the reconstruction of data streams. Research has shown that middleboxes often disrupt traffic flows in unexpected ways.

Additionally, it is crucial to maintain fairness in resource allocation between multipath transport and other network traffic, primarily traditional TCP flows that dominate Internet traffic. In scenarios with bottleneck links, multipath-enabled connections might use more bandwidth than intended, reducing the availability for single-path transmissions.

Having established an understanding of multipath transport, its key functionalities, the advantages shared by these protocols, and the implementation challenges, the next section will focus on MPTCP. MPTCP is distinguished by its capacity to enhance network reliability and efficiency through simultaneous data transmission across multiple paths.

2.2 Fundamentals of Transmission Control Protocol (TCP)

MPTCP extends traditional TCP by enabling a single connection to utilize multiple network paths simultaneously. This protocol uses multiple IP addresses and interfaces concurrently, presenting a unified TCP interface to applications and seamlessly distributing data across subflows [5]. MPTCP is designed to be compatible with the existing network infrastructure, as it can operate through middleboxes by using its specific design parameters or, if necessary, revert to standard TCP to avoid conflicts [19]. Before analyzing how MPTCP operates, it is important to first understand certain aspects of TCP, as MPTCP extends TCP and preserves its core functionalities.

TCP is a connection-oriented protocol that maintains the state of a connection throughout its duration. The protocol's behavior adapts based on its current state, with specific actions defined for each state to ensure connection integrity. A connection is established and terminated through a series of message exchanges.

Known for its reliability, TCP tracks each byte of data sent and received, detecting and handling any data loss. Since it is connection-oriented, both ends can communicate resource limitations, preventing unnecessary network traffic and duplication of efforts. This is achieved through the sliding-window algorithm.

Additionally, TCP manages congestion control by adjusting data transmission rates according to network traffic conditions, sending data in smaller or larger chunks as needed. This prevents network congestion and optimizes throughput. Various congestion control algorithms enhance its reliability and stability, making it a widely used protocol in internet technology [2].

A TCP segment comprises a TCP header and the TCP data (payload). The header includes protocol control information, connection-specific details, and a field to validate its integrity. Typically, the TCP header is 20 bytes (160 bits) long, but it can vary in length due to optional fields [2].

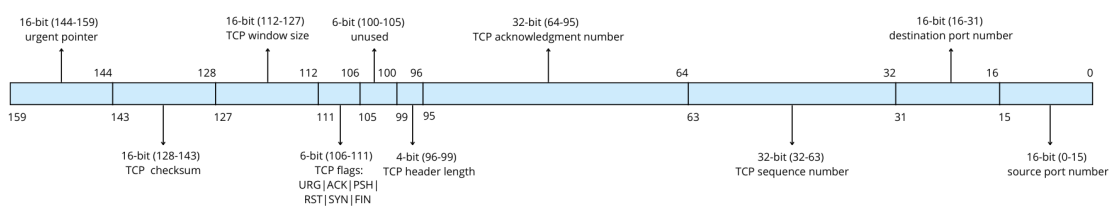


Figure 2.2: TCP header, that has a constant size of 160 bits, or 20 bytes. Adapted from source: [2].

- **Port Numbers** - A TCP connection is identified by destination IP, destination port, source IP, and source port. The TCP header contains the source port (bits 0-15) and the destination port (bits 16-31), each 16 bits long. These port numbers uniquely identify the sockets at each end of a TCP connection.
- **Sequence Number** - A 32-bit field (bits 32-63) in the TCP header indicates the offset of the first byte in the byte stream that the sender intends to send in the current TCP segment. The

sequence number is an offset from the initial sequence number (ISN) and rolls over after reaching $2^{32} - 1$. It helps maintain data integrity and identify retransmissions.

- **Acknowledgment Number** - A 32-bit field (bits 64–95) in the TCP header. It indicates the next byte expected from the sender, ensuring reliable data transfer. Out-of-sequence segments are not acknowledged until the missing data is received.
- **Header Length** - A 4-bit field indicating the number of 4-byte words in the TCP header. The standard TCP header is 20 bytes long without options, and the maximum length is 60 bytes.
- **Unused Field** - A 6-bit field (bits 100–105) reserved for future use.
- **TCP Flags** - A 6-bit field in the TCP header, with each bit representing a flag:
 - **URG** - Urgent pointer field check required.
 - **ACK** - Acknowledgment of the TCP segment.
 - **PSH** - Prioritized consumption of data requested.
 - **RST** - Connection closure without formal handshake.
 - **SYN** - Connection initiation synchronization.
 - **FIN** - Desire to close the connection.
- **Checksum** - A 16-bit field (bits 128–143) used by the receiver to verify the received segment matches the one sent. It covers the TCP header and payload, with the final value expected to be all 1's ($2^{16} - 1$).
- **Urgent Pointer** - A 16-bit field (bits 144–159) indicating the offset from the sequence number where urgent data resides, requiring immediate processing. Set only if the URG flag is set.

2.2.1 TCP Congestion Control

TCP is a reliable protocol that ensures data delivery through acknowledgments (ACKs) for each byte received. TCP detects network congestion by the expiration of the retransmission timer and by receiving duplicate acknowledgments. There are various methods for managing these scenarios:

1. **Retransmission Timer Expiration:** When the retransmission timer expires, it indicates a complete loss of data in the last window, as the timer starts with the first data segment transmission. In this case, all data from the last window must be retransmitted, beginning with the first segment.

2. **Duplicate Acknowledgments:** Receiving duplicate acknowledgments suggests packet loss, prompting the retransmission of the lost segment, assuming no segment reordering occurred. This method, known as fast retransmit and fast recovery, allows for early loss detection and correction.

For congestion control, there are two state variables: the *congestion window* (*cwnd*) and the *slow-start threshold* (*ssthresh*). When TCP enters the loss state, it initializes the slow-start phase. In this phase the congestion window (*cwnd*) is set to 1, and the slow-start threshold (*ssthresh*) is set to half the size of the congestion window before the loss, provided this value is greater than or equal to 2; otherwise, it is set to 2. In the slow-start phase, the data transmission rate depends on the rate at which ACKs are received. Lost packets are retransmitted at a rate that increases exponentially until the congestion window reaches the value of the slow-start threshold (*ssthresh*).

At this point, another phase called *congestion avoidance* begins. During this phase, the congestion window (*cwnd*) is increased linearly by one segment for each RTT, which is the time it takes for a signal to travel from the sender to the receiver and back. Unlike the slow-start phase, the data transmission rate no longer depends on the ACK reception rate.

By managing these phases effectively, TCP aims to optimize data transmission rates while avoiding network congestion [2].

2.2.2 Impact of Round-Trip Time (RTT) on TCP Performance

Round-trip time (RTT) is a critical parameter that influences the TCP performance. RTT is the time it takes for a signal to travel from the sender to the receiver and back. In TCP, every byte of data sent must be acknowledged by the receiver. If the sender doesn't receive an acknowledgment within a specific time frame (based on RTT), it assumes the data segment is lost and retransmits it [2].

- **Underestimation of RTT:** If RTT is set too low for slower networks, the sender might retransmit segments prematurely, assuming the original segment or its acknowledgment is lost while it might still be in transit. This wastes bandwidth increases overhead due to unnecessary packet generation and transmission, and can cause the sender to mistakenly think there is network congestion, leading to reduced data transmission rates.
- **Overestimation of RTT:** Conversely, if RTT is set too high for faster networks, the sender waits too long before retransmitting a lost segment. This delay means recovery from actual data loss is slower, causing overall TCP performance to suffer because the sender does not promptly address the data loss.

Having established a foundational understanding of TCP, we now explore how MPTCP builds on these principles. MPTCP addresses TCP's single-path limitations by enabling the use of multiple network paths, enhancing load balancing, fault tolerance, and overall efficiency.

2.2.3 Effects of Packet Loss on TCP Efficiency in Wireless Networks

As wireless technologies evolve and user demands increase, there is a need for the IP protocol suite to expand its capabilities to include wireless communications. Future all-IP networks will likely be heterogeneous, combining both wired and wireless connections in the communication path from one end to the other. However, the traditional TCP, which has been effective in wired networks, shows limitations in these hybrid environments [3].

There are two types of network losses: congestion losses and non-congestion losses [20]. The first is related when the load on the network is greater than its capacity, the network is called congested. Due to this, some packets may get queued, increasing the delay and some get dropped. It mainly occurs in wired network. To resolve this, TCP reduces its congestion window size to half and retransmit the lost packet.

The second network loss is a named non-congestion loss. In WLANs or cellular networks, packet losses frequently result from wireless channel errors and frequent cell handoffs and random bit or burst error. Random bit errors are due to shadowing effect, fast fading which causes a bit flipping. Moreover, unlike the stable environments of fiber-optic backbones and copper-wired networks, wireless links rely on open air for transmission. This exposes them to various uncontrollable factors such as weather conditions, urban structures, multipath interferences, large moving objects, and the mobility of end devices. Consequently, wireless links tend to have much higher bit error rates (BERs) than their wired counterparts [3].

When there is an error in a frame that the MAC layer cannot fix, it discards the frame. Burst errors, which affect contiguous bits of a frame, can occur due to host mobility, frequent handovers, or prolonged interference. As a result, the congestion control mechanism activates, reducing the size of the congestion window and prompting the retransmission of the packet. This reduction in the congestion window size causes TCP to throttle back, significantly decreasing the throughput [20].

Therefore, TCP often interprets any packet loss as a sign of congestion, which prompts it to reduce the congestion window size by half and start retransmissions. This approach can lead to resource inefficiency. As a result, various applications experience decreased throughput, poor network resource utilization, and frequent disruptions in data transmission [3].

Additionally, the variable conditions of wireless channels significantly affect TCP behavior, influencing the RTT required to send a packet and receive its acknowledgment. Following a retransmission timeout (RTO), packets might be resent even if their acknowledgments are subsequently received, under the assumption that they were lost. This issue, termed spurious retransmission, leads to two significant problems: it results in bandwidth wastage and fails to minimize energy consumption, both of which are critical considerations in wireless networks [20].

Furthermore, TCP in wireless networks faces issues with packet reordering. This occurs when packets arrive in a different order than they were sent, often leading to duplicate acknowledgments and subsequent unnecessary retransmissions.

It is important to note that even a brief disconnection can significantly delay TCP transmissions

for a disproportionately longer time. This phenomenon is illustrated in Figure 2.3, which shows the evolution of the congestion window reflecting TCP transmission throughput, with time measured in RTTs.

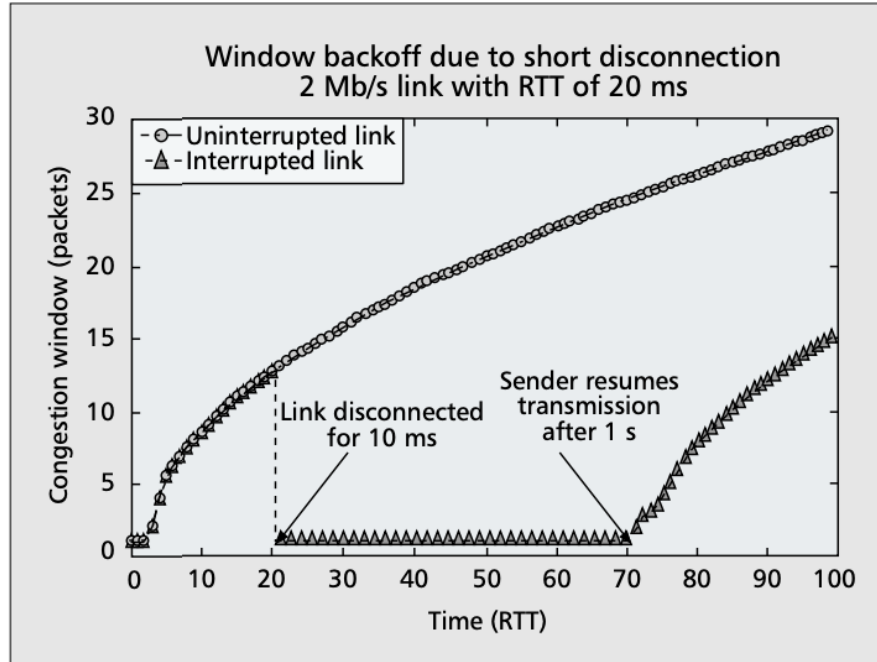


Figure 2.3: Effect of a short disconnection in TCP transmission. Source: [3].

In this scenario, a TCP connection exists between two hosts with a bandwidth capacity of 2 Mb/s and a RTT of 20 milliseconds. During a period of disconnection, both data packets sent by the sender and ACKs from the receiver are lost. This means that every attempt by the sender to transmit data fails because it doesn't receive the necessary ACKs.

Due to these continuous transmission failures, the TCP sender increases the retransmission timer exponentially, typically in increments of 500 milliseconds. After each failed attempt, the sender waits an additional 500 milliseconds before retrying, and this waiting period grows exponentially with each failure.

As a result of this exponential increase, the delays between retransmission attempts become significantly longer. Specifically, the total delay can rise to around 1 second. Given that the RTT is 20 milliseconds, 1 second equates to 50 RTTs (since 1 second divided by 20 milliseconds per RTT equals 50). Therefore, the overall delay of these retransmissions increases to approximately 50 RTTs, or about 1 second, due to the repeated failures and the exponential growth of the retransmission timer.

2.3 Multipath Transmission Control Protocol (MPTCP)

MPTCP allows hosts to use multiple paths with different IP addresses to exchange packets within a single connection. Operating at the transport layer, it is designed to support legacy applications

that are not aware of multipath functionality, maintaining transparency to both the upper and lower layers of the network stack. MPTCP is structured into a specific layer and a TCP underflow layer, enabling applications to use multiple paths simultaneously, as shown in Figure 2.4. This protocol enhances standard TCP by introducing several additional features [4]. TCP interacts with various components of the network stack through different interfaces, with the standard interface between TCP and applications being the sockets interface [21].

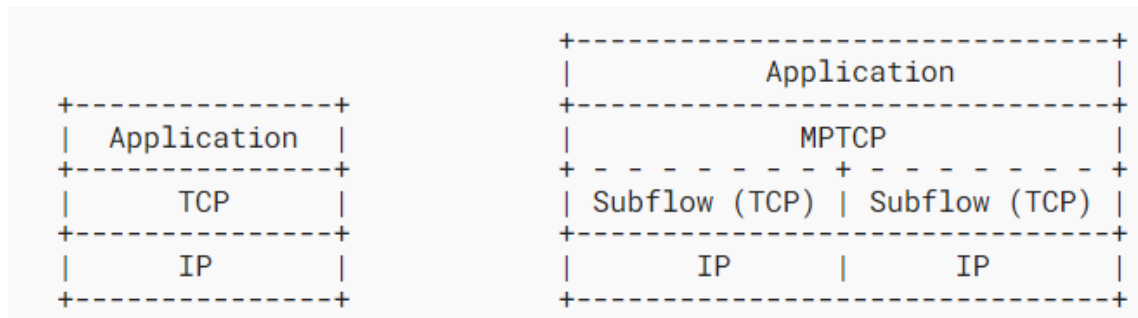


Figure 2.4: Comparison of Standard TCP and MPTCP Protocol Stacks. Source: [4].

At the network layer, each MPTCP subflow resembles a regular TCP flow, with segments carrying a new TCP option type. MPTCP handles the creation, removal, and use of these subflows for data transmission. The number of subflows in an MPTCP connection can change over the connection's duration. All MPTCP operations are indicated using a TCP option that uses a numerical type for MPTCP, along with specific "subtypes" for each MPTCP message [4]. Having understood the main concept of the MPTCP, it is now necessary to understand how it operates.

2.3.1 MPTCP Operation

To fully understand how MPTCP operates, it is essential to first familiarize with several key terms and definitions specific to this protocol, as outlined in [4]. These terms will be referenced throughout the following sections.

- **Path:** A sequence of links between a sender and a receiver, defined by a 4-tuple of source and destination address/port pairs.
- **Subflow:** A flow of TCP segments operating over an individual path, forming part of a larger MPTCP connection. A subflow is initiated and terminated like a regular TCP connection.
- **MPTCP Connection:** A set of one or more subflows used by an application to communicate between two hosts. There is a one-to-one mapping between a connection and an application socket.
- **Data-level:** The payload data is transferred over a connection, which is transported over subflows. "Data-level" is synonymous with "connection-level", in contrast to "subflow-level", which refers to the properties of an individual subflow.

- **Token:** A locally unique identifier given to a multipath connection by a host, also referred to as a "Connection ID".
- **Host:** An end host operating an MPTCP implementation, initiating or accepting an MPTCP connection.
- **Head-Of-Line Blocking:** When packets travel through different paths, they can arrive out of order due to varying delays and loss rates. For instance, if two paths exist between nodes, with one having a lower delay, the packets taking the faster route will arrive first and have to wait in the receiver's buffer for the packets from the slower path. This situation, known as head-of-line blocking, causes the out-of-order packets to delay each other. This can lead to a network burst, which is problematic for streaming applications requiring consistent and smooth data flow [9].
- **Window Limitations:** A problem called window blocking happens when the receive buffer is filled with out-of-order packets, causing the sender to reduce its send window. This situation prevents data transfer on high-performance paths (low delay, high bandwidth) until the blocked packets are acknowledged. Another issue is send buffer blocking. The sender keeps data in its buffer until it receives an acknowledgment. If the receive buffer is full, the sender has to retain the out-of-order packets, waiting for the blocking packets to be processed [9].

Figure 2.5 represents the MPTCP's connection maintenance summarized in five main phases, and Table 2.1 represents the MPTCP options used during these operations. The next Sections will explain in detail how these phases occur.

Table 2.1: MPTCP options subtypes. Source: [4].

Value	Symbol	Name	Reference
0x0	MP_CAPABLE	Multipath Capable	RFC 8684, Section 3.1
0x1	MP_JOIN	Join Connection	RFC 8684, Section 3.2
0x2	DSS	Data Sequence Signal (Data ACK and Data Sequence Mapping)	RFC 8684, Section 3.3
0x3	ADD_ADDR	Add Address	RFC 8684, Section 3.4.1
0x4	REMOVE_ADDR	Remove Address	RFC 8684, Section 3.4.2
0x5	MP_PRIO	Change Subflow Priority	RFC 8684, Section 3.3.8
0x6	MP_FAIL	Fallback	RFC 8684, Section 3.7
0x7	MP_FASTCLOSE	Fast Close	RFC 8684, Section 3.5
0x8	MP_TCPRST	Subflow Reset	RFC 8684, Section 3.6
0xf	MP_EXPERIMENTAL	Reserved for Private Use	

2.3.1.1 Starting an MPTCP Connection

The MPTCP connection starts with the same signaling used to initiate a standard TCP connection. However, the SYN, SYN/ACK, initial ACK packets, and data packets also include the MP_CAPABLE option, which varies in length and serves multiple purposes. First, it checks if the remote host supports Multipath TCP, and, second, it enables hosts to exchange information to authenticate

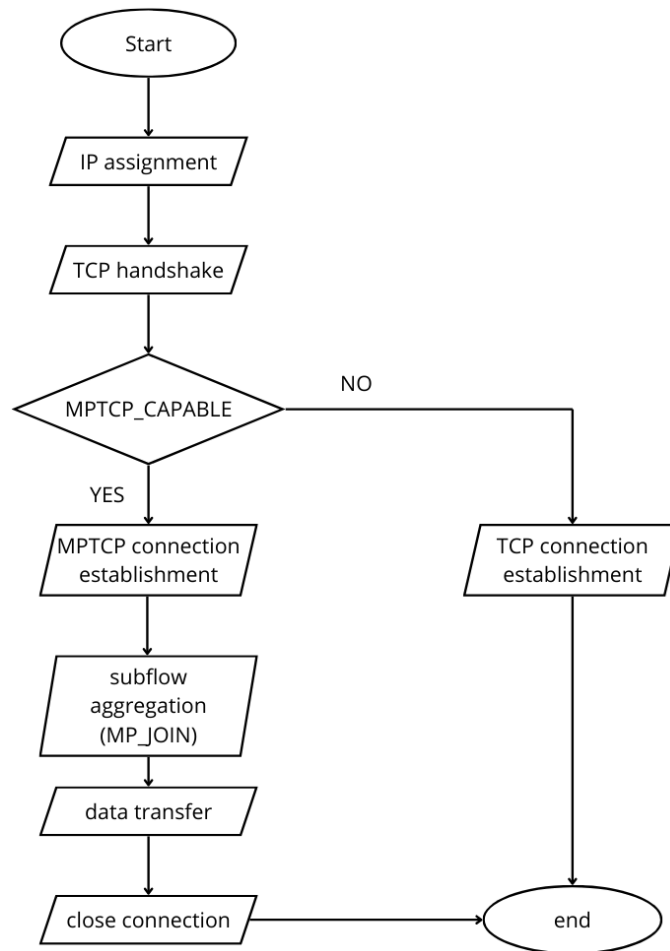


Figure 2.5: Main MPTCP operations to maintain a connection. Adapted from source: [5].

the setup of additional subflows. Retransmission of the `ACK + MP_CAPABLE` can occur if it is uncertain whether it has been received [4].

To associate a new subflow with an existing MPTCP connection, host A starts a new subflow between one of its addresses and one of Host B's addresses. The token, created from the key, identifies which MPTCP connection it joins. The Hash-based Message Authentication Code (HMAC) authenticates the connection using the keys exchanged in the `MP_CAPABLE` handshake and the random numbers (nonces) exchanged in this `MP_JOIN` options. `MP_JOIN` also includes flags and an Address ID that can refer to the source address without the sender needing to know if a NAT (Network Address Translation) has changed it [4]. These elements are further explained in the subsequent section. The diagram in Figure 2.6 illustrates a possible exchange for the initial subflow setup and the association of a new subflow.

2.3.1.2 Notifying the Host of an Additional Address

In an MPTCP connection, a multi-homed host's set of IP addresses can change over the connection's lifetime. MPTCP facilitates both implicit and explicit addition and removal of addresses

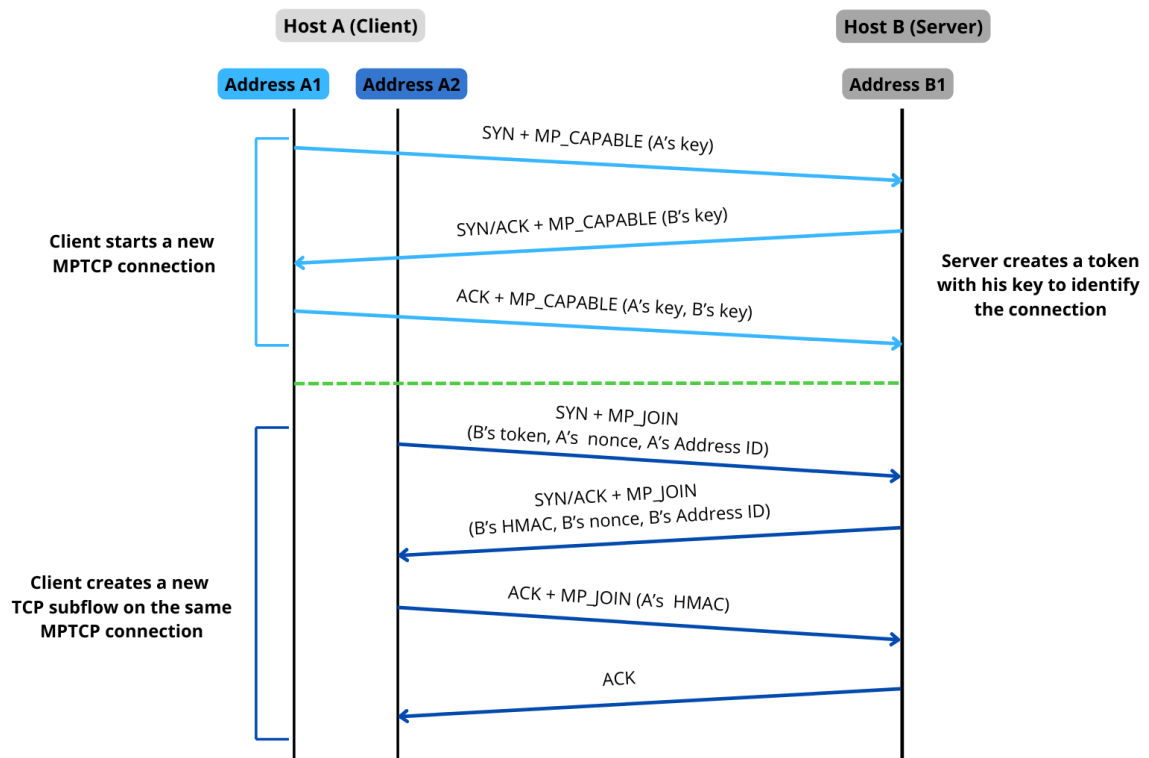


Figure 2.6: Start of a new connection, initial subflow setup, and the process to add a new subflow to an existing MPTCP connection.

on a host. For example, if Host A has an established subflow using the address/port pair IP#-A1 and wants to initiate a second subflow with IP#-A2, it can start the subflow setup as previously outlined. The remote host will then become implicitly aware of the new address.

However, there are scenarios where a host needs to inform the remote host about a new available address without establishing a new subflow—such as when a NAT restricts setup in one direction. In this situation, Host A can notify Host B of its alternative address/port pair (IP#-A2). Host B may later use this information to send an MP_JOIN to the new address. The ADD_ADDR option used for this notification includes an HMAC for authentication, ensuring the address was sent by the connection's originator. The receiver echoes this option back to the sender to confirm successful receipt. To remove an address in an MPTCP connection, a REMOVE_ADDR is used. It leverages the Address ID provided during the ADD_ADDR handshake. This allows the host to inform the remote host about the address removal efficiently. The process of notifying a new IP address and removing an IP address is shown in Figure 2.7.

2.3.1.3 Data transfer with MPTCP

To ensure reliable, in-order delivery of data over multiple subflows that may dynamically change during a connection, MPTCP employs a 64-bit Data Sequence Number (DSN) to track all data transmitted across the entire MPTCP session. Each subflow, operating like a standard TCP connection, maintains its own 32-bit sequence number space using the regular TCP sequence number



Figure 2.7: Adding a new IP from Host A, interface 2 and removing the same IP. Source: [4].

field. Figure 2.8 shows an example with the TCP sequence numbers and DSNs of the MPTCP connection represented [4].

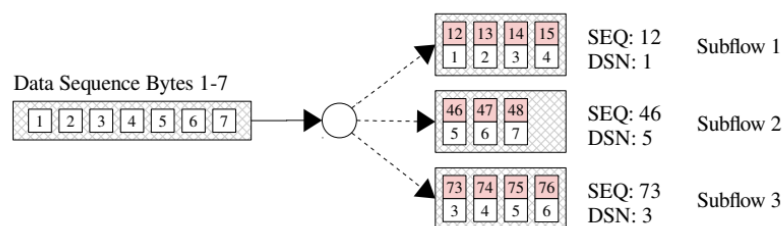


Figure 2.8: Example of mapping - DSN can be mapped with multiple subflows, such as in cases where retransmission is needed. Source: [6].

However, because MPTCP utilizes multiple subflows concurrently, it must map these individual sequence numbers into a unified data sequence space. This is achieved using the "MPTCP option," an additional protocol mechanism embedded within the TCP header. The MPTCP option facilitates mapping the subflow's 32-bit sequence numbers to the 64-bit DSN, effectively linking each subflow's transmitted data to the overall data sequence of the MPTCP connection [4].

By maintaining this mapping, MPTCP can retransmit data across different subflows in the event of a failure. Since the DSN uniquely identifies each segment of data in the overall MPTCP connection, retransmitted data retains its original position within the data stream, regardless of the subflow utilized for retransmission.

The Data Sequence Signal (DSS) carries the Data Sequence Mapping, which includes the subflow sequence number, DSN, and length for which the mapping is valid. This option can also carry a connection-level acknowledgment (the "Data ACK") for the received DSN [4]. Figure 2.9 represents the TCP and MPTCP header, where the DSS is represented.

In MPTCP, all subflows utilize a common receive buffer and advertise an identical receive window. MPTCP employs two acknowledgment levels: at the subflow level, where standard

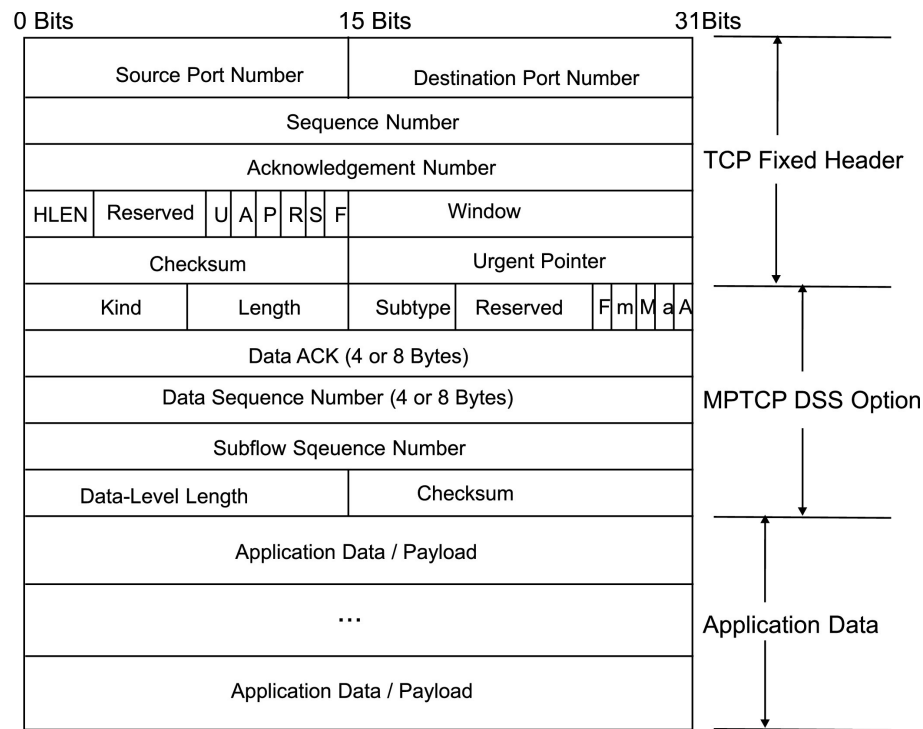


Figure 2.9: The encapsulation of the MPTCP DSS within the TCP option field. Source: [7].

TCP acknowledgments are used to independently confirm the receipt of segments on each subflow, without reference to their Data Sequence Number (DSN); and at the connection level, where MPTCP uses additional acknowledgments specific to the overall data sequence. These connection-level acknowledgments monitor the progress of the byte stream and adjust the receive window accordingly [4].

2.3.1.4 Request for Modification of Path Priority

At the initial setup of a subflow, hosts can specify whether they want the subflow to serve as a regular path or a backup path, with the latter only being utilized if no regular paths are available. During the connection, Host A can use the `MP_PRIO` signal to request Host B to change the priority of a subflow [4].

2.3.1.5 Terminating an MPTCP connection

In regular TCP, a FIN signals that the sender has no more data to send, closing the connection after both sides acknowledge each other's FINs. In MPTCP, a FIN only closes the specific subflow on which it is sent, allowing independent subflow operation. When an application closes an MPTCP socket, a `DATA_FIN` is used to indicate that no more data will be sent. The `DATA_FIN` is a unidirectional signal that verifies all data has been received and is marked by setting the "F" flag in the DSS option [4].

The DATA_FIN occupies the final octet in the connection-level sequence space and is acknowledged once all data is received. Unlike subflow FINs, DATA_FIN operates at the connection level and is independent of individual subflow FINs. A host must not close all subflows until all data is acknowledged or the segment with the DATA_FIN is the only outstanding one [4].

After the DATA_FIN is acknowledged, all remaining subflows should close with standard FIN exchanges. If no DATA_FIN is acknowledged, the MPTCP connection is considered closed only after a timeout. This enables scenarios where connectivity is temporarily lost on all subflows before a new one is established [4].

An alternative method for closing a connection is known as "Fast Close", which is similar to terminating a single-path TCP connection using a RST signal. The MP_FASTCLOSE signal is employed to notify the peer that the connection will be immediately terminated and no further data will be accepted. This signal can be sent either with an ACK (ensuring the reliability of the signal) or with a RST (which does not guarantee reliability) [4].

2.3.2 Path Management

The path manager component determines how connections are established and when additional subflows are created. It contains the logic that regulates the creation and deletion of these flows and controls which IP addresses are available for new subflows.

The decision-making process is influenced by the specific needs of the application. Some applications use multipath for handover, while others use it for load sharing. The coordination between establishing subflows and their usage during the connection is carried out in collaboration with the multipath scheduler [1], which is discussed in the next subsection.

MPTCP implements multi-homing and multipath transport since each MPTCP connection represents one or more TCP sub-flows, each defined by the source/destination IP address (IPv4 or IPv6). MPTCP even allows to simultaneously have IPv4 and IPv6 subflows. Figure 2.10 illustrates a possible connection between two hosts, each with two interfaces. In this example, all possible subflows are established, forming a full mesh of paths [22].

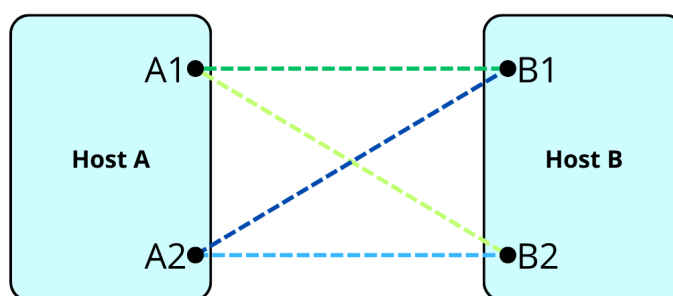


Figure 2.10: Example of MPTCP Connection: Full mesh paths. Adapted from source: [1]

The choice of path establishment, such as using a full mesh or a subset, depends on the protocol's implementation. In Linux MPTCP, for example, four path managers are provided [22]:

- **default:** Does not announce different IP addresses or initiate the creation of new subflows. It only accepts the passive creation of new subflows.
- **fullmesh:** Establishes all possible subflows between all available IP addresses.
- **ndiffports:** Uses the same pair of IP addresses for its paths but utilizes different source and destination TCP ports for each path. This approach aims to bypass middleboxes that limit bandwidth by imitating different TCP connections.
- **binder:** Uses loose source routing to allow senders to influence the path of packets through specified intermediate routers while still leveraging the network's dynamic routing capabilities.

To understand the operations of path management and scheduling in MPTCP, explained in the next subsection, the path management algorithm set up a subflow and the scheduler, often based on measuring the RTT of the subflows, might prefer the subflow with the lowest RTT [1].

2.3.3 Scheduling Algorithms

The packet scheduler is an algorithm responsible for choosing the route for transmitting each packet in a multipath connection, according to a given policy. This mechanism becomes relevant when there are at least two active paths available for a given multipath transport connection. Various packet schedulers have been suggested and applied in multipath transport protocols. The performance of these protocols is significantly influenced by the chosen packet scheduler, as observed in experiences with multipath transport protocols [1] [23]. Figure 2.11 illustrates the organization of various schedulers within the solution space, highlighting the relationships between applications, schedulers, and their overarching objectives.

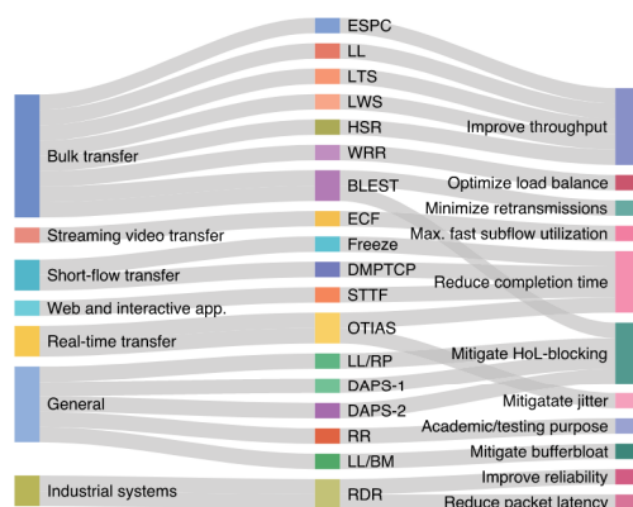


Figure 2.11: Applications, schedulers, and their overall goals. Source: [8].

According to [1], paths can be categorized as either homogeneous or heterogeneous in terms of factors like bandwidth, delay, and loss rates. To better understand the influence that a scheduler has on the performance of multipath transport protocols, consider a basic Round-Robin (RR) scheduler in MPTCP, where packets are sent cyclically over each available path as long as there is space in their Congestion Windows (CWND). However, this simplistic approach, while suitable for homogeneous paths, falls short in handling path heterogeneity. RR does not take path characteristics into account, leading to out-of-order packet arrivals. This can result in receiver buffer blocking and head-of-line blocking, especially when data must be delivered to the application in order, ultimately decreasing overall performance. As the degree of path heterogeneity increases, scheduling across multiple paths becomes more challenging [1].

To tackle these challenges, several strategies can be applied. One method uses transport layer data, such as RTT and CWND, to estimate packet transfer times on each path. The scheduler then distributes packets to maintain order. Another approach involves duplicating packets to improve either latency or reliability, depending on the current path conditions and optimization goals (throughput or latency). Recently, machine learning techniques, including reinforcement and supervised learning, have been explored to optimize latency and/or throughput within a single algorithm. Machine learning features can be derived from transport layer data like RTT, CWND, and in-flight packets [1].

Figure 2.12 (b) and (c) illustrate two scenarios where subflows r_0 and r_1 pass through different bottlenecks, b_0 and b_1 . Despite encountering bottlenecks with lower delays, the r_{best} scheduler cannot enhance multipath performance due to low capacity or high packet loss in these bottlenecks.

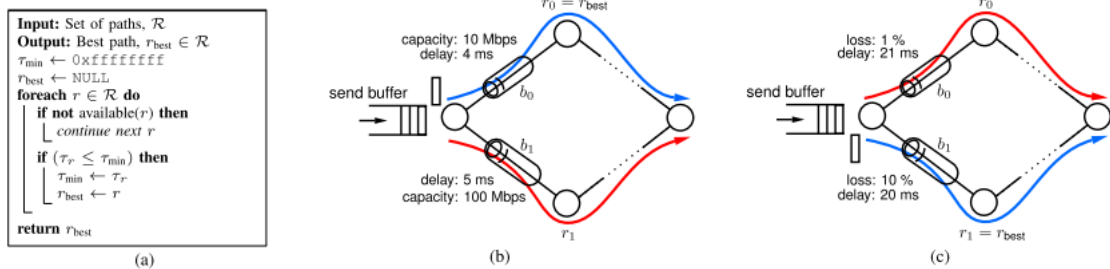


Figure 2.12: Default MPTCP scheduler and examples of its blinder scheduling decisions in scenarios where two paths have heterogeneous characteristics. (a) Lowest Latency—LL (default MPTCP scheduler). (b) r_{best} with less delay, but less capacity. (c) r_{best} with less delay, but higher loss. Source: [8].

MPTCP schedulers primarily ensure that data is delivered in the correct order to the receiver. They determine which subflows to use for sending data among the available options, aiming to prevent head-of-line blocking at the receiver [24]. Some of the MPTCP schedulers are:

- **Default Scheduler (minRTT):** The RTT-based scheduler prioritizes sending data on the subflow with the lowest RTT until its congestion window is full. After this, it sends data on the subflow with the next lowest RTT. Once all congestion windows are filled, the scheduling relies on acknowledgments. As acknowledgments are received, they create space in the

congestion windows, allowing the scheduler to continue sending data on those subflows. Note that using a scheduler that selects the subflow with the lowest RTT enhances the user experience by lowering application delays. This is especially important for interactive applications [25].

- **Round-Robin:** The round-robin scheduler cycles through each subflow in sequence, ensuring an equal distribution of data across all paths. This method can help fully utilize the capacity of each path. However, during large data transfers, the process is not strictly round-robin. Instead, once the application fills the congestion window of all subflows, packets are scheduled based on when space becomes available in each subflow's congestion window, a phenomenon known as ack-clocking [25].
- **Redundant:** The scheduler transmits identical data packets across all subflows that have available space in their CWND. This method of sending duplicate data through multiple communication paths helps to alleviate head-of-line blocking at the receiver, thereby ensuring low data delivery latency. However, transmitting identical packets consumes considerable bandwidth, which could otherwise be used for transmitting other essential data through a more intelligent scheduling scheme [24].
- **Blocking Estimation-Based MPTCP Scheduler (BLEST):** The BLEST scheduler incorporates a mechanism to predict head-of-line (HoL) blocking in advance. By analyzing the RTT and the send window of a subflow, it determines whether transmitting data through that subflow might result in HoL blocking at the receiver. If this risk is detected, BLEST temporarily halts data transmission through the identified subflow. Essentially, BLEST prioritizes waiting for the subflow with the lowest RTT to become available, even if other subflows have space in their CWND, to avoid HoL blocking at the receiver [24].
- **Earliest Completion First (ECF):** The ECF scheduler aims to make intelligent scheduling decisions by taking into account the RTT of the subflows, their available congestion window (CWND), and the size of the data to be transmitted. Abstractly, in making scheduling decisions, it estimates the packet arrival times at the receiver and selects a subflow such that packets arrive in order at the receiver [24].

2.3.4 Congestion Control Mechanisms

In computer networks, congestion arises when the volume of outstanding packets exceeds the capacity of the devices connecting to the Internet. Intermediate devices like routers and switches have limited buffer memory and processing power. Congestion occurs when these devices are pushed beyond their operational limits. If a router receives more traffic than it can handle, it experiences an overload, leading to congestion. Consequently, the router may discard some packets, which is a secondary effect of this condition [26].

Congestion control algorithms work with aspects like packet loss and delay to identify network congestion and respond by modifying the transmission rate. According to [27] (RFC6356), the

following three objectives highlight the key features of a practical congestion control algorithm for multipath communication:

- **Enhance Throughput:** A multipath flow should perform at least as well as a single-path flow on its best available path.
- **Do No Harm:** A multipath flow should not consume more capacity from shared resources than a single flow using only one path. This ensures it does not unfairly impact other flows.
- **Balance Congestion:** A multipath flow should divert traffic away from its most congested paths while meeting the first two objectives.

Together, the first two goals ensure fairness at the bottleneck (part of the network with the lowest capacity, causing it to control the overall data transfer rate), while the third goal encourages resource pooling. Resource pooling involves directing more data through the least congested path, which helps redistribute traffic, enhancing robustness and overall throughput. This is achieved by effectively coordinating the congestion control loops for different subflows.

Multipath congestion control can be broadly classified into two approaches: uncoupled and coupled. In the uncoupled approach, each subflow within a multipath connection is treated independently. This means that the congestion window (CWND) of each subflow is adjusted without considering the impact on other subflows. On the other hand, coupled approaches, as outlined in RFC6356 for standardization purposes, consider all sub-flows within a multipath connection as a unified entity [1].

Regarding TCP and MPTCP congestion algorithms, Figure 2.13 illustrates various examples of single-path TCP and multipath congestion control algorithms [9].

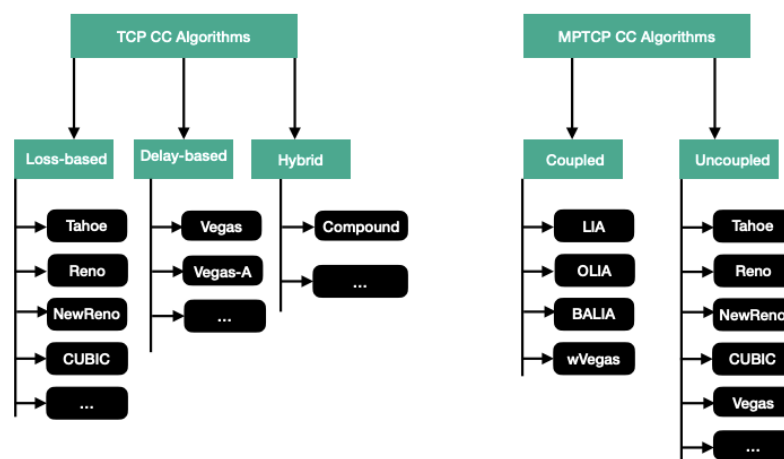


Figure 2.13: TCP and MPTCP CC examples. Source: [9].

One of the congestion control algorithm of MPTCP is Bottleneck Bandwidth and Roundtrip propagation time (BBR). This algorithm utilizes a method distinct from congestion control algorithms that adjust the congestion window based on packet loss. It builds a model using measurements of the connection's delivery rate and round-trip time (RTT). This model dictates data transmission speed and the allowable amount of data in flight. The delivery rate is defined by the data acknowledged within a specific time frame, measured by counting the number of packets acknowledged during that period. To identify the optimal delivery rate at intervals, BBR employs a probing mechanism known as gain cycling. Research indicates that BBR improves throughput in networks with limited buffer capacity and reduces queuing delays in networks with extensive buffering, effectively addressing bufferbloat. BBR is currently implemented in services like Google's Cloud Services and YouTube, which have reported a 4% increase in throughput and a 33% reduction in RTT [28].

2.3.5 Reliable Data Transfer

Transport layer protocols can be grouped into two types: reliable, like TCP which guarantees all data delivery, and unreliable, like UDP which doesn't ensure delivery. In multipath scenarios, protocols like MPTCP maintain reliability by recovering lost packets within the same path or retransmitting through other paths [1].

To enhance reliability in multipath situations, techniques like Forward Error Correction (FEC) and Network Coding (NC) are considered. FEC involves encoding data at the sender to recover lost packets at the receiver, while NC can be applied at both the sender and intermediate nodes [1].

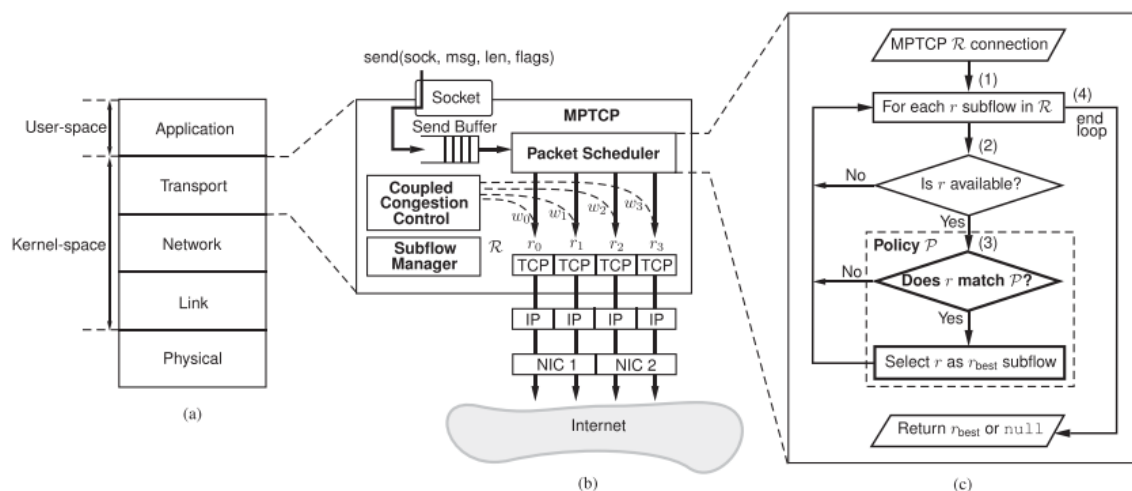


Figure 2.14: General views of the MPTCP Linux kernel implementation regarding a multihomed end system equipped with two network interface cards (NICs). (a) TCP/IP protocol stack. (b) MPTCP architecture. (c) Modular framework of packet scheduling. Source: [8].

Figure 2.14 provides a comprehensive overview of the MPTCP in the context of a multihomed end system equipped with two network interface cards (NICs) and it is possible to observe three functionalities of multipath transport (path management, congestion control, and scheduling).

2.4 Multipath Adaptive Real-time Transport Protocol (MA-RTP) compared to MPTCP

This section provides an overview of RTP and MP-RTP and introduces MA-RTP, an enhanced variant of MP-RTP designed for transmitting real-time, loss-tolerant data streams. MA-RTP addresses MP-RTP's limitations in managing short-term network fluctuations and offers a more suitable alternative to MP-TCP for applications that do not require continuous byte streams. In this study, experiments with MPTCP were conducted in real-world mobile environments to evaluate its performance. The results from the MPTCP experiments are compared with MA-RTP findings from previous research. This section establishes the foundation for the subsequent performance comparison between MPTCP and MA-RTP, presented in Section 4.4.

2.4.1 Overview of Real-time Transport Protocol (RTP) and MPRTCP

Real-time Transport Protocol (RTP) works at the application layer, and it aims to facilitate real-time transmission between sender and receiver. However, it does not ensure complete reliability, and lost data cannot be recovered. Additionally, RTP does not guarantee the order of data packets, as sequencing is managed by higher layers. Consequently, RTP is used only in scenarios where some data loss is acceptable. There is a lack of appropriate transmission protocols for applications that require both real-time delivery and reliability [29].

This protocol was initially developed for data transmission over UDP/IP, but is also compatible with transport protocols like TCP and DCCP. RTP is effectively utilized in applications including live streaming, broadcasting, video-on-demand services, and interactive multimedia communications. To oversee end-to-end media delivery and manage control operations, RTP integrates with the RTP Control Protocol (RTCP) [30].

Multipath Real-time Transport Protocol (MP-RTP) extends RTP by supporting multiple communication paths and operates at the application layer. Figure 2.15 compares the protocol stacks of standard RTP and MP-RTP.

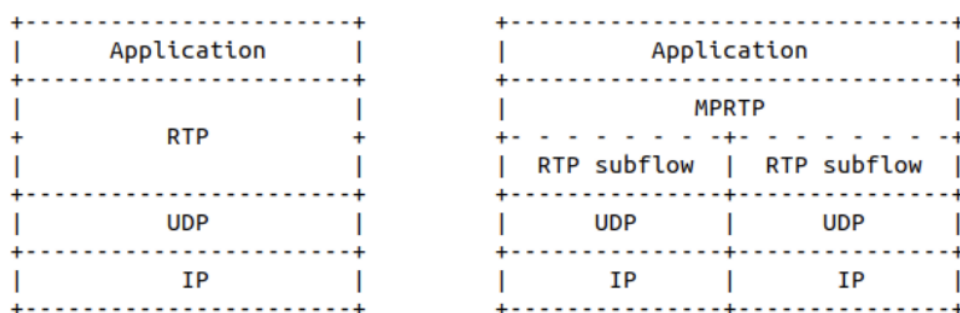


Figure 2.15: Standard RTP and MP-RTP stack. Source: [10].

2.4.2 Introduction to MA-RTP

Real-time data streams, such as video, require transport protocols that ensure timely delivery and minimize data loss. Existing multipath protocols, MP-TCP and MP-RTP, enhance resource utilization, throughput, and fault tolerance by aggregating bandwidth across multiple paths. However, these protocols are inadequate for managing real-time traffic that can tolerate some data loss. MP-TCP emphasizes reliability and congestion control, making it unsuitable for applications like live video streaming that can accept occasional losses. Additionally, MP-RTP is limited by its bandwidth estimation methods and sensitivity to short-term fluctuations.

To address these limitations, MA-RTP has been developed as a new multipath transport protocol specifically designed for real-time, loss-tolerant data streams. As a contribution from Instituto de Telecomunicações to the *Floyd* project, MA-RTP was developed to support data-intensive sensor streams on the uplink, facilitating processing at edge nodes to reduce overall latency. MA-RTP enhances bandwidth estimation, traffic splitting, and connection management, thereby improving performance and efficiently utilizing network resources. This protocol provides a tailored solution for real-time data transmission, meeting the specific requirements of applications that can tolerate some data loss while maintaining low latency [10]. The stack of this protocol is represented in Figure 2.16.

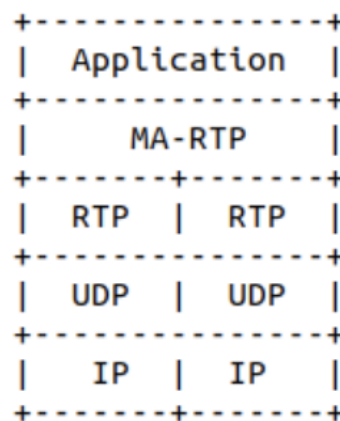


Figure 2.16: Stack of MA-RTP. Source: [10].

MA-RTP works in application layer on top of RTP, and adopts the same methodology of the others multipath protocols by establishing a multiple subflows in order to aggregate network interfaces. Furthermore, it incorporates an additional layer that provides a unified interface to the application.

The MA-RTP layer is responsible for managing multiple RTP sessions, making scheduling decisions, and monitoring available paths for data transmission. It utilizes RTCP reports from each RTP session to collect performance data at the application level, which assists in optimizing traffic splitting and connection management. These connections are illustrated in Figure 3.2. Additionally, MA-RTP can access data from an external database to further enhance traffic splitting strategies.

Following the discussion of MPTCP's foundational aspects, practical implementations of multipath transport protocols must be explored. OpenMPTCProuter (OMR) is a relevant tool designed to enable the aggregation of multiple network connections using MPTCP.

2.5 OpenMPTCProuter (OMR)

OMR is an open-source tool designed to establish a multipath tunnel between a router and a VPS. The router aggregates the available internet connections and tunnels the traffic to the VPS using MPTCP. For this setup to work, both the router and the VPS need to run an MPTCP-enabled kernel. The router can aggregate multiple network interfaces, such as Ethernet, Wi-Fi, and 4G/5G, creating subflows that are transmitted to the VPS. The VPS, in turn, manages the traffic distribution, reassembling packets and routing them to the internet, as shown in Figure 2.17. This architecture allows users to combine different internet connections, thereby optimizing bandwidth and ensuring redundancy.

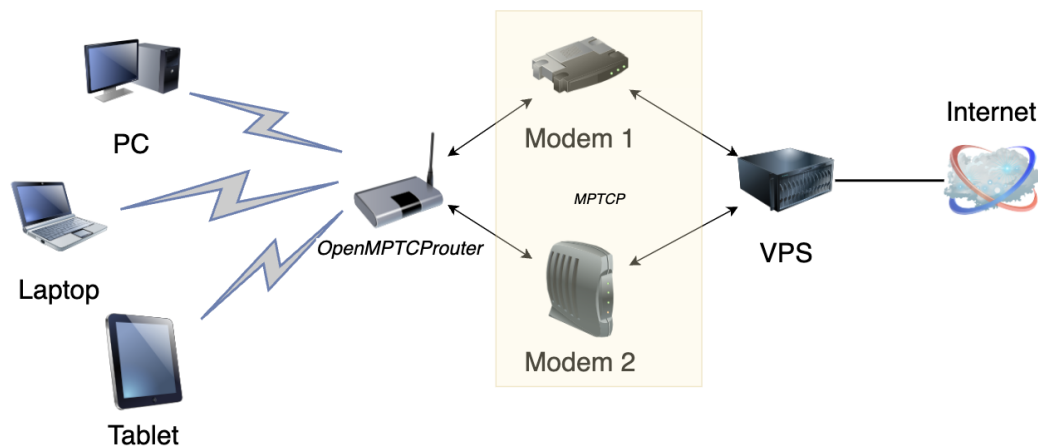


Figure 2.17: Illustration of the OMR operating mechanism. Representation of two points of the multipath tunnel, the router and the VPS. Two paths are used. The Internet connection is achieved after reaching the VPS. Source: [11].

The multipath transport capabilities of OpenMPTCProuter are supported by several essential software components, including OpenWRT, MPTCP, Shadowsocks, and Glorytun. These tools enable network configuration, the aggregation of multiple network connections, and efficient traffic handling with encryption and tunneling features. The following section examines the roles of these components within the OMR architecture.

2.5.1 Main Software Tools

OMR's architecture is built upon four primary components: OpenWRT (platform), MPTCP (core component), Shadowsocks, and Glorytun (supporting tools) [31]. Each component plays a specific

role in enabling the router to aggregate multiple network connections and manage data traffic efficiently. Figure 2.18 illustrates the software architecture of OMR.

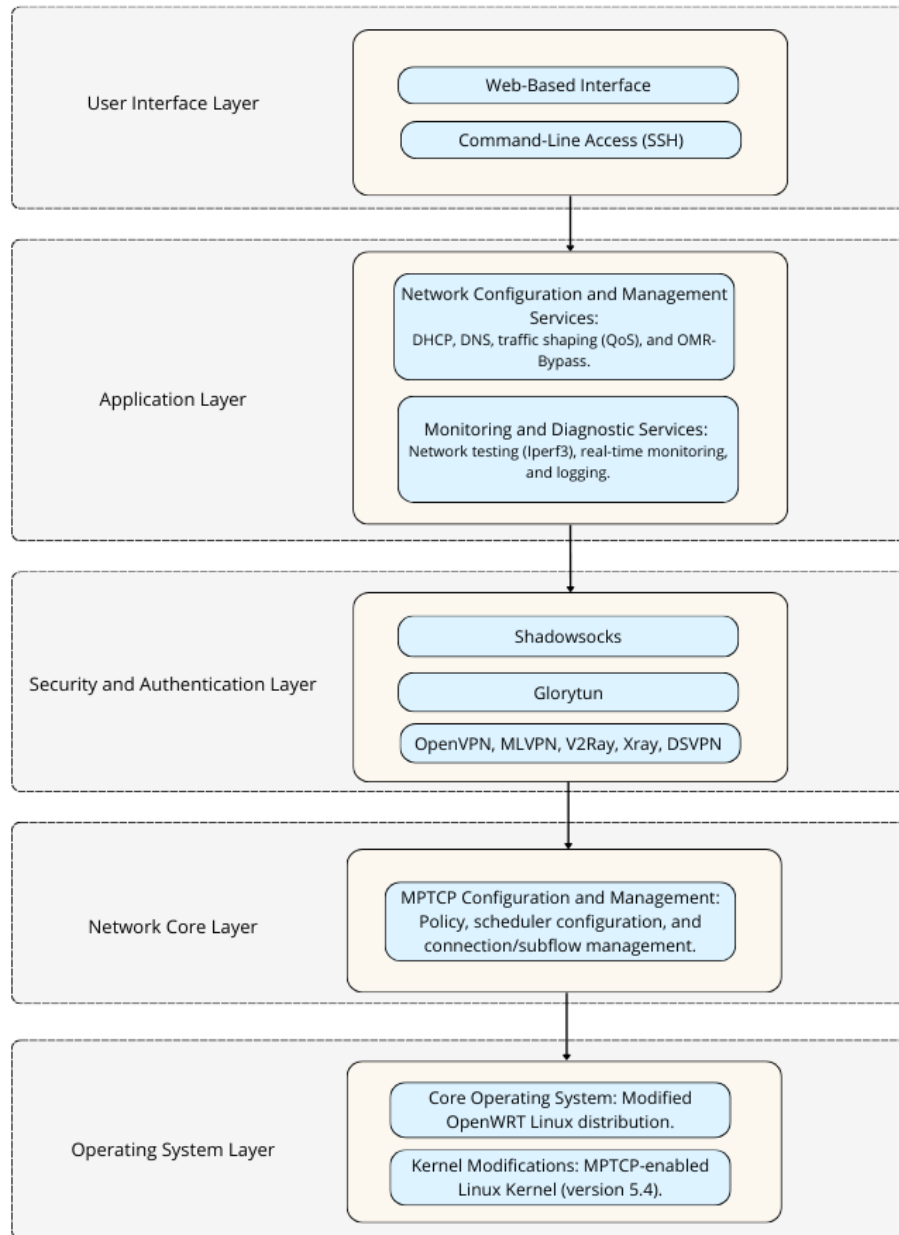


Figure 2.18: OMR software architecture.

OpenWRT is the base operating system on which OMR is built. It is an open-source Linux-based firmware specifically designed for embedded devices such as routers. OpenWRT provides a fully writable filesystem with package management, allowing customization and flexibility in configuring network interfaces and routing policies [32]. In the context of OMR, OpenWRT enables the management of multiple WAN interfaces, facilitating the addition and control of diverse network connections like Ethernet, Wi-Fi, and cellular networks. The modularity of OpenWRT allows for the installation of additional packages and tools required for MPTCP operation, such

as kernel modules and network utilities. By leveraging OpenWRT's capabilities, OMR can run on various hardware platforms, ensuring broad compatibility and adaptability to different network environments.

MPTCP acts as the core protocol that enables OMR to use multiple network paths within a single TCP connection. It does so by dividing user traffic into subflows, each of which can travel through different network routes to reach the VPS. The use of MPTCP requires both the router and the VPS to support an MPTCP-enabled kernel, highlighting the need for compatibility across all devices involved in the network setup.

Shadowsocks is employed as an encrypted proxy within OMR. Its primary function is to secure and obfuscate traffic to bypass network restrictions and maintain data privacy [33]. By encrypting the data and making it resemble regular HTTPS traffic, Shadowsocks ensures that the transmitted information is protected against interception and censorship.

Glorytun is integrated into OMR as a lightweight VPN tunneling solution optimized for aggregating multiple internet connections. It is capable of handling both TCP and UDP traffic efficiently, providing an additional layer of secure transport. Glorytun's role is to enhance the transport layer by establishing secure tunnels that can aggregate bandwidth from different sources. Its inclusion in the system enables the secure and effective transmission of data across multiple network paths [34].

2.5.2 Key Features

OMR offers several key advantages in managing multiple internet connections. It allows the aggregation and encryption of internet connections using the Multipath Transmission Control Protocol (MPTCP), effectively combining bandwidth from various sources like Wi-Fi, Ethernet, and cellular networks into a single virtual connection. This capability results in enhanced bandwidth, improved reliability, and seamless failover, as traffic can dynamically shift between active connections if one becomes unavailable.

One of the standout features of OMR is its ability to support different internet types, such as fiber, ADSL, VDSL, and 4G/5G, making it highly versatile for different network setups. OMR ensures consistent performance, even in high-latency environments, by distributing data streams across multiple network paths. Additionally, OMR can be easily deployed on various hardware platforms, including Raspberry Pi and x86-based systems, as well as on VPS servers using pre-built installation images.

OMR also integrates a web-based graphical interface that simplifies the configuration and management of MPTCP settings. This interface is supported by documentation. Furthermore, OMR's modularity allows it to incorporate additional technologies, such as Shadowsocks for secure proxying and Glorytun for UDP-based traffic management, providing a secure and adaptable network management solution [11].

This open-source also provides scheduling options that enhance its capacity to manage network traffic across multiple internet connections. These schedulers, mentioned in Section 2.3.3, allow the system to optimize traffic distribution based on various policies and conditions. By

leveraging the available schedulers, OMR can dynamically adapt its traffic management strategy, balancing performance, reliability, and redundancy according to the specific needs of the network environment.

Finally, OMR benefits from an active user community. This community provides support, shares configurations, and contributes to the platform's ongoing development.

2.6 State-of-the-Art

This section examines the implementation and performance of multipath transport protocols within 5G networks and their application in vehicular environments. It begins by discussing the role of multipath transport protocols in the ATSSS framework for eMBB and URLLC, emphasizing their ability to manage multiple network paths for improved throughput and reliability. The discussion then extends to the broader significance of these protocols in various settings, highlighting their impact on system performance in wireless environments. The section proceeds to illustrate how multipath transport protocols can enhance data transfer between connected vehicles. Subsequently, the focus shifts to the MPTCP protocol, detailing the design and configuration of a replicable system for evaluating MPTCP in mobility scenarios. The role of packet schedulers in optimizing MPTCP performance is analyzed, supported by studies on different scheduling algorithms. Finally, the section reviews existing performance evaluations of MPTCP in diverse conditions and distinguishes the current work.

The study "A Survey on Multipath Transport Protocols Towards 5G Access Traffic Steering, Switching and Splitting" by Wu et al. [1] addresses the integration of multipath transport protocols, specifically MPTCP and QUIC, within the 5G ATSSS framework, underlining their critical role in enabling the ATSSS functionalities of traffic steering, switching, and splitting. It provides an in-depth analysis of how these protocols can dynamically manage multiple network paths to optimize traffic distribution and resource utilization in a 5G context.

A standout feature of the paper is its comprehensive review of the current standardization efforts around these protocols and their potential modifications to better support the specific requirements of 5G networks. It examines the protocols' capabilities in handling path management, scheduling, congestion control, and reliable transfer across diverse network environments, including the interaction between 3GPP and non-3GPP networks.

Furthermore, the paper discusses the necessity for real-world evaluations of these protocols to truly understand their effectiveness in live network scenarios. It suggests that such evaluations are crucial for adapting these protocols to the dynamic and heterogeneous nature of 5G networks, where they must support a wide range of emerging and demanding applications like URLLC and eMBB.

The paper "The Past, Present, and Future of Transport-Layer Multipath" by Habib et al. [19] provides a comprehensive analysis of multipath communication strategies in network protocols, highlighting the evolution from traditional single-path transport control protocols to modern multipath approaches. A key focus is on the integration of multipath transport protocols such as

MPTCP and Stream Control Transmission Protocol (SCTP) within various network architectures, including mobile and data center environments. These protocols significantly enhance network efficiency by improving bandwidth utilization, reliability, and fault tolerance.

One of the unique contributions of this paper is its in-depth analysis of the challenges and solutions associated with implementing multipath transport protocols. It addresses issues like packet reordering, congestion control, and resource allocation across multiple paths, presenting a detailed discussion on how these protocols manage data flow and maintain stability in complex network conditions.

Furthermore, the paper explores the potential future directions for multipath transport protocols, especially in the context of emerging network demands such as 5G and cloud computing. It emphasizes the need for ongoing research to optimize these protocols for better performance and adaptability in heterogeneous network environments.

This work focuses on the MPTCP protocol, more specifically on designing and configuring a system, that can be replicated, so that this protocol can be evaluated in a mobility scenario. The document titled "A Brief Review of Multipath TCP for Vehicular Networks" by Chao et al. [35] explores the potential and challenges of implementing MPTCP in vehicular network environments. It discusses how MPTCP can significantly improve data transmission efficiency, effectively utilizing available bandwidth and enhancing resilience through multipath data transfers.

A key focus of the paper is on the suitability of MPTCP for vehicular Internet-of-Things (IoT) systems, which require reliable high-speed data transmission and robustness to support applications like cooperative autonomous driving. The paper details how MPTCP addresses the limitations of single-path TCP by better handling the dynamic and complex topologies typical in vehicular networks, thus offering improvements in throughput and network resilience.

This review stands out by providing a thorough exploration of the technical challenges involved in applying MPTCP in vehicular networks, such as handling diverse latency issues and ensuring data flow continuity amid the frequent topology changes inherent to high-mobility environments. The document also highlights the importance of selecting proper transmission paths to minimize interference and optimize performance, an aspect particularly critical in wireless vehicular communications.

In the context of optimizing MPTCP performance, schedulers play a pivotal role in managing how data packets are distributed across multiple sub-flows. Both Kimura et al., in their paper "Packet Scheduling in Multipath TCP: Fundamentals, Lessons, and Opportunities" [8], and Arzani et al., in "Impact of Path Characteristics and Scheduling Policies on MPTCP Performance" [36], emphasize that while MPTCP's joint congestion control mechanisms adjust sending windows based on network conditions, they are insufficient on their own to ensure optimal throughput. Their studies reveal that packet scheduling and path selection significantly influence MPTCP efficiency, especially in heterogeneous network environments where sub-flows have varying RTTs and bandwidths. Effective scheduling decisions can enhance throughput by leveraging the aggregated capacities of available sub-flows, reducing waiting times for out-of-order packets, and minimizing issues like head-of-line blocking and bufferbloat. Conversely, poor scheduling choices can negate

these benefits. These studies underscore the importance of intelligent packet scheduling, which is essential for optimizing MPTCP performance in heterogeneous networks by accounting for path characteristics like RTT and bandwidth.

In the paper "Performance of Multipath TCP Schedulers in Concurrent Use of 5G and 4G Networks," authors Imtiaz Mahmud and You-Ze Cho [24] evaluated various MPTCP schedulers (ECF, Redundant, BLEST, Round-Robin) using NS3-DCE simulations in a scenario where a UE concurrently utilizes 5G and 4G networks at 60 km/h. Their study found that BLEST outperformed others by effectively preventing head-of-line blocking, thereby enhancing throughput and reducing flow completion time (FCT). In contrast, Redundant increased overhead without proportional benefits, ECF struggled with fluctuating 5G throughput, and Round-Robin offered moderate performance but lacked adaptability to dynamic conditions. The findings underscore the importance of intelligent scheduling in heterogeneous and mobile network environments.

Regarding MPTCP's performance, the paper titled "MPTCP Performance over Heterogeneous Subpaths" by Vivek Adarsh, Paul Schmitt, and Elizabeth Belding [37], explores the performance implications of implementing MPTCP over heterogeneous network paths, such as Wi-Fi and cellular connections. The study addresses significant performance variances between different network types, influencing the primary interface selection for DNS resolution and the overall effectiveness of MPTCP.

The researchers conducted two main types of experiments: controlled lab experiments and real-world tests. In the controlled lab experiments, the research team manually adjusted network latency and loss conditions to simulate different network environments. This allowed for precise control and measurement of MPTCP's behavior under varied conditions. For the real-world tests, they replicated these lab conditions in typical user environments to see how MPTCP performs outside of controlled settings, focusing on how path heterogeneity affects throughput and connectivity. These experiments reveal that selecting cellular networks as the primary interface for DNS resolution can lead to suboptimal performance due to higher latency and lower IP reachability compared to Wi-Fi. The main conclusions suggest that heterogeneous network paths can negatively affect MPTCP's ability to improve connectivity and throughput, especially when one path has significant losses in relation to the other.

Although providing valuable insights, these studies did not assess MPTCP in a real-world scenario that simultaneously involved outdoor conditions and mobility and this research is distinctive as it evaluates MPTCP's performance in a that scenario, using key metrics such as throughput, packet loss, and latency. This project involved designing a system composed of several subsystems to implement and evaluate MPTCP performance using specific metrics. This includes real-time data visualization enabled by GPS technology. Additionally, a detailed tutorial has been created to guide the configuration of the system, ensuring it can be replicated.

Chapter 3

Design and Implementation of an MPTCP Evaluation System

This chapter outlines the design and implementation of a system to evaluate the performance of Multipath Transmission Control Protocol (MPTCP) within the context of connected driving. It reviews previous research on the Multimedia Aggregation Real-Time Transport Protocol (MA-RTP) and highlights the necessity of conducting a benchmark. MPTCP was chosen as the protocol for this benchmark due to its status as the state-of-the-art in multipath protocols. Benchmark metrics including latency, packet loss, and throughput are clearly defined to ensure a consistent evaluation approach. The experimental setup is comprehensively detailed, describing the system architecture, configurations for Wi-Fi and 5G networks, and hardware specifications. Implementation limitations are also discussed. Furthermore, the configuration of system components and the measurement subsystem are elaborated upon, including the synchronization of time using the Network Time Protocol (NTP), the integration of Global Positioning System (GPS) data, and the methods for data processing used to compute performance metrics. This foundational work sets the stage for the subsequent evaluation and analysis of MPTCP's performance.

3.1 Problem Definition and Methodology

This section outlines the core objectives and methodological approach of the project, detailing the problem being addressed and the specific strategies employed to achieve the desired outcomes. The methodology is designed to ensure a thorough evaluation of the proposed solutions within the context of real-world use cases, leveraging advanced technologies and protocols to address key challenges. Additionally, the section draws upon prior research and establishes the metrics and benchmarks that will be used to assess the effectiveness of the solutions.

3.1.1 Context and Use Case

This project is part of the Route25 initiative, specifically within Work Package 2 (WP2): Digital Experience and Adaptive Connectivity for Cooperative Mobility. It focuses on enhancing road

safety for vulnerable road users (VRUs), such as pedestrians and cyclists, by deploying Cooperative Perception systems. These systems enable vehicles, roadside sensors, and mobile devices to collaboratively share and process their sensor data in real time, thereby creating a comprehensive and precise environmental view that surpasses the capabilities of individual entities.

Given the dynamics of the system, many nodes, such as vehicles and bicycles, use video cameras that generate substantial data. Processing this data directly on mobile platforms is computationally intensive and potentially slow. To address this challenge, the project aims to leverage edge computing for processing video data, which necessitates rapid data transfer to edge nodes. This setup is expected to facilitate significant uplink flows, characterized by the specific demands of video data transmission.

The primary use case of this work involves transmitting large video flows via uplink to edge computing nodes. These nodes are equipped to swiftly process computer vision algorithms that are essential for detecting objects in traffic scenes, thus enhancing the efficiency and responsiveness of the Cooperative Perception systems.

Moreover, the project explores multi-network connectivity by utilizing both WiFi and 5G communication technologies. This dual-network approach aims to optimize data transmission paths, balancing factors such as bandwidth availability, signal strength, and network reliability. The a more concrete use case is illustrated in Figure 3.1.

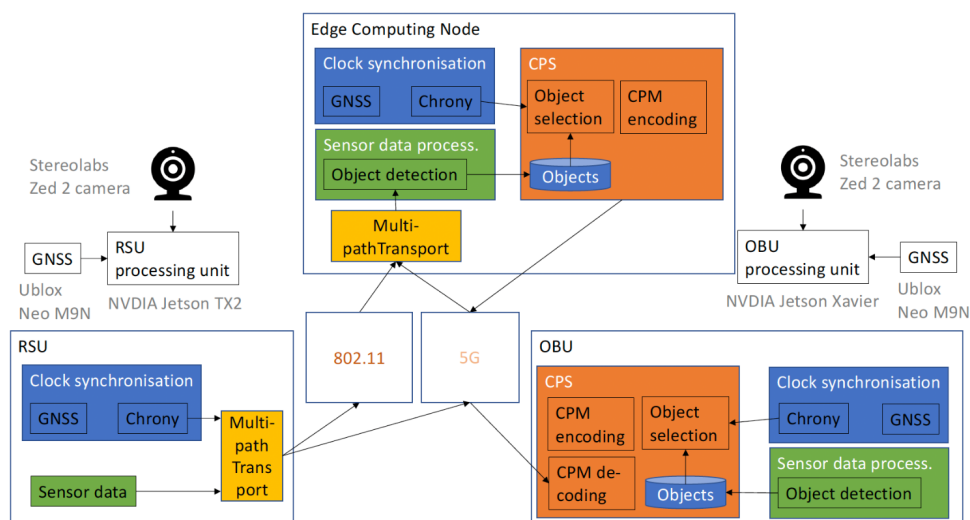


Figure 3.1: Use Case Scenario. This work is responsible for implementing multipath transport (highlighted in yellow) using Wi-Fi and 5G for data transmission.

Central to this effort is the development and evaluation of multipath transport protocols such as MP-TCP and MA-RTP. These protocols are designed to manage the distribution of video traffic across diverse access networks efficiently. The implementation of advanced packet scheduling algorithms plays a crucial role in this context. Specifically, the project aims to minimize latency and packet loss, thereby enhancing the overall performance and reliability of Cooperative Perception services. Furthermore, these algorithms help in relieving the load on cellular networks by making the "bonded" wireless connection more resilient to fluctuations in signal quality. This resilience

is critical for ensuring the timely arrival of video frames at the edge node, which is essential for maintaining the integrity and effectiveness of real-time video applications.

In the experimental setup of this work, the system is designed to simulate these real-world conditions. A mobile node, representing a VRU, transmits video data over multiple network paths, to a video receiving service hosted at an edge computing node. By integrating multipath transport protocols and scheduling algorithms, the project aims to guarantee that the video frames arrive on time for processing at the edge node.

3.1.2 Previous work

In the context of this project, prior work focused on evaluating the MA-RTP protocol as a solution for the efficient transmission of real-time video streams over multiple network paths, namely WiFi and 5G. This work aimed to explore how MA-RTP could be leveraged to improve road safety by facilitating real-time Cooperative Perception services, particularly for vulnerable road users such as pedestrians and cyclists.

Results from these experiments demonstrated that by leveraging MA-RTP, video traffic could be efficiently distributed between WiFi and 5G, significantly reducing delays at the application level [10].

In summary, the previous work established a basis for the current emphasis on evaluating MPTCP. This effort aims to provide a benchmark for MA-RTP, rather than enhancing its capabilities. The current phase of the project builds on these findings by comparing the effectiveness of MPTCP for real-time video transmission under similar conditions to those addressed by MA-RTP.

3.1.3 Benchmark Metrics

For the evaluation of both MA-RTP and MPTCP, it is crucial to maintain a consistent experimental setup and methodology to ensure comparable results. Similar to the MA-RTP evaluation, the setup involves a sender transmitting video data over Wi-Fi and 5G to an edge computing node [10]. Several packet scheduling algorithms will be applied, and the same performance metrics, such as latency, packet loss, and throughput, will be used to assess the performance of both protocols. However, there is a key difference: MA-RTP was evaluated at the application level, whereas MPTCP will be assessed at the transport layer at the endpoints of the multipath tunnel.

These metrics are necessary to evaluate the performance of each protocol in dynamic environments. Latency measures the delay in video data transmission to match the rate at which data is generated, essential for real-time video streams. Packet loss assesses how often data fails to reach its destination, and throughput measures the speed of data transmission. By maintaining consistent benchmarks across both protocols, the project will provide a clear comparison of their respective capabilities in transmitting video sensor streams. The evaluation focuses solely on the performance of MP-TCP for this purpose.

3.2 Experimental Setup

The goal of the experimental setup is to evaluate an uplink scenario where a mobile node, acting as the sender, transmits data in the form of a fragmented video. The video is sent through the multipath gateway, which uses the MPTCP tunnel to access the internet and deliver the data to a receiver outside the mobile node's network.

Furthermore, the experiments, aimed at comparing the performance of MPTCP with MA-RTP, will be conducted outdoors at a designated single location. This setting requires that the devices used for network access and MPTCP aggregation be portable to accommodate the outdoor environment. The experimental setup for both MPTCP and MA-RTP will be identical to ensure comparable results.

This section outlines the experimental setup, including any modifications implemented to facilitate MPTCP. It also includes detailed specifications of the components used, emphasizing the importance of their compatibility and functionality in an outdoor setting. Challenges such as limited electricity and device-software compatibility are addressed to maintain the reliability of the implementation.

3.2.1 Architecture

The test environment is configured with an MPTCP tunnel connecting the multipath gateway to a Virtual Private Server (VPS), forming the endpoints of the tunnel. This tunnel is managed by the OpenMPTCProuter (OMR) software, which divides the data stream into separate TCP subflows and transmits them via MPTCP. In this setup, MPTCP combines two networks: Wi-Fi and 5G, so there are two TCP subflows. The installation and configuration of the multipath gateway and VPS are described in Section 3.3.1.

The system employs client applications on the mobile node (sender) and server applications on a server (receiver) to manage video data. The sender leverages TCP sockets, facilitating the multipath gateway's use of MPTCP to aggregate network connections efficiently. Similarly, the receiver utilizes TCP sockets to receive the video data. The software used for the setup is illustrated in Figure 3.2.

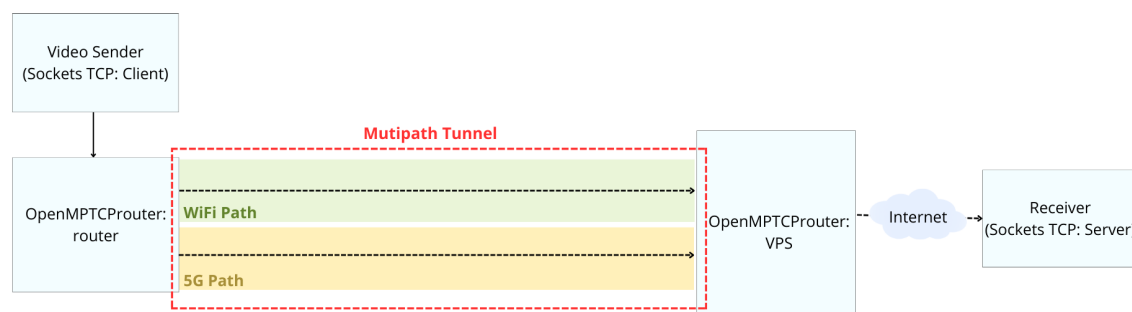


Figure 3.2: Software components used for the MPTCP evaluation experimental setup.

Once the software is understood, the appropriate hardware must be selected. The OMR software will be installed on a Raspberry Pi (RPI), enabling it to act as an MPTCP-capable multipath gateway. The RPi was chosen for its portability, which is essential for experiments involving mobility. The VPS will run on a Google Cloud Compute Engine virtual machine (VM), selected for its user-friendly interface and free trial availability. The mobile node, responsible for generating and sending traffic, will be a desktop, selected for its mobility during experiments. The server receiving the traffic will be hosted on an Amazon Web Services (AWS) VM. The hardware configuration is illustrated in Figure 3.3.

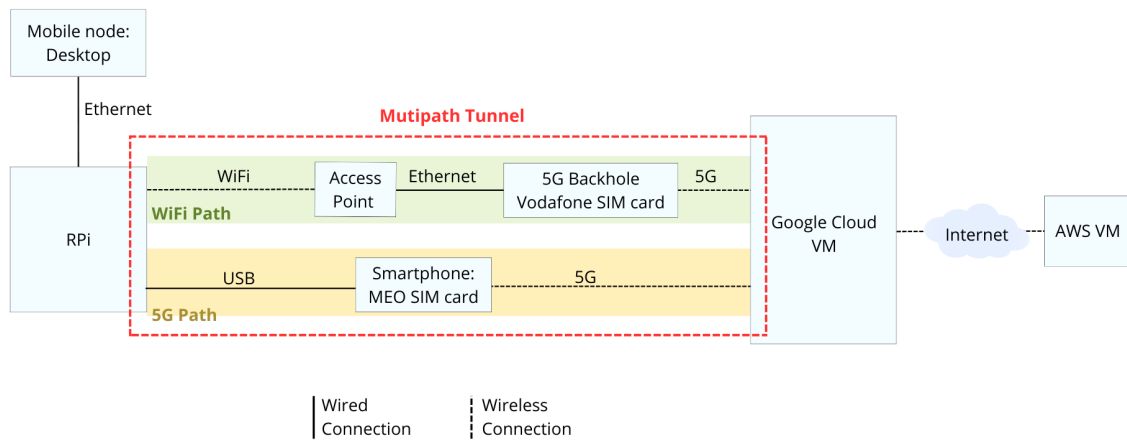


Figure 3.3: Hardware components used for the MPTCP evaluation experimental setup.

Once the MPTCP setup is operational, the next step is to determine how to collect data. To evaluate signal quality, key metrics such as throughput, latency, and packet loss will be measured. Packet data will be captured using `tcpdump`, running on both the RPi terminal and the Google Cloud VM terminal. This approach provides data from both the sending and receiving ends of the communication.

In addition to monitoring these metrics over time, understanding their variation over space is also crucial. To achieve this, a GPS antenna was integrated into the experimental setup to record GPS coordinates.

The GPS antenna is connected to the RPi via USB to log location data during the experiment. To ensure continuous operation of the RPi during movement, a power bank was included to provide a reliable power source. A laptop serves as the interface for establishing SSH connections to both the RPi and the server, allowing control, configuration, and execution of the experiments. Additionally, a power extension was added to supply power to the access point (AP) and the 5G HAT. The newly incorporated component, along with the locations where captures were conducted, is depicted in Figure 3.4.

It is important to note that the VPS is physically separate from the receiver. If both the receiver and the VPS were hosted on the same VM, the setup would not function correctly. This means, during packet capture on the VPS using `tcpdump`, only standard TCP packets were observed, with

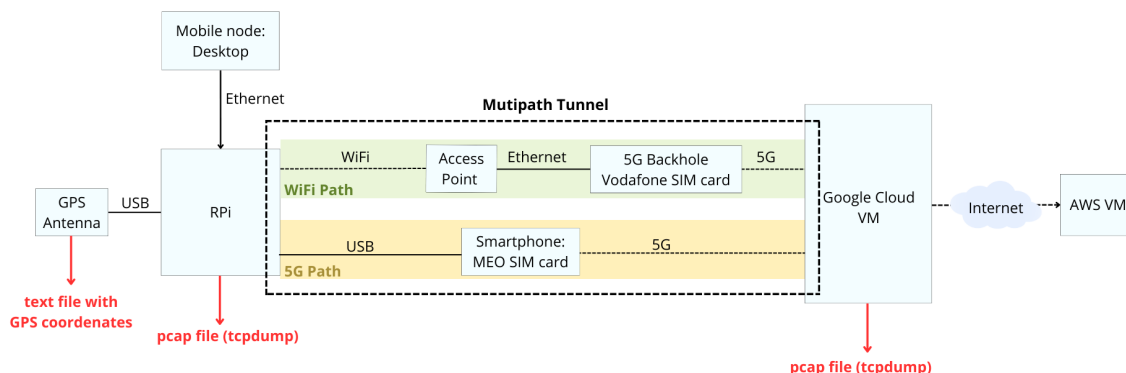


Figure 3.4: Integration of GPS antenna into the experimental setup and representation of data files collected during experiments.

no visibility of MPTCP packets. This issue arises due to OMR’s deep integration with the Linux kernel, which manages MPTCP traffic.

OMR modifies the network stack, blocking traffic on ports not involved in its core operations. Although the VM’s firewall rules allow all incoming traffic, the VM does not listen on ports unrelated to OMR’s internal processes. As a result, the only way to send packets to the VPS is through the internal operation ports, mentioned in Section 3.3.1, which prevents full utilization of MPTCP’s multipath features. Consequently, incoming packets were treated as standard TCP rather than MPTCP.

Thus the right solution is separating the target server from the VPS. The VPS remained on the Google Cloud VM, while the target server was moved to an external AWS instance. After this adjustment, tcpdump on the VPS successfully captured MPTCP packets from OMR, while the packets forwarded to the AWS server appeared as TCP. By separating the destination from the VPS, MPTCP subflows were properly established, allowing full use of MPTCP’s multipath capabilities and enabling tcpdump to capture MPTCP packets as they arrived at the VPS.

Figure 3.5 presents a real-world perspective of the complete experimental setup. The photograph on the left illustrates the mobile components, which include the multipath gateway, desktop, GPS unit and external GPS antenna, smartphone (providing the 5G path), and a power bank. The photograph on the right displays the stationary components that remained fixed during the experiments. These components consist of the Wi-Fi subsystem (AP and 5G HAT), which establishes the Wi-Fi path to the multipath gateway.

3.2.2 Wi-Fi Path

Regarding the Wi-Fi path, the initial setup used the Wi-Fi network provided by the Faculty of Engineering of the University of Porto (FEUP). An access point (AP) was connected to the FEUP network via an Ethernet cable, similar to the configuration used in the MA-RTP evaluation. However, the FEUP network’s routers do not support MPTCP. As a result, MPTCP packets were downgraded to standard TCP packets, leading to the loss of MPTCP functionality. Additionally, the

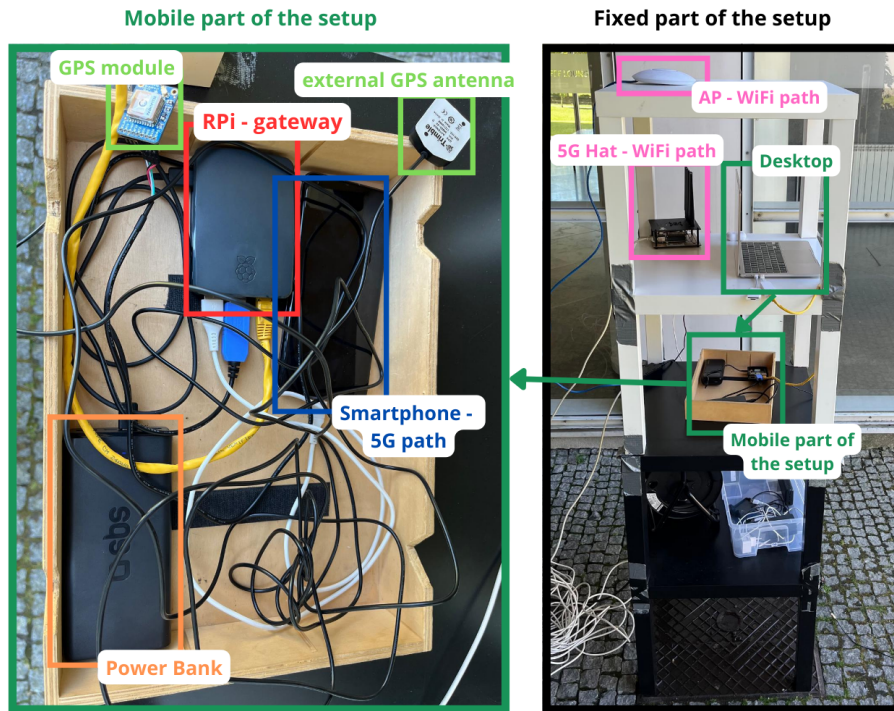


Figure 3.5: Photograph of mobile components on left and fixed components of setup on right.

server acting as the video receiver in the MA-RTP experimental setup was originally within the FEUP network.

To get around this routers limitation and maintain full MPTCP capacity, the VPS and the receiver were moved outside of this network, and a Wi-Fi network was implemented. This new network, is made up of an AP connected to a 5G router, constituting a subsystem, guaranteeing an independent and fully functional network environment, free from the restrictions of FEUP's routers. In this document, this subsystem is referred to as the Wi-Fi path subsystem.

The AP creates a local wireless network, with the 5G network serving as its backhaul, facilitated by the 5G Hardware Attached on Top (HAT) module (represented in Figure 3.6). A 5G SIM card is inserted into this module to enable 5G connectivity. The 5G HAT is a hardware accessory designed to deliver 5G backhaul support to the WiFi network.

This device comprises the following components:

- **5G HAT (with a module):** HAT provides cellular connectivity by using a module, allowing a device to connect to 4G or 5G networks. The module component delivers 5G connectivity by managing the communication with the 5G network. It connects with the antennas and is responsible for data transmission and reception and includes the necessary radio frequency (RF) components and circuits.
- **Antennas:** Multiple antennas are attached to the HAT, which is used for receiving and transmitting 5G signals. They ensure strong and stable connectivity with the cellular network. They are high-gain, omnidirectional antennas.

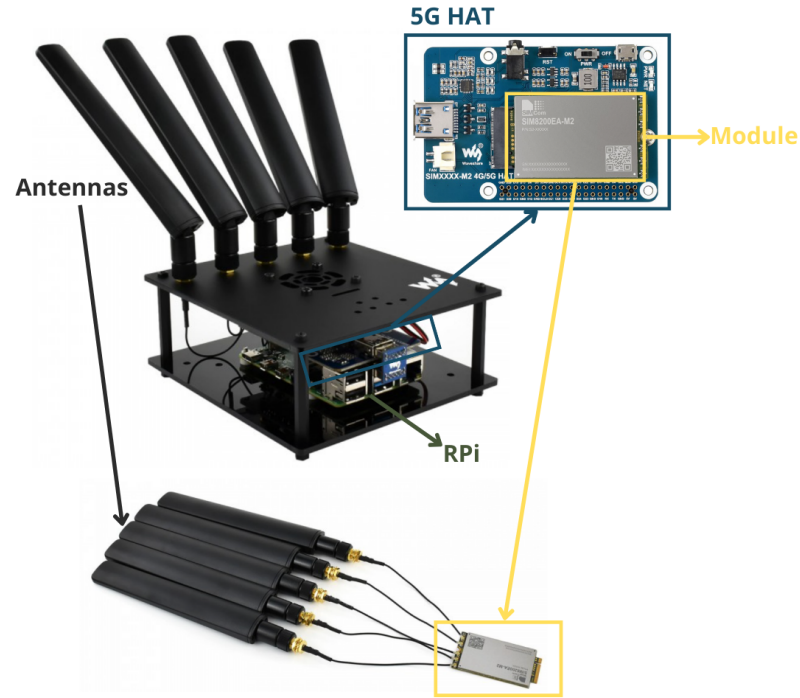


Figure 3.6: 5G router and its constituent parts. Adapted from source: [12].

- **RPi:** Serving as the main processor, the RPi interacts with the 5G module. It manages data processing, application execution, and communication tasks. The 5G HAT connects to the RPi through GPIO pins or other connectors.

3.2.3 5G Path

For the 5G path, the RPi accesses the 5G network, via a smartphone equipped with a 5G SIM card. The smartphone shares its mobile data connection with the multipath gateway via a USB connection, thus providing access to the 5G network.

3.2.4 Specification of Components

The specifications of each device, including the desktop computer, RPi, power bank, smartphone, GPS module, and external GPS antenna, are described to provide a clear understanding of the system's capabilities and requirements. Below is a detailed description of the specifications of each component.

- **Desktop:** The desktop is a MacBook Air M2, 2022, running macOS version 14.3.1 (23D60). It is equipped with an Apple M2 processor and has 16 GB of RAM.
- **RPi (multipath gateway):** The RPi 4 Model B Rev 1.1 has a quad-core Cortex-A72 (ARMv8) 64-bit CPU, 2GB RAM, dual-band Wi-Fi (2.4 GHz and 5 GHz), and needs a 5V USB-C power supply with at least 3A current.

- **OMR:** Image version used is v0.59.1 with kernel version is 5.4 (it must correspond to the VPS kernel installed in Google Cloud VM) for schedulers purpose. This version uses legacy MPTCP, version 0.96, which corresponds to an out-of-tree release (differing from the upstream MPTCP version used in the kernel 6.1 setup). The firmware implemented on the router is based on the OpenWrt framework v21.02.
- **Power Bank:** The Energizer 22.5 Power Bank has a 20,000 mAh capacity and features two USB-A ports (5V-3A output) and one USB-C port (20W fast charging). The USB-A port used for the RPi provides 5V-3A, meeting the RPi's power requirements.
- **Smartphone:** The Xiaomi Mi 10 5G has 8GB of RAM, an octa-core processor, and runs on Android 11.
- **GPS Antenna:** The Adafruit Ultimate GPS module offers high-precision GPS with -165 dBm sensitivity, a 10 Hz update rate, and tracks up to 22 satellites. It supports NMEA 0183 output at 9600 baud, operates on 3.0-5.5VDC with a 20 mA current draw, and includes an internal patch antenna, built-in data logging, and a status LED.
- **GPS External Antenna:** The external active GPS antenna operates on 3-5V, provides 28 dB gain, and comes with a 5-meter cable and SMA connector. It's compatible with Ultimate GPS modules and improves sensitivity in restricted or shielded environments.
- **Access Point:** The Ubiquiti Unifi AC Lite (UAP-AC-LITE) supports Wi-Fi 802.11ac with speeds up to 300 Mbps on 2.4 GHz and 867 Mbps on 5 GHz. It features 2x2 MIMO, two 3 dBi dual-band antennas, one Ethernet port, and a reset button. Its maximum TX power is 20 dBm, with a range up to 122 meters.
- **Wi-Fi Backhaul:**
 - **5G HAT:** The HAT for RPi with the SIM8200EA-M2 module supports 5G NSA and SA networks, offering speeds up to 4 Gbps downlink and 500 Mbps uplink. It includes a SIM card slot, connects via a 40-pin GPIO header, and operates on Sub-6G, LTE, and WCDMA bands. It requires a 5V 3A power supply and has six IPEX-4 antenna connectors. It connects to the RPi using a USB 3.0 adapter [12].
 - **Antennas:** High-gain, omni-directional antennas suitable for 4G/5G frequencies. SMA connectors to attach them to the HAT. 5 antennas to support MIMO (Multiple Input Multiple Output) technology.
 - **RPi:** RPi 4 Model B Rev 1.1 has a quad-core Cortex-A72 (ARMv8) 64-bit CPU, 2GB RAM, dual-band Wi-Fi (2.4 GHz and 5 GHz), and needs a 5V USB-C power supply with at least 3A current.
- **Google Cloud VM (VPS):** To run kernel 5.4, we selected a Google Cloud VM with Debian 11 Bullseye x86_64, as this kernel version is not compatible with Debian 12 Bookworm.

Although Debian 11 Bullseye defaults to kernel 5.10—which is closer to kernel 5.4 than Debian 12’s kernel 6.1—kernel 5.4 is not included by default in Debian 11 and required manual installation, which is done during the VPS installation. This step is detailed in Section 3.3.1. The VM is located in the europe-west2-b zone (London) and uses an e2-medium machine type with Intel Broadwell CPUs. It has 2 CPUs (Intel Xeon @ 2.20 GHz), 3.8 GiB of memory, and no swap space. The internal IP is 10.154.0.2, and the temporary external IP is 34.142.4.89.

- **AWS VM:** The AWS EC2 instance has a public IPv4 address of 3.8.132.141 and a private IPv4 address of 172.31.45.141. The instance type is t2.micro, running Amazon Linux, using AMI ID ami-0fe310dde2a8fdc5c.

Once the setup is defined, all components must be configured to function together as required. As outlined previously, the setup includes subsystems, and each one must be configured to ensure that the final system can effectively implement and evaluate MPTCP.

3.3 Setup Components Configuration

This section details the configuration process for all subsystems. All commands necessary for these configurations are listed in Appendix A.

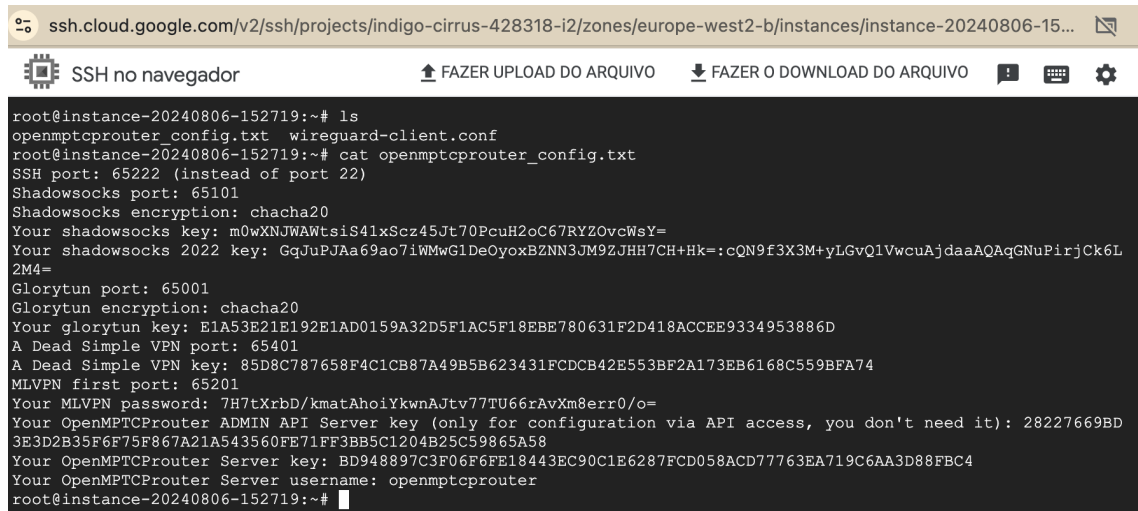
3.3.1 Multipath Tunnel: VPS and Router

As detailed in Section 3.2.1, OMR was installed on a RPi to act as a multipath gateway, being one of the endpoint of the multipath tunnel. The other tunnel endpoint, the VPS, was deployed on a Google Cloud VM.

Google Cloud was selected due to its free trial offering for VMs and its straightforward SSH access configuration, particularly useful since the OMR script changes the default SSH port from 22 to 65222 during installation. To set up the VPS, on the VM terminal, a command was executed to download and run a script from <https://www.openmptcprouter.com/>. This command installs the kernel 5.4. Upon installation, encryption keys required for data security were generated and stored in a file on the VM, as illustrated in Figure 3.8.

The script installs and configures the MPTCP kernel, Shadowsocks, Glorytun, explained in Section 2.5.1, and Shorewall (as a firewall). It also generates keys for Shadowsocks and Glorytun. OMR utilizes the following internal ports [38]:

- SSH: 65222 (TCP)
- Shadowsocks: 65101 (TCP and UDP)
- Shadowsocks-Rust: 65280 (TCP and UDP)
- Glorytun: 65001 (TCP and UDP)



```

root@instance-20240806-152719:~# ls
openmptcprouter_config.txt  wireguard-client.conf
root@instance-20240806-152719:~# cat openmptcprouter_config.txt
SSH port: 65222 (instead of port 22)
Shadowsocks port: 65101
Shadowsocks encryption: chacha20
Your shadowsocks key: m0wXNJWAWtsiS4lxScz45Jt70PcuH2oC67RYZOvcWsY=
Your shadowsocks 2022 key: GqJuPJAA69ao7iWMwG1DeOyoxBZNN3JM9ZJHH7CH+Hk=:cQN9f3X3M+yLGvQ1VwcuAjdAAQAqGNuPirjCk6L
2M4=
Glorytun port: 65001
Glorytun encryption: chacha20
Your glorytun key: E1A53E21E192E1AD0159A32D5F1AC5F18EBE780631F2D418ACCEE9334953886D
A Dead Simple VPN port: 65401
A Dead Simple VPN key: 85D8C787658F4C1CB87A49B5B623431FCDCB42E553BF2A173EB6168C559BFA74
MLVPN first port: 65201
Your MLVPN password: 7H7tXrbD/kmatAhoiYkwnAJtv77TU66rAvXm8err0/o=
Your OpenMPTCProuter ADMIN API Server key (only for configuration via API access, you don't need it): 28227669BD
3E3D2B35F6F75F867A21A543560FE71FF3BB5C1204B25C59865A58
Your OpenMPTCProuter Server key: BD948897C3F06F6FE18443EC90C1E6287FCD058ACD77763EA719C6AA3D88FBC4
Your OpenMPTCProuter Server username: openmptcprouter
root@instance-20240806-152719:~#

```

Figure 3.7: Keys generated by the script, required for router configuration.

- OMR JSON Admin: 65500 (TCP)
- OpenVPN: 65301 (TCP)
- MLVPN: 65201-65208 (UDP)
- Iperf3: 65400 (TCP and UDP)
- DSVPN: 65401 (TCP)
- V2Ray: 65228, 65229, 65230, 65231 (TCP)
- Xray: 65248, 65249, 65250, 65251, 65252, and 443 when VLESS-Reality is used (TCP)
- WireGuard: 65311, 65312 (UDP)
- ICMP traffic should not be blocked

For simplicity, the VM firewall was configured to allow TCP and UDP traffic on ports 65000 to 65535, and ICMP packets were also allowed. Following these configurations, the VM was rebooted. The relevant commands can be found in Section A.1.1 of the appendix.

After the VPS installation and key extraction, the multipath gateway was installed on the RPi 4. A precompiled image was downloaded from <https://www.openmptcprouter.com/> and copied to an SD card using the Etcher graphical interface. The multipath gateway image version used for the experiments, as referred to in Section 3.2.4, was *openmptcprouter-v0.59.1-5.4* (where v0.59.1 denotes the OMR version, and 5.4 refers to the kernel version, which must correspond to the VPS kernel).

The RPi is running OMR firmware v0.59.1-5.4, based on the OpenWrt framework and optimized for the RPi 4's 64-bit ARM architecture. It uses the Linux 5.4 kernel and the opkg package

management system, unlike Debian-based distributions like Raspbian or Ubuntu, which use apt-get. Once these steps were completed, the multipath gateway was installed on the RPi 4 and became accessible at IP address 192.168.100.1, along with its graphical interface.

The next step is the configuration, which involves accessing the graphical interface and entering the keys in the designated fields. Following this, the WLAN (for the Wi-Fi path) and USB (for the 5G path) interfaces were configured. Once configured, the OMR system was operational. If everything is functioning correctly, the general system status will appear as shown in Figure 3.8.

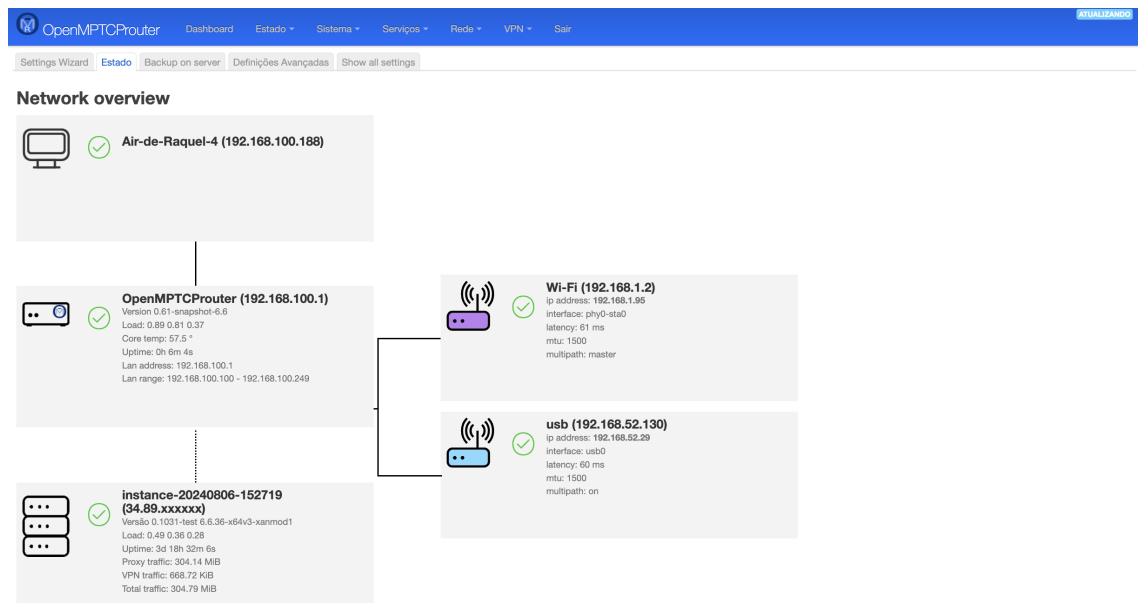


Figure 3.8: General system status when everything is functioning correctly. The Air-de-Raquel is the desktop used, OMR is the RPi that is running the multipath gateway, the Wi-fi is the wifi path, USB is the 5G path, and the instance is the VPS running on the Google Cloud VM.

MPTCP was configured to use full-mesh for path management and the BBR algorithm for congestion control, explained in Section 2.3.4. Additionally, all schedulers available in kernel version 5.4 were tested, including Round Robin, Redundant, Default, ECF, and BLEST.

3.3.2 Wi-Fi Subsystem

The initial step involved configure the *UniFi* AP, i.e. to make it create a network, the *UniFi* mobile application (app) was used. This app makes it easy to configure the network created by the AP and requires only one configuration. To begin with, the AP must be connected to a network via Ethernet, and the mobile device running the *UniFi* app must be connected to the same network. This allows the application to effectively detect, associate with, and configure the AP.

The next step is configuring the SIM8200EA-M2 module on the RPi, enabling network communication on it. To configure the module, it must first be connected to an RPi via USB, and then it is necessary to check that the module is recognized by the operating system and appears as a USB device, with its associated serial interfaces listed in the device directory.

After confirming the connectivity of the hardware, the next step is to configure the software environment. A Python script was used to communicate with and configure the SIM820X modules by sending a series of AT (Attention) commands over a serial connection. The pyserial library was installed to enable serial communication between Python and the device.

The commands sent by the script perform specific functions are the following:

- **Firmware Check:** The `AT+SIMCOMATI` command retrieves the firmware version of the module, which helps verify the module's software state.
- **Signal Strength Query:** The `AT+CSQ` command queries the signal strength, providing information about network signal quality.
- **Network Status Retrieval:** The `AT+CPSI?` command retrieves current network status, including network registration information.
- **Network Mode Configuration:** The `AT+CNMP=2` command sets the network mode, configuring the module to operate in the desired network mode, such as LTE.

Furthermore, as mentioned in Section 3.2.1, connectivity to the 5G network is established through the SIM card inserted into the 5G HAT module. To unlock the SIM card, Minicom, a terminal emulator and text-based modem control program that allows communication with devices via a serial interface, was used. Minicom was employed to send the command for entering the SIM PIN and to confirm that the card had been successfully unlocked.

Table 3.1 summarises all the AT commands used in both the Python code and *minicom*. All the required commands sequence for this process is detailed in Section A.2.1 of the appendix.

Table 3.1: Summary of *minicom* commands and their functions.

Command	Description
<code>AT+SIMCOMATI</code>	Retrieves the version information of the module.
<code>AT+CSQ</code>	Queries the signal quality, including Received Signal Strength Indicator (RSSI) and Bit Error Rate (BER).
<code>AT+CPSI?</code>	Inquires the system information of the User Equipment (UE), such as network status.
<code>AT+CNMP</code>	Sets the preferred network mode (e.g., 2G, 3G, 4G).
<code>AT+CUSBCFG</code>	Configures USB settings such as USB ID, ADB mode, EDL mode, and bootloader mode.
<code>AT+CPIN</code>	Sends a PIN code to unlock the SIM or other security features (e.g., SIM PIN, SIM PUK).

The next step was to configure the RPi so that, when connected via Ethernet to the AP, it would allow the AP to provide internet access to the devices connected to it via wireless, over the 5G network via the 5G HAT. This configuration involved routing packets between the AP (which is connected to the Ethernet interface, `eth0`) and the 5G HAT module (connected to the USB interface).

To enable this, the RPi was configured for IP forwarding, which is necessary to route traffic between its different network interfaces. Without IP forwarding, the RPi would not pass network traffic between the AP and the 5G network, thus preventing devices connected to the AP from accessing the internet.

Network Address Translation (NAT) was then implemented using `iptables` to allow packets from the AP (via `eth0`) to be routed to the 5G network (via `usb0`) and to ensure that return traffic from the 5G network is properly routed back to the AP. NAT modifies the source IP addresses of outgoing packets to maintain the correct routing of responses.

Finally, the network interfaces were configured. A static IP address was assigned to the Ethernet interface (`eth0`) to ensure a fixed address for the AP, and `dnsmasq` was set up to manage Dynamic Host Configuration Protocol (DHCP), enabling the AP to assign IP addresses to devices connecting to it. These configurations ensure that the AP provides WiFi access, with network traffic routed through the 5G network via the RPi as the backhaul. The specific commands that ensure the RPi can correctly route traffic through the 5G HAT shield to establish a connection to the 5G network are provided in the appendix Section A.2.2.

3.4 Measurement Subsystem

To evaluate MPTCP and compare it with MA-RTP, a quantitative system performance assessment was conducted, focusing on throughput, latency, and packet loss. The traffic under consideration in these experiments consists of real-time data flows, as MPTCP and MA-RTP are tested in a video streaming scenario. This section details the collection of the necessary variables during the experiments and the subsequent processing steps required to derive these metrics.

3.4.1 Data Collection

MPTCP performance was evaluated using metrics. These metrics were computed using data collected with `tcpdump` during experiments on the RPi and on the Google Cloud VM. The captured data was saved in packet capture (*pcap*) files, which include sequence numbers, total packet counts, source IP addresses (to identify the network interface used), timestamps, and byte counts. These capture files provide detailed records of the packets transmitted between the multipath gateway and VPS (on the Google Cloud VM), allowing performance metrics such as throughput, latency, and packet loss to be calculated. Furthermore, as already mentioned in Section 3.4.3, GPS coordinates are also stored in the RPi, so that it is possible to make a map with information on the distribution of the values of these metrics in space. Therefore, two files are stored on the multipath gateway, the GPS coordinates and the *.pcap* file generated by `tcpdump`. On the VPS, only the last file is generated.

Due to specific challenges encountered during the experiments, such as insufficient memory problems on the RPi, the following command `tcpdump` was used on it to capture network packets for performance evaluation:

```
1 tcpdump -i any -s 100 -w /tmp/experiencias/gateway.pcap
```

This command was essential for ensuring accurate and reliable data collection, and preventing the experience from stopping because the RPi does not have enough space to write more data. Here is a breakdown of what the command does:

- `tcpdump -i any`: Captures packets on all available network interfaces.
- `-w /tmp/experiencias/gateway.pcap`: Write the captured packets to this file, which corresponds to the file with the most free space in memory. The archive with the most space is previously checked and chosen using the `df -h` command.
- `-s 100`: Limits the captured packet size to 100 bytes, enough to capture the headers but not the full payload, since only the packet headers are of interest when calculating the metrics and so only useful information is stored.

Concerning data storage on the VPS, as the Google Cloud VM does not impose any memory limits, the command used was as follows:

```
1 tcpdump -i any -s 100 -w vps.pcap
```

To transfer the file from the Google Cloud VM to the computer for processing, the following command is used:

```
1 dbxcli put vps.pcap /vps.pcap
```

This command uploads the file `.pcap` from your local machine (VM) to a Dropbox account, specifically to the root directory of Dropbox, using the `dbxcli` command-line interface.

To save the GPS coordinates, the Python code mentioned in Section 3.4.3 is executed and the `.txt` file it generates is also stored in the file `-w /tmp/experiencias`. To transfer the files from the RPi to the computer for processing, the following command is used:

```
1 scp gateway.pcap raquellopes@192.168.100.188:/tmp/
```

This command transfers the file `.pcap` from the local machine (RPi) to the remote machine at IP address 192.168.100.188, placing it in the `/tmp/` directory on the remote system using the secure copy protocol (SCP) over SSH. The same method is used to transfer the coordinates file.

3.4.2 Time Synchronization

Precise time synchronization is essential for an accurate evaluation of MPTCP performance, particularly for measuring latency. In this setup, NTP was configured on both the RPi and the Google Cloud VM to ensure consistent timing across devices. The configuration steps are detailed in Appendix A.4. NTP was chosen over GPS for synchronization, for example, due to synchronizing the clock of the RPi with GPS would interfere with the OMR image installed on the RPi, making NTP the safer choice. This section provides a brief introduction to how NTP works and explains how an NTP server was selected to synchronize both devices.

3.4.2.1 Network Time Protocol (NTP)

The Network Time Protocol (NTP) defines a time service architecture and protocol for distributing time over the Internet. Its primary objectives are [13]:

- Provide accurate synchronization to Coordinated Universal Time (UTC): NTP filters synchronization data and assesses its quality despite Internet delays.
- Ensure reliability during prolonged connectivity losses: NTP uses redundant servers and paths, reconfiguring as needed to maintain service.
- Compensate for clock drift by allowing frequent resynchronization: The service efficiently scales to many clients and servers.
- Protect against interference: NTP employs authentication to verify time data sources and validates message return addresses.

The NTP service operates through a network of servers distributed across the Internet, organized hierarchically in a synchronization subnet with levels called strata [13] (Figure 3.9). The hierarchy is structured as follows:

- Primary servers (stratum 1): Directly connected to the time source, such as a radio clock receiving UTC.
- Secondary servers (stratum 2): Synchronized with primary servers.
- Servers in stratum 3: Synchronized with stratum 2 servers, continuing in subsequent strata.
- Leaf servers: Located on user workstations at the lowest level.

Servers with higher stratum numbers are less accurate than those with lower strata due to errors introduced at each synchronization level. NTP considers total round-trip delays to the root when assessing a server's timekeeping quality [13].

The synchronization subnet reconfigures in response to server failures or inaccessibility. If a primary server loses its UTC source, it can downgrade to stratum 2. Similarly, a secondary server can switch to another synchronization source if its usual one becomes unavailable.

NTP servers synchronize using three modes: multicast, procedure-call, and symmetric.

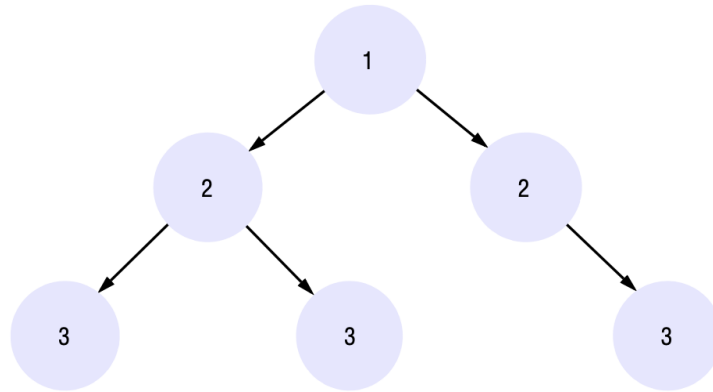


Figure 3.9: Synchronization Subnet in an NTP Configuration. Arrows represent the flow of synchronization control, with numbers indicating different strata levels. Source: [13].

- **Multicast mode:** For LANs, servers broadcast time to other computers, which adjust their clocks assuming minimal delay. This mode provides low accuracy but is sufficient for many applications.
- **Procedure-call mode:** A server responds to requests with its timestamp. Suitable when higher accuracy is needed than multicast provides or when multicast is unsupported. For example, file servers on the same or nearby LAN requiring precise timing can use this mode.
- **Symmetric mode:** Used by servers providing time within LANs and at higher levels of the network, aiming for highest accuracy. Two servers exchange timing messages and store this data as part of an association, enhancing synchronization accuracy.

In all modes, messages are transmitted using UDP, which does not guarantee reliable delivery [13].

3.4.2.2 Configuration of NTP on Devices

As outlined in the previous section, NTP provides a robust method for synchronizing devices over the Internet. It employs a hierarchical structure of time servers, known as strata, where devices typically synchronize to one primary server while having others as backups [39]. However, when two devices need to generate synchronized timestamps to calculate performance metrics, it is necessary to ensure that both devices are connected to the same NTP server. Synchronizing with a single, consistent time source reduces the risk of discrepancies arising from different time sources.

NTP synchronization accuracy heavily depends on the precision with which it can measure the RTT of time-stamped packets exchanged between a client and a server [40]. The RTT measurement is essential because it helps determine the network delay, which must be accurately accounted for to adjust the client's clock correctly.

When the RTT is stable and consistent, the client can more precisely synchronize its clock with the server. However, fluctuations in RTT, often caused by network congestion or varying network conditions, introduce uncertainty, making it more difficult for NTP to accurately calculate the delay, thereby reducing synchronization accuracy [40].

In this work, the primary focus is on the RTT between the client device and the NTP server. High or inconsistent RTT values can significantly affect the client's ability to synchronize accurately with the server. Furthermore, if multiple NTP servers are involved, variability in RTT between them can further complicate synchronization. For this reason, it is often recommended to synchronize with a single, reliable NTP server to reduce discrepancies and ensure more consistent synchronization.

To select the NTP server, it is necessary to determine the network location of both devices. To do this, we used the `curl ipinfo.io` command, which obtains a device's public IP address and geographic information by requesting the `ipinfo.io` API. This identifies the network region from which the devices connect to the Internet, helping in the optimal selection of the NTP server to reduce latency. The results, shown in Figures 3.10 and 3.11, indicate that both the multipath gateway and the VM are located in London, England. Although the multipath gateway is physically in Porto, Portugal, it connects via VPS in London, aligning its network location with the VM. Selecting an NTP server near London is therefore recommended to minimize network-induced delays.

```
rakelmp1@debian11legacymptcp:~$ curl ipinfo.io
{
  "ip": "35.242.178.53",
  "hostname": "53.178.242.35.bc.googleusercontent.com",
  "city": "London",
  "region": "England",
  "country": "GB",
  "loc": "51.5085,-0.1257",
  "org": "AS396982 Google LLC",
  "postal": "E1W",
  "timezone": "Europe/London",
  "readme": "https://ipinfo.io/missingauth"
}
rakelmp1@debian11legacymptcp:~$
```

Figure 3.10: Location output of the Google Cloud VM using the `curl ipinfo.io` command, showing that the VM is in London, England. This information helps in selecting an optimal NTP server for time synchronization based on proximity.

To select the NTP server with the lowest latency, the RTT was measured by pinging the selected servers from both devices for 60 seconds. This test was repeated five times at different times of the day to improve accuracy. The NTP servers chosen for these RTT measurements were `ntp5.leontp.com`, a Stratum 1 NTP server located in Milton Keynes, UK, synchronized via GPS; `pool.ntp0.cam.ac.uk`, a Stratum 2 NTP server hosted by the University of Cambridge in the UK, providing time synchronization as part of the global NTP pool; and `ntp1.tecnico.ulisboa.pt`, a Stratum 2 server based at the Instituto Superior Técnico (IST) in Lisbon, Portugal.

The RTT measurements shown in Table 3.2 for Milton Keynes server, 3.3 for the University of Cambridge, and 3.4 for Instituto Superior Técnico provide a comparison of the RTT of the three

```
[root@OpenMPTCProuter:~# curl ipinfo.io
{
  "ip": "35.242.178.53",
  "hostname": "53.178.242.35.bc.googleusercontent.com",
  "city": "London",
  "region": "England",
  "country": "GB",
  "loc": "51.5085,-0.1257",
  "org": "AS396982 Google LLC",
  "postal": "E1W",
  "timezone": "Europe/London",
  "readme": "https://ipinfo.io/missingauth"
}root@OpenMPTCProuter:~# █
```

Figure 3.11: The multipath gateway's network location, identified through the *curl ipinfo.io* command, indicates it operates from London, England, despite its physical presence in Porto.

different NTP servers when used by the RPi and the Google Cloud VM, assessing the impact of server selection on synchronization accuracy, with a focus on latency and network proximity. By analyzing the RTT values, it becomes possible to identify the most suitable NTP server to achieve optimal time synchronization between the devices.

Table 3.2: Average Round-Trip Time (RTT) measurements between the RPi and Google Cloud VM to the Milton Keynes NTP Server over five tests showing the performance of using the same NTP server for both devices.

RTT AVG (ms) - Milton Keynes, England NTP Server (same NTP server to both devices)						
	Test 1	Test 2	Test 3	Test 4	Test 5	AVG value
Router	90,472	82,968	85,148	77,465	80,679	83,346
Google Cloud VM	4,022	4,009	3,956	3,949	4,145	4,016

Table 3.3: Average Round-Trip Time (RTT) measurements between the RPi and Google Cloud VM to the Cambridge NTP Server over five tests.

RTT AVG (ms) - Cambridge England NTP Server (same NTP server to both devices)						
	Test 1	Test 2	Test 3	Test 4	Test 5	AVG value
Router	81,299	79,929	107,496	87,492	104,010	92,045
Google Cloud VM	7,185	7,677	7,668	7,799	7,798	7,625

The RTT results for the three NTP servers highlight variations in latency depending on the server's geographic location. The Técnico Lisbon NTP server shows an average RTT of 109.027 ms for the RPi and 31.986 ms for the Google Cloud VM, indicating relatively higher latency compared to the other servers. The Cambridge NTP server records an average RTT of 92.045 ms for the RPi and 7.625 ms for the Google Cloud VM, showing lower round-trip times. The Milton Keynes NTP server has the lowest average RTT values, with the RPi showing an average

Table 3.4: Average Round-Trip Time (RTT) measurements between the RPi and Google Cloud VM to the Técnico Lisbon NTP Server over five tests

RTT AVG (ms) - Técnico Lisbon NTP Server (same NTP server to both devices)						
	Test 1	Test 2	Test 3	Test 4	Test 5	AVG value
Router	125,065	99,091	122,150	95,027	103,800	109,027
Google Cloud VM	31,936	32,039	31,926	32,078	31,951	31,986

of 83.346 ms and the Google Cloud VM showing 4.016 ms. These results suggest that, among the three NTP servers, the Milton Keynes server provides the lowest latency, and also a more stable value, followed by the Cambridge server, while the Técnico Lisbon server exhibits the highest RTT values for both the RPi and the Google Cloud VM. So, the Milton Keynes NTP server, among all those analyzed, is the one that will guarantee the highest accuracy in the time synchronization of devices with the server. This makes it a more suitable choice for time synchronization between the devices.

3.4.3 Global Positioning System (GPS)

A Global Positioning System (GPS) module, represented in Figure 3.12, was connected via USB to the RPi to obtain and store GPS coordinates. This setup requires executing commands in the terminal to install necessary GPS-related packages and then running a Python script to save the coordinates to a text file.

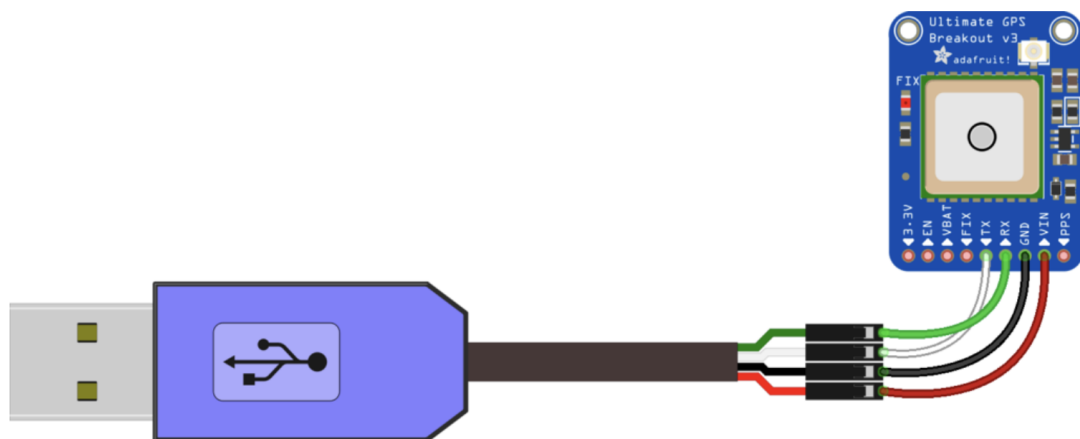


Figure 3.12: Ultimate GPS Breakout with the adapter cable. Source: [14].

After connecting the USB cable to the RPi, it was necessary to install software on the RPi that can interpret the serial data from the GPS module, typically provided through the `/dev/ttyUSB*` interface. The required software is GPS Daemon (GPSD), an intermediary between the GPS hardware and applications, managing parsing errors and offering a standardized interface for any GPS module. To check if the module was receiving a signal and getting the coordinates the

command `cgps -s` was executed. When everything was receiving the data Figure 3.13 represents what appears on the terminal.

Time: 2024-06-26T11:14:02.000Z (0)	GNSS	PRN	Elev	Azim	SNR	Use
Latitude: 41.17847167 N	GP 2	2	61.0	299.0	21.0	Y
Longitude: 8.59444333 W	GP 8	8	68.0	122.0	23.0	Y
Alt (HAE, MSL): 184.700, 133.400 m	GP 10	10	24.0	44.0	28.0	Y
Speed: 0.84 km/h	GP 14	14	24.0	310.0	19.0	Y
Track (true, var): 66.4, -2.1 deg	GP 21	21	72.0	342.0	16.0	Y
Climb: 12.00 m/min	GP 27	27	32.0	124.0	36.0	Y
Status: 3D DGPS FIX (11 secs)	GP 32	32	23.0	78.0	33.0	Y
Long Err (XDOP, EPX): 0.86, +/- 3.2 m	GP 3	3	25.0	207.0	15.0	N
Lat Err (YDOP, EPY): 1.29, +/- 4.8 m	GP 16	16	2.0	167.0	0.0	N
Alt Err (VDOP, EPV): 0.89, +/- 5.1 m	GP 22	22	9.0	324.0	19.0	N
2D Err (HDOP, CEP): 1.37, +/- 6.5 m	SB123	36	32.0	136.0	34.0	N
3D Err (PDOP, SEP): 1.63, +/- 7.7 m						
Time Err (TDOP): 1.52						
Geo Err (GDOP): 2.89						
Speed Err (EPS): +/- 34.7 km/h						
Track Err (EPD): n/a						
Time offset: 0.606841894 s						
Grid Square: IN51qe82						

Figure 3.13: The output of the `cgps -s` command when the GPS module is receiving a satellite signal, with coordinate data presented on the left and satellite information on the right.

The initial GPS configuration was performed indoors before the entire system was fully operational. As mentioned in Section 3.2.4, the GPS module used was the Ultimate GPS Breakout, which has a sensitive antenna but struggles to capture signals in indoor environments. To address this, an external GPS antenna was connected to the module. This setup allows the RPi and GPS Breakout to remain indoors, while the external antenna is positioned near a window to improve signal reliability and integrity.

After conducting outdoor GPS tests along the experiment route, it was observed that the GPS could not capture signals at certain points. To resolve this issue, an external GPS antenna was used during the experiments, ensuring continuous GPS signal throughout the entire process.

To log GPS coordinates over time, a Python script (as detailed in Section A.3 of the Appendix) is executed. This script captures GPS coordinates (latitude and longitude) and logs them with timestamps into a text file. It utilizes the `gps3` library to interact with the GPSD service, with the key functionalities outlined below:

- **Data Collection Loop:**

1. The script connects to the GPSD service and continuously listens for GPS data.
2. Upon receiving valid GPS data, it formats the timestamp with millisecond precision and logs it along with the coordinates to both the file and console.
3. The loop ensures a 1-second interval between data captures by adjusting the sleep time based on the processing time.

- **Error Handling:** The script includes mechanisms to stop gracefully if a `KeyboardInterrupt` is detected or if the device runs out of storage space.

3.4.4 Data Processing Methodology

The captured data, in pcap file, will be processed using Python scripts to compute the performance metrics of interest: throughput, latency, and packet loss. The results will be displayed in graphical form, with performance parameters plotted on the y-axis and time on the x-axis. The Python scripts used for calculating throughput, latency, and packet loss over time are provided in Appendices B.2, B.3, and B.4, respectively.

It is important to clarify that the codes assume the pcap files have been pre-filtered to include only the specific packets relevant for analysis. In this case, the filtering was based on sources IPs (different from Wi-Fi and 5G interface), destination IP, source port, and protocol. This approach ensures that the performance metrics are computed exclusively from the data of interest.

In this case, the pre-filtered process was made by executing the following commands:

For the Gateway PCAP:

- **To filter only the packets from the WiFi interface:**

```
1 tshark -r "gateway.pcap" -Y "ip.src == 192.168.1.94 && ip.  
    dst == 34.89.61.130 && tcp.srcport == 64329 && mptcp" -w  
    gateway_wifi.pcap
```

- **To filter only the packets from the 5G interface:**

```
1 tshark -r "gateway.pcap" -Y "ip.src == 192.168.182.204 && ip.  
    .dst == 34.89.61.130 && tcp.srcport == 54310 && mptcp" -w  
    gateway_5g.pcap
```

For the VPS PCAP:

- **To filter only the packets from the WiFi interface:**

```
1 tshark -r "vps.pcap" -Y "ip.src == 87.103.48.246 && ip.dst  
    == 10.154.0.5 && tcp.srcport == 13182 && mptcp" -w  
    vps_wifi.pcap
```

- **To filter only the packets from the 5G interface:**

```

1  tshark -r "vps.pcap" -Y "ip.src == 92.250.99.85 && ip.dst ==
    10.154.0.5 && tcp.srcport == 18118 && mptcp" -w vps_5g.
    pcap

```

3.4.4.1 Throughput

The code calculates network throughput by reading two pcap files, one for the Wi-Fi path and one for the 5G path, from the multipath gateway. Throughput is calculated as the total number of bits transferred over fixed time intervals (e.g., 1 second) and is measured in megabits per second (Mbits/s).

For each packet, the packet size (in bytes) is multiplied by 8 to convert it to bits. The throughput for each time interval is then calculated using the formula:

$$\text{Throughput}(t) = \frac{\text{Total bits in the interval}}{\text{Time interval} \times 10^6}$$

where the time interval is typically set to 1 second, and the result is in Mbits/s.

The function processes each packet and stores the calculated throughput values and their corresponding timestamps relative to the first packet's time. These timestamps and throughput values allow for a detailed analysis of the network's performance over time.

Additionally, the combined throughput of both the Wi-Fi and 5G paths is calculated as:

$$\text{Combined Throughput}(t) = \text{Throughput}_{\text{Wi-Fi}}(t) + \text{Throughput}_{\text{5G}}(t)$$

This method enables precise tracking of data transfer rates across different network interfaces.

3.4.4.2 Latency

To calculate the network latencies, the code developed starts by reading four pcap files: two for the 5G interface and two for the Wi-Fi interface, distinguishing between sender and receiver data in each case. The reading process is made to obtain raw packet data for both sender and receiver on each network interface.

Next, the sequence numbers and timestamps of the TCP packets are extracted. The sequence number in TCP is crucial, as it allows packets to be identified and sorted correctly across a network. It is crucial to filter the pcap file according to the source port, so as not to pick up sequence numbers from other packets. The timestamp, on the other hand, records the time at which each packet is sent, providing a basis for calculating latency.

Latency calculations are performed by comparing the timestamps of packets with the same sequence number from sender and receiver files. For each matching pair of sequence numbers, the latency is computed as the difference between the receiver's and sender's timestamps.

The mathematical representation of the latency L between a packet sent and received with the same sequence number s is given by:

$$L = (t_{\text{recv}} - t_{\text{send}}) \times 1000$$

where t_{recv} and t_{send} are the timestamps of the packet when received and sent, respectively. The result is multiplied by 1000 to convert the time from seconds to milliseconds.

3.4.4.3 Packet Loss

The code examines TCP packet errors from pcap files. It reads two distinct pcap files, one representing Wi-Fi network traffic and the other for 5G network traffic, both on the sender side. First the code loads each pcap file separately and calculates error percentages based on the last 10 packets. It identifies errors such as lost segments or retransmissions in each packet.

Lost segments are detected through the attribute ‘analysis_lost_segment’, which indicates that a packet analyzer detected a sequence number in the stream that does not match the expected number based on previous packets. Similarly, the attribute ‘analysis_retransmission’ signals that the packet is a retransmission of an earlier packet. If a packet has any of these errors, it is noted, and a running tally of the last 10 packets is kept to calculate the error percentage at each point in time. The timestamps and error percentages are then stored.

Mathematically, the error percentage $E(t)$ at any given time t is calculated by considering the number of error-containing packets within the last 10 packets. This percentage is computed as follows:

$$E(t) = \left(\frac{\text{number of error-containing packets in the last 10 packets}}{10} \right) \times 100$$

Here, $E(t)$ represents the error percentage at time t , and the denominator (10) reflects the total number of packets being considered in the sliding window. The fraction calculates the proportion of packets that contain errors, and multiplying by 100 converts this proportion into a percentage. For example, if 3 out of the last 10 packets contain errors, the error percentage at that point in time would be:

$$E(t) = \left(\frac{3}{10} \right) \times 100 = 30\%$$

Chapter 4

Experiments, Results and Discussion

This chapter presents the experiments, the results and their analysis, evaluating the Multipath Transmission Control Protocol (MPTCP) and comparing its performance with the Multi-Access Real-Time Transport Protocol (MA-RTP) in heterogeneous network environments. The experiments involved transmitting video data from a mobile node over simultaneous Wi-Fi and 5G connections, utilizing various MPTCP schedulers such as Round Robin (RR), Redundant, Earliest Completion First (ECF), and Blocking Estimation (BLEST). Key performance metrics including throughput, latency, and packet loss were measured to assess each scheduler's effectiveness. Additionally, MPTCP's performance was benchmarked against MA-RTP to highlight differences in efficiency and reliability. The results section details the collected data, followed by a discussion that interprets the findings and explores their implications for optimizing multipath transport protocols in real-world scenarios.

4.1 Experiment Overview

The experiment involved the transmission of a video file from a mobile node, in this case, a personal computer, to a server hosted on Amazon Web Services (AWS). The test scenario focused exclusively on the uplink transmission process.

Packet transmission originates from a computer, which forwards data to the multipath gateway via an Ethernet connection. The multipath gateway, implemented on a Raspberry Pi (RPi), functions as one endpoint of the multipath tunnel and manages packet routing. Routing decisions are based on a scheduling algorithm that selects between two available paths, Wi-Fi and 5G, and determines whether packets are transmitted via Wi-Fi, 5G, or both. During the experiment, the multipath gateway continuously logs data on the sender side, including performance metrics and GPS coordinates.

Packets from the multipath gateway are transmitted to the other endpoint of the multipath tunnel, the Virtual Private Server (VPS) deployed on a Google Cloud Virtual Machine (VM). At the VPS, packets are forwarded to their final destination through a Transmission Control Protocol (TCP) connection to an AWS server. Throughout the experiment, the VPS continuously logs data

required for performance evaluation and relays all incoming packets to the AWS server located outside the Google Cloud VM.

4.1.1 Experimental Route

During the packet flow, the mobile part of the setup, as described in Section 3.2.1, followed a predefined route. This mobile setup include the computer, GPS antenna, multipath gateway, and a smartphone which provides 5G network access.

The predefined route was designed to introduce variability in network conditions by incorporating movement and deliberately impacting the stability of the Wi-Fi connection. The concept behind the experimental route was that as the distance from the access point (AP) increased, the quality of the Wi-Fi signal would degrade, while the 5G connection remained stable due to its direct USB connection to the multipath gateway. The mobile setup would then move back towards the AP to take advantage of Wi-Fi signal quality. The objective was to observe how MPTCP adapts to changing network conditions while maintaining at least one available connection.

For each experiment, the route consisted of four fixed points, as depicted in Figure 4.1, with the 5G Next Generation Node B (gNB) marked as point F (red). The Wi-Fi subsystem, including the AP and the 5G HAT, remained stationary at point W throughout the experiment.

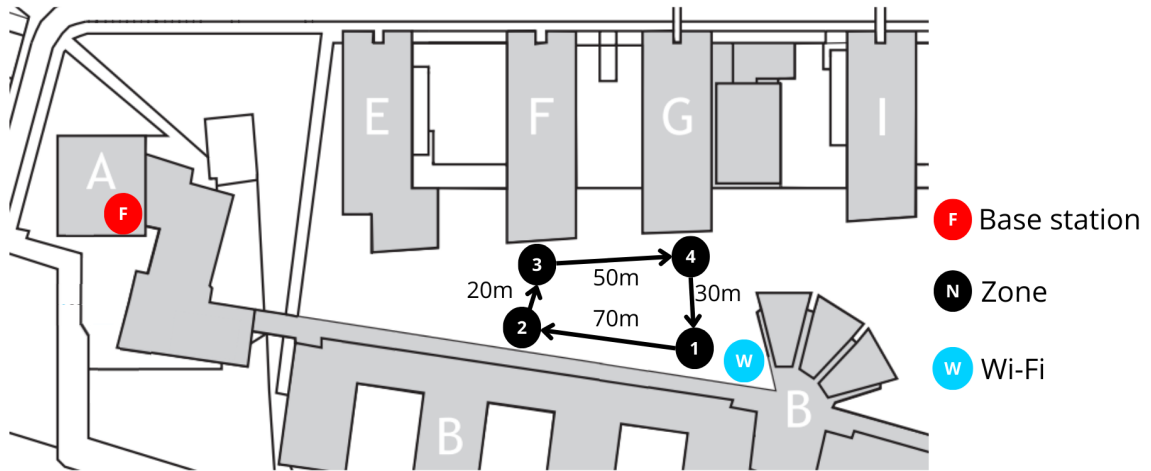


Figure 4.1: Experimental Route. Adapted from source: [10].

The experiment began at zone 1, which was the closest location to the Wi-Fi AP (point W), as this was the nearest point where a GPS signal could be acquired. The route then proceeded sequentially through zone 2, zone 3, and zone 4, before returning to zone 1.

The distances between the zones were defined such that $d_{\text{zone2}} < d_{\text{zone3}} < d_{\text{zone4}} < d_{\text{zone1}}$, corresponding to increasing distances from the AP (0 meters, 30 meters, approximately 59 meters, and 70 meters, respectively). The movement between these zones was conducted at a slow and constant speed. The time taken to travel between each point on the route was consistently 1 minute and 25 seconds, carefully timed to ensure uniformity. Additionally, the traffic transmitted during each session was the same.

It is important to emphasize that the fixed points, distances, and movement parameters were established to ensure accurate comparisons of experimental results. Multiple experiments comparing different schedulers were conducted under identical conditions to maintain result integrity. Both the movement pattern and the route used in the MA-RTP evaluation were replicated to ensure consistency.

During each experiment, a pre-recorded video of a street scene with passing vehicles and pedestrians was transmitted as a stream over a duration of 328 seconds. The video was encoded at a resolution of 640×480 pixels using the H.264 codec with the High 4:2:2 profile, at 30 frames per second and a bit rate of 1676 kbps. The video stream was structured with a Group of Pictures (GoP) size of 4 [10]. This is the same video used for the MA-RTP performance evaluation.

However, the video transmission process encountered a significant issue due to the use of an unsupported protocol in the current setup. Specifically, the original video sender code relied on the RTP protocol over UDP, which proved incompatible with the hardware constraints of the system. The multipath gateway, implemented on a RPi4, lacks the necessary Central Processing Unit (CPU) resources to efficiently run the V2Ray proxy required for handling both TCP and UDP traffic. As a result, the RTP-based video sender could not be utilized.

To address this limitation, a custom Python script was developed to simulate the behavior of the video sender, using TCP. This script manages packet transmission by accurately replicating packet sizes and maintaining precise delays between consecutive packets, thereby ensuring consistent throughput and without overloading the RPi4's limited CPU and memory resources. This alternative approach enabled effective simulation of video transmission within the hardware constraints of the system. A detailed explanation of the problem and the implemented solution is provided in Section 4.2.1.

4.1.2 Planned Testing Scenarios

As mentioned in the previous Section, the experiments were conducted along a consistent route, repeated for each trial. The only variable between experiments was the scheduler. This variation was introduced to evaluate the impact of different schedulers on the performance of multipath transport protocols. The influence of schedulers has been documented in previous studies, as referenced in Section 2.6. Consequently, several schedulers, more specifically the ones that OpenMPTCProuter (OMR) provides (RR, Redundant, ECF, and BLEST, explained in Section 2.3.3), were tested and compared.

Before experimenting with the schedulers, two more extra experiments, using only TCP, were conducted under the same conditions. First extra experiment was for evaluating TCP performance over Wi-Fi network, and the other was TCP over to the 5G network; the setup and results are presented later in Section 4.3. This approach aimed to understand the performance of TCP without multipath along the experimental route, allowing for an assessment of the quality of each network at each point along the route. This also helps to avoid false interpretations. The experimental plan is summarized in Table 4.1. Only one measurement was taken for each scheduler.

Table 4.1: Planned experimental scenarios.

Experiment Number	Experiment	Description
1	TCP only: Wi-Fi	Testing TCP performance using Wi-Fi network only
2	TCP only: 5G	Testing TCP performance using 5G network only
3	MPTCP with Round Robin (RR) Scheduler	Testing MPTCP performance using RR scheduler
4	MPTCP with Redundant Scheduler	Testing MPTCP performance using Redundant scheduler
5	MPTCP with Earliest Completion First (ECF) Scheduler	Testing MPTCP performance using ECF scheduler
6	MPTCP with Blocking Estimation (BLEST) Scheduler	Testing MPTCP performance using BLEST scheduler

4.1.3 Experimental Setup and Location Constraints

This section outlines the primary limitations encountered during the experimental phase, which are pertinent for future research endeavors. Additionally, it identifies potential challenges that can be mitigated through prior planning and awareness.

Regarding the RPi, on particularly hot days, the system overheated, causing the device to stop functioning, which disrupted code execution as the RPi became unresponsive. Consequently, the RPi's operating system had to be reinstalled, followed by the reinstallation of OMR, reconfiguration of network and server settings, and the re-execution of all commands required to install the necessary libraries. The experimental codes had to be re-uploaded, synchronization tools reinstalled and reconfigured, and the GPS data reading installations had to be completed again. This entire process took approximately 1 hour and 30 minutes and had to be carried out in a crowded garden. Additionally, the RPi's limited storage capacity required some experiments to be interrupted and repeated, as data capture was halted due to insufficient storage space.

During the experiment, the GPS repeatedly lost signal for long periods of time, interrupting the tests and forcing them to be restarted. In addition, the GPS's external antenna was connected via a female SMA to u.FL/I-PEX adapter, which proved to be fragile and susceptible to breakage.

In terms of OMR, developers occasionally provided inconclusive or contradictory responses to technical inquiries. Additionally, OMR's available schedulers were limited. While the system offers scheduling algorithms such as default, round-robin, redundant, BLEST, and ECF, their availability can vary across kernel versions. For example, redundancy mode was unavailable in kernel 6.1 but reintroduced in kernel 6.6. In this study, version 59.1-5.4 was used to access all schedulers mentioned in Section 2.3.3, as they are the most commonly used. In version 61-6.6, only two schedulers (round-robin and redundant) worked, with the others losing internet connectivity.

Concerning the Wi-Fi network, the experimental location experienced heavy foot traffic and movement, leading to congestion on the 5G network and interference with the Wi-Fi network.

Lastly, about overall experimental location and setup, FEUP's closure on weekends and public holidays restricted access to electrical outlets inside the building, making it impossible to conduct experiments on less crowded days, such as weekends. Furthermore, IT resources were located on the third floor, and when the elevators malfunctioned, the entire experimental setup, including four tables, had to be manually transported to the test area. Moreover, the FEUP maintenance team did not permit the storage of equipment closer to the experimental area, which could have optimized

the setup process.

4.1.4 Blueprint

A blueprint was created to serve as a foundational guide for replicating the experiment designed to evaluate the performance of the multipath transport protocols in heterogeneous environments. This blueprint is a concise yet comprehensive document that outlines all necessary materials, equipment specifications, methodologies, and experimental conditions in a systematic manner. By clearly documenting the tools, hardware configurations, and procedural steps, the blueprint ensures that every aspect of the experimental setup can be accurately reproduced.

The main advantage of providing this blueprint is that it enables other researchers or practitioners to follow a consistent and well-structured approach, ensuring reliable replication of the study. This is especially important when attempting to evaluate MPTCP under real-world scenarios involving 5G and WiFi networks, where mobility and network variability are critical factors. The documentation covers all the key details, including device configurations, network conditions, and testing protocols, which are essential to capture the nuances of MPTCP's performance in dynamic, real-life settings.

By offering a standardized reference, the blueprint not only facilitates the reproducibility of results but also lays the groundwork for comparative studies with other multipath transport protocols, such as MA-RTP. This ultimately enhances the reliability and validity of performance assessments, contributing to a broader understanding of multipath transport solutions in next-generation network environments.

In addition to evaluating MPTCP and comparing results across different schedulers, this project aims to compare these findings with previous work on the MA-RTP. To ensure a valid comparison, replicating the experimental conditions was essential. Consequently, both the experimental routes and the time intervals between zones were maintained consistently with those used in the MA-RTP evaluation. However, certain challenges in the experimental setup required modifications. The following section details the differences in the evaluation conditions for MPTCP and MA-RTP and the solutions implemented to ensure that these differences do not impact the test environment between protocols, thereby preserving the validity of the comparison.

4.2 Differences in Evaluation Conditions for MPTCP and MA-RTP

Although the route and metrics used in each experiment were identical, certain parameters in the MPTCP evaluation differed from those in the MA-RTP evaluation. To closely replicate the conditions under which MA-RTP was assessed, these parameters were adjusted. This section details the differences in experimental conditions, specifically the traffic sender, which should be a video, and the experimental setup. It also explains the reasons for these variations and the measures implemented to align the experiments as closely as possible.

4.2.1 Traffic Generator: Video Sender Constraint

To transmit the pre-recorded video, the same code used in the MA-RTP evaluation was intended to be employed. However, this code is designed to use RTP, which relies on UDP for transport. OMR provides two options for managing network traffic, one of them is a proxy called V2Ray, which supports both TCP and UDP. However, V2Ray requires significant CPU power to operate efficiently on the multipath gateway.

In this study, the multipath gateway used is a RPi4, equipped with a quad-core Cortex-A72 (ARMv8) 64-bit CPU and 2GB of RAM. While the RPi4 is capable of handling basic routing tasks, its limited hardware resources present challenges when performing more CPU-intensive operations.

The RPi4 concurrently executes OMR while managing multiple tasks, including logging GPS coordinates via a Python script into a text file, capturing network packets into a pcap file, and generating print statements every second for real-time debugging over a 328 second period. Additionally, two persistent SSH terminals remain open throughout the experiment to enable real-time control and monitoring. This combined workload, encompassing GPS logging, packet capturing, print-based debugging, and SSH sessions, exerts considerable strain on the RPi4's CPU and RAM, underscoring the necessity for efficient resource management.

Given these constraints, OMR provides an alternative, less CPU-intensive option: the use of Shadowsocks proxy, which handles only TCP traffic. In this scenario, UDP traffic must be routed through a VPN such as Glortyun UDP, which is designed to transmit UDP efficiently without overloading the CPU. However, while this approach allows for the handling of UDP traffic, it introduces additional overhead, leading to increased latency due to the larger packet size and the nature of TCP's error-checking and acknowledgment mechanisms.

Due to time constraints and to maintain focus on the primary objective of this thesis—the evaluation of MPTCP performance, a script was developed to simulate the video sender code used in the MA-RTP evaluations. This script replicates the throughput observed in MA-RTP sender to ensure methodological consistency across both evaluations.

4.2.1.1 Simulation of Video Sender Behavior

The primary goal of the developed code is to accurately replicate the behavior of the video sender. This involves maintaining a consistent throughput over time, ensuring that packets are transmitted with the appropriate delay between consecutive packets, and that the size of each packet matches that of the video sender used in the MA-RTP evaluation.

To implement this, the video sender code was executed, and the timestamps, with nano second precision, along with the size of each packet were recorded in a text file, creating the following text file:

1	16:09:07.763212841	857
2	16:09:07.763342410	1514

3	16:09:07.763399832	1514
4	16:09:07.763413204	1514
5	16:09:07.763425401	1514
6	16:09:07.763438929	1514

A script then processed this text file to calculate the delay between consecutive packets, ensuring that each packet is sent at the correct relative time. The script also associated each delay with the corresponding packet size, and both pieces of information were written to a separate text file for further use. Below is a portion of the content from the generated text file demonstrating how the delays and corresponding packet sizes are recorded:

1	0	857
2	0.000129569	1514
3	5.7422e-05	1514
4	1.3372e-05	1514
5	1.2197e-05	1514
6	1.3528e-05	1514

A separate script, referred to as the packet transmission code, was developed to simulate packet transmission with precise delays and sizes for the experiments. This script reads the updated text file containing the calculated delays and sends packets according to the specified intervals. To ensure accurate transmission timing, the delay calculation and packet transmission were implemented as separate processes, preventing any interference from the delay computation on the actual packet transmission timing. Both the delay calculation and packet transmission codes are provided in Annex B.5 and B.6, respectively.

4.2.1.2 Packet Transmission Characteristics

To accurately assess the throughput of the packet transmission code and conduct a comparative analysis with the video sender employed in the MA-RTP evaluation, both systems were executed independently on the local host. This approach ensures that the results are not affected by external network factors. Figure 4.2 illustrates the throughput performance over time for the packet transmission code, while Figure 4.3 displays the corresponding data for the video sender code.

The video sender code exhibits a higher average throughput and more pronounced peaks on its throughput graph compared to other components. Furthermore, the graphs indicate that the video sender in MA-RTP completed video transmission earlier than the packet transmission code. This observation suggests that the packet transmission code may introduce additional delays between packets. To potentially reduce these delays, rewriting the code in a compiled language such as C has been suggested. Although this approach was not implemented in the MPTCP experiments, it was explored locally for further investigation after all MPTCP experiments were completed. This local testing resulted in a reduction of total transmission time to 387.71 seconds and an increase in average throughput to 1.61 Mbits/s.

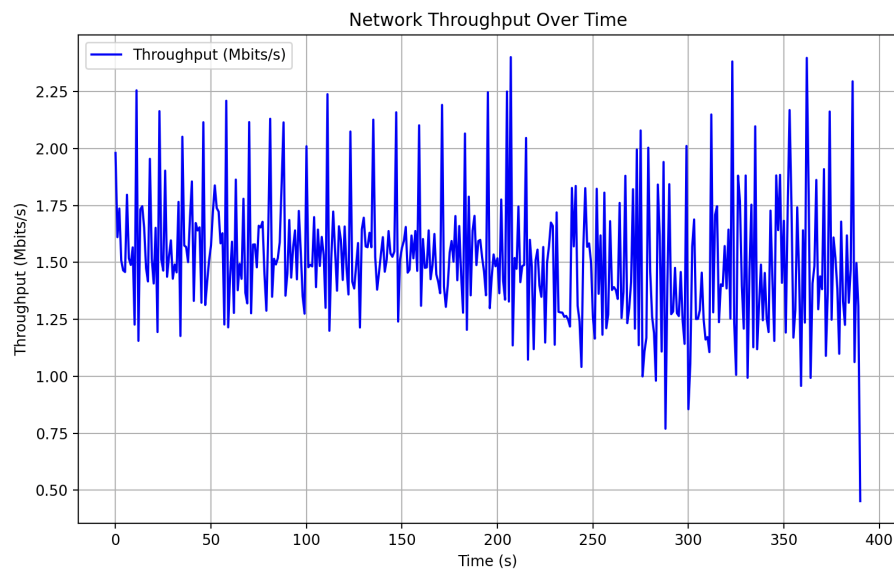


Figure 4.2: Throughput performance over time for the packet transmission code, to simulate the video sender code characteristics, used in MPTCP evaluation under a Loopback interface, averaging 1.52 Mbits/s. Total Time Taken: 390.22 seconds.

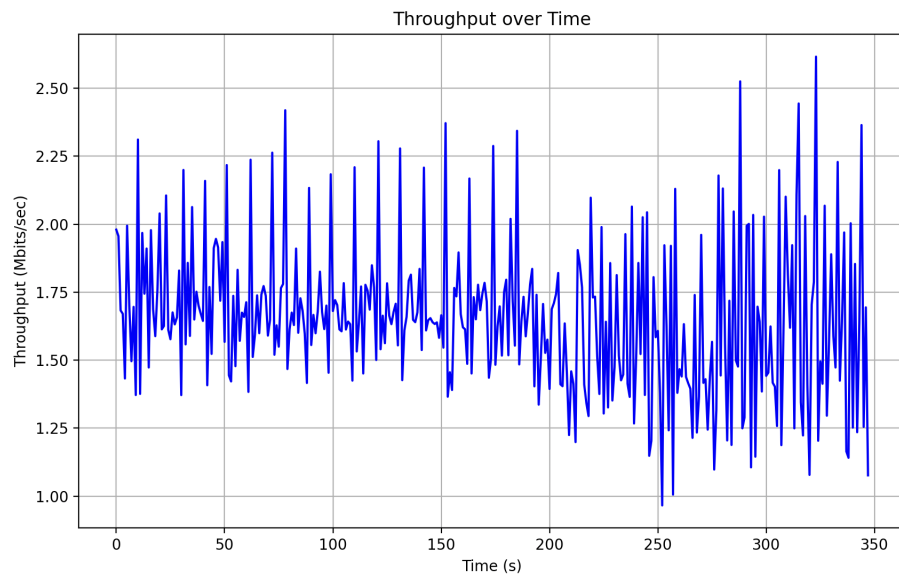


Figure 4.3: Throughput performance over time for the video sender code used in MA-RTP evaluation under a Loopback interface, averaging 1.66 Mbits/s. Total Time Taken: 343.700418 seconds.

However, both throughput graphs exhibit similar variations over time. This similarity ensures that the senders in the experimental setup are equivalent, thereby enhancing the accuracy of the results. Specifically, the sender used in the MA-RTP evaluation has comparable throughput over time to the sender used in the MPTCP evaluation.

4.2.2 Adjustments to the Experimental Setup

Figure 4.4 presents the setup used to evaluate MA-RTP, while Figure 4.5 illustrates the experimental setup employed in this study. Comparing Figures 4.4 and 4.5 reveals two differences that could affect the experimental conditions: the Wi-Fi path and the location of the server serving as the endpoint of the multipath tunnel.

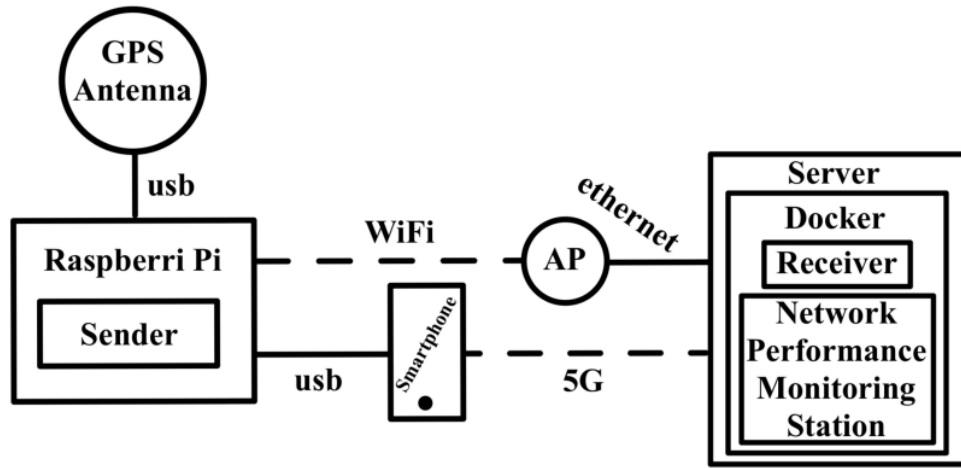


Figure 4.4: Hardware experimental setup from MA-RTP evaluation. Source: [10].

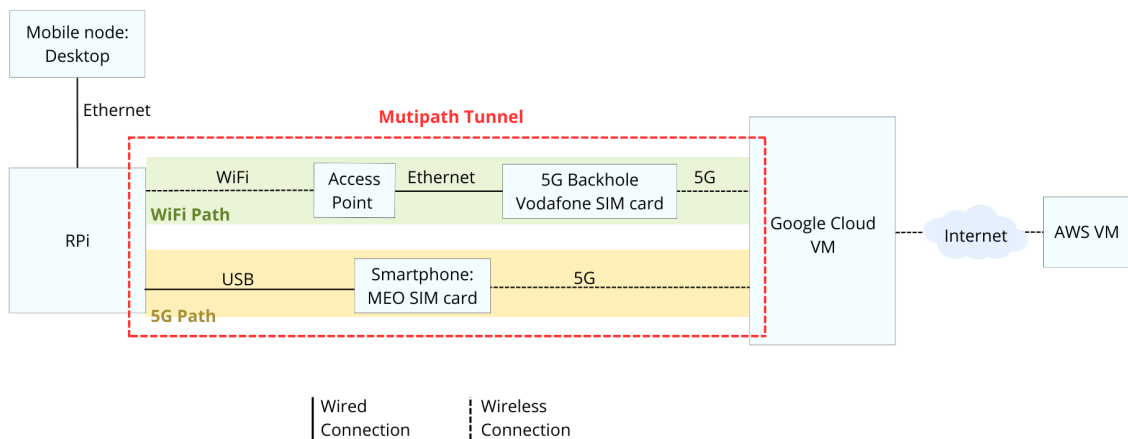


Figure 4.5: Hardware experimental setup used for the MPTCP evaluation experimental setup.

The reasons for the changes in the Wi-Fi path are explained in Section 3.2.2. Briefly, the routers on FEUP's network support only TCP and not MPTCP, necessitating the use of a Wi-Fi network independent of the FEUP infrastructure. Additionally, due to the same limitation, the

server responsible for reassembling the data, the endpoint of the multipath tunnel, had to be located outside the FEUP network.

To ensure that the network conditions were comparable to those in the MA-RTP evaluation, we performed throughput tests on both the Wi-Fi and 5G networks using the *iperf* tool. The objective of these tests was to verify that the Wi-Fi backhaul network, provided by 5G HAT device using a Vodafone SIM card, delivers higher throughput than the 5G network accessed through a smartphone with a MEO SIM card. This configuration aligns with the conditions in the MA-RTP setup.

The *iperf* tests to evaluate the Wi-Fi backhaul were conducted on the RPi associated to the 5G HAT, ensuring that no wireless interference occurred, as could happen if testing connected to the Wi-Fi from the AP. In this setup, the RPi acted as the client, while the server was hosted on the same AWS VM used in the primary experiments. The command executed on the RPi terminal to run the *iperf* client is provided below:

```
1 iperf3 -c 3.8.132.141 -p 5201 -t 60
```

For the 5G network, the smartphone was tethered to a computer via cable for mobile data sharing. In this configuration, the *iperf* client was executed on the computer's terminal, while the server continued to run on the AWS VM. The same client command used for the Wi-Fi tests was also applied in the 5G tests.

For both networks, maximum throughput tests were conducted in the FEUP garden under similar conditions to those of the main experiments, with the exception of mobility and MPTCP, as only the TCP protocol was used. Five *iperf* tests were performed for each network, each lasting 1 minute. The results, presented in Table 4.2 display the maximum average throughput values for Wi-Fi backhaul and 5G.

Table 4.2: Average maximum throughput values for Wi-Fi backhaul and 5G, based on five 1-minute *iperf* tests conducted in the FEUP garden, using the TCP protocol without mobility or MPTCP.

Wi-Fi Backhaul	Experiment Number	Average Maximum Throughput (Mbits/s)	Total Average
Vodafone SIM Card	1	60.7	61.02
	2	60.3	
	3	62.2	
	4	60.7	
	5	61.2	
MEO SIM Card	1	14.2	18.04
	2	16.1	
	3	19.5	
	4	22.0	
	5	18.4	

As shown in the tables, the Wi-Fi backhaul achieved a higher average maximum throughput

(61.02 Mbits/s) compared to the smartphone's 5G network (18.04 Mbits/s). This indicates that the Wi-Fi network provided better performance than the 5G network from the smartphone. Despite the differences in the test setups, the network conditions under which MPTCP was evaluated are similar to those used in the MA-RTP evaluation.

It is important to highlight that, as noted in Section 4.2.1 and presented in Table 4.2, the sender's maximum throughput was 2.50 Mbits/s, with an average of 1.52 Mbits/s. Therefore, the maximum throughput achieved in each network was not constrained by the sender's capacity. The key factor in making the MPTCP evaluation comparable to the MA-RTP setup was ensuring better Wi-Fi performance and more favorable conditions than those provided by the 5G network.

4.3 Comparative Analysis of TCP Performance Over Wi-Fi and 5G Networks

This section provides a detailed analysis of TCP performance by examining throughput, latency, and packet loss metrics. These metrics were derived from a series of experiments outlined in Section 4.1.2. Specifically, two separate experiments were conducted to evaluate the performance characteristics: one utilizing a Wi-Fi network and the other employing a 5G network. The experimental setups for each network type are depicted in Figures 4.6 and 4.7, respectively. In the images presented, it is possible to observe not only the hardware utilized but also the software and tools employed in the experiments.

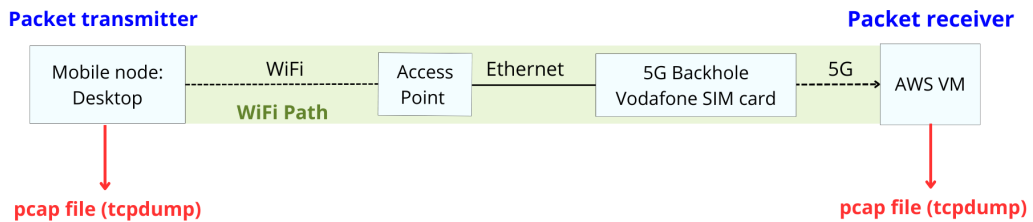


Figure 4.6: Schematic representation of the TCP packet transmission experiment using Wi-Fi network.

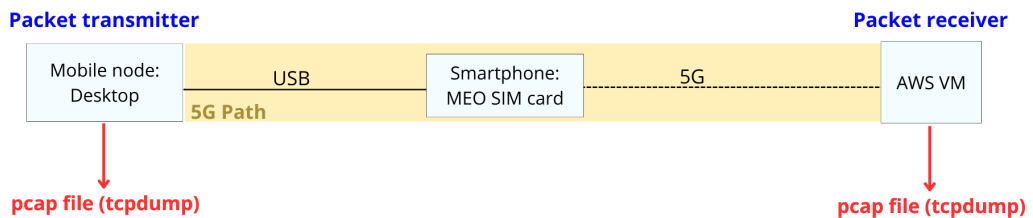


Figure 4.7: Schematic representation of the TCP packet transmission experiment using 5G network.

4.3.1 Throughput

In Figure 4.8, the network throughput is measured over time for TCP connections via Wi-Fi and 5G between a computer and an AWS VM. The graph records throughput fluctuations in Mbits/s across the duration of the experiment.

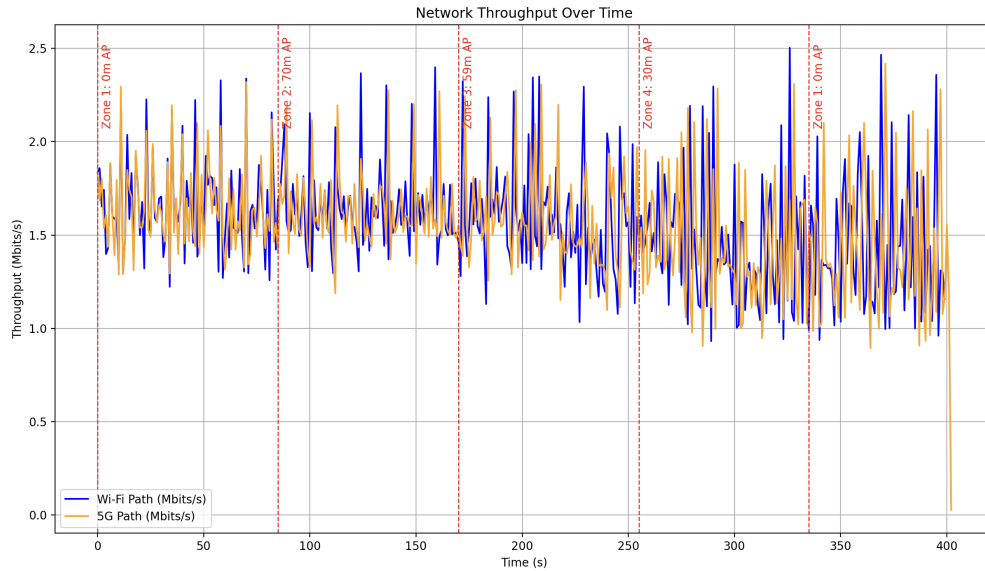


Figure 4.8: TCP over Wi-Fi and TCP over 5G connection. Throughput between computer and AWS VM. Average Throughput for Wi-Fi: 1.56 Mbits/s. Total Transmission Time for Wi-Fi: 399.89 s. Average Throughput for 5G: 1.55 Mbits/s. Total Transmission Time for 5G: 402.03 s.

The graph demonstrates that both connections exhibit significant variability with spikes and drops, yet they maintain remarkably similar average throughput rates over the evaluated period: 1.56 Mbit/s for Wi-Fi and 1.55 Mbit/s for 5G. Furthermore, it can be verified that Wi-Fi concludes packet transmission first, indicating that it was faster than 5G.

These results confirm the observations detailed in Section 4.2.2, which indicated that although the maximum throughput of the Wi-Fi network was higher than that of the 5G network, there was no significant difference in throughput between the two networks during this experiment, as the packet transmitter operated below 2.5 Mbits/s. Furthermore, the consistent stability of Wi-Fi throughput, regardless of movement, suggests that the distance between the mobile setup and the AP was insufficient to cause significant attenuation of the Wi-Fi signal, which could otherwise result in a decrease in throughput. However, as previously mentioned, it is essential to maintain the same route used in the MA-RTP evaluation to ensure comparability of results. Additionally, an abrupt decline in Wi-Fi signal quality is not required; the presence of mobility alone is sufficient to induce changes in network quality.

4.3.2 Latency

The one-way Latency over time of TCP over Wi-Fi and TCP over 5G paths are represented in Figure 4.9. This graph illustrates the fluctuating latencies encountered between a computer and an AWS server over two communication paths.

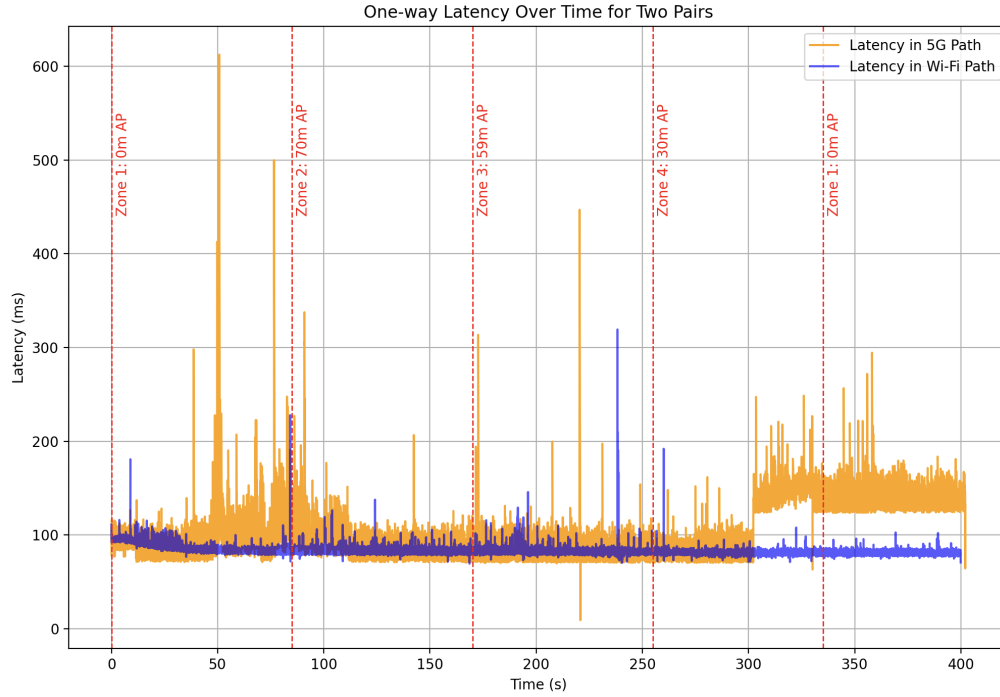


Figure 4.9: TCP over Wi-Fi and TCP over 5G connection. Latency between computer and AWS VM. Average Latency for Wi-Fi Path: 84.36 ms. Average Latency for 5G Path: 109.80 ms.

The data suggests that despite similar average throughput rates for both Wi-Fi and 5G networks, Wi-Fi consistently exhibits lower latency compared to 5G. Latency in the Wi-Fi path shows an average of 84.36 ms, while the 5G path records a higher average of 109.80 ms. This suggests that Wi-Fi provided a lower latency connection in this experimental setup. The lower spikes in latency on the Wi-Fi path also indicate a more stable connection compared to the 5G path, which exhibits several sharp peaks. These spikes can mean momentary deteriorations in 5G connectivity or other external interference that affects the performance of the 5G network.

This disparity in latency measurements points to inherent differences in the network architecture and performance characteristics between Wi-Fi and 5G, highlighting their heterogeneity. Furthermore, the observations suggests that schedulers incorporating RTT, such as ECF and BLEST, might favor packet transmission over Wi-Fi rather than 5G to optimize performance.

4.3.3 Packet Loss

Figure 4.10 depicting TCP packet loss over time provides a direct comparison between the Wi-Fi and 5G network paths in terms of their stability and reliability.

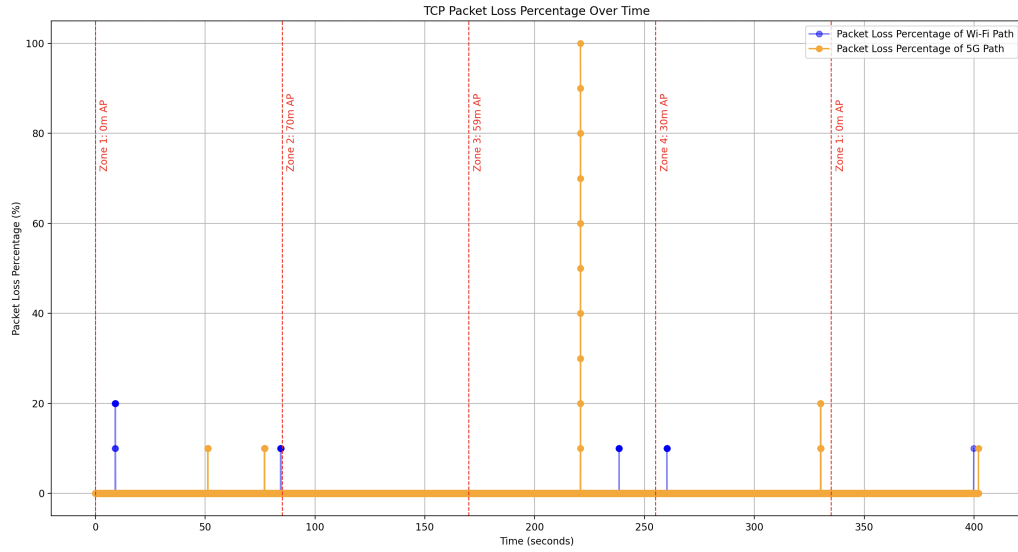


Figure 4.10: TCP over Wi-Fi and TCP over 5G connection. Packet loss percentages based on the last 10 packets between computer and AWS VM. Lost packets in Wi-Fi: 6. Lost packets in 5G: 15.

The data shows that packet loss percentages for Wi-Fi are considerably lower throughout most of the experiment compared to 5G, where several spikes indicate higher packet loss rates. Specifically, the Wi-Fi network experienced a maximum of 6 lost packets, whereas the 5G network saw up to 15 lost packets, highlighting a less stable connection.

The higher packet loss in the 5G network correlates with the increased latency and occasional throughput instability observed in the previous figures. Packet loss, especially at the rates observed in the 5G network, usually requires retransmissions, which can significantly increase latency and have a variable effect on throughput, confirming the interconnection of these metrics.

The combined analysis of throughput, latency and packet loss elucidates the conditions under which each network performs optimally or faces challenges. Additionally the analysis of these TCP-only metrics provides a comprehensive view of the paths' heterogeneity. Although 5G can offer a throughput comparable to Wi-Fi, its higher packet loss and latency suggest limitations in its current deployment and that Wi-Fi will perform better in schedulers' experiences.

4.4 Evaluation of MPTCP Performance Metrics with Different Schedulers

This section aims to thoroughly understand the system's behavior, evaluate the performance of MPTCP, and compare its effectiveness across various scheduling algorithms. The schedulers reviewed in this study, outlined in Section 2.3.3, are the RR, Redundant, ECF and BLEST.

For MPTCP, throughput and latency are measured at each time point, while packet loss is calculated by determining the percentage of TCP packet errors (either lost segments or retransmissions) within a sliding window of the last 10 packets, as detailed in Section 3.4.4.

4.4.1 Round-Robin

4.4.1.1 Throughput Using RR

The graph in Figure 4.11 depicts the network throughput over time for an MPTCP connection utilizing the RR scheduler. It presents three throughput metrics: Wi-Fi Path, 5G Path, and Sum Throughput (which is the sum of throughput of Wi-Fi and 5G in each time point), measured in Mbits/s.

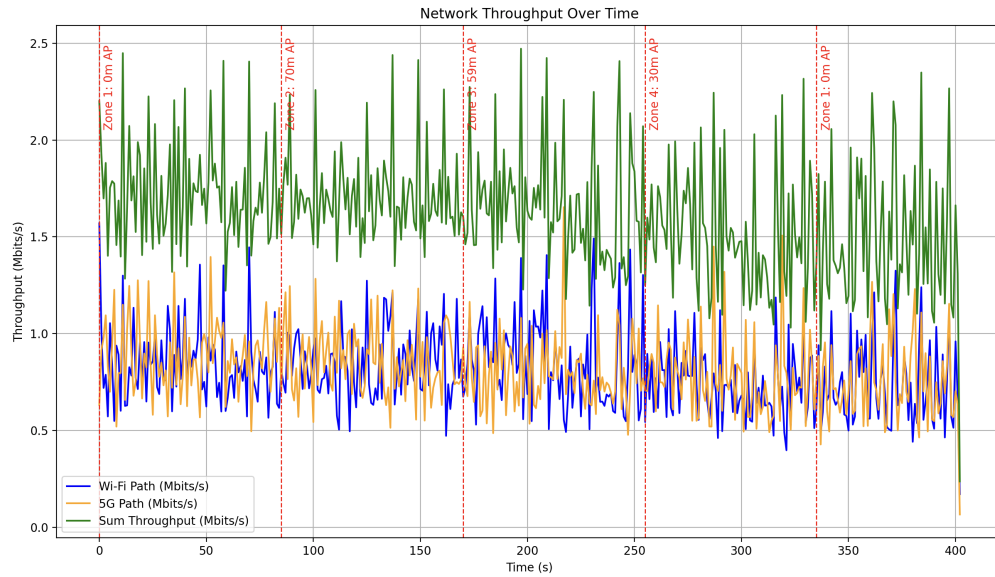


Figure 4.11: MPTCP connection throughput using RR scheduler. Average throughput for Wi-Fi Path: 0.81 Mbits/s. Average throughput for 5G Path: 0.82 Mbits/s. Combined Average Throughput: 1.62 Mbits/s. Total Transmission Time: 402.34s.

The data illustrates the RR scheduler manages traffic, allocating packets alternately between the two paths. The average throughput recorded for the Wi-Fi path is 0.81 Mbits/s, and for the 5G path, it is 0.82 Mbits/s. These closely matched values indicate that the RR scheduler effectively equalizes the load between the two paths, despite the inherent differences in their network characteristics.

The average throughput for the MPTCP connection is 1.63 Mbits/s, which surpasses the average throughput of a single-path TCP connection at 1.56 Mbits/s for Wi-Fi and 1.55 Mbits/s for 5G. However, the total transmission time for the MPTCP connection was longer, recorded at 402.34 seconds. This extended duration suggests that more data was transmitted, likely due to increased packet loss and subsequent retransmissions.

4.4.1.2 Latency Using RR

The graph in Figure 4.12 represents the latency over time for MPTCP connections using the RR scheduler.

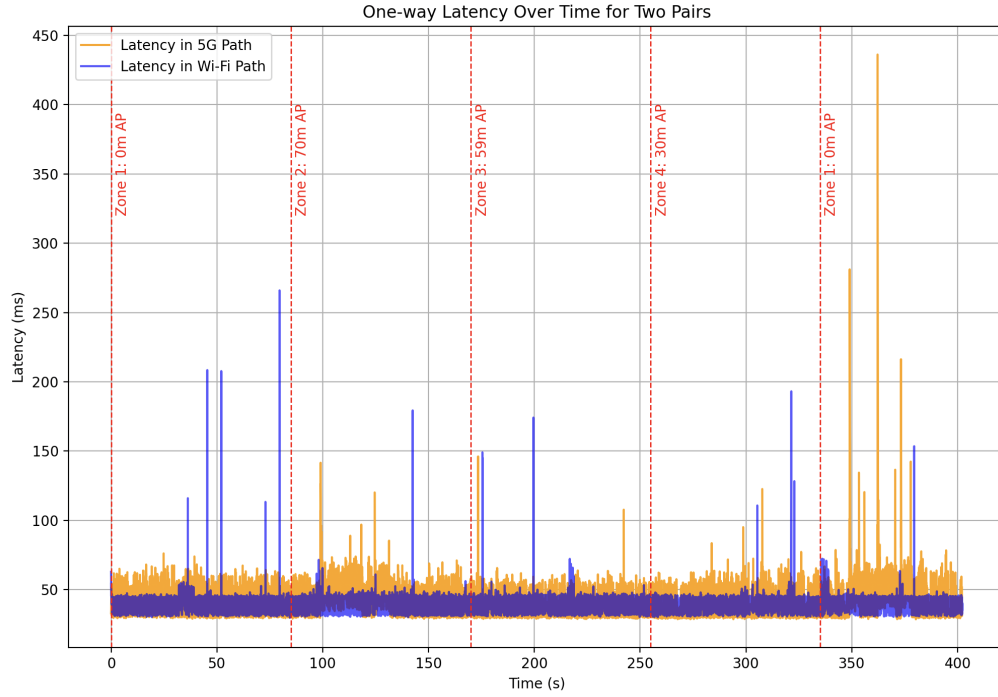


Figure 4.12: MPTCP connection latency using RR scheduler. Average latency for Wi-Fi interface: 41.48 ms. Average latency for 5G interface: 42.74 ms.

The latency graph shows periodic spikes which suggest that both interfaces occasionally experience delays, likely due to external network conditions such as congestion or interference. Despite these spikes, it is possible to observe a general stability of the latency across both interfaces. Most of the latency peaks were observed in the Wi-Fi network as the system approached zone 2. In contrast, the 5G network exhibited very high latency when the mobile configuration moved back toward the AP starting at $t=350$ seconds, corresponding to zone 4. As is possible to seen on the graph the average latency of 5G is higher than in the Wi-Fi, and the values were 42.74 ms and 41.48 ms, respectively.

4.4.1.3 Packet Loss Using RR

Figure 4.13 shows the packet loss over time for an MPTCP connection using the RR scheduler, for both the Wi-Fi and 5G paths.

The total number of lost packets on the Wi-Fi path was 43, while the 5G path experienced only 3 lost packets during the experiment. Consequently, the MPTCP connection had a total of 46 lost packets, which is higher than the packet loss observed when using TCP over each individual path. However, it is not feasible to draw definitive conclusions from individual experiments involving the individual TCP experiments, as they have reported higher packet loss in 5G network.

The Wi-Fi path experienced significantly more packet loss compared to the 5G path, particularly during the initial segments of the experiment. This discrepancy indicates that the Wi-Fi connection faced greater reliability issues than the 5G path during this period, likely due to the

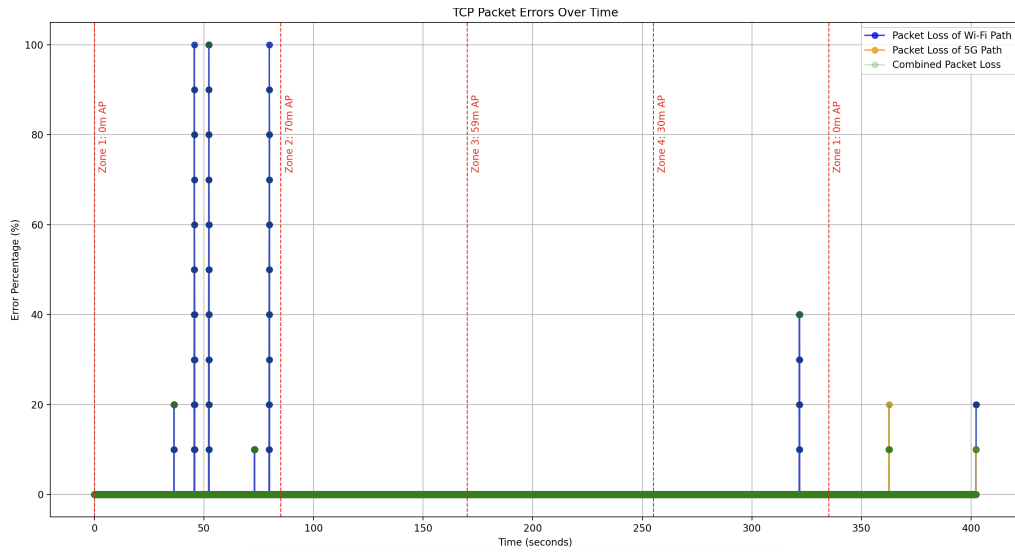


Figure 4.13: MPTCP connection packet loss using RR scheduler. Total lost packets: 46 (43 packets lost in Wi-Fi and 3 in 5G).

progressive movement away from the AP. Since the RR scheduler does not consider current network conditions, it continues to alternate packets between paths regardless of their quality, leading to increased errors and lost packets during movement. Packet loss directly resulted in increased latency. In the case of the RR scheduler, evaluating the impact on throughput is not practical since the scheduler varies iteratively between paths.

It is evident that the RR scheduler's strategy of alternately sending packets over the networks, without accounting for their conditions, contributed to packet loss. The retransmission of these lost packets led to an increase in latency.

4.4.2 Redundant

Figure 4.14 illustrates the throughput over time for an MPTCP connection using the Redundant scheduler across Wi-Fi and 5G paths.

4.4.2.1 Throughput Using Redundant

The graph demonstrates that the scheduler operates as intended. The green line, representing the sum of the throughput of the Wi-Fi and 5G paths, displays values approximately twice those of the individual paths, averaging 3.25 Mbits/s. In addition, the throughput lines for these networks overlap, reflecting the scheduler's strategy of replicating data evenly on both paths to maximize the overall throughput by using both paths simultaneously.

This analysis highlights the Redundant scheduler's capability to provide a more stable and efficient throughput by leveraging both Wi-Fi and 5G paths. It compensates for potential degradation or packet loss on one path by ensuring that data is also transmitted on the other path.

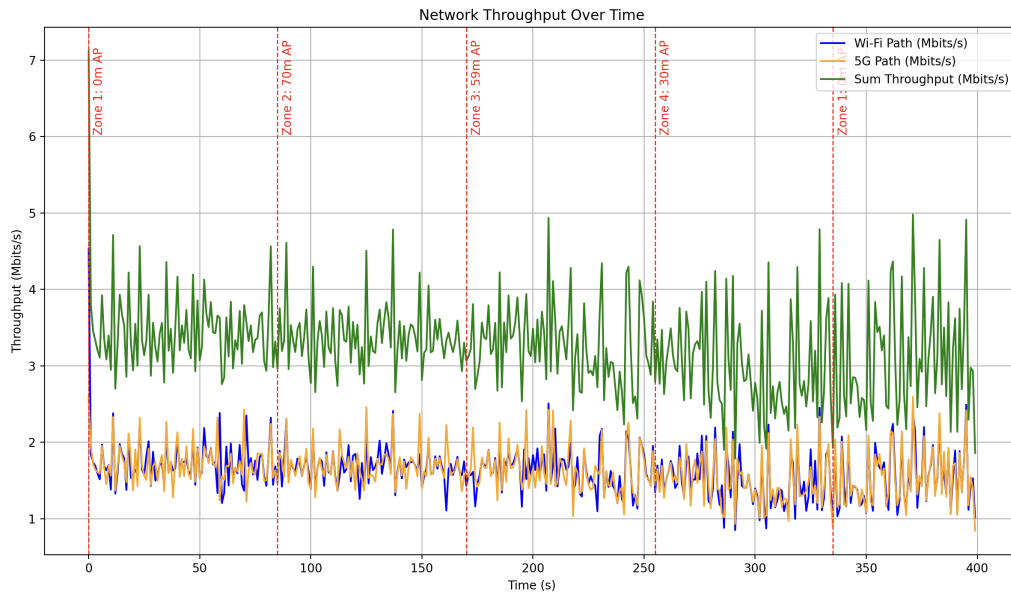


Figure 4.14: MPTCP connection throughput using Redundant scheduler. Average throughput for Wi-Fi Path: 1.63 Mbits/s. Average throughput for 5G Path: 1.63 Mbits/s. Combined Average Throughput: 3.25 Mbits/s.

4.4.2.2 Latency Using Redundant

Figure 4.15 illustrates the one-way latency over time for an MPTCP connection using the Redundant scheduler across both Wi-Fi and 5G paths.

The average latency for the Wi-Fi path is 47.95 ms, and for the 5G path, it is 49.80 ms. While these averages indicate similar latency values between the two paths, the graph shows significant variability, particularly in the Wi-Fi path, which experiences large spikes in latency at several points during the experiment.

The Redundant scheduler enhances reliability by transmitting duplicate packets simultaneously over both network paths. This approach ensures that latency spikes on one path are less likely to affect overall connection performance, as the duplicate packet may still arrive via the other path.

However, the observed variability in the Wi-Fi path's latency indicates that during certain periods, particularly between zone 2 and zone 3, where the distance to the AP is higher, this path experienced less connectivity. With this scheduler, both paths were equally used to transmit the which means that the Wi-Fi path is continuously used to transmit packets throughout the experiments. Consequently, more latency spikes are observed.

Consistent with the latency results observed for the RR scheduler, the Wi-Fi path exhibited higher latency between zones 2 and 3, while the 5G path showed increased latency from zone 4 onwards. However, in the average, 5G latency is higher than Wi-Fi, as is possible to observe in the graph.

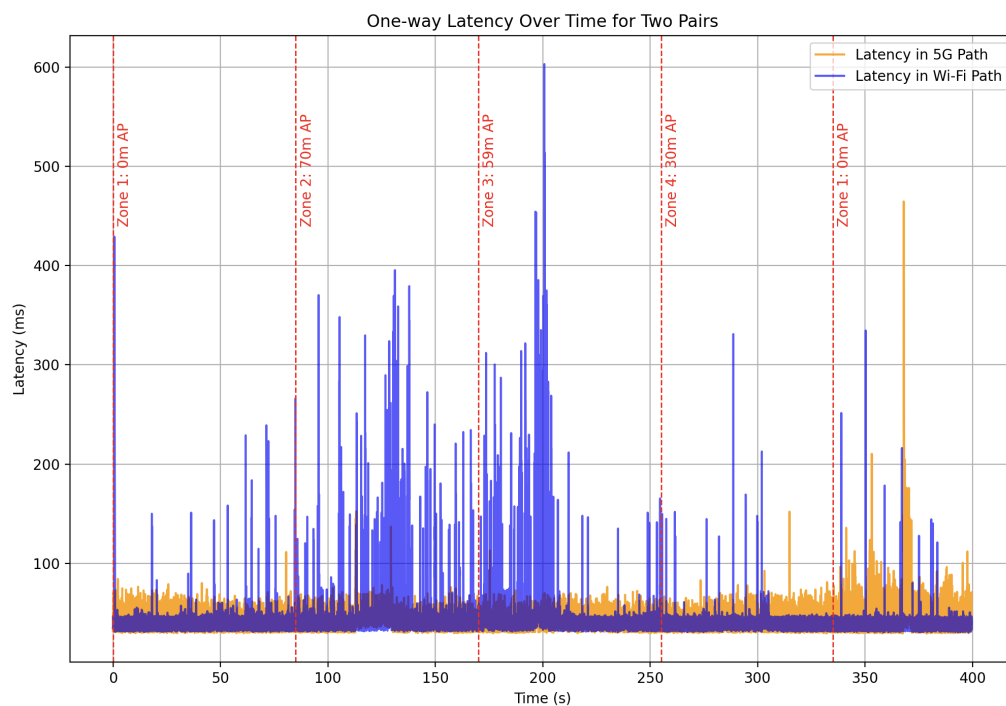


Figure 4.15: MPTCP connection latency using Redundant scheduler. Average latency for Wi-Fi interface: 47.95 ms. Average latency for 5G interface: 49.80 ms.

4.4.2.3 Packet Loss Using Redundant

Figure 4.16 shows the packet loss over time for an MPTCP connection using the Redundant scheduler across both Wi-Fi and 5G paths.

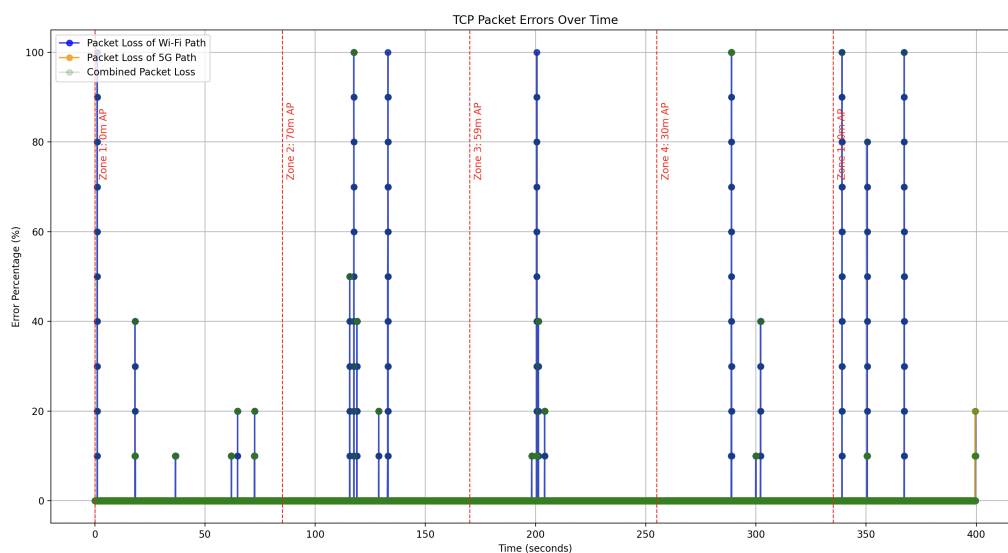


Figure 4.16: MPTCP connection packet loss using Redundant scheduler. Total lost packets for Wi-Fi.pcap: 140. Total lost packets for 5G: 2.

The Wi-Fi path experiences significantly higher packet loss throughout the experiment, with

multiple spikes reaching up to 100% packet loss (which means that 10 out of 10 packets were lost) in some instances. In contrast, the 5G path shows reduced packet loss, with only a few occurrences of packet loss. The combined packet loss, represented by the green line, remains consistently close to zero, indicating that the Redundant scheduler effectively mitigates the impact of packet loss by utilizing both paths for transmission.

The observed packet loss in the Wi-Fi path is attributed to link instability caused by mobility. Both the RR and Redundant schedulers do not consider parameters that reflect this instability, leading to packet loss. Additionally, as seen in the case of the RR scheduler, latency and packet loss are related due to the retransmission of packets at the transport layer.

4.4.3 ECF

4.4.3.1 Throughput Using ECF

Figure 4.17 illustrates the throughput over time for an MPTCP connection using the ECF scheduler across both Wi-Fi and 5G paths.

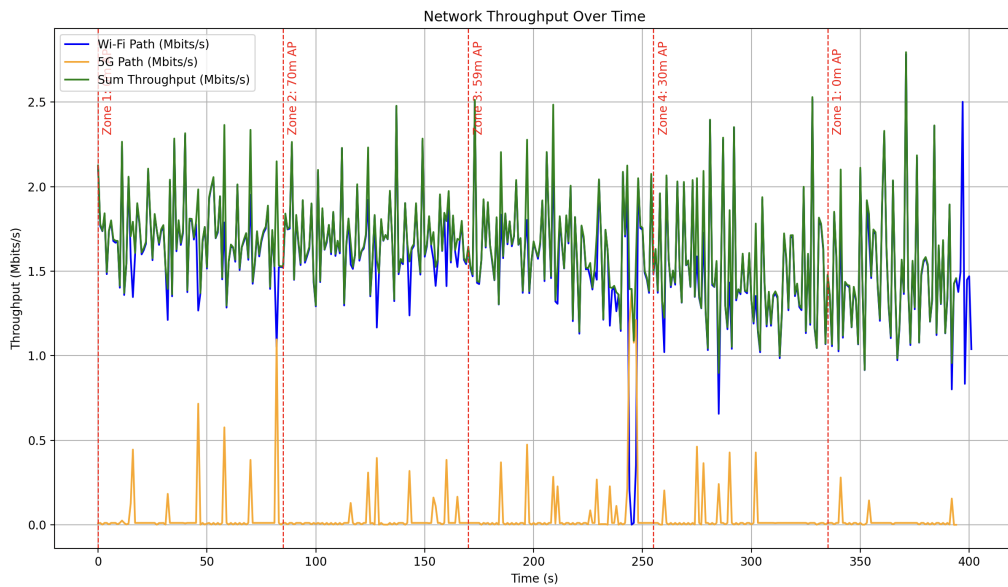


Figure 4.17: MPTCP connection throughput using ECF scheduler. Average Throughput for Wi-Fi Path: 1.58 Mbits/s. Average Throughput for 5G Path: 0.05 Mbits/s. Combined Average Throughput: 1.63 Mbits/s.

The throughput on the Wi-Fi path averages 1.58 Mbits/s, while the throughput on the 5G path is considerably lower, with an average of 0.05 Mbits/s. The combined average throughput of both paths is recorded at 1.63 Mbits/s, indicating that most of the data transmission is handled by the Wi-Fi path, while the 5G path contributes minimally to the overall throughput.

The behavior of the ECF scheduler, which prioritizes packets based on expected completion times, is reflected in this distribution. In this case, the scheduler favors the Wi-Fi path due to its lower latency relegating the 5G path to a secondary role. The sum throughput (green line) follows

closely the Wi-Fi path throughput, which highlights the scheduler's decision to offload most of the data to the more stable and higher-performing path.

4.4.3.2 Latency Using ECF

Figure 4.18 illustrates the one-way latency over time for an MPTCP connection using the ECF scheduler across both Wi-Fi and 5G paths.

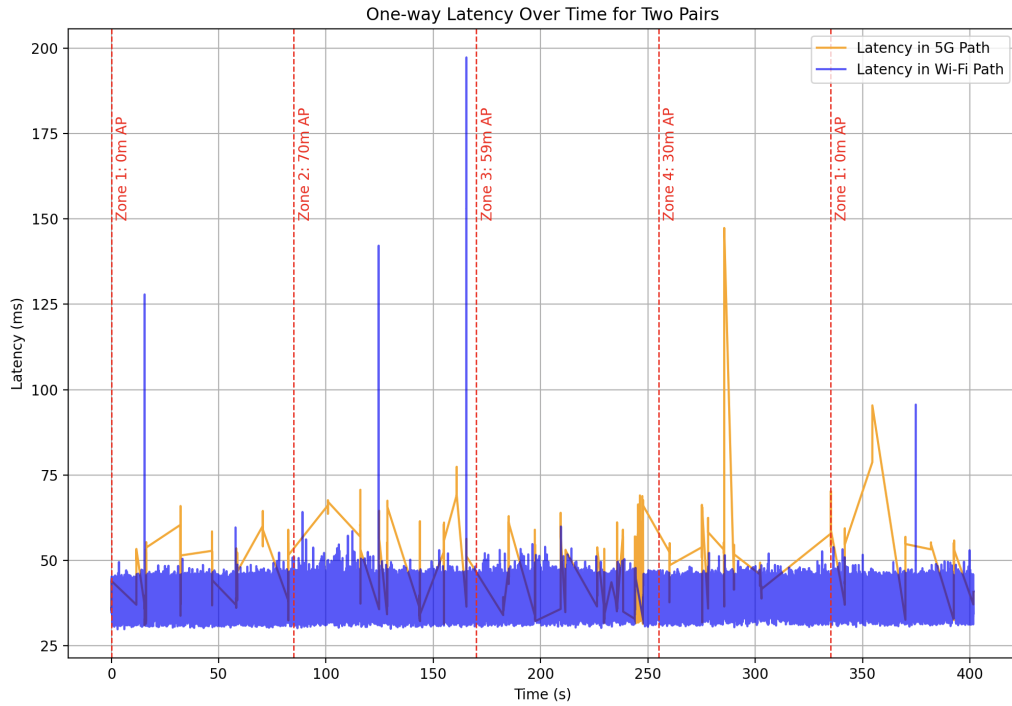


Figure 4.18: MPTCP connection latency using ECF scheduler. Average latency for Wi-Fi interface: 42.04 ms. Average latency for 5G interface: 50.82 ms.

The average latency for the Wi-Fi interface is 42.04 ms, while the average latency for the 5G interface is higher, at 50.82 ms. The Wi-Fi path shows more stability in terms of latency, with fewer spikes compared to the 5G path, which exhibits more significant and frequent increases in latency over the course of the experiment. The ECF scheduler prioritizes paths based on expected completion times, which explains why the Wi-Fi path consistently shows lower latency.

4.4.3.3 Packet Loss Using ECF

Figure 4.19 depicts the packet loss over time for an MPTCP connection using the ECF scheduler across both Wi-Fi and 5G paths.

The Wi-Fi path experiences a total of 12 lost packets during the experiment, while the 5G path records no packet loss. This results indicate that the ECF scheduler effectively mitigates the impact of packet loss by prioritizing packet transmission over the more reliable 5G path when necessary.

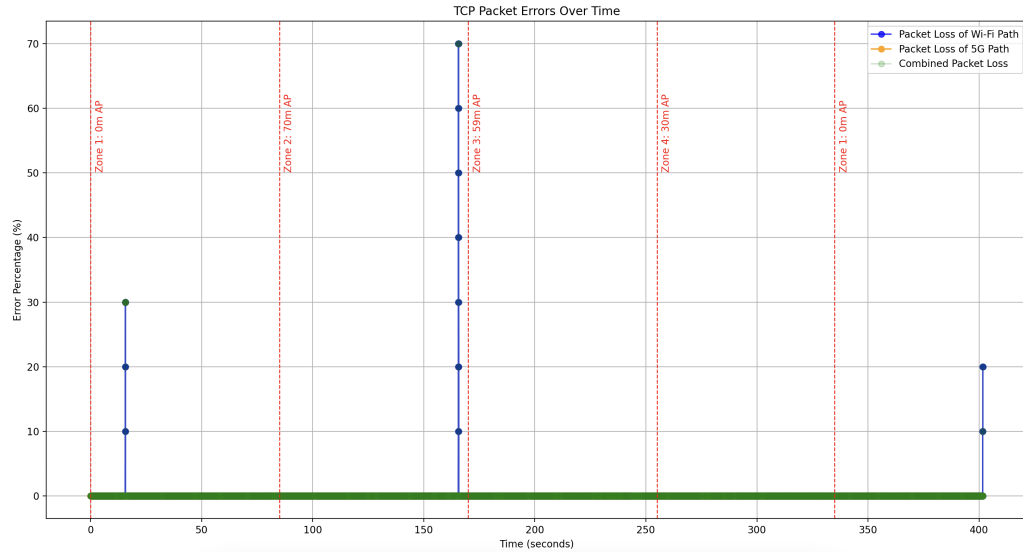


Figure 4.19: MPTCP connection packet loss using ECF scheduler. Total lost packets for Wi-Fi: 12. Total lost packets for 5G: 0.

The spikes in packet loss observed on the Wi-Fi path occur intermittently, suggesting moments of instability or network degradation on that path. However, the ECF scheduler compensates for these lost packets by utilizing the 5G path, which experienced no packet loss during the experiment. This behavior aligns with the ECF scheduler's design, which selects the path with the lowest expected completion time for each packet, allowing it to dynamically adapt to changing network conditions.

4.4.4 BLEST

4.4.4.1 Throughput Using BLEST

Figure 4.20 presents the throughput variation over time for an MPTCP connection utilizing the BLEST scheduler on both Wi-Fi and 5G paths. BLEST's policy is designed to avoid transmitting packets over slower or higher latency paths.

The average throughput for the Wi-Fi path is recorded at 1.21 Mbits/s, while the average throughput for the 5G path is significantly lower at 0.41 Mbits/s. The combined throughput, represented by the green line, fluctuates consistently, demonstrating that the BLEST scheduler is dynamically balancing the use of both paths, though it primarily favors the Wi-Fi path due to its higher throughput.

This is due to the BLEST scheduler's policy, which is designed to avoid transmitting packets over slower or higher latency paths. This objective accounts for the limited utilization of the 5G path observed in this experiment. Consistent with findings from previous MPTCP experiments, the 5G path exhibited a higher average latency compared to the Wi-Fi path. Therefore, the Wi-Fi path handles most of the data transmission, as indicated by the higher and more consistent throughput values. The 5G path is used sparingly, contributing less to the overall throughput, probably due to

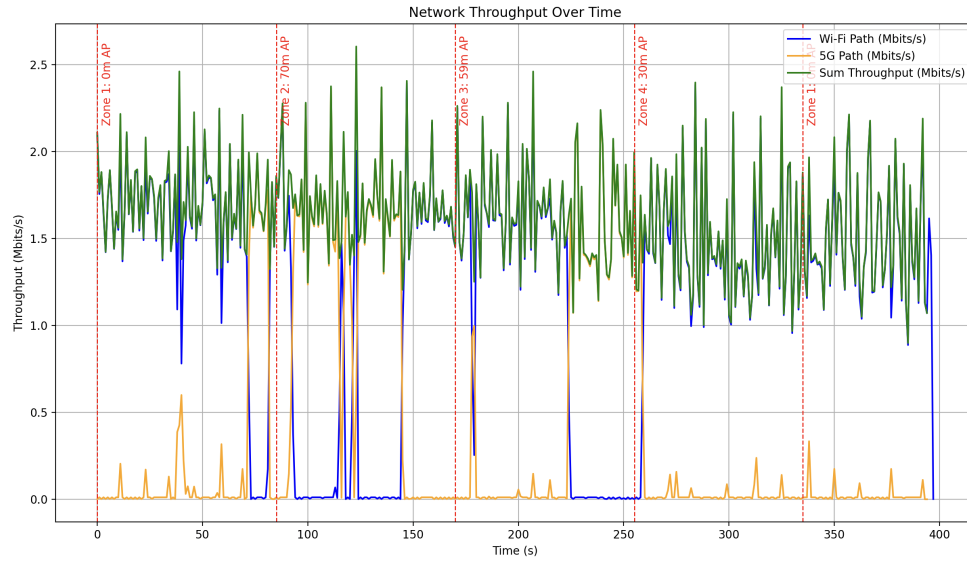


Figure 4.20: MPTCP connection throughput using BLEST scheduler. Average Throughput for Wi-Fi Path: 1.21 Mbits/s. Average Throughput for 5G Path: 0.41 Mbits/s. Combined Average Throughput: 1.61 Mbits/s.

its lower performance compared to the Wi-Fi path. The small peaks in the 5G throughput suggest that BLEST only sends data over the 5G path when necessary or when the Wi-Fi path cannot handle the full load.

In comparison to other schedulers, such as RR or Redundant, which may distribute traffic more evenly across paths, BLEST takes a more selective approach, prioritizing the use of the path with lower latency. The BLEST scheduler achieved an average throughput (1.61 Mbits/s) closely matching the sender's transmission rate of 1.51 Mbits/s. In contrast, the RR and Redundant schedulers recorded higher throughputs, indicating that the MPTCP link experienced instability by transmitting additional packets beyond the intended rate, thereby overloading the network. The ECF scheduler attained an average throughput of 1.63 Mbits/s, which is higher than that of BLEST but remains within a comparable range.

After all the throughput results, it was verified that RR guarantees balanced traffic on both paths, but can be less efficient when the path characteristics are very different. Redundant maximizes reliability by doubling traffic, which leads to higher throughput, but at the cost of some inefficiency. ECF and BLEST prioritize the fastest paths, ensuring more efficient use of resources by focusing on the best-performing links.

4.4.4.2 Latency Using BLEST

Figure illustrates 4.22 the one-way latency over time for an MPTCP connection using the BLEST scheduler across both Wi-Fi and 5G paths.

The average latency for the Wi-Fi interface is 36.18 ms, while the average latency for the 5G interface is higher at 51.40 ms. The Wi-Fi path shows more stable and lower latency throughout

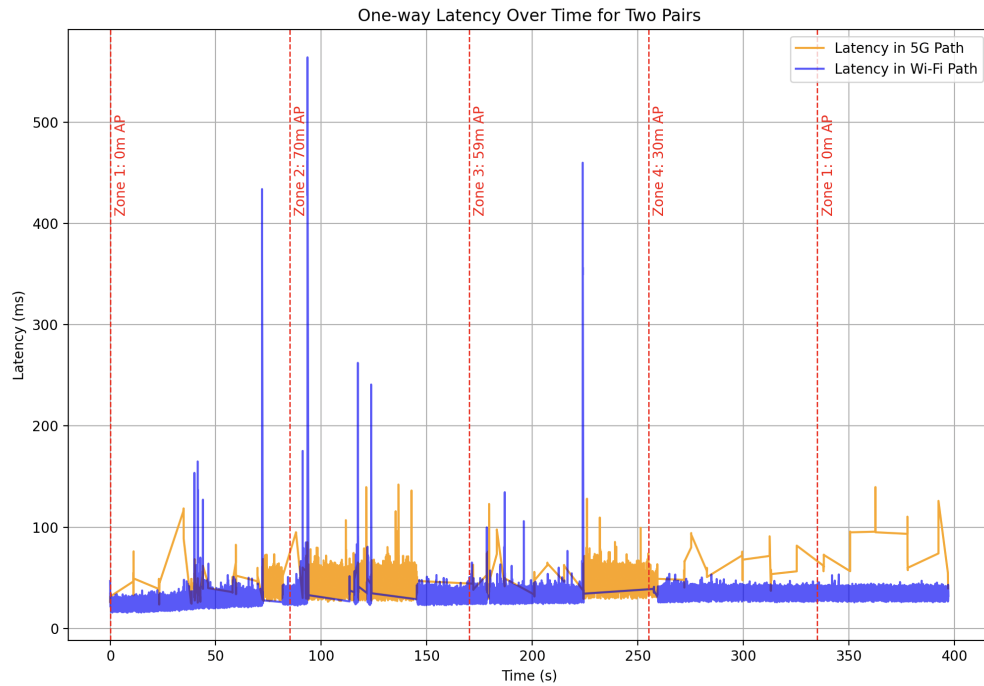


Figure 4.21: MPTCP connection latency using BLEST scheduler. Average latency for Wi-Fi interface: 36.18 ms. Average latency for 5G interface: 51.40 ms.

the experiment, with only occasional spikes, while the 5G path experiences more frequent and larger latency spikes.

The BLEST scheduler is designed to minimize the use of slower paths, which explains the more consistent use of the Wi-Fi path. The Wi-Fi interface maintains lower latency, making it the preferred path for packet transmission in most cases. The 5G path is used less frequently, as reflected in both the lower throughput and the higher, less stable latency values seen in the graph.

When the Wi-Fi network experiences high latency peaks, the throughput graph indicates that BLEST selects the 5G path for packet transmission. By doing so, BLEST minimizes the impact of high latency by limiting the use of the Wi-Fi path when it becomes overloaded or its quality diminishes due to increased distance from the AP.

This adaptive path selection results in a more stable connection with lower average latency. The average latency measurements for each path show that BLEST achieved the lowest latency values for the Wi-Fi path, which was the most utilized path in this scheduler, making the total latency of the MPTCP connection nearly equal to this value.

The BLEST scheduler used the 5G network during critical periods to maintain the overall reliability of the connection. This adaptive behavior ensures that the connection is not overly dependent on a single path

4.4.4.3 Packet Loss Using BLEST

Figure 4.22 shows the packet loss over time for an MPTCP connection using the BLEST scheduler across both Wi-Fi and 5G paths.

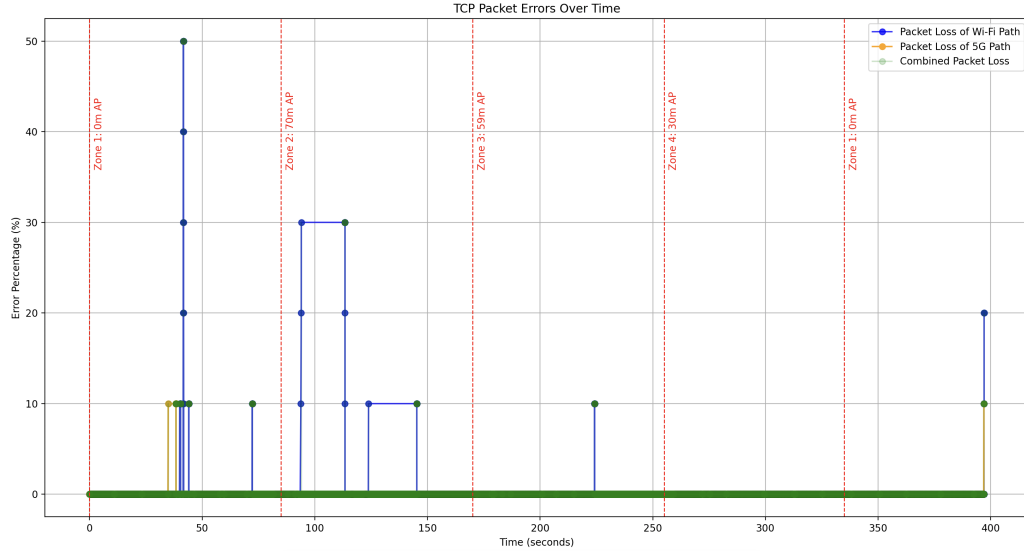


Figure 4.22: MPTCP connection packet loss using BLEST scheduler. Total lost packets for Wi-Fi: 15. Total lost packets for 5G: 2.

The total packet loss recorded is 15 packets for the Wi-Fi path and 2 packets for the 5G path throughout the duration of the experiment. The BLEST scheduler, designed to prioritize paths with the least slack time until the next packet block can be sent, shows a strategic use of the 5G path when the Wi-Fi path experiences higher latency or potential congestion. Once again, most Wi-Fi packet loss occurs near zone 2.

4.5 MPTCP Schedulers Comparison

The ECF scheduler prioritizes the fastest path for data transmission, in contrast to RR and Redundant schedulers, which distribute traffic evenly across available paths. Latency measurements indicate that ECF results in fewer peaks in Wi-Fi latency compared to RR and Redundant schedulers. This outcome arises from ECF's selective utilization of the 5G path based on real-time evaluations of network parameters. RR alternates between paths without considering current network conditions, and the Redundant scheduler employs both paths simultaneously. These approaches can lead to Wi-Fi network overload during changes in distance from the AP. The data suggest that adaptive path selection, informed by parameters such as the congestion window and RTT, enhances connection stability. Therefore, ECF provides a more stable connection with lower latency than RR and Redundant schedulers.

Among the schedulers tested, BLEST achieved the lowest latency. This result is attributed to BLEST's effective use of the 5G path, activating it as necessary, as demonstrated by throughput

and latency graphs. By integrating the 5G path, BLEST obtained lower overall latency. ECF utilized the 5G path less extensively than BLEST, resulting in higher latency compared to BLEST.

In terms of packet loss, the ECF scheduler recorded the lowest total with 12 lost packets, followed by BLEST with 17 lost packets. Despite experiencing higher packet loss than ECF, BLEST demonstrated superior latency performance, as its latency was less impacted by retransmissions. This improvement is due to BLEST's selective activation of the 5G path when required, which optimizes data transmission and mitigates the effects of network fluctuations.

4.6 Comparative Analysis of MPTCP and MA-RTP

This section aims to compare MPTCP with MA-RTP, using equivalent schedulers. RR and Redundant are schedulers that exist for both protocols. The ECF scheduler is similar to the "Context-based Average Throughput" scheduler in MA-RTP, which selects paths based on the highest historical average throughput relative to the current geographical location. The BLEST scheduler is equivalent to the "Timely Arrival" scheduler in MA-RTP which focus on paths with the best RTT and packet loss metrics to ensure timely data delivery to the receiver. These schedulers are tested to determine their effectiveness in managing network traffic and improving the reliability of data transmission under consistent conditions for both MPTCP and MA-RTP. It is important to note that the methods for plotting the metrics for MPTCP and MA-RTP differ. MPTCP metrics are represented graphically at each time point, while MA-RTP uses average values plotted over fixed time intervals. Despite these methodological differences, valid conclusions about the performance of both protocols can still be made. The schedulers compared between this two protocols are represented in Table 4.3.

Table 4.3: Mapping correspondences between MPTCP schedulers and similar MA-RTP schedulers.

MPTCP scheduler	MA-RTP scheduler
RR	RR
Redundant	Redundant
ECF	Context-based Average Throughput
BLEST	Timely Arrival

4.6.1 Round-Robin

Figure 4.23 shows the results obtained for throughput, latency and packet loss in the MA-RTP experiments for the RR scheduler.

In terms of total connection throughput, MA-RTP exhibited a range from approximately 1 Mbits/s to 1.577 Mbits/s, while MPTCP demonstrated a broader range from around 1 Mbits/s to 2.5 Mbits/s. In addition, throughout the experiment, a greater load balance was observed with MPTCP, where the throughput of the Wi-Fi path was closely aligned with that of the 5G path. In contrast, MA-RTP showed a higher disparity, with the Wi-Fi path achieving significantly higher

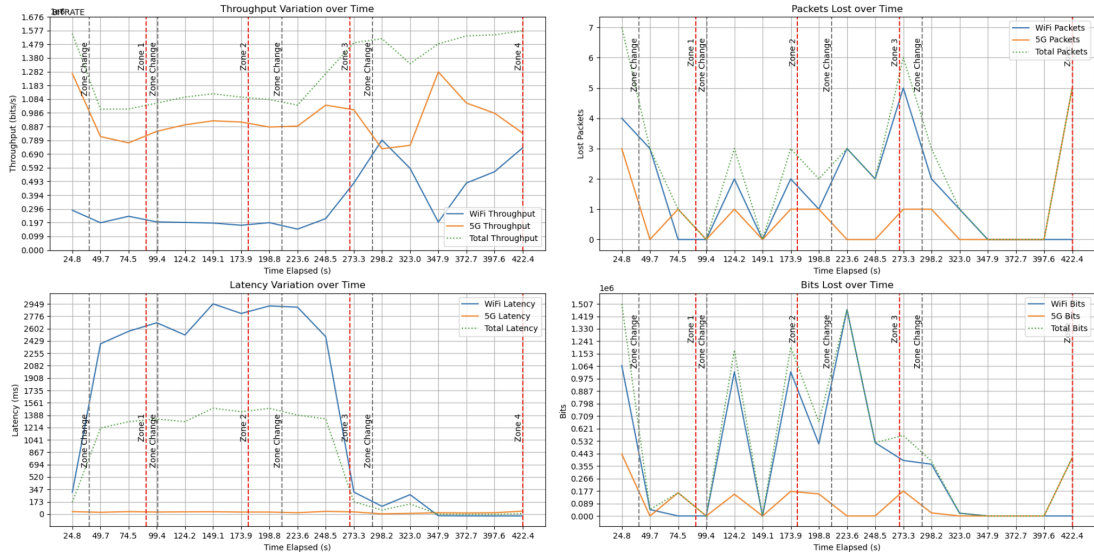


Figure 4.23: MA-RTP connection using RR scheduler. Source: [10].

throughput compared to the 5G path. Only from $t = 347.9$ s, zone 4, the throughput of Wi-Fi and the 5G path come closer together.

Regarding latency, MA-RTP exhibited a significant difference between the two paths, with the Wi-Fi path reaching a maximum latency of approximately 2949 ms, since it had almost no connectivity, while the 5G path had near-zero latency, since it was almost not used to transmit packets. This discrepancy can be attributed to the fact that the RR scheduler in MA-RTP prioritized the Wi-Fi path more frequently than the 5G path for packet transmission. This becomes even more noticeable from time $t = 347.9$ s, zone 4, where it can be seen that the throughput of Wi-Fi and the 5G path have approached each other and therefore the latency decreases, approaching zero.

In contrast, MPTCP showed more balanced latency across both paths, as the RR scheduler distributed traffic equally between them, resulting in effective load balancing. Consequently, the average latency was similar for both paths, with Wi-Fi registering 41.48 ms and 5G at 42.74 ms.

Another observation is that the latency of MA-RTP is higher compared to MPTCP. The observed higher latency values in MA-RTP compared to MPTCP can be attributed to their operation at different protocol layers. Specifically, MA-RTP operates at the application layer, as mentioned in Section 2.4, and calculates performance metrics at this level, as mentioned in [10], potentially introducing additional latency due to higher-level processing. In contrast, MPTCP functions at the transport layer, which is in the kernel space, allowing for more direct handling of data packets. Latency measurements for MPTCP are taken using *tcpdump* at the transport layer, providing a more immediate assessment of network delays.

Regarding total packet loss, MA-RTP resulted in the loss of 34 packets (25 on the Wi-Fi path and 9 on the 5G path), while MPTCP resulted in the loss of 46 packets (43 on the Wi-Fi path and 3 on the 5G path). The higher packet loss in MPTCP compared to MA-RTP can be attributed to the RR scheduler, which equally distributed traffic across both paths without considering real-time

network conditions in MPTCP. This led to continued use of unreliable paths, such as Wi-Fi, even when performance deteriorates, causing higher packet loss. In MA-RTP case, the RR did not use equally both paths, resulting in fewer negative effects from path quality degradation.

Furthermore, using the packet loss data, it is possible to determine that packet loss in MA-RTP has fewer implications for latency compared to MPTCP. This is due to the absence of packet retransmission in MA-RTP, whereas MPTCP, adhering to TCP's reliability mechanisms, triggers retransmissions when packet loss occurs. As a result, MPTCP experiences increased latency during packet loss events, while MA-RTP avoids such delays, although it may suffer from quality degradation in real-time data transmission.

4.6.2 Redundant

Figure 4.23 shows the results obtained for throughput, latency and packet loss in the MA-RTP experiments for the RR scheduler.

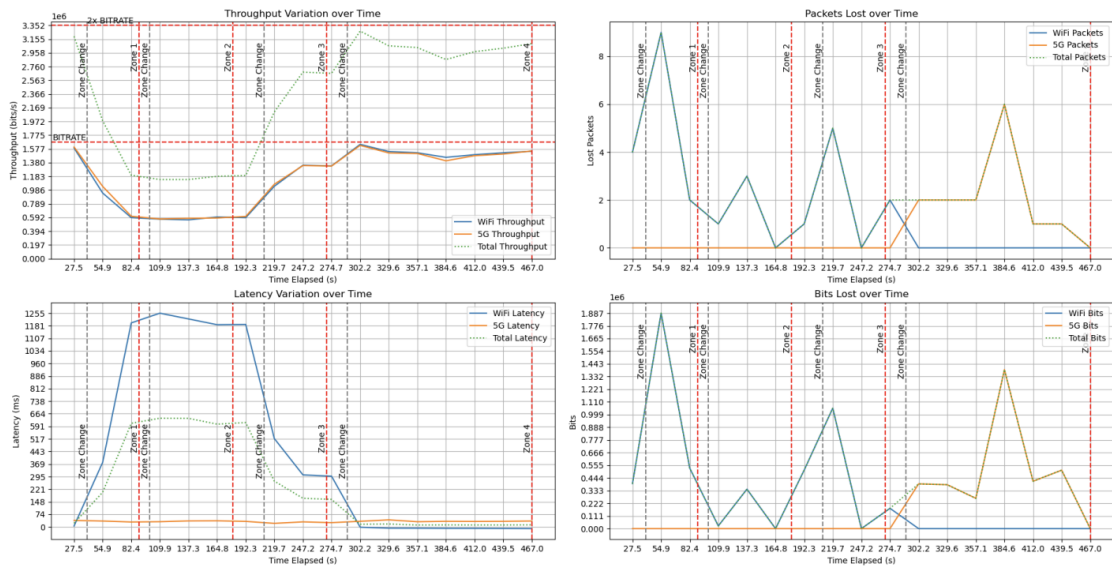


Figure 4.24: MA-RTP connection using Redundant scheduler. Source: [10].

In terms of throughput, the Redundant scheduler followed its methodology of sending the same duplicate packet to both networks, for both MA-RTP and MPTCP, since the throughput lines of both networks (Wi-Fi and 5G) overlap in each case.

However, the average throughput of the MA-RTP connection was 1.17 Mbps (Wi-Fi at 1.168 Mbps), which is lower than that achieved by MPTCP at 3.25 Mbps (1.63 Mbps for both Wi-Fi and 5G). The reduced throughput of MA-RTP primarily occurred between zones 2 and 3, corresponding to the area where the highest latency and packet loss were observed in the Wi-Fi path during the MPTCP experiments.

The lower throughput of MA-RTP in this area is attributed to changes in Wi-Fi network quality as the device moves away from the AP. Since MA-RTP lacks a retransmission mechanism, any degradation in network quality results in a direct drop in throughput.

In contrast, MPTCP includes a retransmission mechanism. When packet loss occurs, MPTCP exhibits increased latency and increased throughput due to retransmissions. This demonstrates that the absence of retransmission in MA-RTP leads to lower throughput when Wi-Fi network quality decreases. Therefore, MPTCP maintains data transmission under mobile conditions owing to its retransmission mechanism.

Regarding latency, the MA-RTP scheduler exhibited higher latency values similar to those observed with the RR scheduler. The maximum latency recorded for the Wi-Fi network under MA-RTP was 1255 ms between zones 2 and 3. In contrast, under MPTCP, the maximum latency observed for the Wi-Fi network was 600 ms between zones 3 and 4. The average latency for MA-RTP was 612.037 ms for Wi-Fi and 32.744 ms for 5G. This discrepancy is attributed to movement away from the AP, which affects the Wi-Fi network but not the 5G network. For MPTCP, the average latency was 47.95 ms for Wi-Fi and 49.80 ms for 5G.

The variance in latency between Wi-Fi and 5G networks is less significant in MPTCP compared to MA-RTP. This discrepancy arises from changes in network conditions between the MA-RTP and MPTCP experiments. During MA-RTP testing, Wi-Fi performance declined notably as the mobile component of the setup distanced from the access point, often nearing a connection loss. Conversely, MPTCP maintained a more stable connection under similar conditions.

Furthermore, the increased latency observed in MA-RTP experiments compared to MPTCP, as detailed in the comparison with the RR scheduler in Section 4.6.1, may stem from the inherent differences in their operational layers. MA-RTP, functioning as an application layer protocol, processes and gathers performance metrics at this higher level, introducing additional latency due to the overhead involved. In contrast, MPTCP operates at the transport layer, which enables direct packet handling and measurement. This arrangement allows for more direct capture of latency data using tcpdump between link and network layers, potentially minimizing delays compared to the application layer approach utilized by MA-RTP.

It is important to note that the latency graph for MPTCP appears more variable compared to that of MA-RTP. This variability is due to differing data processing methodologies. In MPTCP, latency is represented at each data point, whereas in MA-RTP, latency is averaged over relatively large time intervals. Consequently, this averaging process in MA-RTP attenuates the latency peaks observed in MPTCP.

Regarding packet loss, the MPTCP protocol experienced a total of 142 lost packets, whereas the MA-RTP protocol had only 41 lost packets. The higher packet loss observed in MPTCP is attributable to its retransmission mechanism, which involves sending duplicate packets to ensure data delivery. In contrast, MA-RTP does not implement retransmission; therefore, it does not generate additional packet loss through retransmissions, resulting in a lower total number of lost packets.

4.6.3 ECF and Context-based Average Throughput

Regarding the comparison with the MA-RTP protocol, the previous research encountered issues that prevented the collection of results for this scheduler. Specifically, a coding error caused the protocol to terminate shortly after initiating transmission, resulting in unusable data. Consequently, the Context-based Average Throughput, which corresponds to the ECF scheduler, was not evaluated. Because of that, a comparison between MPTCP with ECF and MA-RTP with Context-based Average Throughput is not possible.

4.6.4 BLEST and Timely Arrival

Figure 4.23 shows the results obtained for throughput, latency and packet loss in the MA-RTP experiments for the Timely Arrival scheduler.

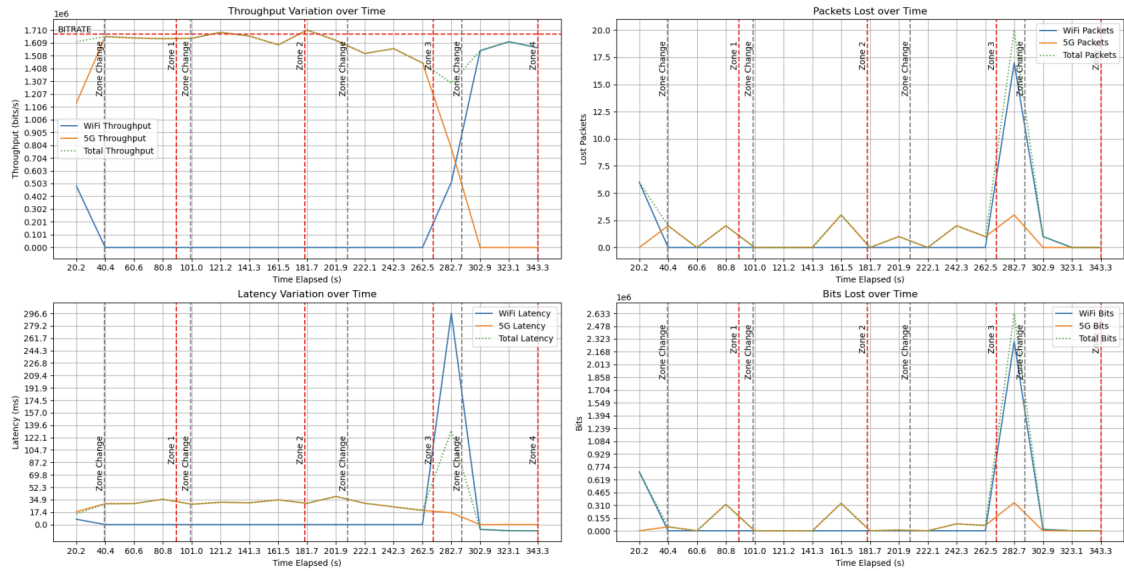


Figure 4.25: MA-RTP connection using Timely Arrival scheduler. Source: [10].

Regarding throughput, the average throughput for MA-RTP was 1.29 Mbits/s, comprising 0.57 Mbits/s over Wi-Fi and 0.857 Mbits/s over the 5G path. In comparison, MPTCP achieved an average throughput of 1.61 Mbits/s, with Wi-Fi contributing 1.21 Mbits/s and the 5G path contributing 0.41 Mbits/s.

Additionally, the Timely Arrival scheduler prioritized the 5G path for the majority of the experiment, as evidenced by the throughput graph. This prioritization indicates that within the experimental setup for MA-RTP, the Wi-Fi path exhibited a higher RTT compared to the 5G path. Consequently, the 5G path was predominantly utilized from zone 1 to zone 4 to optimize data transmission efficiency.

In all experiments conducted using the MPTCP protocol, the Wi-Fi network consistently exhibited lower latency compared to the 5G network. However, due to the inherent differences between these networks, it remains unclear whether the observed latency variations are a result of

the schedulers' decisions or the distinct characteristics of the networks themselves. Consequently, it is not feasible to definitively determine which protocol achieves the lowest overall latency.

However, when comparing schedulers within the same protocol, the scheduler that prioritizes the path with the least delay consistently resulted in the lowest latency among all tested schedulers. This finding indicates that intelligent path selection based on latency metrics can effectively minimize overall latency within a given protocol framework. Furthermore, these demonstrate that, although MA-RTP operates at the application layer, its scheduler's ability to select paths based on RTT parameters and packet loss enables it to achieve significantly lower latency compared to other schedulers.

In terms of lost packets, MA-RTP had a higher value, with a total of 40 lost packets (25 lost packets for Wi-Fi and 15 for 5G). MPTCP had a total of 17 lost packets (15 for Wi-Fi and 2 for 5G).

4.7 Take-Aways

Table 4.4 provides a detailed comparison between TCP and MPTCP protocols across different network conditions, with MPTCP evaluated using various schedulers. Metrics such as average throughput (Mbit/s), average latency (ms), and the number of packets lost are presented for both Wi-Fi and 5G networks.

Table 4.4: Comparative Analysis of Average Throughput, Latency, and Packet Loss for TCP and MPTCP Using Different Schedulers Across Wi-Fi and 5G Networks.

Protocol	Experiment	Network	Average Throughput (Mbits/s)	Average Latency (ms)	Nº Packets Lost	Total Transmission Time (s)
TCP	TCP only: Wi-Fi	Wi-Fi	1,56	84,36	6	399.89
	TCP only: 5G	5G	1,55	109,80	15	402.03
MPTCP	RR scheduler	Wi-Fi	0,81	41,48	43	402.34
		5G	0,82	42,74	3	
		Total	1.62	42,11	46	
	Redundant scheduler	Wi-Fi	1,63	47,95	140	399.68
		5G	1,63	49,80	2	
		Total	3.25	47,95	142	
	ECF scheduler	Wi-Fi	1,58	42,04	12	401.79
		5G	0,05	50,82	0	
		Total	1.63	42,04	12	
	BLEST scheduler	Wi-Fi	1,21	36,18	15	397.09
		5G	0,41	51,40	2	
		Total	1.61	36,18	17	

The performance analysis of MPTCP with different schedulers demonstrated specific operational characteristics. The RR scheduler demonstrated balanced throughput between Wi-Fi and 5G paths but resulted in higher packet loss due to its lack of adaptability to network conditions. The Redundant scheduler achieved the highest throughput by transmitting duplicate packets across

both paths, though it also incurred increased packet loss. The ECF scheduler prioritized the Wi-Fi path based on historical throughput data, resulting in stable latency and reduced packet loss compared to RR and Redundant schedulers. BLEST effectively balanced throughput and minimized latency by selectively utilizing the 5G path when necessary, leading to the lowest overall latency and moderate packet loss.

Table 4.5 presents a comprehensive comparison of performance outcomes from experiments conducted using the MA-RTP and MPTCP protocols across various scheduling techniques, including RR, Redundant, and BLEST/Timely Arrival, within mixed Wi-Fi and 5G environments.

Table 4.5: Performance evaluation results summary of MA-RTP and MPTCP protocols using Round Robin, Redundant, and BLEST/Timely Arrival schedulers, detailing average throughput, maximum and average latency, and packet loss in Wi-Fi and 5G environments.

Scheduler	RR		Redundant		BLEST/Timely Arrival	
Protocol	MA-RTP	MPTCP	MA-RTP	MPTCP	MA-RTP	MPTCP
Average Throughput (Mbits/s)	1.28	1.62	1.17	3.25	1.59	1.61
Wi-Fi vs 5G Throughput Disparity	Wi-Fi > 5G	Wi-Fi ≈ 5G	Wi-Fi ≈ 5G	Wi-Fi ≈ 5G	5G > Wi-Fi	Wi-Fi > 5G
Maximum Latency (ms)	Wi-Fi = 2949 5G ≈ 32.61	Wi-Fi = 265 5G = 435	Wi-Fi = 1255 5G ≈ 32.74	Wi-Fi = 600 5G ≈ 460	Wi-Fi = 219.35 5G ≈ 29.132	Wi-Fi = 570 5G ≈ 150
Average Latency (ms)	Wi-Fi = 2156.16 5G ≈ 32.61	Wi-Fi = 41.48 5G = 42.74	Wi-Fi = 612.04 5G ≈ 32.74	Wi-Fi = 47.95 5G ≈ 49.80	Wi-Fi = 296.6 5G ≈ 40	Wi-Fi = 36.18 5G ≈ 51.40
Packet Loss (# of Packets)	Total = 34 Wi-Fi = 25 5G = 9	Total = 46 Wi-Fi = 43 5G = 3	Total = 57 Wi-Fi = 43 5G = 14	Total = 142 Wi-Fi = 140 5G = 2	Total = 40 Wi-Fi = 25 5G = 15	Total = 17 Wi-Fi = 15 5G = 2

The results reveals significant variations in average throughput, with the Redundant scheduler under MPTCP achieving the highest throughput, reaching 3.25 Mbits/s. Furthermore, latency and packet loss metrics highlight the operational differences and challenges each scheduler encounters in maintaining network efficiency and reliability. Notably, MPTCP combined with the BLEST scheduler and MA-RTP paired with the Timely Arrival scheduler demonstrated lower latency and packet loss values compared to other scheduling techniques. These results indicate that the BLEST and Timely Arrival schedulers are the most effective performers for heterogeneous networks in real-world environments that include mobility.

Due to time constraints, it was not possible to create the code intended to process GPS-collected coordinates and generate maps displaying throughput, latency, and packet loss values. However, this task will be completed by the thesis submission deadline.

Chapter 5

Final Remarks

5.1 Conclusion

This thesis conducted a comprehensive evaluation of the performance of the Multipath Transmission Control Protocol (MPTCP) in operational 5G and WiFi networks, focusing on scenarios involving mobility. A key achievement of this work was the design, implementation, and testing of a multi-technology connectivity gateway that integrates 5G and WiFi networks to provide seamless connectivity during network transitions. The setup was carefully planned and justified, considering various counterpoints and design decisions to ensure it met the requirements of dynamic network environments. Each component of the setup was explained in detail, ensuring transparency in the system's architecture and operations.

A significant part of this research involved testing MPTCP in a mobile scenario, which is critical for understanding how the protocol performs under real-world conditions. Through this experimentation, the thesis explored the behavior of MPTCP under different scheduling algorithms, measuring their impact on throughput, reliability, and latency. The results showed that the scheduling algorithms not only improved network resilience but also optimized data transfer rates in environments where mobility creates network challenges. This practical investigation fills a gap in the existing literature regarding the application of multipath transport protocols in high-mobility environments, where maintaining consistent network performance is often difficult.

Additionally, this study made significant contributions to practical applications by providing clear recommendations for the implementation of multipath transport protocols in various network environments. These findings are particularly relevant to emerging applications such as connected vehicles and smart cities, where reliable and uninterrupted network performance is vital. By focusing on real-world scenarios, the thesis offers actionable insights that can guide the deployment of MPTCP in diverse settings.

In conclusion, the thesis successfully demonstrates the benefits of MPTCP in dynamic 5G and WiFi networks, offering a thorough comparison with the Multipath Aggregated Real-Time Protocol (MA-RTP). The research advances the understanding of how multipath transport protocols can

be optimized and integrated into modern network infrastructures, supporting the development of more resilient and efficient networks for high-demand applications.

5.2 Contributions

- **Practical Performance Insights:** Addressed significant gaps in the literature by providing insights into the practical performance of multipath transport protocols in high-mobility scenarios. Offered practical recommendations for deploying these protocols in both 5G and WiFi networks.
- **Implementation of a Multi-Technology Connectivity Gateway:** Designed, implemented, and tested a connectivity gateway that integrates 5G and WiFi networks to ensure seamless service during network transitions.
- **Comprehensive Evaluation of MPTCP:** Detailed comparison of the MPTCP with the MA-RTP, focusing on performance metrics such as throughput, reliability, and latency under various scheduling algorithms.
- **Contribution to Sustainable Development Goals (SDGs):** Aligned work with SDGs related to Industry, Innovation, and Infrastructure; Sustainable Cities and Communities; and Quality Education. Enhanced technical education and skills development and supported reliable and uninterrupted network connectivity.

5.3 Future Work

In the future, it would be useful to investigate the influence of different congestion control algorithms on MPTCP performance and understand how these algorithms interact with schedulers. Additionally, a more in-depth investigation is needed to understand the differences between Wi-Fi and 5G networks. This could involve measuring the round-trip times (RTTs) for each network and conducting further experiments with TCP. These measurements could help optimize MPTCP performance and provide a more comprehensive understanding of its behavior across different network environments.

Bibliography

- [1] Hongjia Wu, Simone Ferlin, Giuseppe Caso, Özgü Alay, and Anna Brunstrom. A survey on multipath transport protocols towards 5g access traffic steering, switching and splitting. *IEEE Access*, page 24, November 2021.
- [2] Sameer Seth and M. Ajaykumar Venkatesulu. *Protocol Fundamentals*, pages 49–99. 2008.
- [3] Ye Tian, Kai Xu, and N. Ansari. Tcp in wireless environments: problems and solutions. *IEEE Communications Magazine*, 43(3):S27–S32, 2005.
- [4] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and C. Paasch. Multipath tcp. <https://www.rfc-editor.org/rfc/rfc8684>, March 2020. Stream: Internet Engineering Task Force (IETF), RFC: 8684, Obsoletes: 6824, Category: Standards Track, Published: March 2020, ISSN: 2070-1721.
- [5] Jecenia Luzuriaga, Rommel Torres, Patricia Ludeña-González, and Katty Rohoden-Jaramillo. *MPTCP Multipath Protocol Evaluation in Packet Networks*, pages 69–81. 03 2020.
- [6] Nigel Williams. Implementing a Multipath Transmission Control Protocol (MPTCP) stack for FreeBSD with pluggable congestion and scheduling control. Thesis (masters by research), 2017.
- [7] V Kumar and Debabrata Das. Data sequence signal manipulation in multipath tcp: The vulnerability, attack and its detection. *Computers Security*, 103:102180, 01 2021.
- [8] Bruno Yuji Lino Kimura, Demetrius C. S. F. Lima, and Antonio Alfredo Ferreira Loureiro. Packet scheduling in multipath tcp: Fundamentals, lessons, and opportunities. *IEEE Systems Journal*, 15:1445–1457, 2021.
- [9] Farinaz Jowkarishasaltaneh and Jason But. An analysis of mptcp congestion control. *Telecom*, 2022.
- [10] Bernardo Carrilho. *Multipath Video Transport for Edge-Supported Cooperative Perception Services*. June 2023.
- [11] OpenMPTCProuter. Openmptcprouter.

- [12] Waveshare. SIM8200EA-M2 5g hat. <https://www.waveshare.com/sim8200ea-m2-5g-hat.htm>.
- [13] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011.
- [14] Kevin Townsend. Adafruit ultimate gps on the raspberry pi: Setting everything up.
- [15] Tidiane Sylla, Leo Mendiboure, Sassi Maaloul, Hasnaa Aniss, Mohamed Aymen Chalouf, and Stéphane Delbruel. *Multi-Connectivity for 5G Networks and Beyond: A Survey*. October 2022.
- [16] Pehr Söderman, Johan Eklund, Karl-Johan Grinnemo, Markus Hidell, and Anna Brunstrom. Handover in the wild: The feasibility of vertical handover in commodity smartphones. In *2013 IEEE International Conference on Communications (ICC)*, pages 6401–6406, 2013.
- [17] Ana Aguiar, Afonso Azevedo, and Bernardo Carrilho. Poster: Multipath transport for video streams in heterogeneous wireless environments. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, pages 1–2, 2023.
- [18] Olivier Bonaventure and Gregory Detal. Multipath transport protocol (mptp) schedulers: Taxonomy, design, and performance evaluation. Internet-Draft draft-bonaventure-iccr-g-schedulers-02, Internet Engineering Task Force, October 2020. Work in Progress.
- [19] Sana Habib, Junaid Qadir, Anwaar Ali, Durdana Habib, Ming Li, and Arjuna Sathiaselalan. The past, present, and future of transport-layer multipath. *Journal of Network and Computer Applications*, 75, 01 2016.
- [20] Monika U. Daigavhane and Manish D. Chawhan. Congestion control algorithm for tcp in wireless network. In *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, pages 1–4, 2018.
- [21] Michael Scharf and Alan Ford. Multipath TCP (MPTCP) Application Interface Considerations. RFC 6897, March 2013.
- [22] Kun Wang, Thomas Dreibholz, Xing Zhou, Fu Fa, Yuyin Tan, Xi Cheng, and Qining Tan. On the path management of multi-path tcp in internet scenarios based on the nor-net testbed. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 1–8, 2017.
- [23] O. Bonaventure and M. Piraux and Q. De Coninck and M. Baerts and C. Paasch and M. Amend. Multipath schedulers. ietf.org, October 2021. Available at <https://www.ietf.org/archive/id/draft-bonaventure-iccr-g-schedulers-02.html>.
- [24] Imtiaz Mahmud and You-Ze Cho. Performance of multipath tcp schedulers in concurrent use of 5g and 4g networks. In *2022 27th Asia Pacific Conference on Communications (APCC)*, pages 550–553, 2022.

- [25] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. Experimental evaluation of multipath tcp schedulers. In *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, CSWS '14, page 27–32, New York, NY, USA, 2014. Association for Computing Machinery.
- [26] Changqiao Xu, Jia Zhao, and Gabriel-Miro Muntean. Congestion control design for multipath transport protocols: A survey. *IEEE Communications Surveys Tutorials*, 18(4):2948–2969, 2016.
- [27] C. Raiciu, M. Handley, and D. Wischik. Coupled congestion control for multipath transport protocols. RFC 6356, October 2011. Available at <http://www.rfc-editor.org/rfc/rfc6356.txt>.
- [28] Phillipe Austria, Chol Hyun Park, Ju-Yeon Jo, Yoohwan Kim, Rahul Sundaesan, and Khanh Pham. Bbr congestion control analysis with multipath tcp (mptcp) and asymmetrical latency subflow. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1065–1069, 2022.
- [29] Guangjin Pan and Hanhan Xue. Real time analysis of current transport protocols in high loss networks. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 4, pages 2743–2746, 2011.
- [30] Varun Singh, Saba Ahsan, and Jörg Ott. Mprtp: multipath considerations for real-time media. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, page 190–201, New York, NY, USA, 2013. Association for Computing Machinery.
- [31] Ysurac and Contributors. Openmptcprouter. <https://github.com/Ysurac/openmptcprouter>. GitHub repository, accessed: 17 September 2023.
- [32] OpenWrt Project. <https://openwrt.org/>.
- [33] Shadowsocks. Shadowsocks: A secure socks5 proxy. <https://shadowsocks.org/>.
- [34] Alexandre Guillot. Glorytun. <https://github.com/angt/glorytun>.
- [35] Luomeng Chao, Celimuge Wu, Tsutomu Yoshinaga, Wugede Bao, and Yusheng Ji. A brief review of multipath tcp for vehicular networks. *Sensors*, 21(8), 2021.
- [36] Behnaz Arzani, Alexander Gurney, Shuotian Cheng, Roch Guerin, and Boon Thau Loo. Impact of path characteristics and scheduling policies on mptcp performance. In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 743–748, 2014.
- [37] Vivek Adarsh, Paul Schmitt, and Elizabeth Belding. Mptcp performance over heterogenous subpaths. In *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2019.

- [38] Yannick Chabanois. Install or update the vps. <https://github.com/Ysurac/openmptcprouter/wiki/Install-or-update-the-VPS>.
- [39] D.L. Mills, J. Martin, J. Burbank, and W. Kasch. Network time protocol version 4: Protocol and algorithms specification. Request for Comments 5905, Internet Engineering Task Force, June 2010.
- [40] Martin Langer, Kai Heine, Rainer Bernbach, and Dieter Sibold. Analysis and compensation of latencies in nts-secured ntp time synchronization. In *2020 Joint Conference of the IEEE International Frequency Control Symposium and International Symposium on Applications of Ferroelectrics (IFCS-ISAF)*, pages 1–10, 2020.

Appendix A

Commands for Setup Configuration

A.1 OpenMPTCProuter Configuration

A.1.1 Virtual Private Server (VPS) Installation and keys extraction

- For the development server script:

```
1 wget -O - https://www.openmptcprouter.com/server-test/debian-  
x86_64.sh | KERNEL="6.6" sh
```

After installation, the keys, required to configure OpenMPTCProuter on the router, are saved in `/root/openmptcprouter_config.txt`.

- Finally, to finalize this installation, the VPS was rebooted, using the following command (After reboot the ssh port is 65222):

```
1 reboot
```

A.2 Wi-Fi Path Configuration

A.2.1 Setting Up the 5G Hat: Configuring the SIM820X on the Raspberry Pi

To verify that the SIM820X module, Raspberry Pi, and other peripherals are properly connected, use the following commands. Successful connection will display Qualcomm devices and `ttUSB0` through `ttUSB4`:

- List USB devices connected to the system:

```
1 lsusb
```

- List available device files:

```
1 ls /dev
```

If the devices are recognized correctly, proceed with the following steps to configure the SIM820X module:

- Download the necessary setup files:

```
1 wget https://www.waveshare.com/w/upload/1/1e/SIM820X_RNDIS.zip
```

- Install Python 3 pip and the required serial communication library:

```
1 sudo apt-get install python3-pip
2 sudo pip3 install pyserial
```

- Install unzip utility and extract the downloaded archive:

```
1 sudo apt-get install unzip
2 unzip SIM820X_RNDIS.zip
```

- Modify file permissions and run the setup script:

```
1 sudo chmod 777 SIM820X_RNDIS.py
2 sudo python3 SIM820X_RNDIS.py
```

After executing these commands, verify that the `usb0` interface has acquired an IP address using:

- Check the network interface configuration:

```
1 ifconfig
```

In the event of connectivity problems on the RPi, these commands ensure that it can access the 5G network provided by the 5G HAT shield.

- To initiate network access through the `usb0` interface, run:

```
1 sudo dhclient -v usb0
```

- If the `udhcpd` utility is not installed, install it with:

```
1 sudo apt-get install udhcpd
```

- To obtain an IP address for the `usb0` interface using `udhcpd`, execute:

```
1 sudo udhcpd -i usb0
```

- Finally, add a default route through the `usb0` interface with:

```
1 sudo route add -net 0.0.0.0 usb0
```

To send the PIN code of the SIM card to the module, the following commands were executed

- Install `minicom` (in this phase, the module is still not giving access to the 5G network, so use any wifi network):

```
1 sudo apt install minicom
```

- To open a terminal interface for communicating through the specified serial port, allowing you to send and receive data, this command is used (try all `ttyUSB*`, till it responds to the AT commands):

```
1 run sudo minicom -D /dev/ttyUSB2
```

- To send the SIM PIN, use the following command:

```
1 AT+CPIN?  
2 AT+CPIN="xxxx"
```

- Finally, to check if it is connected to the 5G network

```
1 AT+CPSI?
```

A.2.2 Setting Up the AP subsystem

A.2.2.1 Enable IP Forwarding

- Edit the `/etc/sysctl.conf` file to enable IP forwarding:

```
1 sudo nano /etc/sysctl.conf
```

Add or modify the line:

```
1 net.ipv4.ip_forward = 1
```

This allows the system to forward IP packets between network interfaces.

- Apply the changes to enable IP forwarding:

```
1 sudo sysctl -p
```

This command reloads the configuration from `sysctl.conf`.

A.2.2.2 Set Up NAT with iptables

- Configure NAT using iptables:

```
1 sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

This modifies outgoing packets on `eth0` to use its IP address.

- Allow packets to be forwarded from `usb0` to `eth0`:

```
1 sudo iptables -A FORWARD -i usb0 -o eth0 -j ACCEPT
```

- Allow established or related packets to be forwarded from `eth0` to `usb0`:

```
1 sudo iptables -A FORWARD -i eth0 -o usb0 -m state --state  
   ESTABLISHED,RELATED -j ACCEPT
```

A.2.2.3 Verify Configuration

- Check the status of IP forwarding:

```
1 sysctl net.ipv4.ip_forward
```

Verify that `net.ipv4.ip_forward` is set to 1.

- Verify `iptables` rules:

```
1 sudo iptables -L -v -n  
2 sudo iptables -t nat -L -v -n
```

Confirm that NAT and forwarding rules are correctly set up.

A.2.2.4 Configure Network Interfaces and Services

- Configure a static IP address for `eth0`:

```
1 sudo nano /etc/dhcpd.conf
```

Add or modify the following lines:

```
1 interface eth0  
2 static ip_address=192.168.1.2/24
```

- Set up `dnsmasq` for DHCP:

```
1 sudo nano /etc/dnsmasq.conf
```

Add or modify the following lines:

```
1 interface=eth0
```

```
2 DHCP-range=192.168.1.50,192.168.1.150,12h
3 dhcp-option=3,192.168.1.2
4 dhcp-option=6,192.168.1.2
```

- Restart the `dhcpcd` and `dnsmasq` services (or if there is any DNS problem):

```
1 sudo systemctl restart dhcpcd
2 sudo systemctl restart dnsmasq
```

A.2.2.5 Update Routing Table

- Display the IP routing table:

```
1 show ip route
```

- Remove the default route from `eth0`:

```
1 sudo ip route del default dev eth0
```

A.3 GPS Configuration

The command syntax for installing GPS-related packages on OpenWrt is as follows:

- Update the package list:

```
1 opkg update
```

The following commands are used to install GPS-related packages, configure the GPS daemon, and monitor GPS data:

- Install the GPS daemon and client utilities:

```
1 opkg install gpsd gpsd-clients
```

- List connected USB devices:

```
1 ls /dev/ttyUSB*
```

- View raw data from the GPS device (assuming it is connected to `tttyUSB0`):

```
1 cat /dev/ttyUSB0
```

- As the Raspberry Pi used was Stop the GPS daemon service:

```
1 /etc/init.d/gpsd stop
```

- Disable the GPS daemon from starting automatically:

```
1 /etc/init.d/gpsd disable
```

- Terminate all running instances of the GPS daemon (optional):

```
1 killall gpsd
```

- Start the GPS daemon with the GPS device connected to `tttyUSB0`:

```
1 gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
```

- Display GPS data in a human-readable format:

```
1 cgps -s
```

A.4 Time Synchronization

A.4.1 NTP Configuration on Raspberry Pi

- Install the ‘chrony’ package:

```
1 opkg install chrony
```

- Check the list of NTP sources:

```
1 chronyc sources
```

- Edit the Chrony configuration file to set the NTP server:

```
1 nano /etc/chrony/chrony.conf
2 # confdir /var/etc/chrony.d
3 # sourcedir /var/run/chrony-dhcp
4 server ntp1.tecnico.ulisboa.pt burst
```

- Restart the Chrony service:

```
1 /etc/init.d/chronyd restart
```

- Verify the updated NTP sources:

```
1 chronyc sources
```

A.4.2 NTP Configuration on Google Cloud VM

- In the Google Cloud Console, click on SSH for the desired instance.
- Edit the Chrony configuration file:

```
1 sudo nano /etc/chrony/chrony.conf
```

- Find and replace any existing server or pool lines with your specific NTP server. I used the ntp1.tecnico.ulisboa.pt server.

```
1 server ntp1.tecnico.ulisboa.pt iburst
```

- Save the file and exit the editor.
- Restart the Chrony service:

```
1 sudo systemctl restart chrony
```

- Verify the newly configured NTP server:

```
1 chronyc sources
```

Appendix B

Codes

B.1 GPS Data Logging Script

```
1 from gps3 import gps3
2 from datetime import datetime, timedelta
3 import time
4 import shutil
5
6 filename = './coordinates_0.txt'
7
8 while True:
9
10     # Open the file in write mode
11     with open(filename, 'w') as file:
12         # Write the header
13         file.write('Timestamp, Latitude, Longitude\n')
14
15         # Create GPS connection object
16         gps_socket = gps3.GPSSocket()
17         data_stream = gps3.DataStream()
18         gps_socket.connect()
19         gps_socket.watch()
20
21     try:
22         while True:
23             # Record the start time of each loop iteration
24             loop_start_time = time.time()
25
26             # Get GPS data
27             for new_data in gps_socket:
28                 if new_data:
29                     data_stream.unpack(new_data)
30                     latitude = data_stream.TPV['lat']
31                     longitude = data_stream.TPV['lon']
```

```

32
33         if latitude != 'n/a' and longitude != 'n/a':
34             # Get the current local time
35             current_local_time = datetime.utcnow()
36
37             # Add 1 hour to the local time
38             adjusted_time = current_local_time + timedelta(hours=1)
39             timestamp = adjusted_time.strftime('%Y-%m-%d %H:%M:%S.%
              f')[:-3] # Format with milliseconds
40
41             # Print the coordinates to the console
42             print(f"Timestamp: {timestamp}, Latitude: {latitude},
              Longitude: {longitude}")
43
44             # Write the coordinates and timestamp to the file
45             file.write(f'{timestamp}, {latitude}, {longitude}\n')
46             file.flush() # Ensure data is written to file
47
48             # Break out of the inner loop after processing one data
              point
49             break
50
51         else:
52             print("Waiting for GPS data...")
53
54             # Calculate the time taken in the current loop iteration
55             loop_elapsed_time = time.time() - loop_start_time
56
57             # Ensure there's a 1-second interval before starting the next
              iteration
58             if loop_elapsed_time < 1:
59                 time.sleep(1 - loop_elapsed_time)
60             else:
61                 print(f"Warning: Script took {loop_elapsed_time:.3f} seconds to
              process.")
62
63
64     except KeyboardInterrupt:
65         print("GPS reading stopped.")
66         break
67     except OSError as e:
68         if e.errno == 28:
69             print("Error: No space left on device.")
70             break

```

B.2 Throughput

```

1 import pyshark
2 import matplotlib.pyplot as plt
3 import numpy as np
4 from tqdm import tqdm
5
6 def calculate_throughput(pcap_file, time_interval=1.0):
7     capture = pyshark.FileCapture(pcap_file)
8     first_packet_time = None
9     last_packet_time = None
10    current_time = None
11    bits_in_interval = 0.0
12    throughputs = []
13    times = []
14
15    for packet in tqdm(capture, desc=f"Processing {pcap_file}"):
16        if not first_packet_time:
17            first_packet_time = float(packet.sniff_timestamp)
18            current_time = first_packet_time
19            packet_time = float(packet.sniff_timestamp)
20            last_packet_time = packet_time
21            packet_size_bits = int(packet.length) * 8
22
23            if packet_time - current_time > time_interval:
24                throughputs.append(bits_in_interval / (time_interval * 1000000)) #
25                Convert to Mbits/s
26                times.append(current_time - first_packet_time)
27                current_time += time_interval
28                bits_in_interval = 0
29
30            bits_in_interval += packet_size_bits
31
32    # Append the last calculated throughput if any data remains
33    if bits_in_interval > 0:
34        throughputs.append(bits_in_interval / (time_interval * 1000000))
35        times.append(current_time - first_packet_time)
36
37    capture.close()
38
39    # Print the total time taken to send packets
40    total_time = last_packet_time - first_packet_time
41
42    return times, throughputs, first_packet_time, last_packet_time
43
44 def align_data(times1, throughputs1, times2, throughputs2):
45     combined_times = sorted(set(times1) | set(times2))
46     combined_throughput1 = np.interp(combined_times, times1, throughputs1, left=0,
47                                     right=0)
48     combined_throughput2 = np.interp(combined_times, times2, throughputs2, left=0,
49                                     right=0)

```

```

47     combined_throughput = combined_throughput1 + combined_throughput2
48     return combined_times, combined_throughput1, combined_throughput2,
        combined_throughput
49
50 def plot_throughput(times1, throughput1, times2, throughput2, combined_times,
    combined_throughput):
51     plt.figure(figsize=(12, 6))
52     plt.plot(times1, throughput1, label='Wi-Fi Throughput (Mbits/s)', color='blue')
53     plt.plot(times2, throughput2, label='5G Throughput (Mbits/s)', color='orange')
54     plt.plot(combined_times, combined_throughput, label='Combined Throughput (Mbits
        /s)', color='green')
55     plt.xlabel('Time (s)')
56     plt.ylabel('Throughput (Mbits/s)')
57     plt.title('Throughput Over Time')
58     plt.legend()
59     plt.grid(True)
60     plt.show()
61
62 def print_average_throughput(throughput1, throughput2, combined_throughput):
63     avg_throughput1 = np.mean(throughput1) if np.size(throughput1) > 0 else 0
64     avg_throughput2 = np.mean(throughput2) if np.size(throughput2) > 0 else 0
65     avg_combined_throughput = np.mean(combined_throughput) if np.size(
        combined_throughput) > 0 else 0
66
67     print(f"Average Throughput for Wi-Fi Path: {avg_throughput1:.2f} Mbits/s")
68     print(f"Average Throughput for 5G Path: {avg_throughput2:.2f} Mbits/s")
69     print(f"Combined Average Throughput: {avg_combined_throughput:.2f} Mbits/s")
70
71 if __name__ == "__main__":
72     pcap_file_1 = '../BLEST_bbr/gateway_wifi.pcap'
73     pcap_file_2 = '../BLEST_bbr/gateway_5g.pcap'
74     times1, throughput1, first1, last1 = calculate_throughput(pcap_file_1)
75     times2, throughput2, first2, last2 = calculate_throughput(pcap_file_2)
76
77
78     if (first1<first2):
79         first = first1
80     else:
81         first = first2
82
83     if(last1>last2):
84         last = last1
85     else:
86         last = last2
87
88     total = last - first
89
90     # Print total time taken from the first to the last packet
91     print(f"Total time taken {total} seconds")

```

```

92
93     combined_times, combined_throughput1, combined_throughput2, combined_throughput
          = align_data(times1, throughput1, times2, throughput2)
94     plot_throughput(times1, throughput1, times2, throughput2, combined_times,
          combined_throughput)
95     print_average_throughput(throughput1, throughput2, combined_throughput)

```

B.3 Latency

```

1  from scapy.all import rdpcap, TCP
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from tqdm import tqdm # Progress bar
5
6  # File paths for the pcap files
7  sender_pcap_path_1 = '../ECF_bbr/gateway_5g.pcap'
8  receiver_pcap_path_1 = '../ECF_bbr/vps_5g.pcap'
9  sender_pcap_path_2 = '../ECF_bbr/gateway_wifi.pcap'
10 receiver_pcap_path_2 = '../ECF_bbr/vps_wifi.pcap'
11
12 # Read the pcap files for both pairs
13 sender_packets_1 = rdpcap(sender_pcap_path_1)
14 receiver_packets_1 = rdpcap(receiver_pcap_path_1)
15 sender_packets_2 = rdpcap(sender_pcap_path_2)
16 receiver_packets_2 = rdpcap(receiver_pcap_path_2)
17
18 # Filter for only TCP packets
19 def filter_tcp_packets(packets):
20     return [pkt for pkt in packets if TCP in pkt]
21
22 sender_tcp_packets_1 = filter_tcp_packets(sender_packets_1)
23 receiver_tcp_packets_1 = filter_tcp_packets(receiver_packets_1)
24 sender_tcp_packets_2 = filter_tcp_packets(sender_packets_2)
25 receiver_tcp_packets_2 = filter_tcp_packets(receiver_packets_2)
26
27 # Extract sequence numbers and timestamps with progress bar
28 def extract_seq_timestamps(packets):
29     seq_timestamps = []
30     for pkt in tqdm(packets, desc="Extracting TCP sequence numbers and timestamps"):
31         :
32         if TCP in pkt:
33             seq = pkt[TCP].seq
34             ts = pkt.time
35             seq_timestamps.append((seq, ts))
36     return seq_timestamps

```

```

37 sender_seq_times_1 = extract_seq_timestamps(sender_tcp_packets_1)
38 receiver_seq_times_1 = extract_seq_timestamps(receiver_tcp_packets_1)
39 sender_seq_times_2 = extract_seq_timestamps(sender_tcp_packets_2)
40 receiver_seq_times_2 = extract_seq_timestamps(receiver_tcp_packets_2)
41
42 # Function to calculate latencies for a given pair of sequences and timestamps
43 def calculate_latencies(sender_seq_times, receiver_seq_times):
44     latencies = []
45     for sender_seq, sender_time in tqdm(sender_seq_times, desc="Calculating
46         latencies"):
47         # Find the matching receiver sequence number
48         for receiver_seq, receiver_time in receiver_seq_times:
49             if sender_seq == receiver_seq:
50                 latency = (receiver_time - sender_time) * 1000 # Convert to
51                     milliseconds
52                 if latency >= 0: # Only consider non-negative latencies
53                     latencies.append((sender_time, latency))
54                 break
55     # Sort latencies by sender timestamp
56     latencies.sort(key=lambda x: x[0])
57     return latencies
58
59 latencies_1 = calculate_latencies(sender_seq_times_1, receiver_seq_times_1)
60 latencies_2 = calculate_latencies(sender_seq_times_2, receiver_seq_times_2)
61
62 # Separate timestamps and latencies for plotting
63 def prepare_plot_data(latencies):
64     timestamps = [t[0] for t in latencies]
65     latency_values = [t[1] for t in latencies]
66     # Convert timestamps to relative time (from 0)
67     relative_times = np.array(timestamps) - timestamps[0]
68     return relative_times, latency_values
69
70 relative_times_1, latency_values_1 = prepare_plot_data(latencies_1)
71 relative_times_2, latency_values_2 = prepare_plot_data(latencies_2)
72
73 # Plotting the latency graph without moving average for both pairs
74 plt.figure(figsize=(12, 8))
75 plt.plot(relative_times_1, latency_values_1, label='Latency in 5G Path', color='
76     orange')
77 plt.plot(relative_times_2, latency_values_2, label='Latency in Wi-Fi Path', alpha
78     =0.65, color='blue')
79
80 plt.xlabel('Time (s)')
81 plt.ylabel('Latency (ms)')
82 plt.title('One-way Latency Over Time for Two Pairs')
83
84 # Adding vertical lines for zones
85 zones = {

```

```

82     0: "Zone 1: 0m AP",
83     85: "Zone 2: 70m AP",
84     170: "Zone 3: 59m AP",
85     255: "Zone 4: 30m AP",
86     335: "Zone 1: 0m AP"
87 }
88
89 for x, label in zones.items():
90     plt.axvline(x=x, color='red', linestyle='--', linewidth=1)
91     # Adjusting the y-coordinate for the labels to make them lower
92     plt.text(x + 2, plt.ylim()[0] + (plt.ylim()[1] - plt.ylim()[0]) * 0.7, label,
93             color='red', ha='left', fontsize=10, rotation=90)
94
95 plt.legend()
96 plt.grid(True)
97 plt.show()

```

B.4 Packet Loss

```

1 import matplotlib.pyplot as plt
2 import pyshark
3
4 def analyze_pcap(file_name):
5     # Load the pcap file
6     cap = pyshark.FileCapture(file_name, keep_packets=False)
7
8     # Lists to hold timestamps and error percentages
9     timestamps = []
10    error_percentages = []
11
12    # Queue to hold the last 10 packets' error states
13    error_queue = []
14
15    # Initialize the base timestamp
16    base_time = None
17
18    total_packets = 0
19    total_errors = 0
20
21    for packet in cap:
22        try:
23            # Check if the packet is TCP
24            if 'TCP' in packet:
25                total_packets += 1
26                current_time = float(packet.sniff_timestamp)
27                if base_time is None:

```

```

28         base_time = current_time
29
30         has_error = hasattr(packet.tcp, 'analysis_lost_segment') or hasattr
31             (packet.tcp, 'analysis_retransmission')
32
33         if has_error:
34             total_errors += 1
35
36         # Update the queue of the last 10 packets
37         if len(error_queue) >= 10:
38             error_queue.pop(0)
39             error_queue.append(has_error)
40
41         # Calculate the percentage of error packets in the last 10 packets
42         error_percentage = (sum(error_queue) / len(error_queue)) * 100
43
44         # Append the relative time and error percentage
45         timestamps.append(current_time - base_time)
46         error_percentages.append(error_percentage)
47
48     except AttributeError:
49         # Handle packets that do not contain TCP layer gracefully
50         continue
51
52     cap.close()
53
54     # Print total packet loss for the interface
55     print(f"Total lost packets for {file_name}: {total_errors}")
56
57     return timestamps, error_percentages
58
59 def plot_error_percentages(timestamps_1, error_percentages_1, timestamps_2,
60     error_percentages_2):
61     # Combine the error percentages for plotting
62     combined_timestamps = sorted(set(timestamps_1) | set(timestamps_2))
63     combined_errors = []
64
65     for t in combined_timestamps:
66         error_1 = next((error_percentages_1[i] for i in range(len(timestamps_1)) if
67             timestamps_1[i] == t), 0)
68         error_2 = next((error_percentages_2[i] for i in range(len(timestamps_2)) if
69             timestamps_2[i] == t), 0)
70         combined_errors.append(error_1 + error_2) # Sum of error percentages for
71             combined packet loss
72
73     # Plotting all three sets of data in one figure
74     plt.figure(figsize=(15, 8))
75
76     # Plot the first PCAP

```

```

72     plt.plot(timestamps_1, error_percentages_1, marker='o', linestyle='--', label='
       Packet Loss of Wi-Fi Path', alpha=0.75, color='blue')
73
74     # Plot the second PCAP
75     plt.plot(timestamps_2, error_percentages_2, marker='o', linestyle='--', label='
       Packet Loss of 5G Path', color='orange')
76
77     # Plot the combined errors
78     plt.plot(combined_timestamps, combined_errors, marker='o', linestyle='--', label=
       ='Combined Packet Loss', alpha=0.2, color='green')
79
80     # Adding vertical lines for zones
81     zones = {
82         0: "Zone 1: 0m AP",
83         85: "Zone 2: 70m AP",
84         170: "Zone 3: 59m AP",
85         255: "Zone 4: 30m AP",
86         335: "Zone 1: 0m AP"
87     }
88
89     for x, label in zones.items():
90         plt.axvline(x=x, color='red', linestyle='--', linewidth=1)
91         # Adjusting the y-coordinate for the labels to make them lower
92         plt.text(x + 2, plt.ylim()[0] + (plt.ylim()[1] - plt.ylim()[0]) * 0.7,
                   label, color='red', ha='left', fontsize=10, rotation=90)
93
94     # Adding titles and labels
95     plt.title('TCP Packet Errors Over Time')
96     plt.xlabel('Time (seconds)')
97     plt.ylabel('Error Percentage (%)')
98     plt.grid(True)
99     plt.legend()
100
101     # Show the plot
102     plt.tight_layout()
103     plt.show()
104
105 # Analyze first pair of PCAPs
106 timestamps_1, error_percentages_1 = analyze_pcap('../BLEST_bbr/gateway_wifi.pcap')
107
108 # Analyze second pair of PCAPs (modify the file name as needed)
109 timestamps_2, error_percentages_2 = analyze_pcap('../BLEST_bbr/gateway_5g.pcap')
110
111 # Plot the results
112 plot_error_percentages(timestamps_1, error_percentages_1, timestamps_2,
                        error_percentages_2)

```

B.5 Packet Delay Calculation

```
1 from datetime import datetime
2
3 def parse_file(filename):
4     with open(filename, 'r') as f:
5         lines = f.readlines()
6
7     data = []
8     for line in lines:
9         parts = line.strip().split('\t')
10        timestamp_str = parts[0]
11        size = int(parts[1])
12        data.append((timestamp_str, size))
13
14    return data
15
16 def convert_timestamp_to_nanoseconds(timestamp_str):
17     seconds, nanoseconds = timestamp_str.split('.')
18     dt = datetime.strptime(seconds, '%H:%M:%S')
19     total_nanoseconds = (dt.hour * 3600 + dt.minute * 60 + dt.second) * int(1e9) +
20        int(nanoseconds)
21    return total_nanoseconds
22
23 def calculate_delays(packet_data):
24     new_data = []
25     for i, (timestamp_str, size) in enumerate(packet_data):
26         timestamp_ns = convert_timestamp_to_nanoseconds(timestamp_str)
27
28         if i > 0:
29             prev_timestamp_ns = convert_timestamp_to_nanoseconds(packet_data[i -
30                1][0])
31             delay = (timestamp_ns - prev_timestamp_ns) / 1e9
32             if delay < 0:
33                 delay = 0
34             else:
35                 delay = 0
36
37             new_data.append((delay, size))
38
39    return new_data
40
41 def save_delays_to_file(new_data, output_file):
42     with open(output_file, 'w') as f:
43         for delay, size in new_data:
44             f.write(f'{delay}\t{size}\n')
```

```

45     input_file = 'packets.txt'
46     output_file = 'delays_and_sizes.txt'
47
48     packet_data = parse_file(input_file)
49     new_data = calculate_delays(packet_data)
50     save_delays_to_file(new_data, output_file)
51     print(f'Delays and sizes saved to {output_file}')
52
53 if __name__ == '__main__':
54     main()

```

B.6 Packet Transmission

```

1  import socket
2  import time
3  import matplotlib.pyplot as plt
4
5  # Shared data structures for logging timestamps and sizes
6  timestamps = []
7  sizes = []
8
9  def parse_delay_file(filename):
10     with open(filename, 'r') as f:
11         lines = f.readlines()
12
13     data = []
14     for line in lines:
15         parts = line.strip().split('\t')
16         delay = float(parts[0])
17         size = int(parts[1])
18         data.append((delay, size))
19
20     return data
21
22 def busy_wait(seconds):
23     time.sleep(seconds)
24
25 def send_packets(host, port, packet_data):
26     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
27         sock.connect((host, port))
28         for delay, size in packet_data:
29             busy_wait(delay)
30             message = b'\0' * size
31             sock.sendall(message)
32
33 def main():

```

```
34     filename = 'delays_and_sizes.txt'
35     server_ip = '18.170.31.216' #'127.0.0.1'
36     server_port = 5201
37
38     packet_data = parse_delay_file(filename)
39
40     if packet_data:
41         send_packets(server_ip, server_port, packet_data)
42     else:
43         print("No packet data found in the file.")
44
45 if __name__ == '__main__':
46     main()
```
